Universidad Rey Juan Carlos

Universidad Rey Juan Carlos
Madrid, Octubre 2020

Conducción autónoma de un robot con visión mediante aprendizaje por refuerzo.

Trabajo fin de Máster

Máster Oficial en Visión Artificial

Autor: Ignacio Arranz Águeda
Tutor: Jose María Cañas Plaza
Co-tutor: Eduardo Perdices

Departamento de Sistemas Telemáticos y Computación,
Universidad Rey Juan Carlos

Resúmen

Este Trabajo de Fin de Máster describe el desarrollo llevado a cabo para la construcción de un entorno de pruebas donde un modelo de un Fórmula-1, a través de algoritmos de Aprendizaje por Refuerzo es capaz de resolver el ejercicio del Sigue-Líneas utilizando visión computacional. Mediante diferentes ensayos con diferentes configuraciones de parámetros de acciones y percepciones se hará un estudio de la configuración que mejor resuelve el circuito en términos de tiempos por vuelta comparado con un robot que lo resuelve de manera explícita. Toda esta infraestructura se hará utilizando estándares del software libre en el mundo de la robótica como son ROS, para las comunicaciones, Gazebo para la representación tridimensional de la escena y Python como lenguaje de programación.

Palabras clave: Aprendizaje por refuerzo, Q-Learning, robótica, visión artificial, conducción autónoma, Python.

Abstract

This Master's Thesis describes the development carried out for the construction of a test environment where a model of a Formula-1, through Reinforcement Learning algorithms is able to solve the exercise of the Follow-the-Lines using computer vision. By means of different tests with different configurations of action and perception parameters, a study will be made of the configuration that best resolves the circuit in terms of lap times compared to a robot that resolves it explicitly. All this infrastructure will be done using free software standards in the world of robotics such as ROS for communications, Gazebo for the three-dimensional representation of the scene and and Python as a programming language.

Keywords: *Reinforcement Learning, Q-Learning, robotics, computer vision, autonomous driving, Python.*

Índice general

Índice de figuras

Índice de tablas

1
Introducción

Este Trabajo de Fin de Máster aborda el aprendizaje por refuerzo para el control visual de un robot. Se encuadra en la intersección de tres áreas de interés creciente: la visión artificial, el aprendizaje automático y la robótica.

En este capítulo se introduce el tema general del proyecto, aprendizaje por refuerzo para control visual de un robot y la motivación que lleva a la resolución del problema y se fijan los objetivos concretos a abordar.

1.1. Motivación y contexto

En la última década se está viviendo una explosión de los campos de la inteligencia artificial y la robótica en muchos de los ámbitos que nos rodean: social, político, médico, tecnológico, etc. Muchos de estos cambios promovidos por la inteligencia artificial que vive constantes renovaciones y revisiones de sí misma donde los repositorios que contienen el código con meses de antigüedad parezcan piezas de museo haciendo que el tiempo, en vez de años, parezcan siglos.

Por ejemplo, en el campo del procesamiento natural del lenguaje parece que fue el «siglo pasado» cuando OpenAI lanzó su popular modelo de generación de texto en su versión 2; (*Generative Pre-trained Transformer* o GPT-2) que tanto impresionó al mundo con los textos y conversaciones que tenían dos máquinas dándoles únicamente una pequeña semilla. Esto ocurrió a principios del 2019. Este modelo ha pasado a un segundo plano por culpa de su hermano mayor, el generador de texto en su versión 3 (GPT-3) que contiene una cantidad mayor de parámetros y es capaz de mantener conversaciones similares a las de un humano sin aparente esfuerzo.

El incremento del uso de la *inteligencia artificial* en cada vez más ámbitos viene también impulsado por el aumento de la capacidad de cómputo de los procesadores, mayor número de núcleos para el procesamiento, capaces de paralelizar más procesos, aceleración por hardware, aumento del número de librerías que facilitan el uso de esta tecnología (TensorFlow, Pytorch, etc) así como la reducción del coste de los componentes y sensores y la accesibilidad para realizar programas complejos con elementos casi domésticos.

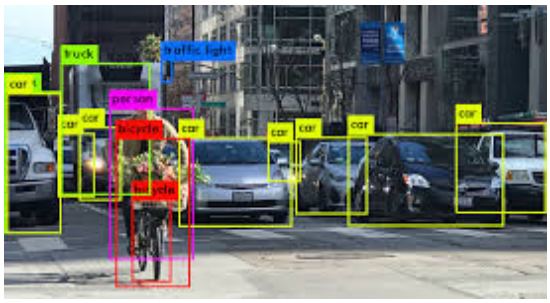
Esta explosión en el campo de la inteligencia artificial viene cimentada sobre una base que comenzó a construirse desde la creación de la considerada primera neurona artificial, el perceptrón (1958). Desde su nacimiento, el interés por el aprendizaje automático ha ido en constante crecimiento con la creación de diferentes algoritmos de aprendizaje automático *supervisado* como los clasificadores, árboles de decisión, máquinas de vector soporte

(*Support Vector Machine*, SVM), *no supervisado* como: agrupación (*clustering*), o la rama más psicológica del *aprendizaje por refuerzo* con algoritmos como QLearn, SARSA, etc.

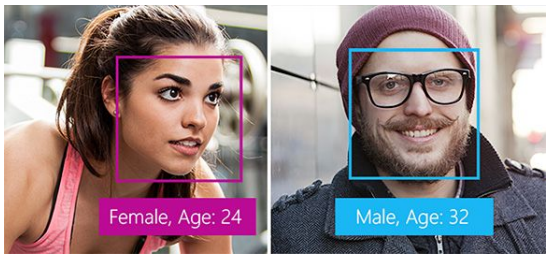
En el año 2012 se produce un nuevo crecimiento en este campo con la solución presentada en el reto del reconocimiento visual a gran escala con el conjunto de datos[1] «Image Net»[2], el *aprendizaje profundo* o *Deep Learning*. Con esta solución, muchos de los algoritmos conocidos evolucionan a su versión con la nueva arquitectura donde los resultados mejorarían drásticamente debido a la capacidad de generalización que se consigue, donde es la propia red neuronal la que aprende las características de los datos que van pasando por ella.

En al ámbito de la *visión por computador*, los avances han sido igualmente sorprendentes para un gran abanico de tareas como son: la segmentación (figura 1.1a), detección (figura 1.1b), reconocimiento (figura 1.1c), restauración (figura 1.1d), pose corporal, gestos, etc. En todos ellos, el estado actual está muy avanzado debido a las constantes revisiones de los trabajos y el incremento en capacidad de cómputo así como el interés que despierta el campo de la visión en muchos sectores como puede ser: industrial, civil, social, etc.

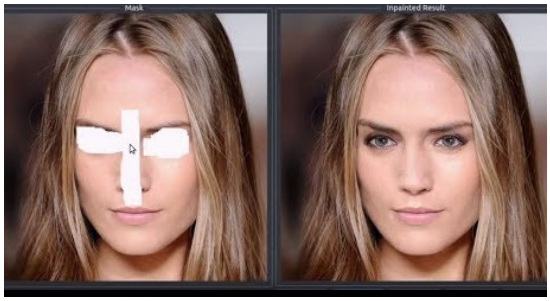
(a) Segmentación.



(b) Detección.



(c) Reconocimiento facial.



(d) Restauración.

Figura 1.1: Ejemplos de soluciones de visión artificial.

Por otro lado, la combinación de la robótica y la inteligencia artificial, genera soluciones que podemos ver en muchos campos como el médico (robot de cirujía DaVinci[3]), doméstico (aspiradores Roomba[4]), logístico (movimiento de mercancía en almacenes, como

[1] `http://image-net.org/challenges/LSVRC/`
[2] `http://www.image-net.org/`
[3] `http://www.abexsl.es/es/robot-da-vinci/que-es`
[4] `https://www.irobot.es/`

Amazon[5]), automovilístico (conducción semi-autónoma de Tesla Motors, Inc[6]), dotando de distintos tipos de sensores y en muchos de ellos con uno en común, el sensor de imagen o cámara. Y es en este punto donde se ha potenciado el uso de la inteligencia artificial y su capacidad para extraer *información valiosa* (características) de imágenes con precisión en función de la aplicación, habiéndose entrenado previamente para solucionar esa tarea en concreto. La reducción del coste de este componente facilita además su integración en todo tipo de dispositivos por lo que es más accesible integrar soluciones basadas en este campo de la inteligencia artificial, la *visión por computador*.

(a) Robot DaVinci. **(b)** Robots Aspiradores.

(c) Robots de almacén. **(d)** Coches semi-autónomos.

Figura 1.2: Aplicaciones robóticas.

La compañía Nvidia[7], la mayor empresa en la creación de tarjetas aceleradoras hardware, cuenta también con soluciones que combinan ambos campos. Cuenta con un departamento de enseñanza robótica a través de su robot Jetbot[8] (figura 1.3), que incorpora una tarjeta gráfica para la aceleración de los cálculos en los algoritmos de inteligencia artificial en la propia placa (Jetson Nano, Xavier, TX, etc) así como una cámara como sensor principal. Esto permite construir vehículos gobernados por algoritmos inteligentes a un coste cada vez más ajustado para resolver problemas como seguir una carretera o leer señales de tráfico para detener el robot gracias al sensor visual que le permite «ver» y «entender» los elementos que le rodean.

[5] https://www.wired.com/story/amazon-warehouse-robots/
[6] https://www.tesla.com/es_es
[7] https://www.nvidia.com/es-es/
[8] https://www.nvidia.com/es-es/autonomous-machines/embedded-systems/

Figura 1.3: Robot para robótica educativa de Nvidia, Jetbot.

A nivel industrial, el crecimiento de robots ha ido en aumento desde el 2018 [33] (un 6 % de incremento) y se pronostica que ese incremento vaya ampliándose año tras año. Además de la apuesta de países como China, Japón, República de Corea, Estados Unidos y Alemania por la robótica, el sector automovilístico está siendo el que más demanda la robótica a nivel mundial, promovido por la actualización de las cadenas de montaje así como el creciente impulso que se está destinando a los nuevos vehículos eléctricos. En la figura 1.4 puede verse un gráfico con la tendencia creciente en la última década en el sector.

Figura 1.4: Previsión de crecimiento de la robótica industrial desde el 2013. Fuente: World Robotics 2019.

La industria automovilística está revolucionando el mercado incorporando en los coches mecanismos de conducción, por el momento, semi-autónoma que gestionan el comportamiento y ayudan en la conducción y proporcionan un componente más de seguridad. Esta revolución dentro del sector ha sido protagonizado los últimos años por la empresa Tesla Motors, Inc, que combina todo lo mencionado en este punto: varias cámaras para recoger información, hardware que procesa la información del sensor, software que procesa, gestiona y actúa en base a unas decisiones tomadas por algoritmos de inteligencia artificial que procesan estas imágenes de entrada hasta en condiciones extremas.

jetbot-ai-robot-kit/

En la figura 1.5 puede verse un ejemplo muy simplificado de cómo las cámaras del vehículo puede detectar otros a su alrededor así como segmentar el carril por el que circula y sus aledaños.

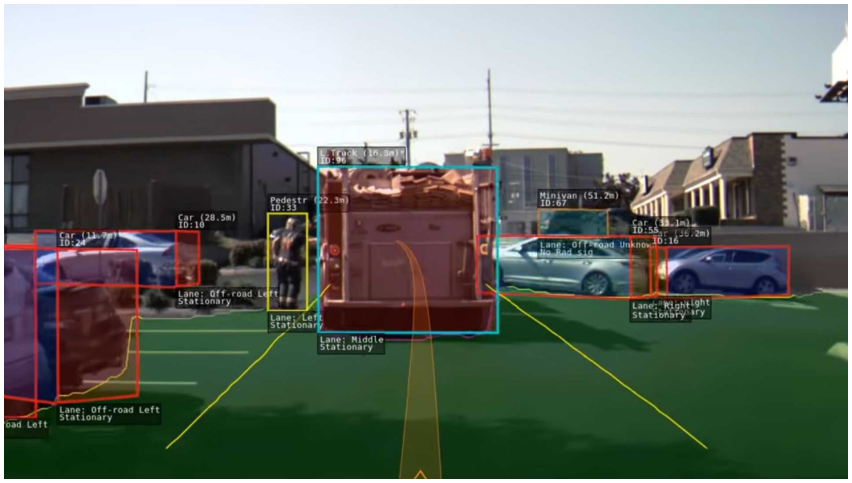


Figura 1.5: Visión de un vehículo Tesla a través de la cámara.

Esta detección puede extenderse a la de peatones, bicicletas, distancia de seguridad con vehículos posteriores, navegación automática, cambio de carril, etc.

Para conseguir este hito de la historia contemporánea ha hecho falta entrenar algoritmos de inteligencia artificial que sean capaces de tomar las mejores decisiones en cada una de las situaciones a las que se enfrentan. Para ello es necesario alimentar al sistema con imágenes de todo tipo de iluminación y visibilidad: lluvia, nieve, niebla, soleado, nublado, etc.

Estos procesos de entrenamiento pueden llevarse a cabo de varias maneras como puede ser: clasificar una imagen tomada de lluvia, segmentar el carril y tomar una decisión o, el tema de este trabajo, enseñar a vehículos a resolver el problema de seguir un carril o una línea utilizando otro tipo de aprendizaje automático: *el aprendizaje por refuerzo.*

1.2. Objetivos y requisitos

El objetivo principal del proyecto es entrenar un vehículo en un simulador para seguir la línea dibujada en el asfalto utilizando aprendizaje por refuerzo. El resultado de estos entrenamientos será un vehículo capaz de recorrer un circuito de manera autónoma y comprobar si es capaz de solucionar circuitos para los que no ha sido entrenado. Estos objetivos se dividen en:

1. Programar un entorno de aprendizaje por refuerzo con visión y robots.

2. Entrenar un controlador visual para conducción autónoma siguiendo una línea.

3. Validación experimental y análisis de las posibilidades del aprendizaje por refuerzo.

Todos los puntos mencionados son resueltos usando siempre la cámara como sensor de entrada de información que, mediante un procesamiento de cada imagen, se ofrecerá como

respuesta a los motores en forma de movimiento.

Para la elaboración del trabajo del fin de máster se listan los siguientes requisitos:

- Uso del sistema operativo robótico (*Robot Operating System*, ROS) como interfaz para la comunicación con el robot.
- Uso del simulador Gazebo para realizar las pruebas en diferentes circuitos.
- Emplear aprendizaje por refuerzo como herramienta de entrenamiento del robot a través del entorno Gym-Gazebo.

1.3. Metodología

Para este trabajo de fin de máster se ha empleado el método de desarrollo en espiral, muy usado en ingeniería de software y que permite establecer pequeños objetivos para cada una de la iteraciones, formado por un conjunto de actividades. Estas actividades consisten en establecer unos objetivos, planificados previamente en función de los cumplidos en la semana anterior.

Estructurados por prioridad, se estudia su diseño y se programa una solución. Una vez resuelto se procede a pruebas de los componentes y se evalúan los resultados y objetivos marcados de cara a proponer unos nuevos, comenzando una nueva iteración en el bucle. En la figura 1.6, se muestra un diagrama con la estructura de la metodología seguida durante el desarrollo del proyecto.

Figura 1.6: Método en espiral

Desde el comienzo de este trabajo, se han planificado reuniones semanales con los tutores en el que se analizaba el progreso y objetivos y se marcaban los de la siguiente semana siguiendo el esquema en espiral.

El código desarrollado se almacena en un repositorio público de GitHub (`https://github.com/RoboticsLabURJC/2019-tfm-ignacio-arranz`), perteneciente a

RoboticsLabURJC[9] de la Universidad Rey Juan Carlos donde se guarda un registro con todos los cambios. Además, en el mismo repositorio se crea un *blog*[10] sobre el proyecto en el que se escriben los avances semanales/mensuales desde el comienzo del desarrollo hasta la solución final.

1.4. Estructura del documento.

La estructura del trabajo presentado está compuesta por los siguientes capítulos:

- **Capítulo 1**: Se presenta una introducción al proyecto así como la motivación que ha llevado a resolver la tarea y los objetivos que se pretenden alcanzar.
- **Capítulo 2**: En el estado del arte se presenta el posicionamiento del proyecto dentro del marco de trabajos similares.
- **Capítulo 3**: Describe cada una de las herramientas empleadas en el proyecto.
- **Capítulo 4**: Programación del código de aprendizaje por refuerzo para el control visual.
- **Capítulo 5**: Estudios y resultados experimentales.
- **Capítulo 6**: Se extraen conclusiones de todo lo desarrollado anteriormente y la línea de trabajos por la que se pretende continuar con el proyecto.

Los anexos a este proyecto se utilizan para proporcionar la información de tablas, imágenes o fragmentos de código que complementen los resultados.

[9] `https://github.com/RoboticsLabURJC`
[10] `https://roboticslaburjc.github.io/2019-tfm-ignacio-arranz/`

2
Estado del arte

En este capítulo se hace una breve revisión de los fundamentos del aprendizaje por refuerzo y del actual estado del arte, presentando un panorama general de los métodos y utilidades que se abordan en este trabajo.

Como se presentó en el punto 1.1 este trabajo presenta una solución que combina el aprendizaje por refuerzo con un vehículo dentro de un simulador para recorrer un circuito siguiendo una franja en el suelo. En este capítulo se recogen algunos de los métodos planteados así como las ventajas e inconvenientes de cada uno de ellos hasta el uso actual.

2.1. Aprendizaje por refuerzo

Existen diferentes áreas de aprendizaje automático que permiten a un sistema generar una salida dada una entrada desconocida. En función del problema que se quiere solucionar, los entrenamientos en aprendizaje automático generan una u otra salida como puede ser un valor numérico, una probabilidad, una imagen (dentro del campo de la visión por computador), o una *recompensa*. Es en este punto donde se encuentra la rama del *aprendizaje por refuerzo* (*reinforcement learning* o RL) que tiene un enfoque psicológico en el proceso de entrenamiento del algoritmo. En este aspecto se diferencia de los métodos de entrenamiento supervisados y es el propio agente el que se encarga de encontrar la mejor relación de parámetros para la solución del problema.

El aprendizaje por refuerzo puede definirse como: aprender qué hacer o cómo relacionar las situaciones (o *estados*) con las *acciones*, con el objetivo de maximizar una señal numérica de *recompensa*. En este ámbito se describe uno de los elementos principales del aprendizaje, *el agente*. El agente debe ser capaz de adquirir información del entorno y tomar decisiones que afecten a su estado. No se le dice qué acciones debe tomar, sino que debe descubrir qué acciones producen la *mayor recompensa* cuando se llevan a cabo. Estas acciones pueden afectar no solo a la recompensa inmediata sino también a la siguiente situación y, a través de ella, a todas recompensas venideras.

Las tareas de toma de decisiones secuenciales cubren una amplia gama de posibles aplicaciones como: la robótica, asistencia sanitaria, las redes inteligentes, videojuegos, finanzas, conducción autónoma y muchas otras.

El aprendizaje por refuerzo proporciona a los agentes la capacidad de realizar experimentos para comprender mejor su entorno, lo que les permite aprender incluso relaciones causales de alto nivel. Las mejoras en *renderizadores* visuales y motores de física permiten

replicar cada vez con más fiabilidad entornos reales donde dejar que los agentes exploren y aprendan la mejor política o combinación de parámetros para aprender la tarea en cuestión. En la figura 2.1 se ilustra un entorno de pruebas del juego del 'pilla-pilla' donde los agentes que se esconden pueden mover paneles para ocultarse mejor. Pasados muchos episodios, los agentes rastreadores aprenden a mover esos paneles para encontrar a sus rivales.

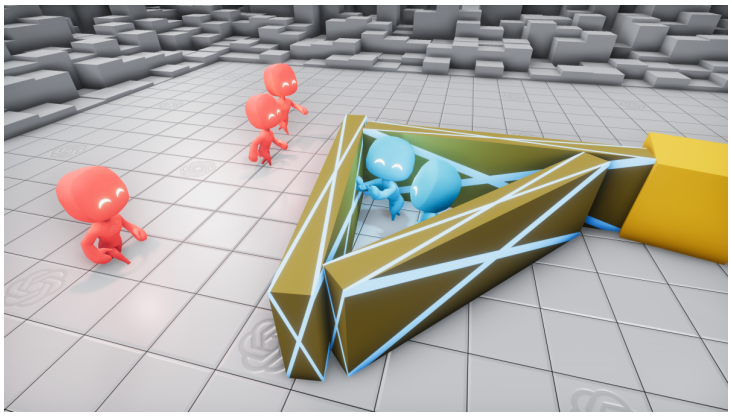


Figura 2.1: Juego de "pilla-pilla". Unos agentes buscan y otros se esconden.

En el ámbito concreto que ocupa este Trabajo Fin de Máster, combinar este aprendizaje con un vehículo para solucionar el problema de control visual de completar una vuelta al circuito, supone un entorno de pruebas seguro donde el vehículo puede llegar a los límites admitidos y permite reiniciarse el entorno de simulación si llega a un extremo no deseado.

Aún quedan retos por delante para que esto sea posible en el mundo real, pero se está progresando en la idea de que los agentes aprendan los principios fundamentales del mundo a través de la observación y la acción. Esto permitirá aproximarse a sistemas de inteligencia artificial que aprendan y actúen de manera más humana en entornos cada vez más complejos.

2.1.1. Comportamiento recompensado

La esencia del aprendizaje por refuerzo es aprender a través de la interacción con el entorno, observando las consecuencias de haber realizado una acción para poder alterar su propio comportamiento en respuesta a las recompensas recibidas. Este paradigma hereda de la psicología y es conocido como *ensayo-error* siendo una de las principales características en el campo del aprendizaje por refuerzo.

Como se explica en [8], la interacción con el entorno es llevada a cabo por un *agente*, controlado por algoritmos de aprendizaje automático que observa el estado s_t de su entorno en un instante t. El agente interactúa con el entorno realizando una acción a_t en el estado s_t. Cuando el agente ha realizado la acción a_t, el entorno y el agente transitan a un nuevo estado s_{t+1} basado en el estado actual y la acción elegida. La mejor secuencia de acciones está determinada por las recompensas que ofrece el entorno.

El objetivo del agente es *aprender una política* o estrategia que maximice la recompensa. El reto del aprendizaje por refuerzo es conseguir que el agente aprenda de las consecuencias de las acciones en el entorno a través de *ensayo-error* ya que no existe un método de transición entre estados. Este bucle de *percepción-acción-aprendizaje*, explicado en [8], puede verse en la figura 2.2.

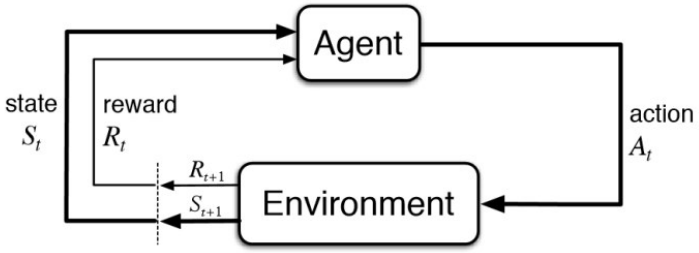


Figura 2.2: Bucle de percepción-acción-aprendizaje llevado a cabo por el agente.

Estas dos características (la búsqueda a través de *ensayo-error* y la recompensa) son las más diferenciales del aprendizaje de refuerzo.

2.1.2. Soluciones basadas en matrices o tablas

Uno de los tipos de soluciones del aprendizaje por refuerzo es aquel en el que el estado y los espacios de acción son lo suficientemente pequeños para que las funciones de valor aproximado sean representadas como matrices o tablas. En estos casos, los métodos suelen encontrar soluciones o la función de valor óptima y la política óptima. Los tipos de soluciones basados en tablas se mencionan en [3] y son los siguientes:

- ***Multi-armed bandits*** [17]: Denominado así por analogía con máquinas tragaperras (bandido con un solo brazo) con la salvedad de que se tienen k palancas en lugar de una. La selección de una acción es como una tirada en una máquina siendo las mejores recompensas los mejores premios. El objetivo es concentrarse en las máquinas que retornan mejores premios (valores) descartando las que retornan peores.

- **Procesos de decisión finitos de Markov (MDP)** [19]: Este caso implica también una retroalimentación evaluativa (como el caso anterior) y además una asociativa. Es una formalización de la tomar de decisiones secuenciales donde la elección de distintas acciones en diferentes situaciones hacen que los resultados no solo influyan en las recompensas inmediatas sino que también en las futuras.

- **Programación Dinámica (DP)** [20]: Conjunto de algoritmos que pueden ser utilizados para calcular políticas óptimas requiriendo de un modelo muy preciso del entorno. Este método resuelve problemas MDP. Es un algoritmo que se toma como base para la comprensión del resto de algoritmos, siendo estos intentos por mejorar el coste computacional y la suposición del modelo perfecto del entorno que se impone en la programación dinámica.

- **Métodos de Monte Carlo** [17]: Este método se usa para resolver el MDP. No requiere conocimiento del entorno y logra comportamientos muy buenos. Se basa en el promedio de los resultados de las muestras. Conceptualmente es sencillo pero

no es adecuado para el cálculo incremental paso a paso.

- **Aprendizaje de Diferencia Temporal** [18]: Es un método más novedoso. Siendo una combinación de la idea de Monte Carlo y programación dinámica puede aprender directamente de la experiencia en bruto sin un modelo de la dinámica del entorno. Igualmente, se usan para resolver problemas MDP. Son más complejos de analizar.

2.1.3. Métodos de solución aproximada

En muchas de las tareas en la que se pretende aplicar aprendizaje por refuerzo, el espacio de estados es combinatorio y enorme: ingentes cantidades de imágenes tomadas de distintas cámaras. En tales casos, no puede esperarse encontrar una política óptima o una función de valor óptima incluso con un límite de tiempo infinito. En contraste con el método anterior el objetivo es encontrar una buena *solución aproximada* utilizando recursos computacionales limitados.

Para tomar decisiones sensatas en grandes conjuntos de estados es necesaria la generalización. ¿Cómo puede la experiencia con un subconjunto limitado de estados producir una buena aproximación sobre un subconjunto mucho más grande?. Tal y como se menciona en [3], utilizando *funciones de aproximación*.

Existen diversos tipos de funciones de aproximación como:

- **Predicción de valores**: usado en general en cualquier tipo de aprendizaje supervisado como árboles de decisión, regresión multivariante, etc, donde suele emplearse la métrica de la raíz del error cuadrático medio (RMSE) para la evaluación de las medidas.

- **Métodos de descenso-de gradiente**: Está entre los métodos más comunes de aproximación y es una de las piezas claves del aprendizaje profundo. Se calcula el error a través de las derivadas parciales en sentido opuesto al flujo de la red neuronal.

- **Métodos lineales**: Es un caso particular del método del descenso del gradiente donde la función de aproximación es una función lineal.

2.2. Algoritmos clásicos de aprendizaje por refuerzo

En el aprendizaje por refuerzo el objetivo es aprender sin tener un modelado completo del entorno. Sin embargo, las interacciones con el entorno pueden ser usadas para aprender funciones de valor, políticas y, también, modelos.

En general existen dos aproximaciones para solventar problemas de aprendizaje por refuerzo:

- Métodos que **no** están basados en modelos.
- Métodos basados en modelos.

2.2.1. Métodos sin modelos

Los métodos que no están basados en modelos aprenden directamente de las interacciones con el entorno, como pueden ser: los métodos basados en *funciones de valor* (*value functions*), la *búsqueda de políticas* (*policy search*) o métodos híbridos como el *actor-crítico* (*actor-critic*).

- **Funciones de valor**: Estos métodos están basados en la estimación del valor (rendimiento esperado) de estar en un estado s dado. Se denomina función de *valor-estado* $V^{\pi}(s)$. La política óptima se define como π^*. La función $Q^{\pi}(s, a)$ representa la función de calidad o *state-action-value* (similar a V^{π}). La mejor política dará como resultado la mejor $Q^{\pi}(s, a)$. Esta función de calidad se mejora por diversos métodos de estimación dando lugar a la función denominada $Q - learning$[4] y al algoritmo SARSA[8] (*state-action-reward-state-action*) que sigue la ecuación:

  $$Q^{\pi}(s_t, a_t) \leftarrow Q^{\pi}(s_t, a_t) + \alpha\delta.$$

  Donde:
  - $\alpha \rightarrow$ es la tasa de aprendizaje,
  - $\delta \rightarrow$ es el error en la diferencia temporal (TD).

  El valor $Q - learning$ se aproxima directamente a Q^* y se encuentra a base de iteraciones con una evaluación y mejora de las políticas que mejora la estimación de la *función de valor*. A medida que la estimación mejora la política puede mejorarse eligiendo acciones basadas en la *función de valor* actualizada.

- **Métodos de búsqueda de políticas**: No necesitan mantener un modelo de la *función de valor* sino que se busca directamente una política óptima π^*. Las redes neuronales que codifican las políticas han tenido excelentes resultados usando en el entrenamiento ambos métodos: sin el uso del gradiente[11] y basado en gradiente[9]. El primero de ellos cubre eficazmente los parámetros cuando las dimensiones son reducidas pero con superficies grandes, el entrenamiento basado en gradiente es el mejor método actualmente para la mayoría de los algoritmos de aprendizaje por refuerzo profundo (*Deep Reinforcement Learning*, (DRL)) cuando las políticas tienen un gran número de parámetros. Los métodos que no están basados en gradientes requieren búsquedas heurísticas para encontrar la mejor política pero, tienen como ventaja que también puede optimizar políticas no diferenciables.

  Las *políticas de gradientes* [12] proporcionan una tasa de aprendizaje muy fuerte para mejorar políticas que llevan asociadas parámetros. La configuración más común de aprendizaje por refuerzo sin modelos es el uso del algoritmo de *Monte Carlo*.

- Los métodos híbridos de actor-crítico (*actor-critic*): combinan los valores con una representación explícita de la política. El *actor* (política) aprende utilizando la retroalimentación del crítico (valor). En este proceso, los métodos compensan la desviación de reducción de los gradientes con las políticas con introducción de sesgo a partir de las funciones de valor.

2.2.2. Métodos basados en modelos

Pueden simular transiciones usando el modelo aprendido [13] que se traduce en una mayor eficiencia en la muestra. Estos métodos basados en modelos son particularmente importantes en entornos donde cada interacción con el entorno es especialmente costosa aunque introduce más complejidad en la creación y estructura del sistema (es más delicado porque al existir más parámetros el modelo puede contener errores). La solución a los problemas basados en modelos que se aporta en esta nueva etapa es el uso de las redes neuronales profundas que permiten crear modelos muy complejos y ricos de manera eficiente y que se explican con más detalle a continuación.

2.3. Aprendizaje profundo en aprendizaje por refuerzo

Denominado como *aprendizaje por refuerzo profundo* (*Deep Reinforcement Learning*, DRL en adelante), muchos de los éxitos de esta técnica se han basado en la *ampliación de los trabajos previos* de aprendizaje por refuerzo *a problemas de alta dimensión* permitiendo el aprendizaje de representaciones de características de baja dimensión así como la aproximación funcional de las redes neuronales [1]. Mediante el aprendizaje de la representación se hace frente al problema de la maldición de la dimensionalidad [14] que ocurría en los métodos basados en tablas y los tradicionales no paramétricos. El uso de redes neuronales convolucionales (*convolutional neural network*, CNN) hace que puedan utilizarse como componentes de los agentes permitiéndoles aprender directamente de las imágenes crudas (componentes visuales de alta dimensión) o con menor procesamiento. En general, el aprendizaje por refuerzo profundo trata de aproximar una política óptima de π^* o las funciones de valor óptimo V^*, Q^* y A^*.

Los trabajos actuales se basan mayoritariamente en métodos basados en gradiente [10] dado que proporcionan una fuerte señal de aprendizaje. El éxito del DRL radica en el aprendizaje de la representación y la aproximación a las funciones. Este auge en el aprendizaje profundo ha inspirado nuevas formas de pensar y trabajar sobre el aprendizaje profundo lo que lleva a revisar las funciones de valor y búsqueda de políticas. Una de las más conocidos es *Deep Q-network* (DQN).

2.3.1. Métodos no basados en modelos

Al igual que en la sección 2.2, se visitan los métodos basados en valores y en políticas:

- **Métodos basados en funciones de valor**: Desde los primeros métodos de función de valor en DRL, que tomaban estados simples como entrada, los métodos actuales son ahora capaces de abordar entornos visual y conceptualmente complejos.

  El algoritmo DQN se convirtió en uno de los más populares en esta evolución en el uso de redes neuronales ya que logró conseguir *records* de puntuación en los procesos de aprendizaje llevados a cabo en la videoconsola Atari [5], [6] comparables con un jugador profesional utilizando imágenes en escala de gris como entrada (figura 2.3).

Cada una de las capas convolucionales completamente conectadas procesan estas imágenes que codifican los efectos de las acciones en su salida.

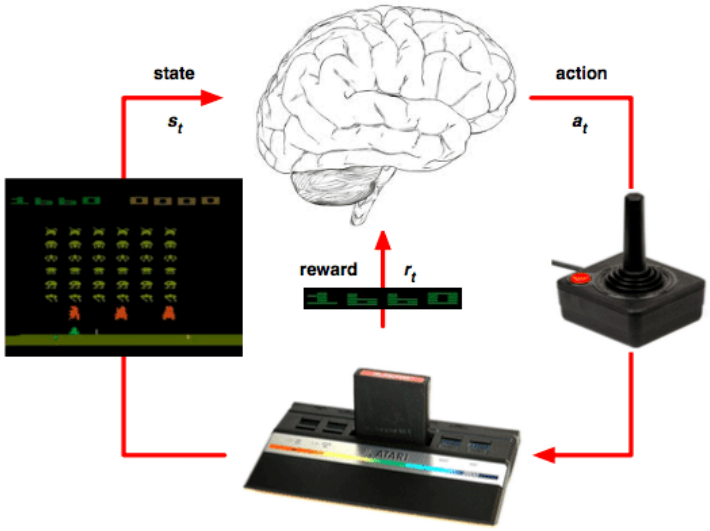


Figura 2.3: Bucle de percepción-acción-aprendizaje llevado a cabo por el agente para la resolución de juegos en la videoconsola Atari.

- **Métodos basados en políticas**: Los métodos de búsqueda de políticas tienen como objetivo encontrar directamente las políticas por medio de métodos o bien *libres de gradientes* o *basados en gradientes*. Antes del actual aumento de interés en DRL, varios métodos en este ámbito evitaron el algoritmo de retropropagación o propagación hacia atrás (en inglés, *backpropagation*) comúnmente utilizado en favor de los algoritmos evolutivos que potencialmente pueden ser distribuidos a escalas mayores que las técnicas que se basan en gradientes.

- **Métodos del actor crítico (*Actor-critic*)**: Combina los beneficios de los métodos de búsqueda de políticas con lo aprendido en las funciones de valor y son capaces de aprender de los errores retornados de TD [15]. En los últimos años, los métodos de DRL de actores críticos se han ampliado desde el aprendizaje de tareas de física simulada a tareas reales de navegación visual robótica, directamente desde los píxeles de la imagen. Un desarrollo más reciente de este tipo de métodos son los gradientes determinísticos (*Deterministic Policy Gradients*, DPG en adelante) de las políticas que extiende los teoremas de política de gradientes. Una de las mayores ventajas es que, mientras que los gradientes de políticas estocásticas se integran tanto en los espacios de estados como en los de acción, los DPG solo se integran en los primeros requiriendo menos muestras en problemas con grandes espacios de acción.

2.4. Investigación actual y retos

Existen múltiples áreas activas de investigación acerca de diferentes métodos y maneras de alcanzar un aprendizaje por refuerzo:

- **Aprendizaje por refuerzo basado en modelo**: Mencionado en el punto 2.2.2, permite aprender los modelos de transiciones que dispone el entorno pero sin la interacción directa con él [22]. No asume conocimiento a priori. Sin embargo, puede incorporar información para acelerar el proceso de aprendizaje. Dado que es poco realista realizar millones de experimentos con un robot en un tiempo razonable existen varios enfoques para aprender modelos predictivos utilizando información de los píxeles. Se incorpora información de alta dimensión que se reduce con autocodificadores (*autoencoders*). En esencia, si se puede aprender un modelo suficientemente preciso del entorno entonces los controladores más sencillos pueden ser usados para controlar un robot directamente desde las imágenes de la cámara. Utilizando aprendizaje profundo es más viable simular entornos sobre cientos y cientos de muestras que, la mayoría de las veces, son caras o difíciles de conseguir. Es una vía muy eficiente de entrenar modelos y mejorar la eficiencia de los parámetros. Un ejemplo de uso de este tipo del aprendizaje por refuerzo basado en modelo es el presentado por Łukasz Kaiser y su equipo [31] donde se pretende enseñar a un agente a completar videojuegos de la Atari a través de muy pocas visualizaciones para intentar igualar en tiempo de aprendizaje a un humano, en apenas 2 horas. El trabajo se llama SimPLe y en la figura 2.4 puede verse el bucle principal del programa.

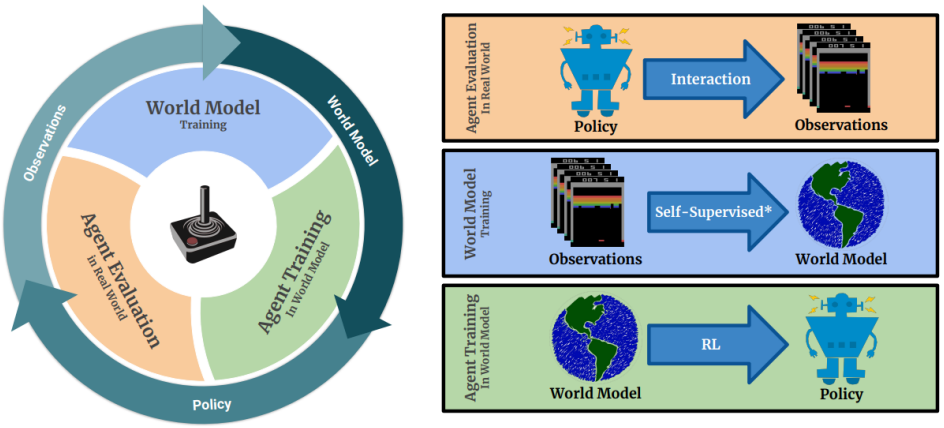


Figura 2.4: Simulación de entorno para la aceleración del aprendizaje.

En cada iteración:

1. El agente comienza a interactuar con el entorno real siguiendo el última política (inicializada al azar).

2. Las observaciones recogidas se utilizarán para entrenar (actualizar) el mundo actual modelo.

3. El agente actualiza la política actuando dentro del modelo mundial. La nueva política será evaluada para medir el rendimiento del agente así como recoger más datos.

4. Vuelve al primer estado.

Destacar que el entrenamiento del modelo global es auto-supervisado para los estados observados y supervisado para la recompensa.

- **Entrenamientos a través de los sueños**: El grupo de investigación DeepMind de Google[1] presentó una manera alternativa de entrenar algoritmos de aprendizaje por refuerzo. La situación que planteaban era sustituir un entorno de aprendizaje por refuerzo real por uno generado, entrenando al controlador del agente dentro de ese entorno creado por él, adquiriendo el aprendizaje en el mundo interno y transferir la política al entorno real. En el propio artículo de *World Models* [32] permiten «jugar» con los parámetros para ver estos resultados[2]. Puede verse una representación del mundo generado internamente en la figura 2.5.

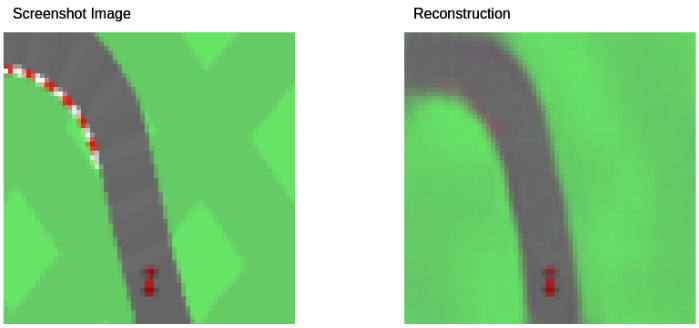


Figura 2.5: «World Models». Generación de un entorno artificial para aprendizaje por refuerzo.

- **Aprendizaje por refuerzo jerarquizado**: Igual que el aprendizaje profundo, se basa en las jerarquías de características. El aprendizaje por refuerzo jerarquizado (*Hierarchical Reinforcement Learning*, *HRL*) se basa en jerarquías de políticas donde, además de las acciones primitivas, también pueden aplicarse otras políticas (acciones en varias etapas). Este enfoque permite que las políticas de alto nivel se centren en objetivos de más alto nivel, mientras que las subpolíticas son responsables de un control preciso. El descubrimiento y la generalización de los objetivos es también un área importante de investigación en curso.

  En el trabajo de [29] se ven diferentes entornos donde un agente tiene que resolver un problema principal mientras resuelve problemas secundarios pero necesarios para finalizar el ejercicio. Puede verse un fragmento de una ilustración del trabajo en la figura 2.6.

- **Imitación y aprendizaje por refuerzo inverso**: El objetivo del aprendizaje por refuerzo inverso *Inverse Reinforcement Learning*, IRL en adelante) e imitación (*Imitation Learning*) es estimar una función de recompensa desconocida de las trayectorias observadas que caracterizan una solución [24]. IRL puede ser usado en combinación con aprendizaje por refuerzo para mejorar en un comportamiento demostrado. Con el uso de las redes neuronales es posible aprender recompensas complejas y no lineales. Se demostró que las políticas se caracterizan de manera

[1] `https://deepmind.com/`
[2] `https://worldmodels.github.io/`

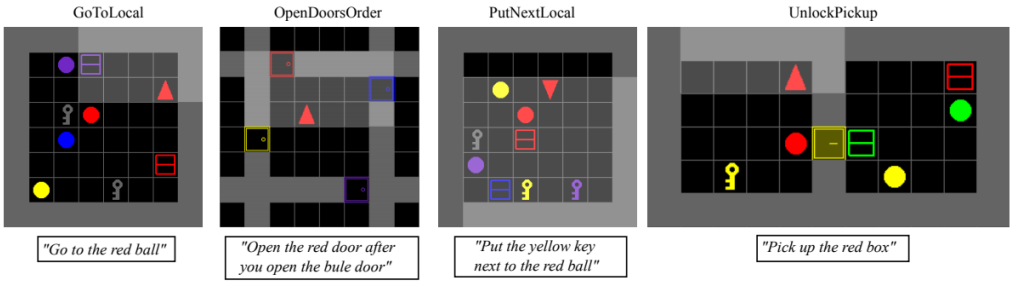


Figura 2.6: Aprendizaje por refuerzo jerarquizado. Resuelve pequeñas sub-tareas para resolver una mayor.

única por sus ocupaciones (estados visitados y distribución de acciones), lo que permite reducir el IRL al problema de la correspondencia de las medidas. Con esto y el uso de redes adversarias generativas se facilita recompensar el aprendizaje funcional de una manera más flexible, lo que resulta en el algoritmo de aprendizaje por imitación generativa adversaria (*Generative Adversarial Imitation Learning*, GAIL). Este tipo de técnicas son muy empleadas para el entrenamiento de redes para el aprendizaje de coches autónomos, donde dado un comportamiento el coche «hereda» la información como en el trabajo de [30] con los coches autónomos.

- **Multi-agente**: El aprendizaje por refuerzo multiagente (*Multi-Agent Reinforcement Learning*, MARL en adelante) considera el aprendizaje de múltiples agentes [26] y la no estacionariedad introducida por otros agentes cambia su comportamiento a medida que aprenden. En DRL, el enfoque se ha centrado en permitir la comunicación (diferenciable) entre los agentes, lo que les permite cooperar. Un trabajo muy popular del multiagente es la solución que presentó OpenAI-Gym al juego del pilla-pilla (o *hide and seek*) [32]. En esta solución, un grupo de agentes adquiere el rol de «buscadores» para tratar de encontrar a otro grupo de agentes que se ocultarán en el entorno permitiéndose incluso el movimiento de objetos dentro del escenario como ayuda de camuflaje (paredes, cajas, etc). Una vez pasado el periodo de entrenamiento los agentes encargados de buscar consiguen encontrar a sus rivales que a su vez adquieren mejores habilidades para esconderse. Puede verse un conjunto de situaciones en la figura 2.7.

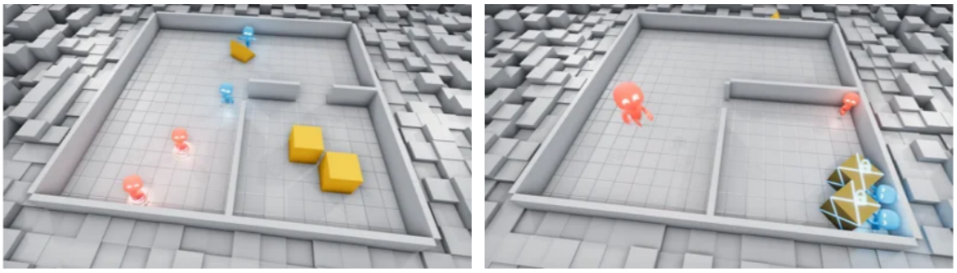


Figura 2.7: Juego de *hide and seek* utilizando aprendizaje por refuerzo profundo (OpenAI-Gym).

Lo sorprendente de este trabajo es que cuando aumentan considerablemente los episodios de entrenamiento (las partidas) ambos grupos llevan al extremo el entorno de simulación llegando incluso a sobrepasar los límites de las físicas desarrolladas

originalmente y todo con el objetivo de buscar o esconderse mejor. En la figura 2.8 puede verse al agente encargado de buscar cómo utiliza uno de los objetos para elevarse y encontrar mejor a los agentes escondidos.

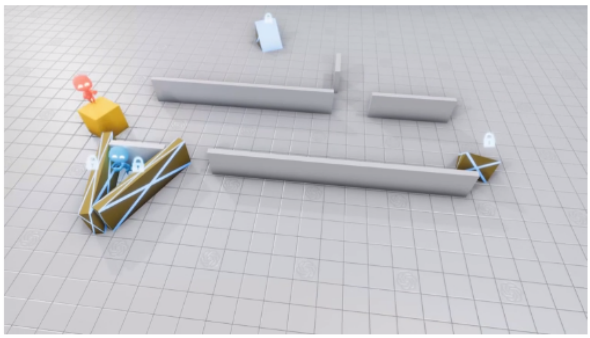


Figura 2.8: Juego de *hide and seek* utilizando aprendizaje por refuerzo profundo (OpenAI-Gym).

- **Transferencia de conocimiento (*transfer learning*)**: Aunque los algoritmos DRL pueden procesar entradas de alta dimensión, raramente es factible entrenar a los agentes de RL directamente en entradas visuales en el mundo real, debido al gran número de muestras requeridas. Para acelerar el aprendizaje en DRL, es posible explotar los conocimientos adquiridos previamente a partir de tareas relacionadas, que se presentan en varias formas: transferencia aprendizaje, aprendizaje multitarea [27] y aprendizaje curricular [25], entre otros. Es muy interesante transferir el aprendizaje de una tarea a otra, particularmente del entrenamiento en simuladores de física con renderizadores visuales para el posterior ajuste de los modelos en el mundo real. Esto puede lograrse utilizando directamente la misma red tanto en la fase de simulación como en la fase real con procedimientos de formación más sofisticados que intentan mitigar directamente el problema de las redes neuronales "olvidando" el viejo conocimiento añadiendo capas adicionales al transferir el dominio (cabezal).

  En la figura 2.9 puede verse un diagrama donde de izquierda a derecha se entrena la misma red con pesos vacíos (sin conocimiento previo), una red central con un entrenamiento para resolver el problema de «Image-Net» y ajustando en un nuevo entrenamiento la última red al problema que se quiere resolver. Por ejemplo, clasificar imágenes no vistas por el conjunto de datos original pero con todo el conocimiento adquirido del entrenamiento previo.

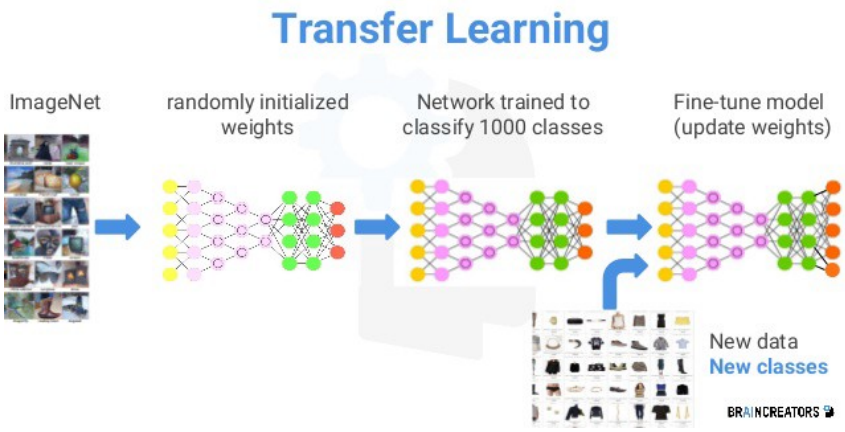


Figura 2.9: Transferencia de aprendizaje.

3
Infraestructura

En este capítulo se presentan las tecnologías que se han usado para el desarrollo de este Trabajo de Fin de Máster, el entorno donde se realizan las pruebas, las librerías utilizadas, los algoritmos y el lenguaje de programación.

Los entornos de pruebas elegidos son actualmente el referente del software libre de la robótica a nivel mundial. Para este trabajo se ha elegido este conjunto por la buena integración que tienen entre sí y la accesibilidad para que otros elementos puedan integrarse en ellos

3.1. Lenguaje Python

Python[1] es un lenguaje de programación interpretado cuya filosofia se basa en una sintaxis que favorezca un código legible. Es de tipado dinámico, multiplataforma y soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Ofrece versatilidad y alta compatibilidad con el resto de componentes de la infraestructura.

Python se ha adoptado como un lenguaje estándar para el desarrollo de algoritmos de inteligencia artificial. Gracias a la sintaxis clara, permite crear prototipos y proyectos de una manera ágil y relativamente sencilla lo que aumenta el interés por su uso. Este asentamiento de Python favorece la creación de librerías que ayudan a la creación de elementos más grandes, evitando un desarrollo inicial en cada proyecto.

Este Trabajo de Fin de Máster utiliza Python en su versión 2.7 (debido a las limitaciones del resto de componentes) para el desarrollo de todo el código del algoritmo de aprendizaje por refuerzo.

3.2. Biblioteca Numpy

Estándar asentado como librería de procesamiento de datos, es una extensión del lenguaje Python para operaciones con vectores y matrices. Numpy[2], en su versión 1.16.6 se utiliza en el proyecto para facilitar los cálculos en la etapa de procesamiento de la imagen de entrada.

[1] `https://www.python.org/download/releases/2.7/`
[2] `https://numpy.org/`

Numpy permite trabajar con estructuras matriciales N-dimensionales con relativa facilidad si se compara con las herramientas nativas de Python. Cuenta con un gran repertorio de métodos para gestionar los datos como: sistema de indexado y búsqueda, formateo de los datos, manipulación de la forma del dato, etc, lo que simplifica las operaciones con matrices o imágenes en este caso.

3.3. Biblioteca OpenCV

Para el procesamiento de las imágenes que toma el robot se utiliza la librería estándar en aplicaciones con visión artificial OpenCV[3] (*Open Computer Vision*) en su versión 4.2.0. Fue creada inicialmente por Intel y liberada bajo software libre con licencia BSD. Es una librería para aplicaciones de visión artificial de propósito general que incluye muchísimas funciones de procesamiento de imágenes de todo tipo desde erosión, dilatación, convolución, flujo óptico, seguimiento, etc. Es multiplataforma y disponible en los lenguajes C++, Python y Java.

OpenCV hace hincapié en la baja latencia en su procesamiento para conseguir tiempos muy bajos y asegurar el tiempo real en los algoritmos que tiene incluidos por lo que es perfecta para usarse en software robótico.

Esta librería es usada en el código para el procesamiento de la imagen de entrada que captura la cámara incorporada en el robot.

3.4. Entorno robótico ROS

Como librería en comunicación entre el robot (simulado) y el entorno donde se llevan a cabo el entrenamiento con aprendizaje por refuerzo se elige el *sistema operativo robótico* (*Robot Operating System*, *ROS*[4] en adelante), en su versión *Melodic Morenia*, programado en C++ y Python y que está mantenido por la Fundación de Robótica de Código Abierto (*Open Source Robotics Foundation*, OSRF). ROS es un entorno para el desarrollo de software para robots que adquiere la funcionalidad de un sistema operativo dado que contiene elementos que así lo caracterizan como son: abstracción a nivel de hardware, control de dispositivos de bajo nivel, implementación de funcionalidad de uso común, paso de mensajes entre procesos y mantenimiento de los paquetes adicionales que pueden integrarse en él.

ROS gestiona los robots como *nodos* que permiten ejecutar programas independientes en cada uno de ellos separando la lógica entre *publicador* y *suscriptor*. El *publicador* (el robot) emite eventos que ROS denomina *topics* a los cuales el programa se conecta como *suscriptor*. Para este proyecto el sensor utilizado en el vehículo es una cámara simple adherida en la parte delantera de un robot modelado como un Fórmula 1 y que recogerá la información frontal mientras avanza. El suscriptor se conecta a uno de los *topics* emitido por ROS para la cámara y recoger así la información sensorial y a otro *topic* para la

[3] `https://opencv.org/`
[4] `https://www.ros.org/`

gestión de los motores poder ordenarle movimiento al robot.

El puente que se emplea entre la cámara en el vehículo y el programa es un paquete denominado `cv_bridge` que *traduce* la información recogida por la cámara a valores numéricos interpretables por el lenguaje de programación y compatible con las librerías de Numpy y OpenCV.

3.5. Simulador robótico Gazebo

El entorno de pruebas en el cual se refleja el comportamiento del robot es el simulador robótico de código libre **Gazebo**[5] en su versión 9, mantenido también por la OSRF. Las principales características de este simulador son:

- Es en 3D.
- Permite simular multitud de sensores como cámaras, láser, etc.
- Muy buena integración con ROS ofreciendo interfaces para los robots simulados.
- Es multirobot.
- Incluye un motor de físicas realistas[6] como: Bullet[7], *Open Dynamics Engine (ODE)*[8], *Dynamic Animation and Robotics Toolkit* (DART) [9], etc.
- Soporta muchísimos modelos diferentes de robot y permite modelarse utilizando URDF[10].

Por todas las características mostradas, es uno de los simuladores más usados en la comunidad investigadora internacional en robótica.

A través de los lanzadores de ROS (`roslaunch`) se configura el entorno donde se lanza la simulación y el mundo que contiene el robot así como los sensores y actuadores. En la figura 4.2b puede verse un ejemplo de un *mundo* de Gazebo empleado en este Trabajo Fin de Máster.

3.6. OpenAI Gym

OpenAI Gym[11] es una plataforma perteneciente a la empresa OpenAI que permite construir *entornos* de investigación para probar diferentes algoritmos de aprendizaje por refuerzo. Está escrito en lenguaje Python y ofrece una capa de abstracción que permite al usuario elaborar espacios donde construir su código utilizando el mismo repertorio de funciones para cada uno de los entornos convirtiéndose en la plataforma de referencia en el aprendizaje por refuerzo.

[5] `http://gazebosim.org`
[6] `http://gazebosim.org/blog/four_physics`
[7] `https://pybullet.org/wordpress/`
[8] `https://www.ode.org/`
[9] `https://dartsim.github.io/`
[10] `http://wiki.ros.org/urdf`
[11] `https://gym.openai.com/`

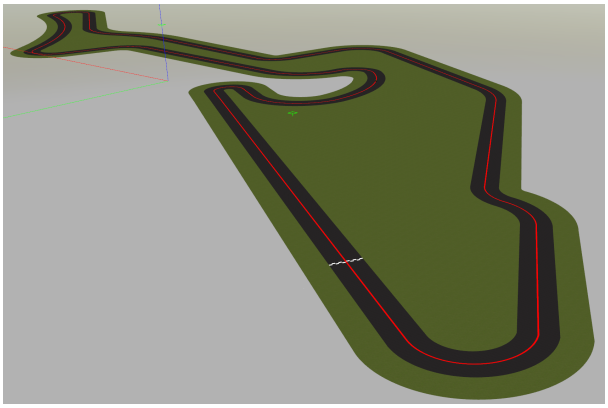


Figura 3.1: Réplica del circuito de Nürburgring en el simulador Gazebo.

Gym es popular por tener en su repertorio entornos de videojuegos procedentes (en su mayoría) de la consola Atari (figura 3.2). A través de los distintos algoritmos de aprendizaje por refuerzo se demuestra que en cada entorno (o videojuego) funciona mejor un algoritmo en concreto para solucionar o completar el juego. Con los escenarios preparados, el usuario únicamente tiene que encargarse de programar la lógica del agente.

En la figura 3.2 se muestran algunos de los entornos de los que dispone OpenAI Gym para la ejecución de diferentes algoritmos de aprendizaje por refuerzo y por refuerzo profundo como: el juego de Breakout de la consola Atari, el problema del Cartpole, el ejemplo de un agente que aprende a caminar en el entorno de Bipedal Walker, etc.

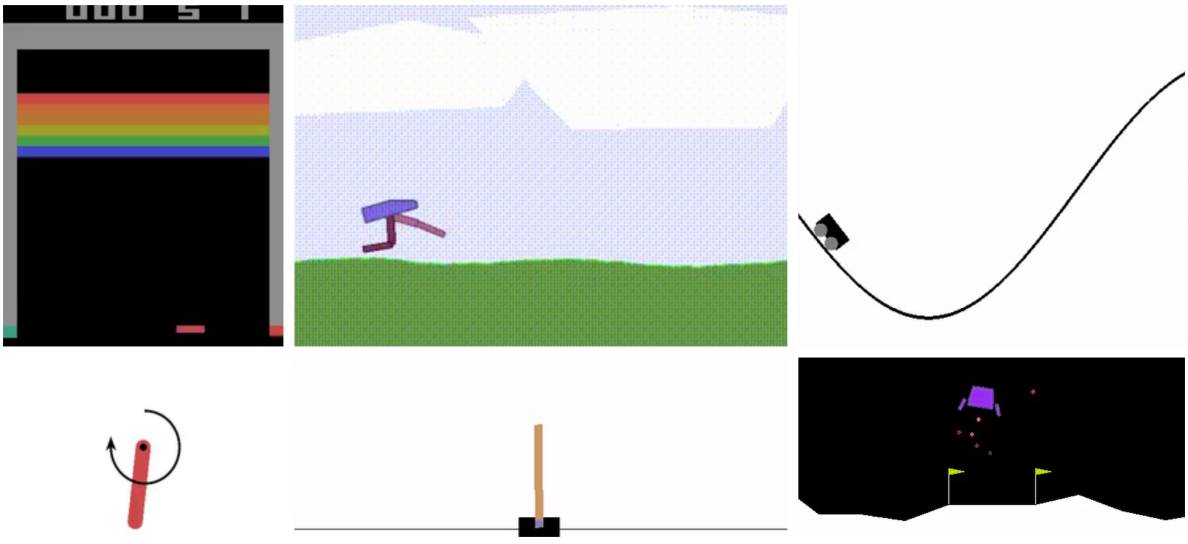
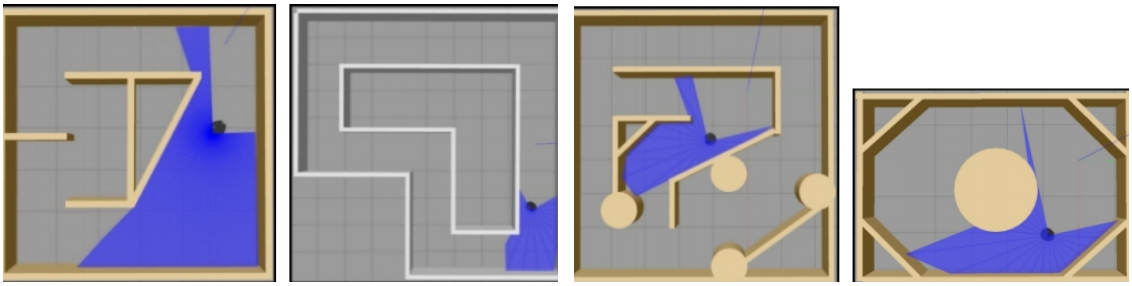


Figura 3.2: Diferentes entornos de OpenAI Gym.

En este Trabajo Fin de Máster se utiliza la interfaz de OpenAI Gym en su versión 0.16.0 para construir un entorno similar pero añadiendo la funcionalidad necesaria para acoplar ROS como herramienta de comunicación y Gazebo como herramienta de representación. Esta agregación se materializa en forma de librería, llamada Gym-Gazebo.



(a) Laberinto. **(b)** Laberinto con obstáculos.

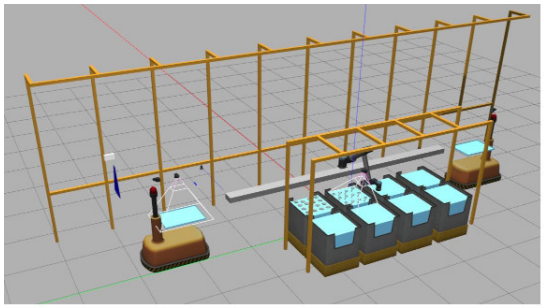
Figura 3.3: Diferentes entornos de Gym Gazebo.

3.7. Gym Gazebo

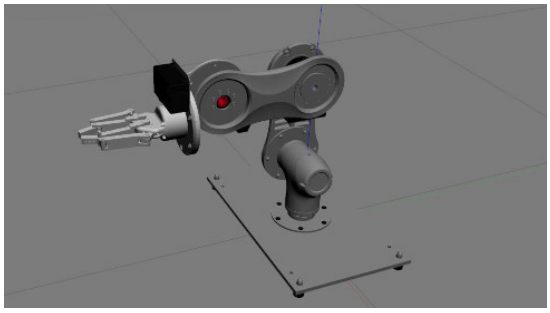
El núcleo principal del proyecto donde se desarrolla el código que resuelve el problema planteado se localiza en la librería Gym-Gazebo [35], escrita en Python en su versión 2.7. La librería pertenecía a la empresa *Erle Robotics* hasta que fue adquirida por Acutronic[12]. En ese momento, el repositorio[13] dejó de tener soporte y fue cerrado para la contribución pero continúa a disposición de la comunidad para ser utilizado como plataforma para el aprendizaje por refuerzo.

Gym-Gazebo facilita el desarrollo de algoritmos de aprendizaje por refuerzo aplicados a la robótica usando el mismo interfaz de funciones que utiliza OpenAI Gym y combinándolos con estándares como ROS y Gazebo para las comunicaciones y simulación. Inicialmente parte de varios escenarios donde se utilizan distintos tipos de algoritmos para la resolución como Q-learning o SARSA empleando distintos tipos de sensores como LIDAR o láser combinado con los motores de Gazebo para la resolución de problemas como esquivar obstáculos, navegación, etc. En la figura 3.3 pueden verse algunos de estos escenarios.
Los robots disponibles para resolver estos entornos son variados teniendo brazos mecánicos móviles y articulados para ejercicios de coger y soltar y por otro lado también robots *turtlebot*, para la resolución de ejercicios de tipo esquivar obstáculo o avanzar sin colisionar con las paredes, etc. En la figura 4.10 pueden verse algunos de estos robots.

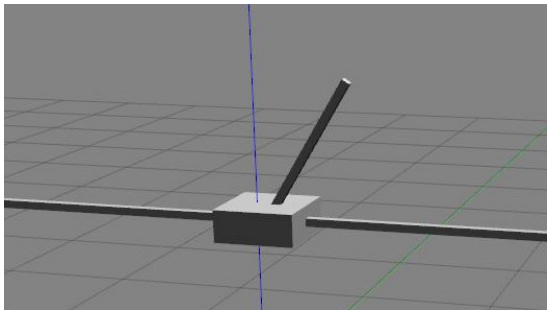
[12] `http://www.acutronic.com/ch/`
[13] `https://github.com/erlerobot/gym-gazebo`



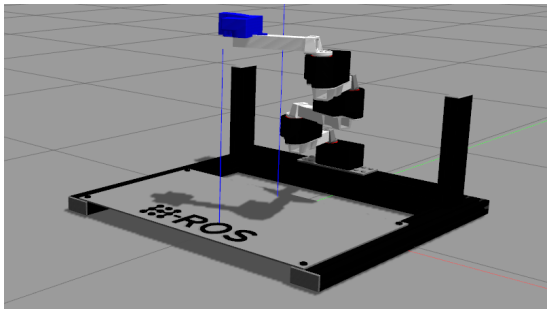
(a) Robot ARIA.



(b) Brazo articular



(c) Cartpole.



(d) Robot Scara.

Figura 3.4: Robots disponibles de Gym-Gazebo.

4

Aprendizaje por refuerzo de un controlador visual

En este capítulo se describe la solución desarrollada. Se detalla cómo cada una de las pequeñas partes ha tenido peso para llegar a la solución del problema. En las siguientes secciones se cubren todos estos aspectos hasta llegar a la fase de aprendizaje donde se llevarán a cabo los entrenamientos con aprendizaje por refuerzo en los diferentes entornos.

4.1. Diseño

La estructura del proyecto está compuesta por tres grandes piezas: Gym-Gazebo, ROS y Gazebo. Como puede verse en la figura 4.1 izquierda, Gym-Gazebo aparece en la parte superior del diagrama como orquestador de la información proporcionada por el simulador en primer lugar y por ROS después a través del *topic*: `img_raw` para actuar entregando los valores correspondientes al simulador a través del *topic* `cmd_vel` en orden inverso para los entrenamientos. Para situaciones que requieran reiniciar la simulación se usan los topics de Gazebo para el posicionamiento del robot en la zona deseada. Una vez se tiene el algoritmo de aprendizaje por refuerzo entrenado, se ejecuta el mismo programa pero cargando la información aprendida en la etapa anterior. A nivel estructural tiene el mismo diseño dado que se encuentra también dentro de la librería de Gym-Gazebo pero es ejecutado por otro programa. Puede verse la arquitectura en la figura 4.1 derecha.

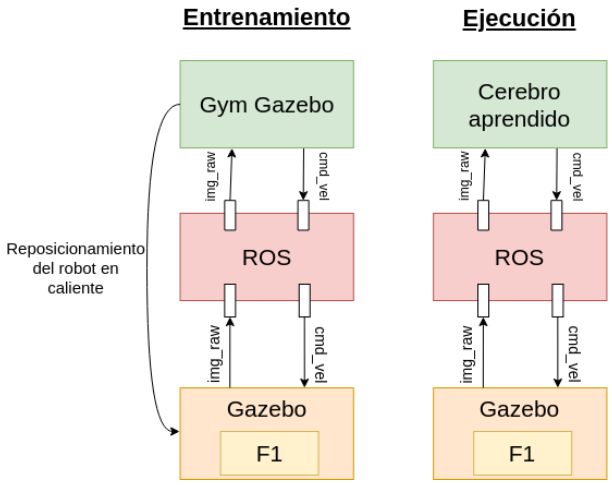


Figura 4.1: Estructura del proyecto.

Los entornos ROS y Gazebo son los encargados de representar el robot en el mundo generado con Gazebo para que sea Gym-Gazebo el que gestione y gobierne el robot.

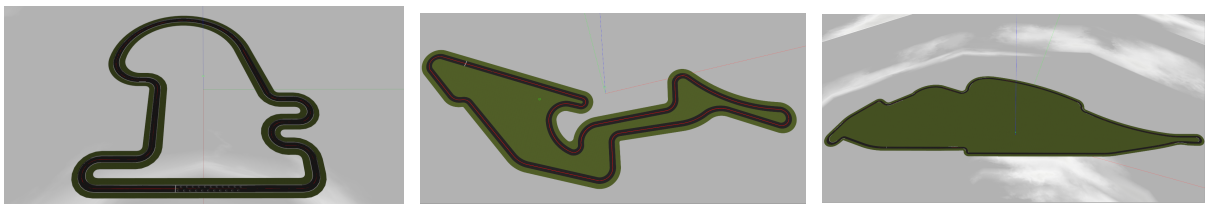
4.2. Escenarios

Los escenarios robóticos empleados en el proyecto para entrenar y para evaluar los cerebros aprendidos son mundos creados con el simulador Gazebo. Cada uno presenta diferentes características de longitud, número de curvas y rectas que pueden resumirse en la tabla 4.1.

Circuito	**metros**	↰	↑	↱
Simple	465	4	5	5
Nürburgring	597	6	9	8
Montreal	1563	7	9	9

Tabla 4.1: Características de los circuitos.

Puede verse una vista cenital de los circuitos (figura 4.2) donde la complejidad de cada uno es diferente. El «circuito simple» (figura 4.2a) es un recorrido sintético que se utiliza en la plataforma RoboticsAcademy[1] para ejercicios de navegación de robots. Las curvas son de complejidad baja y generalmente abiertas. En las réplicas de los circuitos reales de Fórmula-1 de «Nürburgring» (figura 4.2b) y «Montreal» (figura 4.2c) vemos que existen curvas más cerradas, cambios bruscos en mitad de una curva o pequeños *zig-zags* que ponen a prueba los límites del algoritmo para situaciones con cambios más repentinos.



(a) Circuito simple. **(b)** Nürburgring. **(c)** Montreal.

Figura 4.2: Escenarios de Gazebo.

En todos ellos se utiliza el mismo modelo de coche basado en un Fórmula-1 (figura 4.3). El modelo del coche cuenta con una cámara situada en la parte delantera como sensor principal.

Figura 4.3: Modelo de Fórmula-1 empleado en el entrenamiento.

[1] `https://github.com/JdeRobot/RoboticsAcademy`

Se capturan imágenes a unos 20 fotogramas por segundo (fps) y está equipado con un sensor láser de distancia, pero no se usará en este TFM dado que está más enfocado en un control estrictamente basado en visión. Los motores del coche se gobiernan comandándoles continuamente la velocidad de avance deseada $v(m/s)$ y la velocidad de giro deseada $w(rad/s)$.

4.3. Adaptación de Gym-Gazebo

La librería principal desde donde se gestionan todos los entornos y comportamientos así como el registro de los resultados una vez finalizados los experimentos es Gym-Gazebo. Para el desarrollo de este proyecto se ha modificado la librería por fases y agregado un nuevo ejercicio entrenable al repertorio: la conducción autónoma usando una cámara como sensor. Para este nuevo ejercicio del «gimnasio» se han creado varios escenarios (mencionados en el punto 4.2) con diferentes complejidades que enriquecen el conjunto de pruebas.

Para la resolución el ejercicio se crea un nuevo agente, el Fórmula-1, que tiene asociado un entorno (en términos de la librería OpenAI Gym, un *environment*) que es donde se realizan los cálculos y se evalúa cada paso (o *step*) que se da en el circuito. En él se describen los mismos métodos que en el resto de entornos de la librería pero con diferentes operaciones en su interior, adaptadas al ejercicio en cuestión.

La biblioteca incluye todos los ingredientes de un aprendizaje por refuerzo y se encarga de reinicia al simulador Gazebo y/o la posición del robot en el escenario simulado cuando es necesario.

No ha sido posible la incorporación al repositorio original de Gym-Gazebo de todas las mejoras creadas para el proyecto como son: la creación del nuevo ejercicio, los escenarios, entornos, mundos de Gazebo, modelos, etc, dado que el repositorio oficial de la librería se encuentra bloqueado. Actualmente todas las mejoras sí se encuentran integradas en el repositorio del Trabajo Fin de Máster[2].

4.3.1. Aprendizaje por refuerzo con Gym-Gazebo

Para la creación de ejercicios con algoritmos de aprendizaje por refuerzo dentro de la librería Gym-Gazebo es necesario crear una serie de métodos obligatorios, ya que esta biblioteca actúa de intermediaria entre el software robótico (ROS y Gazebo) y la librería de OpenAI Gym. La estandarización que creó OpenAI Gym donde se unificaban todos los ejercicios del gimnasio bajo una misma estructura de métodos definida se hereda también en Gym-Gazebo. Los métodos más relevantes para la ejecución de un ejercicio en Gym-Gazebo son:

- **Método** `step`: Es uno de los métodos más importantes de la librería y es donde se programa la lógica del robot en cada paso que da. Recibe como parámetro de entrada una acción que ejecuta (da un paso) y analiza el entorno con los sensores (cámara, láser, etc), buscando características en él y generando una recompensa acorde en

[2] `https://github.com/RoboticsLabURJC/2019-tfm-ignacio-arranz/tree/master/gym-gazebo`

función de una serie de condiciones. Este método devuelve el estado siguiente $s_t + 1$ (`state`), la recompensa (`reward`), la variable booleana `done` indicando si se sigue la ejecución o hay que reiniciar y un parámetro opcional (`info`) con información adicional.

- **Método** `reset`: Es otro de los métodos importantes de todo algoritmo de aprendizaje por refuerzo y representa el retorno del agente a la posición inicial del entorno, el reinicio. Este método ocurre cuando en el método `step` se ha dado una condición que impide seguir avanzando, por ejemplo, el robot se ha chocado. El retorno a la posición de origen calcula también el nuevo estado de partida s_t desde el cual comenzará a llamarse al método `step`.

- **Métodos** `pause` y `unpause`: Al inicio de cada paso o reinicio generado por los métodos anteriores es necesario detener la simulación de Gazebo durante un instante para generar el estado siguiente $(s + 1)$ para evaluar lo recogido por los sensores en un instante y evitar sobrecarga en el código y en el robot y tener que desechar alguna lectura. Una vez calculados se llama al método `unpause` para reanudar de nuevo la ejecución.

Con estos métodos definidos, se tiene una estructura que permite crear ejercicios que resuelvan situaciones utilizando algoritmos de aprendizaje por refuerzo. El proceso de modificación de Gym-Gazebo hasta crear el ejercicio principal que resuelve este Trabajo Fin de Máster ha seguido una serie de fases que se detallan a continuación.

4.3.2. Fase 1: Sorteo de obstáculos con sensor de distancia y robot *TurtleBot*

Para la construcción del entorno deseado (mundos, modelo y cámara como sensor), se partió inicialmente de un ejercicio con el que contaba la librería de Gym-Gazebo en el que un robot *Turtlebot* avanzaba por un laberinto sin colisionar con las paredes usando el sensor del láser, como puede verse en la figura 4.4.

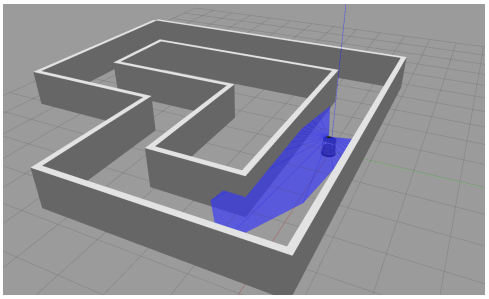


Figura 4.4: Robot *Turtlebot* con el sensor láser en el laberinto.

Este robot, pasados una serie de episodios conseguía circular por el laberinto por el centro del pasillo evitando colisionar. Este ejercicio es la base para construir la solución a la navegación basada en visión dado que el objetivo de ambos es el mismo, evitar colisionar con la pared siguiendo un carril.

Este ejercicio consiguió replicarse con éxito desde la última versión congelada de la biblioteca y ejecutándose en un sistema operativo más moderno.

4.3.3. Fase 2: Sorteo de obstáculos con sensor de distancia y un Fórmula-1

Un segundo paso para adaptar la biblioteca Gym-Gazebo a la aplicación de control visual ha sido replicar las condiciones del ejercicio del *Turtlebot* pero con el mundo del «Circuito Simple», con el modelo del Fórmula-1 como agente y manteniendo el láser como sensor. Puede verse el resultado de trasladar la configuración del *Turtlebot* al Fórmula-1 en la figura 4.5.

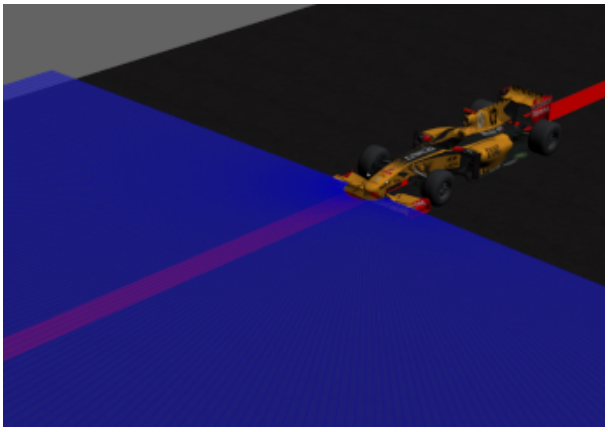


Figura 4.5: Fórmula-1 con el sensor láser en el circuito simple.

El objetivo de esta variante era completar el circuito sin colisionar con las paredes. En este caso la línea roja, pese a estar encima de la carretera era ignorada y únicamente se hacía un cálculo del centro a través de los haces del láser dividiendo el barrido de 180 medidas láser en 5 sectores, como se puede ver en el fragmento 4.1.

```
laser_len =  len(data.ranges)
left_sum =  sum(data.ranges[laser_len - (laser_len / 5):laser_len - (laser_len / 10)])
right_sum =  sum(data.ranges[(laser_len / 10):(laser_len / 5)])

center_detour = (right_sum - left_sum) / 5
```

Fragmento 4.1: Cálculo del centro del carril usando el láser.

Los actuadores para el modelo del Fórmula-1 son los mismos que se utilizan en el ejercicio del *Turtlebot*, velocidad lineal (m/s) y velocidad angular (rad/s). En este sentido, la adaptación al Fórmula-1 ha sido directa.

4.3.4. Fase 3: Información visual

Con la variante del ejercicio con el sensor láser resolviendo el entorno satisfactoriamente el tercer paso fue cambiar ese sensor por la cámara para entrenar con información visual, registrando y creando el escenario en el fichero de entornos, como puede verse en el fragmento 4.2.

```
register(
    id='GazeboF1QlearnCameraEnv-v0',
   entry_point='gym_gazebo.envs.f1:GazeboF1QlearnCameraEnv',
)
```

Fragmento 4.2: Registro de un nuevo entorno en Gym-Gazebo.

El cambio del sensor láser a la cámara es un cambio sencillo debido a la modularidad del código de Gym-Gazebo. Para la captura de la imagen del entorno se crea un bucle en el método `step` que itera hasta tener una imagen en el *buffer* de imágenes para poder extraer. Para convertir los datos en bruto que se captan desde el sensor se utiliza la librería de ROS `cv_bridge`. Esta biblioteca convierte los datos a matrices que las librerías de OpenCV y Numpy pueden entender. Puede verse el código en el fragmento 4.3.

```
while not image_data or not success:
   image_data = rospy.wait_for_message('/F1ROS/cameraL/image_raw', Image, timeout=5)
   cv_image = CvBridge().imgmsg_to_cv2(image_data, "bgr8")
   f1_image_camera = self.image_msg_to_image(image_data, cv_image)
   if f1_image_camera:
      success = True
```

Fragmento 4.3: Extracción de la imagen tomada desde la cámara.

4.3.5. Fase 4: Reinicios aleatorios

Otro elemento añadido a la librería Gym-Gazebo es la posibilidad de hacer reinicios aleatorios en unos puntos seleccionados del circuito. El motivo de esta mejora es acelerar el entrenamiento dándole al agente situaciones que solo se encontrará cuando complete parte del circuito. Para casos en los que la convergencia no es tan rápida permite al agente aprender de una manera más ágil en la etapa inicial del entrenamiento donde la capacidad de exploración está todavía muy viva. Sin estos reinicios, si al final del recorrido hay una situación que nunca ha explorado y requiere reiniciar el episodio necesitará recorrer de nuevo todo el circuito para volver a encontrarse esa situación.

Los escenarios donde el agente entrena son: el «Circuito Simple» y Nürburgring. Para ambos, se ha creado un conjunto de posiciones desde las cuales se considera que se cubren estas necesidades. En la figura 4.6 pueden verse las posiciones de reinicio para el «Circuito Simple» así como la dirección que toma el vehículo en esa posición. En algunos casos se invierte el sentido para enriquecer aún más el repertorio y balancear el número de curvas. De igual manera, pueden verse los puntos de reinicio donde el agente se posicionará para el circuito de Nürburgring en la figura 4.7.

4.4. Entrenamiento

Con las modificaciones realizadas sobre la librería y el nuevo ejercicio entrenable integrado en Gym-Gazebo se elabora un plan de entrenamiento que tendrá en cuenta principalmente dos factores: el *número de percepciones* y el *número de acciones*.

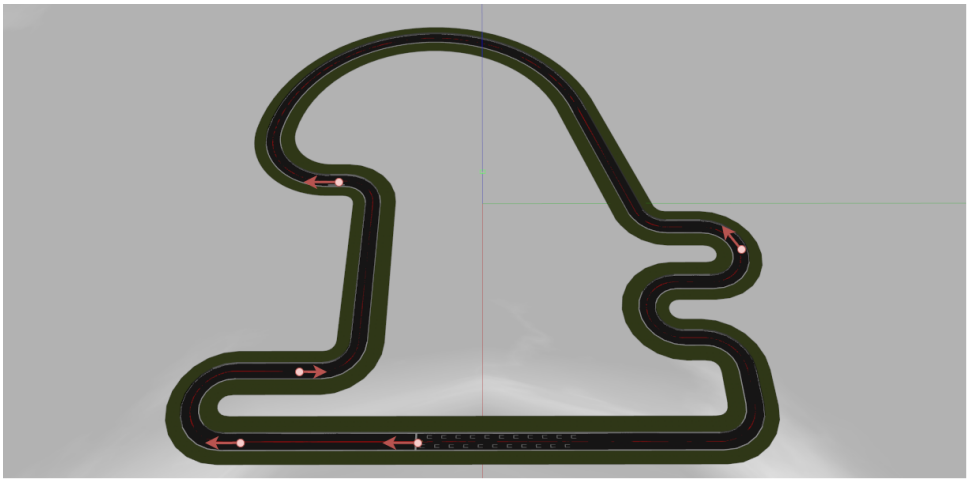


Figura 4.6: Posiciones del circuito simple

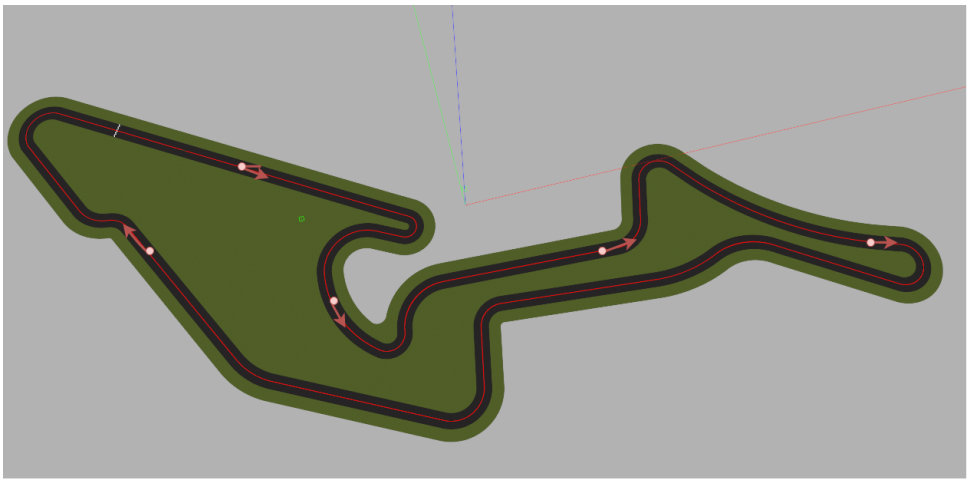


Figura 4.7: Posiciones de Nürburgring

4.4.1. Algoritmo Q-Learning

En el núcleo del problema se encuentra el algoritmo encargado de recibir los pares (*estado*, *acción*) que retorna la librería de Gym-Gazebo y almacenar la *recompensa* asociada para rellenar la «Tabla-Q» que contenga todas las situaciones que ha visto. Esta tabla contiene el par (*estado, acción*) como clave y el valor de la *recompensa* como valor. Este sistema de registro encaja muy bien en los controladores reactivos para gobernar el comportamiento de robots. Puede verse el flujo de registro en la tabla en la figura 4.8.

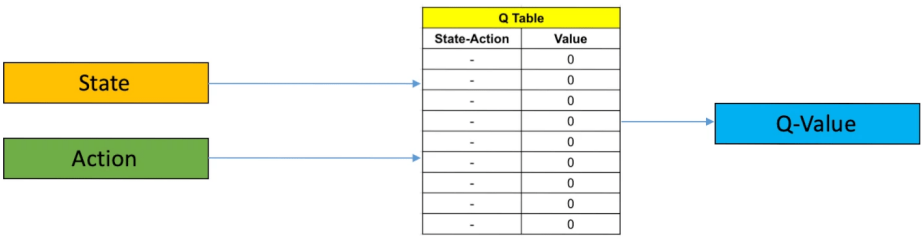


Figura 4.8: Estructura de Q-Learning

En este trabajo se utiliza el algoritmo Q-Learning. Es un algoritmo de aprendizaje muy usado para la resolución de problemas con entornos no modelados. En general el algoritmo converge muy rápido, es eficiente, funciona a través de las observaciones que realiza el agente durante el entrenamiento y es relativamente sencilla su implementación. Estas características hacen que encaje perfectamente con el problema de control visual que se quiere solucionar.

Durante el entrenamiento al robot se le presentan diferentes situaciones (*estados*), el algoritmo va probando posibles actuaciones (*acciones*) como respuesta a ellas y va recogiendo las recompensas observadas en el simulador robótico a esas combinaciones. El resultado neto es que el algoritmo aprende una Tabla-Q, $Q(s, a)$, que contiene cuál es la mejor respuesta para cada situación a la luz de lo experimentado durante el entrenamiento. A mayor número de pares (*estado, acción*) se tiene una mayor Tabla-Q con más posibilidades. Esto tiene como contrapartida que, a mayor número de acciones mayor complejidad para resolver el problema y mayor tiempo de entrenamiento hasta la convergencia.

La ecuación que define el comportamiento de este algoritmo es la siguiente:

$$Q(s, a) \leftarrow (1 - \alpha)Q_{s,a} + \alpha(r + \gamma \cdot \max_{a'} Q_{s',a'}) \tag{4.1}$$

Donde:

- $\alpha(0 < \alpha \leq 1) \rightarrow$ es el índice de aprendizaje o *learning rate*. Si $\alpha = 0$, $Q(s, a)$ nunca es actualizada y, por tanto, no se produce aprendizaje. Por el contrario si $\alpha = 1, Q(s, a)$ se actualiza lo máximo posible en cada iteración. Aunque pueda parecer tentador, tener un valor de $\alpha = 1$ implica que el resultado puede olvidar lo que ha ido aprendiendo en etapas anteriores. Es necesario tener un ajuste en este parámetro para equilibrar ambas situaciones. Para este proyecto se selecciona el valor de $\alpha = 0.8$ por ser el más equilibrado.

- $r \rightarrow$ o *recompensa* (*reward)* es el valor devuelto como consecuencia de tomar una acción y pasar del estado s_t al estado s_{t+1}.

- $\gamma(0 < \gamma \leq 1)$ es el factor de descuento (*discount factor*) que relaciona acciones y recompensas a largo plazo (*long term reward*). Es un valor que consigue relacionar si una secuencia de acciones dará como resultado una mayor recompensa, actuando como una «memoria» para dar más valor a las recompensas anteriores que a las futuras. Entra dentro también de este ámbito más psicológico de esta rama del aprendizaje donde la experiencia te asegura un estado y lo que está por venir es desconocido. Se puede interpretar como la probabilidad de tener éxito en cada paso que se toma.

Un entrenamiento utilizando el algoritmo de Q-Learning sigue la siguiente secuencia hasta conseguir la convergencia:

1. Se comienza con la tabla Q vacía para $Q(s, a)$.

2. Se obtiene de un paso (*step*) los valores: estado, acción, recompensa y estado siguiente (s, a, r, s'). En este paso se decide qué acción se toma, atendiendo a las condiciones de exploración o explotación.

3. Se actualiza la ecuación de Bellman: $Q(s,a) \leftarrow (1-\alpha)Q_{s,a} + \alpha(r + \gamma \cdot \text{máx}_{a'}\, Q_{s',a'})$.

4. Se comprueban las condiciones de convergencia. Si no se cumplen se vuelve al paso 2.

Cabe destacar del procedimiento anterior el paso 2. Durante el entrenamiento se lleva a cabo un proceso de exploración del entorno, representado como una cadena de Markov. Durante esta etapa el algoritmo irá tomando aleatoriamente acciones que le llevan a descubrir los límites de sus posibilidades. Este valor viene representado por el parámetro *epsilon* (ϵ) que en los inicios del entrenamiento tiene un valor cercano a 1. En el fragmento 4.4 de código puede verse como inicialmente se depende de un valor aleatorio muy alto para que se obtenga de la Tabla-Q la mejor recompensa:

```
q = [self.getQValues(state, a) for a in self.actions]
maxQ =  max(q)

if random.random() < self.epsilon:
   minQ =  min(q)
   mag =  max( abs(minQ),  abs(maxQ))
   # add random values to all the actions, recalculate maxQ
   q = [q[i] + random.random() * mag - .5 * mag for i in  range( len(self.actions))]
   maxQ =  max(q)

count = q.count(maxQ)
if count > 1:
   best = [i for i in  range( len(self.actions)) if q[i] == maxQ]
   i = random.choice(best)
else:
   i = q.index(maxQ)

action = self.actions[i]
```

Fragmento 4.4: Selección de acciones: exporación vs explotación.

A medida que el entrenamiento avanza y se va paulatinamente completando la tabla $Q(s,a)$ (el vehículo se encuentra con nuevas situaciones), el valor de ϵ disminuye. Cuando este parámetro es lo suficientemente bajo las acciones que tomará el agente ya no son aleatorias sino que buscará en la tabla aquellas que retornan la máxima recompensa.

Este elemento es una pieza muy importante dentro del algoritmo dado que da al agente la posibilidad de ejecutar acciones que no retornan la mejor recompensa pero le permite actualizar estados no explorados de $Q(s,a)$ para después, en la etapa de explotación, reforzar y perfeccionar aquellas combinaciones de $(estado, acción)$ que retornan mejores valores. Esta política se le conoce como política codiciosa o *epsilon-greedy*.

En la mejora añadida a la librería de Gym-Gazebo este parámetro *epsilon* comienza con un valor de 0.99 y en cada episodio se le aplica un factor de descuento de 0.998 que lo hace disminuir paulatinamente. En entrenamientos donde el número de episodios es pequeño dado que consigue estar mucho tiempo sobre la línea roja, este factor se aplica cada 1000 pasos (`steps`) que es, aproximadamente, una cuarta parte del circuito. Este factor

de descuento añade la posibilidad de que el agente pueda en un mismo entrenamiento explorar (reconocer el mayor número de estados o situaciones posible) y explotar (ajustar los valores de las situaciones conocidas).

Con todos estos elementos solo se necesita conocer qué conjunto de (*estado, acción*) devuelve mejores recompensas así como el conjunto de valores óptimo de *alpha* (tasa de aprendizaje), *gamma* (factor de descuento) y *epsilon* (para medir el grado de explotación y exploración) que configura el mejor entrenamiento y posterior ejecución.

Siendo esta la estructura general del algoritmo Q-Learning hay que describir las concreciones necesarias para aplicarlo al problema de conducción autónoma, del control visual deseable:

1. Los estados son las percepciones simplificadas desde la imagen obtenida por la cámara del coche,
2. las acciones son las actuaciones posibles sobre los motores,
3. la recompensa habrá que definirla y,
4. habrá que dar unos valores razonables a los parámetros de aprendizaje *alfa*, *gamma* y *epsilon*.

4.4.2. Percepción simplificada

El objetivo del vehículo es conducir de manera autónoma por encima de la línea roja dibujada en el asfalto del circuito dada una imagen a través de la cámara situada encima. Esta imagen de entrada tiene unas dimensiones de $640x480$ píxeles en formato BGR (disposición típica de los canales de color en la librería de OpenCV).

Un ejemplo de esta imagen de entrada en una recta puede verse en la figura 4.9. Esta imagen contiene mucha información pero no toda ella es valiosa. La parte de interés se encuentra en la mitad inferior, donde están la línea roja y el asfalto. En esta mitad además, no todo es importante. Dado que el objetivo del vehículo es seguir la línea roja se aplica un filtro de color para la detección sobre una imagen de entrada convertida al formato HSV ya que la extracción del color rojo queda definida en el canal H (tono o *Hue*). El resultado es la imagen en formato de una máscara (negro y blanco o 0 y 255) con los valores de interés.

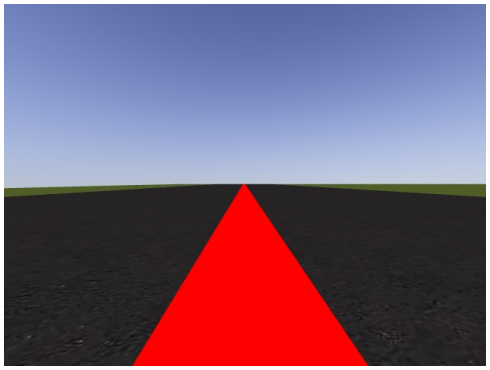
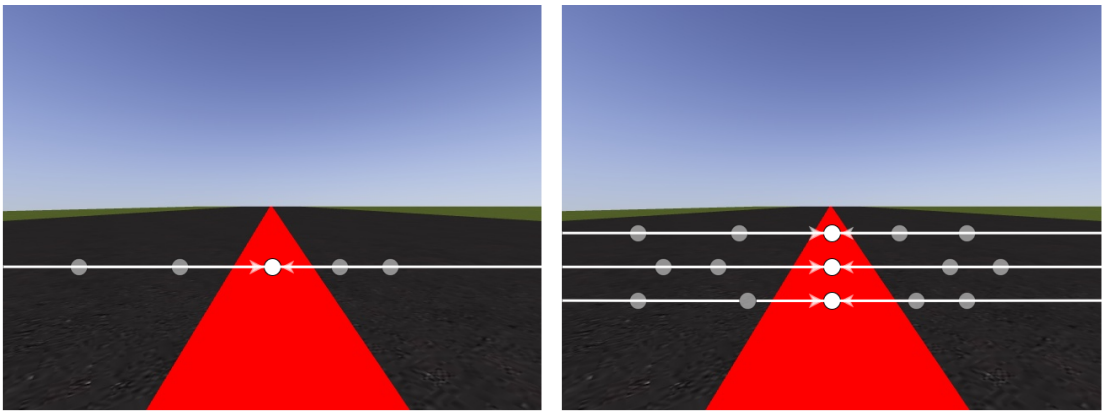


Figura 4.9: Imagen en bruto de la cámara del Fórmula-1.

Con la máscara obtenida se busca en diferentes filas el valor positivo (255) inicial y final para restarlos y obtener el centro de cada una de ellas. Esto retorna un valor (en píxeles) de la posición del centro en la imagen de entrada. Puede verse un ejemplo de los estados simplificados para uno y tres percepciones en las figuras 4.10a y 4.10b.



(a) 1 puntos de percepción. **(b)** 3 puntos de percepción.

Figura 4.10: Diferentes puntos de percepción.

Para evaluar la diferencia entre el valor observado del centro y el deseado se crea una plantilla que divide el espacio de la imagen en diferentes sectores, del 1 al 8 en valores positivos y negativos (figura 4.11). Esta rejilla servirá para generar un valor de «posición» del centro de la línea roja dentro de la imagen. Así pues, si un valor del centro cae en la región 4, el valor que devuelve todo el procesamiento será 4 en vez del número de la columna del píxel de la imagen. El motivo de esta simplificación es reducir los valores de píxeles a un subconjunto simplificado que permita al algoritmo de aprendizaje por refuerzo encontrar una relación entre los valores de manera más rápida.

Como estados posibles para el algoritmo de aprendizaje se van a probar tres configuraciones: tomando la posición del centro en una sola fila de la imagen (un punto de percepción único), en dos (dos puntos de percepción) y en tres (tres puntos de percepción). En ese valor único, doble o triple se resume completamente la imagen de entrada, puesto que condensa la información necesaria para seguir la línea.

Por tanto, si tomamos la percepción simplificada usando únicamente una fila, el número de posiciones en los que puede estar el centro de la línea será de 17 (de -8 a 8 contando el 0) por el número de acciones que tenga el conjunto de estudio (3, 5 o 7) por el número de percepciones. Destacar que la gran mayoría no pueden darse por las limitaciones que ofrece la disposición de la línea sobre la imagen y la condición inicial que plantea el problema: seguir la línea. Puede verse en la figura 4.11 la secuencia de procesamiento sobre la imagen de entrada hasta la salida para un punto de percepción.
Para estados con dos o tres puntos de percepción se obtiene el valor central de la segmentación de la línea en dos o tres filas de la imagen de modo similar. De esta manera el estudio del comportamiento del agente será igual en todos los casos, como puede verse en la figura 4.12.

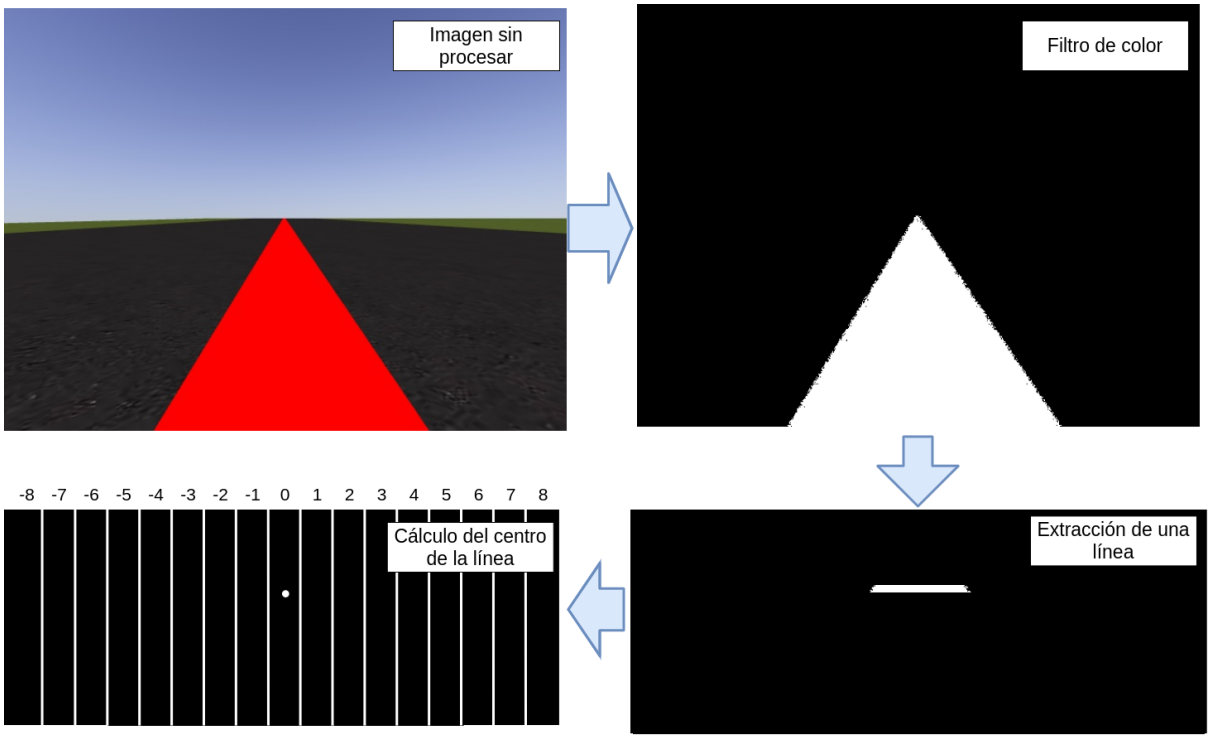


Figura 4.11: Cadena de procesamiento para un punto de percepción.

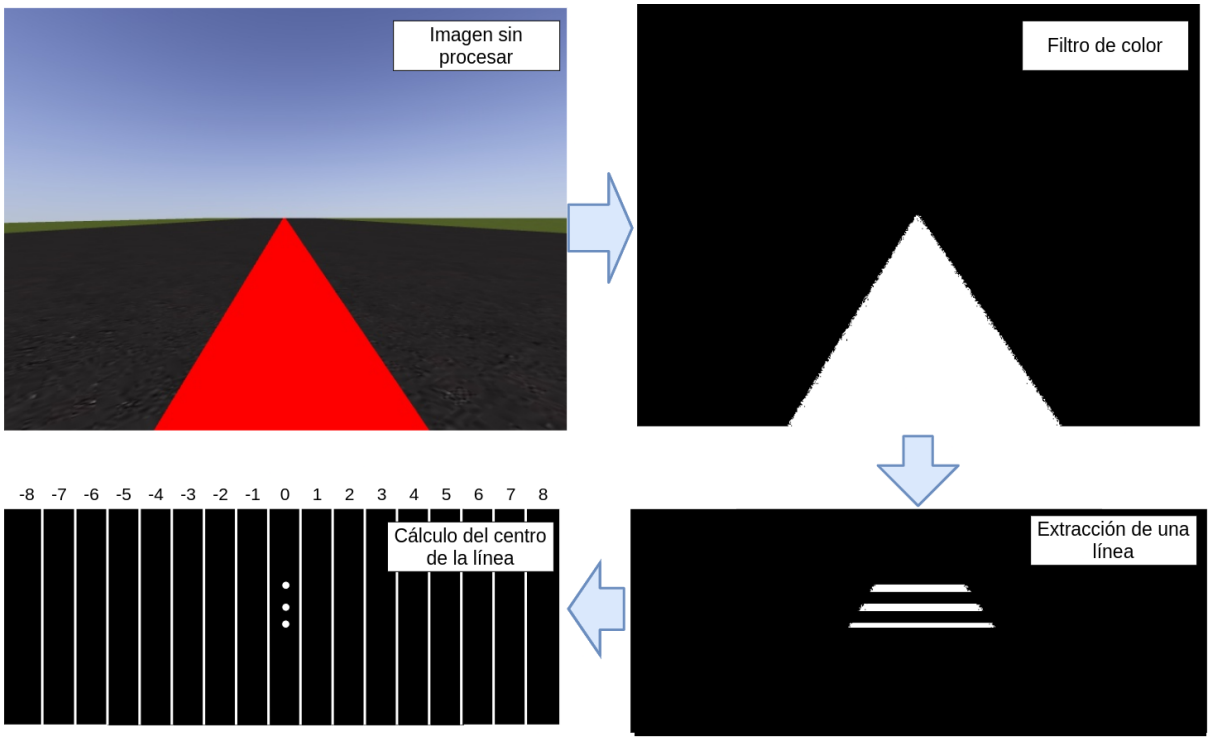


Figura 4.12: Cadena de procesamiento para tres puntos de percepción.

Los valores simplificados son devueltos en forma de lista de enteros de Python constituyendo así los *estados*. Esta lista de estados contiene el número de *situaciones* a las que se enfrenta el algoritmo dada una imagen. Un ejemplo del resultado de esta sección puede verse en la figura 4.13 donde el resultado de ese estado corresponde con la secuencia $[1, 1, 0]$ en orden descendente.

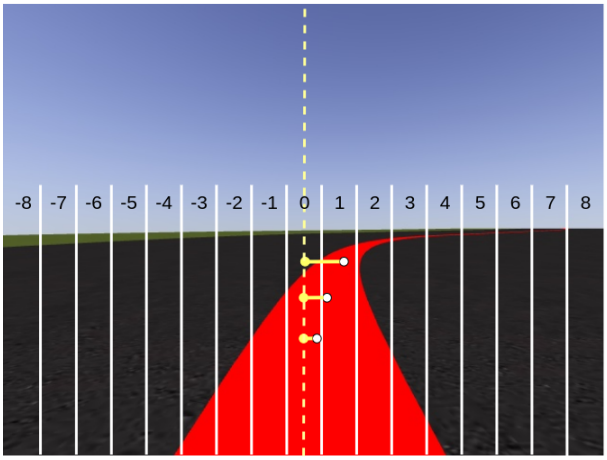


Figura 4.13: Ejemplo de estados retornados dado un paso.

De manera similar a como aprende el ser humano, cuanto más tiempo de entrenamiento se dedique a una tarea, más «estados» se verán y más experiencia se adquiere. A estados nunca vistos el comportamiento será impredecible.

4.4.3. Conjunto de acciones posibles

Las acciones es otro de los grandes pilares del aprendizaje por refuerzo. En función de la situación en la que se encuentre el agente se tomará una u otra acción para corregir la dirección (inicialmente de manera aleatoria). En un entorno real, el conjunto de acciones tiene un espacio continuo de posibilidades. Pueden hacerse modificaciones infinitamente pequeñas para corregir la desviación cuando se conduce por una carretera o cuando se corrige el manillar de una bici para mantenerla en equilibrio. En el caso de este Trabajo Fin de Máster se seleccionan tres conjuntos de acciones discretas suficientes como para que el coche pueda completar todas las curvas del circuito.

Un conjunto tiene tres acciones posibles (tabla 4.2), otro tiene 5 acciones posibles (tabla 4.3) y el tercero contiene 7 acciones posibles (tabla 4.4).

Para encontrar el conjunto que mejor resuelve el problema se crean estos tres repertorios, cada uno más complejo que el anterior. En la tabla 4.2 puede verse la distribución para el conjunto «simple» de acciones, únicamente dos valores de giro y uno de velocidad lineal.

Acción	**0**	**1**	**2**
V. Lineal (m/s)	3	2	2
V. Angular (rad/s)	0	1	-1

Tabla 4.2: Conjunto simple.

Aumentando ligeramente la dificultad, se añaden dos acciones más de velocidad angular para cubrir casos en los que las curvas puedan ser más cerradas. Es un conjunto «medio» de acciones. Puede verse esta distribución en la tabla 4.3.

Acción	**0**	**1**	**2**	**3**	**4**
V. Lineal (m/s)	3	2	2	1	1
V. Angular (rad/s)	0	1	-1	1.5	-1.5

Tabla 4.3: Conjunto medio.

Por último, se reasignan valores y se añaden dos más, para casos todavía más extremos y que requieran de más velocidad angular. Este conjunto de acciones ya cuenta con 7 y se denomina «dificil». Pueden verse los valores en la tabla 4.4.

Acción	**0**	**1**	**2**	**3**	**4**	**5**	**6**
V. Lineal (m/s)	3	2	2	1.5	1.5	1	1
V. Angular (rad/s)	0	1	-1	1	-1	1.5	-1.5

Tabla 4.4: Conjunto difícil.

Para la selección de los valores concretos de v y w para cada acción se ha ejecutado una solución al problema utilizando un algoritmo creado manualmente y se han comprobado los valores máximos alcanzados de giro para tener uno representativo que pueda solucionar la dificultad más extrema.

4.4.4. Función de recompensa

Para completar un *paso* (o *step*) en el aprendizaje por refuerzo se necesita disponer de una función de recompensa que materialice numéricamente los valores retornados de la imagen procesada en un valor que sirva de guía para que la acción elegida como respuesta se repita o se evite.

En el punto 4.4.2 se describían los pasos hasta obtener el centro de la línea. Para el cálculo de la recompensa se utiliza la diferencia entre ese valor central de la línea y el centro de la imagen (corresponde con el valor de píxel 320 dado que la imagen es de 640 píxeles de ancho). En función de la distancia que haya al centro se calcula la recompensa de regreso.

Al igual que ocurre con la percepción simplificada, se le otorga al robot cierta holgura o rango para fijar una recompensa. El objetivo es que el coche haga coincidir su centro con el centro de la línea por lo que en ese punto y sus cercanías la recompensa será máxima. Un poco más lejos la recompensa será un valor mediano y aún más lejos un último rango donde la recompensa será mínima pero aún válida. En otro caso, el episodio termina y reinicia la simulación. Los valores de recompensa usados en este trabajo pueden verse en la figura 4.14 recuadrados. El código de colores representa cómo de buena es la posición del centro en la imagen, siendo el verde muy buena, la amarilla un punto intermedio y la naranja un rango bajo. El color rojo se reserva para el rango no válido.

Mientras el agente mantenga el centro de la parte delantera dentro del rango válido ($[-0.9, 0.9]$) en la imagen se irán dando pequeños pasos (*steps*) en el circuito siguiendo el algoritmo planteado en el punto 4.4.1. En el momento que el centro de la línea segmentada supere los valores admitidos ($centro > 0.9$) se da por terminado el *episodio* y se *reinicia la simulación.*

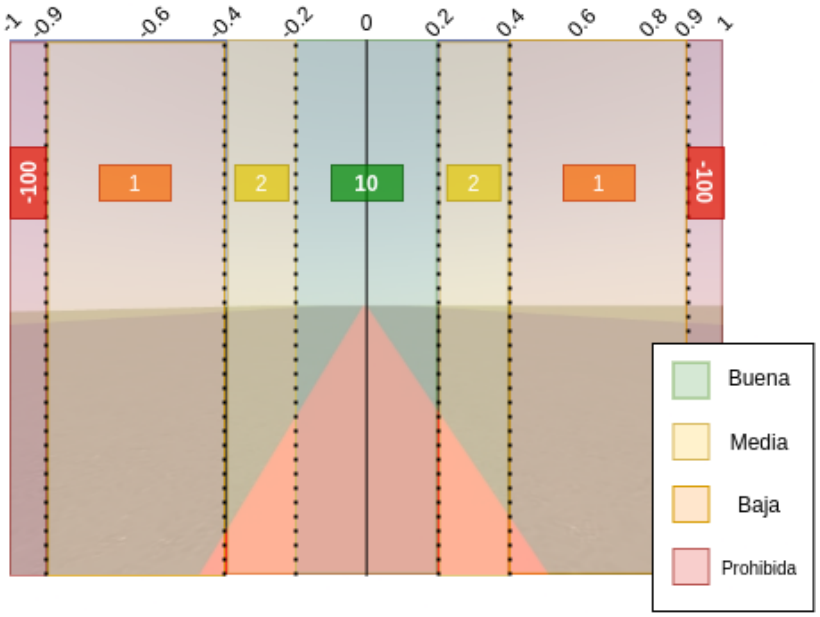


Figura 4.14: Distribución de los valores de recompensa.

4.4.5. Aprendizaje y ajustes

Durante esta sección se han ido viendo las distintas partes de las que está compuesto el algoritmo que soluciona el problema planteado utilizando aprendizaje por refuerzo. En este punto se agrupan todas esa partes para entender la relación entre ellas y los resultados que devuelve. También se extrae conocimiento del comportamiento que sigue el coche durante el entrenamiento que aporta al desarrollador información «psicológica» para afinar los parámetros del aprendizaje.

Siguiendo el flujo del programa en un entrenamiento se detallan algunos de los comportamientos derivados de la configuración y que han sido resueltos mediante iteraciones para el ajuste de los parámetros y de las condiciones del entrenamiento.

Cuando el entrenamiento empieza y se establece la comunicación con los componentes ROS y Gazebo, comienza un episodio dentro de un bucle `for` de Python. El comienzo de un episodio consiste en reiniciar la posición del robot y tomar una imagen de inicio, para tener un estado inicial con el que tomar decisiones. Siguiendo el paralelismo con una situación real es despertar o abrir los ojos y analizar lo primero que se ve. Seguidamente entra en otro bucle que son los *pasos* o *steps* y que se pasan a detallar a continuación:

1. Se selecciona una acción del conjunto disponible. Inicialmente aleatoria y según va avanzando el programa se empiezan a seleccionar de la Tabla-Q que se está confeccionando.

2. Se ejecuta el método `step` que recibe una acción:

   - **Se convierte la acción en movimiento del robot**: Tomando los valores registrado de cada acción en forma de velocidad lineal y angular que se entregan a ROS y se ejecutan en el robot dentro del simulador Gazebo.

- **Se toma un fotograma de la cámara**: Esta imagen representa el estado $s+1$. La acción que se convierte en reacción en el robot ya ha sido procesada en el comienzo del episodio.

- **Se procesa la imagen obteniendo el centro de la línea**. Este punto guarda una característica muy interesante del comportamiento del agente. La altura seleccionada para calcular el centro de la línea (más cerca del horizonte o del robot) repercute en el comportamiento del agente, por lo que se ha seleccionado un valor que guarda un compromiso entre la posición del vehículo con respecto a la línea y la resolución del problema.

  Alturas por encima de la elegida (parte superior de la línea roja o el horizonte) hacen que el agente se quede muy separado de la línea roja sobre el asfalto, pegándose a las paredes como puede verse en la figura 4.15. Por el contrario, alturas inferiores hacen que el comportamiento sea muy agresivo dado que los puntos que están más cerca de la cámara se mueven más rápido. Idealmente, el punto inferior sería el que hay que seguir dado que coloca el vehículo justo encima de la línea pero implica que el entrenamiento no se complete. Esta altura intermedia permite mantener el coche cerca de la línea roja a la vez que completa entrenamientos.

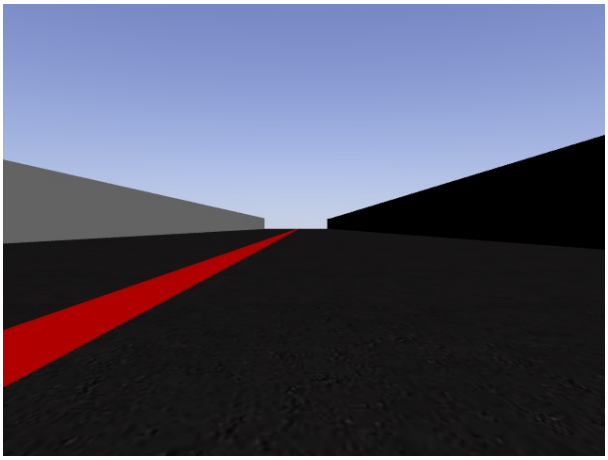


Figura 4.15: Valores de centro de la línea cerca del horizonte.

- **Cálculo del valor de recompensa**: en función de la posición del centro de la línea y el centro de la imagen se retorna una recompensa mayor o menor (explicado en esta sección).

- **Se retorna al programa principal**: los valores del estado o percepción simplificada (vistos en la sección 4.4.2), la recompensa y el booleano `done`, que indica si este cálculo del centro ha sido exitoso, es decir, se encuentra dentro del rango permitido. Un valor de esta variable negativo (`False`) reiniciaría la simulación (el coche se ha salido de la línea).

3. Se agrega la recompensa a la variable de recompensa acumulada. Un valor muy alto de esta variable indica que el coche ha ido la mayor parte del entrenamiento con

el morro y el centro de la línea alineados, es decir, ha conseguido la recompensa máxima en cada paso. Un valor menor indica que el agente ha estado oscilando en torno al centro, probando las áreas que retornan menor recompensa.

El resultado final del entrenamiento es un modelo en formato de diccionario de Python que contiene los pares (*estado, acción*) como clave y la recompensa como valor. Puede verse un ejemplo de la Tabla-Q para un conjunto de acciones simple (3 acciones) en la tabla 4.5.

(estado, acción)	**Recompensa**
(4, 2)	2
(-1, 0)	63
(-5, 0)	-62
(3, 2)	32

Tabla 4.5: Extracto de una Tabla-Q.

A la vista de los resultados del extracto de la Tabla-Q puede verse cómo un par (*estado, acción*) que representa la situación de casi ir en la línea recta implica tomar la acción de ir recto. Es el caso de la segunda fila de la tabla: $(-1, 0)$ que devuelve una recompensa relativamente alta. Por el contrario, valores de (*estado, acción*) como el de la fila tres $(-5, 0)$ devuelve una recompensa negativa dado que este conjunto significa haber ido recto cuando el punto estaba en una curva o se había producido una corrección. Es un estado que se quiere evitar y por eso el algoritmo la registra con un valor negativo. Comportamientos intermedios como la última fila $(3, 2)$ devuelve recompensas intermedias donde si el centro se encuentra en un cuarto de la imagen se debe tomar la acción 2, girar a la derecha para corregir la desviación. Un estado muy parecido es la fila uno $(4, 2)$ pero puede verse cómo el algoritmo intentará no hacer demasiado caso a ese conjunto dado que la recompensa es muy pobre.

Un elemento importante que mejora la calidad y velocidad de entrenamiento han sido los *saltos aleatorios* por cada reinicio del episodio. En la sección 4.3 se vieron los puntos en los que el agente se posiciona al azar cuando se reinicia el episodio. Colocar al agente en estos sitios repercute en la velocidad a la que se rellena la Tabla-Q dado que los estados que únicamente podría ver al final del recorrido los está viendo al principio del entrenamiento. Con unos pocos reinicios, el agente ya dispone de un tamaño mínimo de la Tabla-Q para poder sobrevivir durante más tiempo en cada nuevo episodio.

En este capítulo se han visto detalladamente cada uno de los puntos que influyen en el entrenamiento de un agente en un entorno simulado utilizando el aprendizaje por refuerzo con el algoritmo Q-Learning. Una vez realizados la batería de experimentos se analizan los resultados para conocer en qué situaciones se comporta mejor y peor el agente aprendido.

5

Validación experimental

Este capítulo describe los experimentos realizados en los entrenamientos con distintas configuraciones de parámetros combinando el número de acciones, el número de puntos de percepción y los circuitos. De estos experimentos se obtienen resultados que indican la mejor combinación para cada conjunto de prueba así como la mejor configuración global que resuelve la conducción autónoma y han permitido extraer varias lecciones sobre aplicar aprendizaje por refuerzo al contexto robótico.

5.1. Parámetros generales del aprendizaje

En esta sección se describen las características generales de los entrenamientos por refuerzo y define los diferentes escenarios donde se realizan esos entrenamientos y donde se evalúan los 'cerebros' obtenidos con el aprendizaje por refuerzo.

A nivel general todos los entrenamientos tienen una *duración de 2 horas*. Se define este valor debido a la complejidad creciente que existe cuando se aumentan el número de percepciones (1, 2 y 3) y el número de acciones (3, 5 y 7). La combinación más alta de posibilidades, 3 niveles de percepción con 7 acciones posibles, necesita mayor tiempo de entrenamiento dado que requiere de un mayor tiempo de exploración del espacio de estados. Este valor permite un equilibrio entre convergencia y tiempo razonable de entrenamiento y permite comprar de un modo justo todas las combinaciones de aprendizaje, pues a todas ellas se les concede el mismo tiempo de entrenamiento.

Para todos los entrenamientos se fijan los parámetros globales que definen el algoritmo de aprendizaje por refuerzo Q-Learning (*alpha*, *gamma* y *epsilon*). Los valores de estos parámetros se han ajustado a través de diferentes entrenamientos hasta conseguir unos valores que equilibran el aprendizaje con la resolución del ejercicio. Estos valores son:

- α: 0.8. Este valor es elegido por los buenos resultados que retorna el algoritmo en cuanto a número de episodios, tamaño de la Tabla-Q y tiempo de entrenamiento si se compara con el valor original (0.2).
- γ: 0.9. Fijado por la librería inicialmente así como por diferente bibliografía [37] [38].
- ϵ: 0.99. Con un valor alto inicialmente, en cada episodio se le aplica un factor de descuento de 0.998 que hará que el parámetro ϵ disminuya paulatinamente hasta el final del entrenamiento.

Los entrenamientos se realizan en dos de los tres circuitos creados: en el circuito Simple

y en el circuito de Nürburgring, dejando el circuito de Montreal únicamente para las evaluaciones. En ambos circuitos se realizan los entrenamientos con todas las combinaciones posibles entre los diferentes niveles de percepciones y varios conjuntos de acciones repitiéndose el experimento 3 veces para cada combinación con el objetivo de obtener una media más fiable que hace más representativo el resultado.

Tanto los entrenamientos como las posteriores evaluaciones ha sido realizadas con un ordenador portátil MSI gs63 con un procesador Intel i7 de 7ª generación, 16 Gb de memoria RAM y un disco duro SSD M.2 de 256 Gb.

5.2. Análisis de entrenamientos

En cada entrenamiento se han almacenado diferentes valores que permiten luego extraer conclusiones de cada configuración de parámetros. Estos valores cuentan el número de veces que se repite un estado concreto, conocer por cada episodio cuánta recompensa acumula el agente y saber por cada episodio el número de pasos que se consiguen dar. Estos valores se han volcado en un conjunto de gráficas para una mejor comprensión. Cada grupo de gráficas representa un grupo de entrenamiento que contiene 3 ensayos para cada combinación de número de percepciones y de actuaciones posibles.

La Figura 5.2 tiene un ejemplo de esta secuencia de gráficas. Corresponden a un entrenamiento realizado en el «Circuito Simple», con el conjunto de acciones simple (3 acciones) y un único punto de percepción simplificada.

Puede verse en la Figura 5.1 que el estado más frecuente es el 0 (centro de la línea) seguido de estados como el 2 o el 1. Dado que el circuito simple tiene un mayor número de curvas a derechas que a izquierdas, como se vio en las tablas de la sección 4.2, así como una curva muy larga a derechas, estos valores destacan más que el resto, que apenas ocurren en este circuito.

Dado que todos los entrenamientos tienen la misma duración, (2 horas), en el eje de abscisas puede verse en la Figura 5.2a que 2 de los 3 ensayos convergen muy rápidamente (gráficas verde y amarilla), en apenas 10 épocas y acumulando mucha recompensa, esto es, pasa más tiempo en regiones donde la recompensa retorna los valores más altos (el centro de la línea). El último de ellos (gráfica azul) no consigue una secuencia que le permita converger tan rápidamente.

Se contabiliza una vuelta completa al «Circuito Simple» cuando se superan un número de pasos consecutivos sin reiniciar el entorno. Para el caso del «Circuito Simple» este valor es de 4000 y está marcado en la gráfica de la Figura 5.2b con una línea horizontal roja.

La representación de estas mismas gráficas en valores numéricos puede verse en la Tabla 5.1.

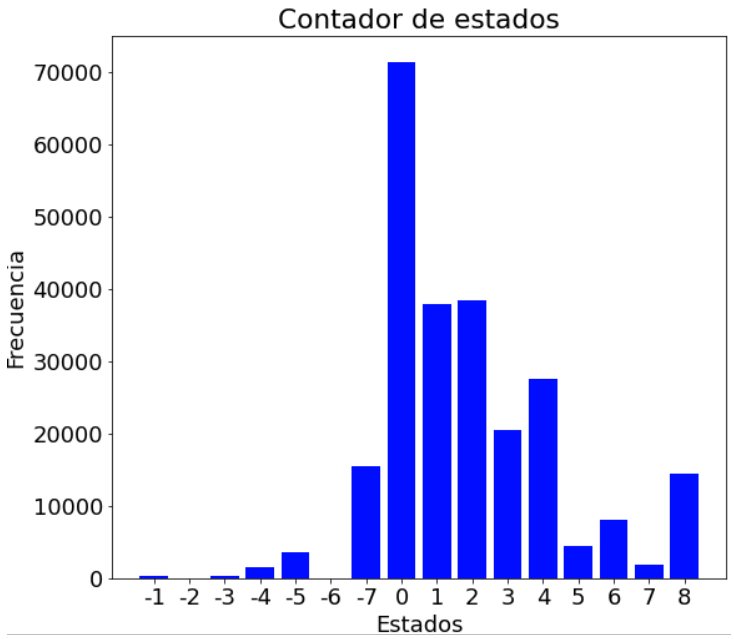
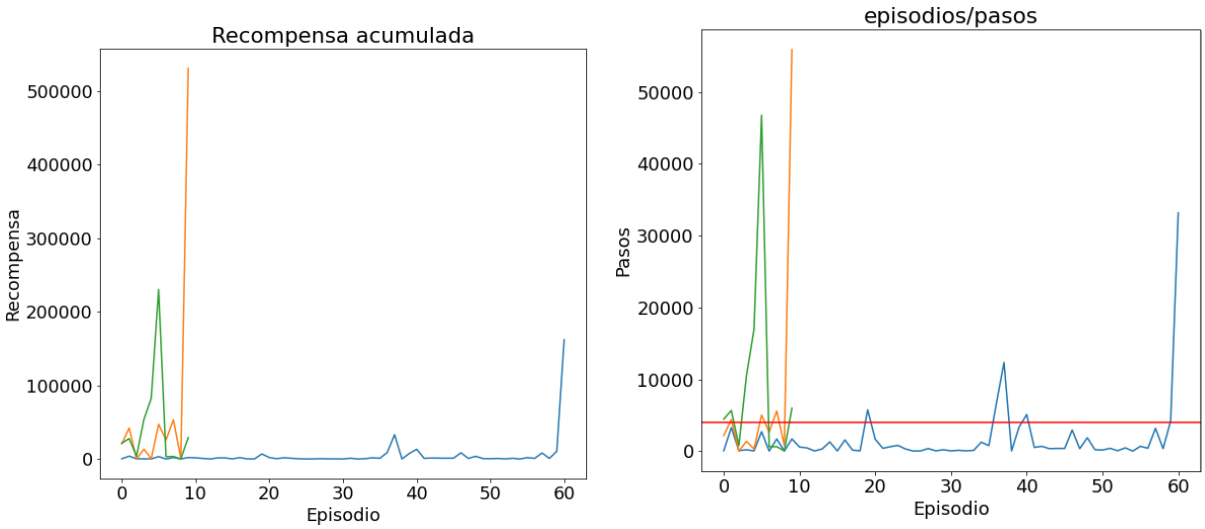


Figura 5.1: Análisis del número de estados de un entrenamiento en el «Circuito Simple», con un nivel de percepción y un conjunto simple de acciones.



(a) Recompensa acumulada por episodio. **(b)** Pasos por episodio.

Figura 5.2: Análisis de un entrenamiento en el circuito simple, con un único nivel de percepción y un conjunto simple de acciones.

Un ejemplo con otra configuración de parámetros puede verse en las Figuras 5.3 y 5.4. En este ejemplo el circuito de entrenamiento es Nürburgring con un conjunto de acciones medio (5 acciones) y 2 puntos de percepción simplificada.

Puede verse que el conjunto de estados para este ensayo (figura 5.3) es más variado comparado con el anterior. Se distribuyen los valores entre más estados enriqueciendo la Tabla-Q. El estado más frecuente es el $(-2, -4)$ con un total de 33065 veces.

En cuanto a la recompensa acumulada durante el entrenamiento (figura 5.4a) pueden

Circuito Simple			
Entrenamiento	**1**	**2**	**3**
Vuelta completada	Sí	Sí	Sí
Tiempo hasta completar	20:06 min	6:53 min	4:26 min
Épocas hasta completar	20	2	1
Valor de ϵ final	0.8	0.84	0.85
Tamaño de la Tabla-Q	40	23	22
Total de épocas	61	10	10

Tabla 5.1: Tabla resumen del conjunto de entrenamiento en el circuito «Circuito Simple» con 1 nivel de percepción y el conjunto de acciones simple.

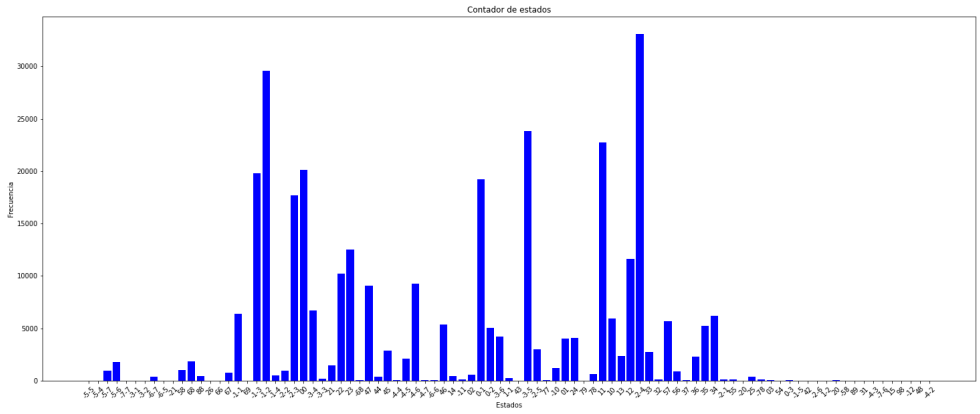
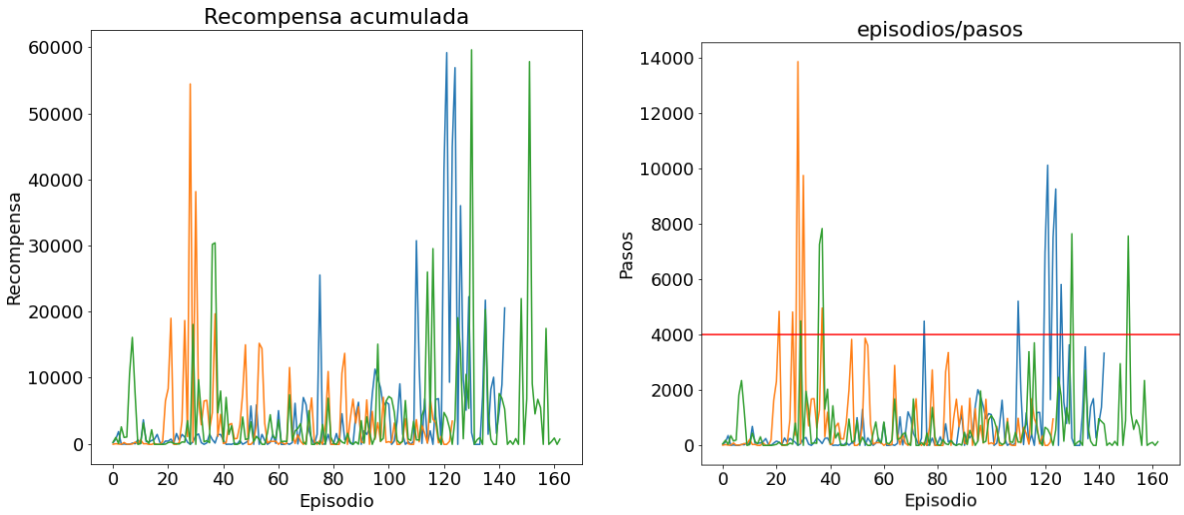


Figura 5.3: Análisis del número de estados de un entrenamiento en el circuito de Nürburgring, con dos niveles de percepción y un conjunto medio de acciones.

verse 3 modas que se replican en el número de pasos donde, con un valor aproximado a 4000 pasos se considera una vuelta al circuito (marcado con una línea horizontal roja en la figura 5.4b). Las tres modas superan este valor por lo que se concluye que hubo convergencia en el entrenamiento para esta configuración.
Los valores numéricos correspondientes a las gráficas pueden verse en la Tabla 5.2 para cada uno de los entrenamientos.

Nürburgring			
Entrenamiento	**1**	**2**	**3**
Vuelta completada	Sí	Sí	Sí
Tiempo hasta completar	19.32 min	9:11 min	13:12 min
Épocas hasta completar	75	21	29
Valor de ϵ final	0.72	0.75	0.72
Tamaño de la Tabla-Q	147	140	115
Total de épocas	143	124	163

Tabla 5.2: Tabla resumen del entrenamiento en el circuito de Nürburgring con 2 niveles de percepción y el conjunto de acciones medio.



(a) Recompensa acumulada por episodio. **(b)** Pasos por episodio.

Figura 5.4: Gráficas del estado de la recompensa y pasos dados por cada episodio.

Pueden verse las gráficas y tablas con todas las configuraciones probadas en el Anexo A, sección A.1.

5.3. Métricas de calidad de la conducción autónoma conseguida

Una vez finalizada cada combinación de entrenamientos se ejecuta el cerebro aprendido por refuerzo para su validación. Únicamente tomará de la Tabla-Q los pares de (*estado, acción*) con la mayor recompensa para cada estado. Esta acción da los valores que se envían de velocidad lineal y de velocidad angular a los motores del Fórmula-1. Este nuevo cerebro aprendido tiene 3 oportunidades para resolver una vuelta en a los circuitos de evaluación.

Las métricas por las que se mide la calidad de la conducción autónoma lograda con cada cerebro aprendido son:

- Tiempo por vuelta.
- Porcentaje del circuito completado.

La métrica que evalúa la calidad del algoritmo entrenado es el 'Tiempo por vuelta' para modelos que completan el circuito y la métrica 'Porcentaje de circuito completado' está más enfocada a la combinación de parámetros que no consigue completar la vuelta. Si una ejecución del modelo entrenado consigue completar una vuelta en un circuito de evaluación (los dos restantes de los tres creados), se considera que la aproximación de aprendizaje por refuerzo es factible y soluciona el problema para esa configuración de parámetros.

Para medir el tiempo que tarda en completar la vuelta se ha diseñado un sistema au-

tomático que registra la salida y llegada a la meta y detiene el programa, mostrando el tiempo que ha tardado en resolverlo y cuántos intentos ha necesitado. Para localizar al Fórmula-1 en el circuito se utiliza la librería de Gazebo `GetModelStates`[1] que retorna la posición del robot en el mundo simulado. Por cada paso del Fórmula-1 se obtienen los valores (x, y) y se comprueba que la distancia euclídea a la línea de salida (origen del mundo de Gazebo) no sea mayor que 0.5 metros, dejando un margen por si el Fórmula-1 atraviesa la línea de meta ligeramente desplazado del centro del circuito. Puede verse en el fragmento 5.1 el cálculo de la distancia del robot a la línea de meta.

```
def finish_line(self):
   x, y = self.get_position()
   current_position = np.array([x, y])

   dist = (self.start_pose - current_position) ** 2
   dist = np. sum(dist, axis=0)
   dist = np.sqrt(dist)

   if dist < max_distance:
      return True
   return False
```

Fragmento 5.1: Cálculo de la distancia al origen del mundo de Gazebo.

Si se cumple esta condición, el programa termina cerrando la sesión del entorno de Gym-Gazebo.

En el caso de que el cerebro entrenado no consiga completar el circuito se toman los puntos de referencia distribuidos regularmente por el circuito para verificar cuántos de ellos ha alcanzado el piloto con aprendizaje por refuerzo. Si ha pasado cerca de un punto se considera que lo ha conseguido. De este modo se tiene una aproximación al porcentaje del circuito que ha logrado completar el piloto con aprendizaje por refuerzo.Puede verse en la figura 5.5 un ejemplo en el «Circuito Simple» de los puntos de control generados por el piloto manual.

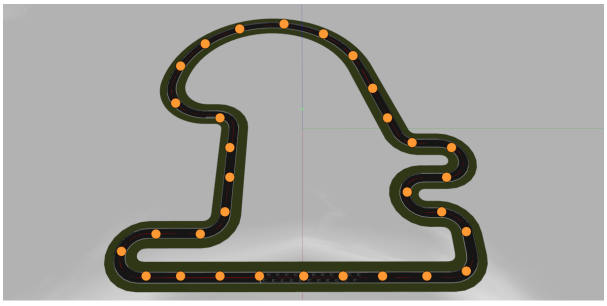


Figura 5.5: Ejemplo de puntos de control generados por el piloto manual.

Estableciendo un radio máximo de distancia de 5 metros entre el punto de referencia y el alcanzado por el entrenado se calcula el porcentaje de puntos de control que atraviesa el piloto de RL. El número de puntos generados para el «Circuito Simple» es de 52 y para Nürburgring de 65.

[1] `http://docs.ros.org/jade/api/gazebo_msgs/html/msg/ModelState.html`

5.4. Experimentos y ejecución típica

Una vez ejecutados todos los cerebros aprendidos en los escenarios donde no fueron entrenados se lleva a cabo un proceso de evaluación.

A nivel individual se han evaluado cómo el conjunto de acciones y los niveles de percepción simplificada afectan a los tiempos por vuelta del Fórmula-1 en cada circuito de evaluación. También se ha visto cómo en función del circuito de entrenamiento la evaluación, al ser más rica, está preparada para estados con menos frecuencia de aparición. Con el objetivo de tener una visión general de todas la evaluaciones, en la siguiente secuencia de tablas se tienen los resultados de todas las evaluaciones en los diferentes escenarios y con las diferentes configuraciones de parámetros de acciones y percepciones.

Las celdas marcadas en verde indican que el circuito se completó con éxito y las marcadas en naranja que solo se completó una parte y por tanto no es contabilizado el resultado como una solución válida.

La Tabla 5.3 muestra los resultados del entrenamiento en el circuito de Nürburgring y la evaluación en el «Circuito Simple».

Entrenamiento en Nürburgring y ejecución en «Circuito Simple»									
Percepciones	**1 punto**								
Acciones	**Simple**			**Medio**			**Difícil**		
Experimento	**1**	**2**	**3**	**1**	**2**	**3**	**1**	**2**	**3**
Tiempo (min)	3:33	3:53	3:18	3:51	6:08	3:33	5:54	5:51	4:10
Intentos	1	1	1	1	4	1	1	2	1
% de circuito	100	100	**100**	100	100	100	100	100	100
Percepciones	**2 puntos**								
Tiempo (min)	∞	3:43	3:55	∞	∞	5:30	5:45	5:03	4:32
Intentos	3	2	1	3	3	1	1	1	1
% de circuito	63.46	100	100	94.23	34.62	100	100	100	100
Percepciones	**3 puntos**								
Tiempo (min)	∞	∞	3.58	∞	∞	∞	∞	∞	∞
Intentos	3	3	3	3	3	3	3	3	3
% de circuito	17.31	95.8876.92	100	34.62	75	78.85	11.54	19.23	21.15

Tabla 5.3: Resultados del entrenamiento en Nürburgring y la evaluación en el «Circuito Simple».

La Tabla 5.4 muestra los resultados del entrenamiento en el circuito de Nürburgring y la evaluación en el Montreal.

Entrenamiento en Nürburgring y ejecución en Montreal									
Percepciones	**1 punto**								
Acciones	**Simple**			**Medio**			**Difícil**		
Experimento	**1**	**2**	**3**	**1**	**2**	**3**	**1**	**2**	**3**
Tiempo (min)	11:38	11:55	10:54	12:25	∞	11:32	20:03	∞	14:03
Intentos	1	1	1	1	3	1	1	3	1
% de circuito	100	100	100	100	7.06	100	100	11.76	100
Percepciones	**2 puntos**								
Tiempo (min)	∞	19:05	13:30	∞	13:38	18:33	∞	15:35	13:50
Intentos	3	3	2	3	3	2	3	1	1
% de circuito	47.06	100	100	12.35	100	100	41.76	100	100
Percepciones	**3 puntos**								
Tiempo (min)	∞	∞	∞	∞	∞	∞	∞	∞	∞
Intentos	3	3	3	3	3	3	3	3	3
% de circuito	8.24	95.88	1.65	6.47	1.65	5.88	1.18	4.71	1.18

Tabla 5.4: Resultados del entrenamiento en Nürburgring y la evaluación en el Montreal.

La Tabla 5.5 muestra los resultados del entrenamiento en el «Circuito Simple» y la evaluación en el Nürburgring.

Entrenamiento en Circuito Simple y ejecución en Nürburgring									
Percepciones	**1 punto**								
Acciones	**Simple**			**Medio**			**Difícil**		
Experimento	**1**	**2**	**3**	**1**	**2**	**3**	**1**	**2**	**3**
Tiempo (min)	4:22	4:13	∞	8:02	4:54	5:45	∞	5:24	∞
Intentos	1	1	3	1	1	2	3	1	3
% de circuito	100	100	55.77	100	100	100	55.77	100	55.77
Percepciones	**2 puntos**								
Tiempo (min)	∞	∞	∞	∞	∞	5:22	∞	6:00	∞
Intentos	3	3	3	3	3	1	3	2	3
% de circuito	7.69	9.62	9.62	9.62	11.54	100	15.38	100	11.54
Percepciones	**3 puntos**								
Tiempo (min)	∞	∞	∞	∞	∞	∞	∞	∞	∞
Intentos	3	3	3	3	3	3	3	3	3
% de circuito	9.62	7.69	7.69	7.69	3.85	7.69	0	0	9.62

Tabla 5.5: Resultados del entrenamiento en el Circuito Simple y evaluación en Nürburgring.

La Tabla 5.6 muestra los resultados del entrenamiento en el «Circuito Simple» y la evaluación en el Montreal.

Entrenamiento en Circuito Simple y ejecución en Montreal									
Percepciones	**1 punto**								
Acciones	**Simple**			**Medio**			**Difícil**		
Experimento	**1**	**2**	**3**	**1**	**2**	**3**	**1**	**2**	**3**
Tiempo (min)	12:47	11:07	14:28	20:30	13:17	15:29	∞	14:30	∞
Intentos	2	1	3	1	1	1	3	1	3
% de circuito	100	100	100	100	100	100	6.47	100	7.65
Percepciones	**2 puntos**								
Tiempo (min)	11:56	∞	11:06	15:49	∞	14:16	∞	15:41	∞
Intentos	1	3	2	1	3	1	3	1	3
% de circuito	100	62.35	100	100	10.59	100	1.18	100	6.47
Percepciones	**3 puntos**								
Tiempo (min)	∞	∞	∞	∞	∞	∞	∞	∞	∞
Intentos	3	3	3	3	3	3	3	3	3
% de circuito	8.24	2.94	6.47	1.76	1.76	1.76	1.18	1.76	5.88

Tabla 5.6: Resultados del entrenamiento en el Circuito Simple y evaluación en Montreal.

En esta sección se han visto distintos experimentos individuales y posteriormente más generales para extraer conclusiones sobre todas las evaluaciones de los cerebros aprendidos por refuerzo. Existen combinaciones de parámetros que resuelven la conducción autónoma en el circuito de manera satisfactoria y próxima en términos de tiempos por vuelta a un cerebro programado explícitamente. El algoritmo Q-Learning funciona bien para este comportamiento y lo resuelve satisfactoriamente.

Teniendo como referencia los tiempos por vuelta en cada circuito de un Fórmula-1 ejecutado con un código programado manualmente que resuelve el circuito de manera explícita, se selecciona el cerebro con la mejor combinación de parámetros que resuelve cada uno de ellos. En la tabla 5.7 puede verse el mejor tiempo para cada circuito así como la diferencia con respecto al piloto de referencia.

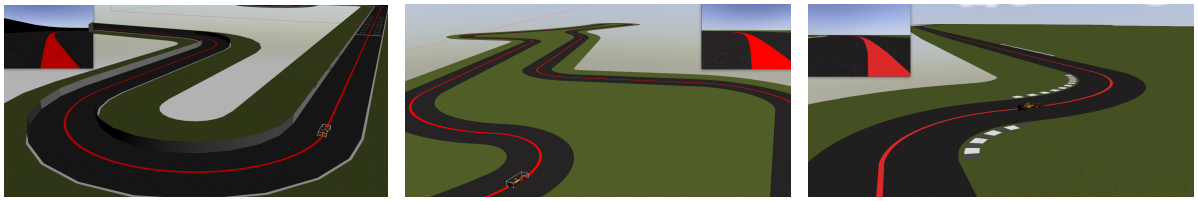
Circuito	Piloto manual	Piloto aprendido por RL	Diferencia
Simple	2.35 min	3.18 min	+43 seg
Nürburgring	3.19 min	4.13 min	+54 seg
Montreal	8.45 min	10.54 min	+2.09 min

Tabla 5.7: Tabla con los mejores tiempos por vuelta.

Para todos los resultados de la tabla la configuración de parámetros que mejor ha resuelto el circuito ha sido la misma: conjunto de acciones simple y 1 punto de percepción simplificada.

En la Figura 5.6 puede verse una captura tomada mientras los algoritmos entrenados por aprendizaje por refuerzo ejecutan en los diferentes circuitos. Está disponible también un vídeo [2] con la secuencia de entrenamiento y evaluación del cerebro con aprendizaje por refuerzo.

[2] `https://youtu.be/3jdxZTjPCss`



(a) Circuito Simple. **(b)** Nürburgring. **(c)** Montreal.

Figura 5.6: Circuitos durante la ejecución.

El circuito de Montreal es el único en el que se prueban los entrenamientos realizados en los otros dos circuitos restantes por lo que tiene el doble de pruebas de evaluación que el resto de circuitos. El mejor tiempo por vuelta en Montreal ha sido conseguido por un modelo entrenado en Nürburgring con una diferencia de 1 minuto en promedio con respecto al mismo modelo pero entrenado en el «Circuito Simple».

La diferencia principal de tiempos por vuelta entre el piloto manual y el aprendido por aprendizaje por refuerzo radica en dos factores:

- En la discretización de las velocidades angulares del Fórmula-1,
- y en su naturaleza reactiva pura, solo tiene en cuenta información instantánea, no tiene memoria ninguna.

El piloto manual cuenta en su código con un mecanismo de control por realimentación proporcional, integral y derivativo (PID) que le permite corregir pequeñas desviaciones sobre la línea e incorporar en su decisión información reciente (memoria). Por ejemplo puede suavizar las respuestas si el error está reduciéndose o aumentar la respuesta si el error está creciendo. Esto se consigue porque el sistema PID puede relacionar el error actual con el error en el pasado para así ajustar más suavemente la corrección y volver a la posición central. En la figura 5.7 puede verse la corrección del piloto manual ante un instante de error y como se realiza la corrección de manera suave hasta la estabilización.

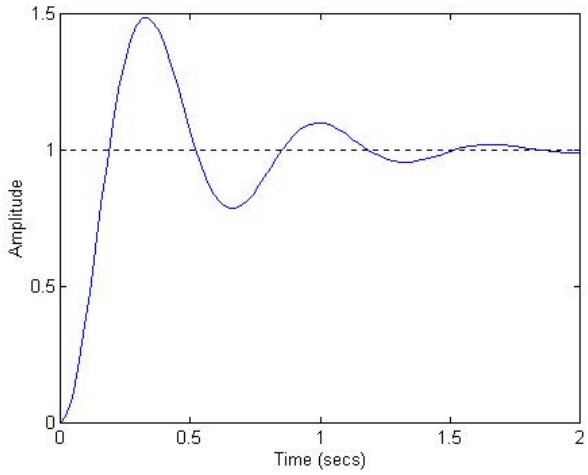


Figura 5.7: Control proporcional, integral y derivativo del piloto manual.

Por el contrario el sistema aprendido con RL es reactivo puro, rellena una tabla en la que

guarda cada estado y le asocia la mejor actuación posible a la luz "únicamente"del estado actual. Sin embargo, como muestran las componentes derivativa e integral de los controladores PID, la mejor actuación posible puede depender también del estado anterior e incluso de estados previos. El sistema de RL, tal y como está construido, no lo puede tener en cuenta.

El comportamiento resultante del agente será más parecido a un control basado en una corrección proporcional. Puede verse en la Figura 5.8 un ejemplo de una secuencia de estados donde el agente de aprendizaje por refuerzo intentar corregir con las acciones de su repertorio el desvío con respecto al centro de la línea del circuito (línea roja central).

Figura 5.8: Control proporcional en el piloto de RL.

El resultado de esto son *zig-zags* sobre la línea roja hasta que consigue estabilizar la dirección, en lugar de un movimiento más suave y paulatino como en el caso del PID. El tiempo de corrección hasta estabilizar el Fórmula-1 en el centro de la línea para un sistema PID es menor al conseguido con aprendizaje por refuerzo con el esquema utilizado.

A nivel de tiempos por vuelta se traduce en que el Fórmula-1 gobernado por un algoritmo de aprendizaje por refuerzo recorre más distancia sobre el circuito, que se traduce en mayor tiempo por vuelta.

5.5. Análisis de cardinalidad de percepciones

Este conjunto de experimentos se centran únicamente en comparar los resultados en las configuraciones donde se modifican los niveles de percepción. Los resultados de los entrenamientos combinando los diferentes niveles muestran que el tamaño de la Tabla-Q crece al igual que la complejidad en el entrenamiento hasta completar la vuelta al circuito. El tamaño de la Tabla-Q indica lo rico que es ese conjunto de estados y que, por tanto, deberá estar más preparado ante situaciones más complejas o repentinas. A su vez el tiempo que se tarda en converger o completar el circuito es mayor y esto se traduce en que para un único nivel de percepción se necesita menor tiempo de entrenamiento y para 3 niveles se necesita, en algunos casos más tiempo de entrenamiento del que se fija en este TFM.

En la tabla 5.8 puede verse cómo a medida que se aumentan el número de percepciones lo hace el tamaño de la Tabla-Q. Esta tabla mide un promedio de todos los entrenamientos realizados para cada una de las configuraciones de percepciones: 1, 2 y 3 puntos de percepción simplificada.

Puntos de percepción	**1**	**2**	**3**
Promedio de tamaño de la Tabla-Q	38	162	1062

Tabla 5.8: Tamaño promedio de la Tabla-Q de todos los entrenamientos

El tamaño del conjunto de estados (o de la Tabla-Q) depende también del circuito donde se entrena dado que el «Circuito Simple» tiene un repertorio de curvas más simple que Nürburgring por lo que hay estados que solo se dan en este segundo circuito, que se traduce en que la Tabla-Q tendrá más casos. Haciendo un promedio de los tamaños de las tablas por nivel de percepción y circuito puede verse que a mismo nivel de percepción, cambiando únicamente el circuito, en ocasiones hay tablas que son más del doble de grandes. Pueden verse estos valores en la Tabla 5.9.

Promedio de tamaño de las Tablas-Q		
Circuito/Percepciones	**Circuito Simple**	**Nürburgring**
1 punto	39	37
2 puntos	143	182
3 puntos	737	1386

Tabla 5.9: Promedio del tamaño de las Tablas-Q para distintos puntos de percepción

En la Tabla 5.10 puede verse la evaluación de los tiempos por vuelta del Fórmula-1 con los distintos niveles de percepción. Para configurar esta tabla se han seleccionado los mejores tiempos para cada punto de percepción para todos los conjuntos de acciones.

Mejor tiempo / Percepciones				
Ejecutado en	**Piloto Manual**	**1 punto**	**2 Puntos**	**3 Puntos**
Simple	2.35 min	3.18 min	3.43 min	3.58 min
Nürburgring	3.19 min	4.13 min	5.22 min	-
Montreal	8.45 min	10.54 min	11.06 min	-

Tabla 5.10: Mejores tiempos con diferentes puntos de percepción simplificada

La tabla de tiempos muestra que aumentando el número de percepciones se incrementan los tiempos por vuelta a diferencia de lo que puede sugerir la intuición a priori. Con el salto de un 1 punto a 2 puntos de percepción simplificada vemos que el Fórmula-1 es capaz de completar el circuito en cambio, aumentando a 3 el número de puntos perceptivos, puede verse que no se ha conseguido resolver en todos los casos.

Con los resultados globales en las tablas puede verse cómo según se va incrementando el número de percepciones simplificadas, en términos generales, la conducción autónoma se vuelve más compleja y no consigue resolverse.

Puede concluirse que *el mejor conjunto de percepciones* para resolver el circuito es con *una única percepción simplificada*.

5.6. Análisis de cardinalidad de acciones

Las diferentes combinaciones de acciones permiten tener un mayor o menor número de reacciones que el agente puede emplear para resolver más o menos situaciones. Aunque inicialmente, con el juego de acciones simple (3 acciones) pueden resolverse los tres circuitos, en algunas ocasiones existe algún estado donde es necesario utilizar alguna acción extra que haga corregir la trayectoria del Fórmula-1 de manera más drástica. Por otro lado, un juego de acciones simple implica perder menos velocidad y completar la vuelta en menos tiempo. Con el conjunto de estados difícil (7 acciones) el agente es más *agresivo*, lo que se traduce en cambios muy bruscos de dirección y una consiguiente reducción de la velocidad de avance y mayor tiempo en completar el circuito. Esa agresividad provoca en ocasiones que corrija en exceso y requiera reiniciar el episodio.

Un ejemplo donde un repertorio medio o difícil de acciones permite corregir más situaciones puede verse en la secuencia de imágenes de la Figura 5.9 donde en un momento del circuito donde hay una curva pronunciada y el Fórmula-1 comienza a corregir tarde. La trayectoria este juego de acciones más extenso permite recuperar la posición, al tener más valores de velocidad angular entre su repertorio.

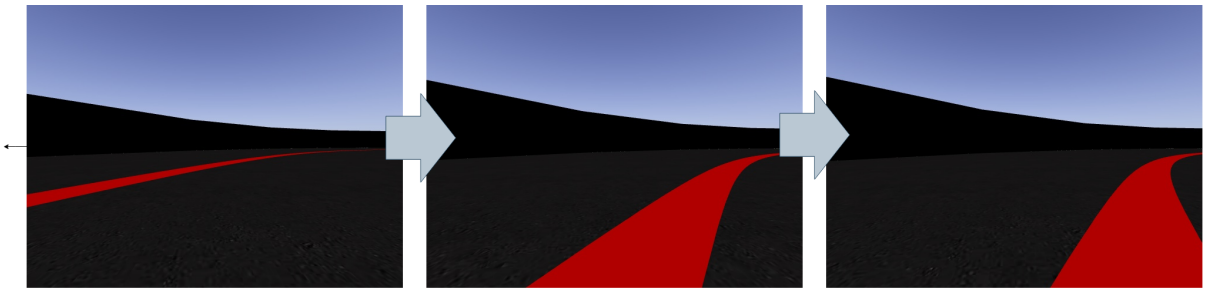


Figura 5.9: Entrada en una curva por mala posición.

Un juego de acciones simple no tendrá entre su repertorio tanta velocidad angular para poder reincorporarse a la línea y tenderá a salirse.

Puede verse en la tabla 5.11 la relación de tiempos por vuelta para los diferentes conjuntos de acciones.

Mejor tiempo / Acciones				
Ejecutado en	**Piloto Manual**	**Simple**	**Medio**	**Difícil**
Simple	2.35 min	3.18 min	3.33 min	4,10 min
Nürburgring	3.19 min	4.13 min	4.54 min	5.24 min
Montreal	8.45 min	10.54 min	11.06 min	14.03 min

Tabla 5.11: Mejores tiempos con diferentes conjuntos de acciones

Puede concluirse que *el mejor conjunto de acciones* para resolver el circuito es con *un conjunto de acciones simple*.

Destacar que, aún perdiendo velocidad en el tiempo por vuelta, un conjunto de acciones medio otorga ese añadido necesario para que el Fórmula-1 pueda solucionar más situaciones por lo que puede ser también válido si se va a conducir autónomamente en circuito con curvas muy reviradas.

5.7. Influencia del circuito de entrenamiento

Otro elemento que se tiene en cuenta en las evaluaciones experimentales es el circuito original donde se entrenaron los algoritmos de aprendizaje por refuerzo. El número de rectas, curvas a izquierdas, curvas a derechas y la complejidad de estas curvas añaden variedad que luego es utilizada en los circuitos de evaluación.

En la tabla 5.12 se muestra el mejor tiempo para el circuito de Montreal con los modelos entrenados en dos circuitos diferentes de entrenamiento: «Circuito Simple» y Nürburgring.

Mejor Tiempo / Circuitos		**Entrenado en**	
		Circuito Simple	**Nürburgring**
Ejecutado en	**Circuito Simple**	3:23 min	3:18 min
	Nürburgring	4:13 min	4:12 min
	Montreal	11:06 min	10:54 min

Tabla 5.12: Entrenamiento y ejecución en diferentes circuitos.

Puede concluirse que entrenamientos en circuitos con más variedad de curvas (estados) repercute en un repertorio más robusto para afrontar cualquier circuito de evaluación. El *mejor circuito* de los dos *de entrenamiento* que devuelve mejores tiempos de vuelta *es Nürburgring*.

Además, esta vista general refuerza la conclusión de que los entrenamientos en circuitos con más variedad de situaciones como Nürburgring, resuelven mejor los circuitos de evaluación. Cruzando la información entre la tabla 5.5 y 5.3 vemos que el segundo resuelve más casos que el primero.

6
Conclusiones y trabajos futuros

En los capítulos anteriores se ha enmarcado este TFM dentro de su contexto, descrito el problema y se ha presentado una solución justificando la elección de cada tipo de tecnología usada, además de las pruebas realizadas. En este último capítulo se presentan las conclusiones finales de este Trabajo de Fin de Máster así como las posibles líneas futuras de desarrollo.

6.1. Conclusiones

Tras analizar el trabajo realizado se puede verificar que se ha cumplido con el objetivo principal. Se ha creado una solución de control reactivo utilizando visión, basada en aprendizaje por refuerzo, para la conducción autónoma. El sistema aprendido es capaz de completar una vuelta a diferentes circuitos propuestos.
Se han satisfecho todos los objetivos enmarcados en el capítulo 1.2 y que se repasan a continuación:

- Se ha modificado y actualizado un entorno para el entrenamiento de algoritmos de aprendizaje por refuerzo utilizando visión y robots partiendo de OpenAI-Gym y Gym-Gazebo. Se ha creado un nuevo ejercicio en Gym-Gazebo que contiene todo lo necesario para llevar a cabo los entrenamientos y evaluaciones de un Fórmula-1 a través de diferentes circuitos y utilizando la cámara como sensor. En función de los valores obtenidos por el procesamiento de la imagen se comandan a los motores del robot los valores de velocidad lineal y angular correspondientes para completar el circuito.

- Se han entrenado diferentes combinaciones que solucionan el problema de conducción autónoma siguiendo una línea configurando distintos parámetros de percepción simplificada de la cámara, número de acciones posibles y el impacto de entrenar en uno u otro circuito, por sus características para el aprendizaje.

- Las mejores configuraciones de parámetros con las que se entrenan los modelos de aprendizaje por refuerzo dan como resultado una solución satisfactoria al problema, completando la vuelta al circuito en tiempos razonablemente buenos comparados con una solución a la conducción programada explícitamente.

También han sido satisfechos los requisitos especificados en la sección 1.2:

- Para las comunicaciones entre el programa y el robot se ha usado el sistema operativo robótico ROS. El uso de las diferentes librerías de este software permite conocer

la posición del coche en el mundo, traducir la imagen captada por la cámara en tipos de datos compatibles con las librerías de visión, como OpenCV y facilidad en el cálculo de las operaciones sobre las imágenes con la librería científica Numpy.

- Como entorno de simulación donde se realizan los entrenamientos y evaluaciones se ha utilizado el simulador en 3D Gazebo, que acompaña a ROS. Un elemento importante desde el punto de vista del rendimiento ha sido poder lanzar únicamente la parte servidora del simulador, liberando carga al ordenador durante los entrenamientos.

- Se ha mejorado y ampliado la librería Gym-Gazebo para la creación de un nuevo entorno que permite entrenar algoritmos de aprendizaje por refuerzo que solucionen el problema del ejercicio Sigue-Líneas planteado en el TFM. El repositorio donde se encuentra el código modificado permite replicar el ejercicio con facilidad así como modificar las configuraciones de parámetros para un nuevo entrenamiento de manera ágil cambiando o añadiendo circuitos, acciones, niveles de percepción, número de estados, etc.

6.2. Trabajos futuros

Durante el desarrollo han ido surgiendo ideas de posible extensión del trabajo. Las principales son:

- **Entrenamiento de parámetros v y w**: El número de acciones presentadas en este TFM eran limitadas y no permitían una conducción muy suave. La discretización de los valores de velocidad lineal y angular no dan posibilidad a que el vehículo acelere gradualmente en las curvas o gire menos si el error es pequeño. Una posible línea por donde se podría extender este trabajo sería incluir un rango continuo de valores que le permita al vehículo aprender y aplicar el mejor valor para cada estado. Dado que el espacio de estados y acciones puede ser muy complejo, limitar la capacidad a un subconjunto acota esa dificultad. Esto mejorará el tiempo por vuelta dado que el Fórmula-1 circularía por encima de la línea constantemente.

- **Extensión del número de problemas robóticos entrenados con aprendizaje por refuerzo**: Incluir en la librería entornos simulados que permitan ejecutar otro tipo de robots como por ejemplo, drones o tareas con brazos mecánicos. Comportamientos como el «Sigue persona» usando drones, atravesar un entorno de pruebas esquivando obstáculos usando visión con drones o clasificación de objetivos usando un brazo mecánico entrenado con algoritmos de aprendizaje por refuerzo son ejemplos simples donde poder aplicar este tipo de técnicas de RL a la robótica.

- **Aprendizaje por refuerzo profundo, DQN**: El siguiente salto lógico en la resolución de problemas de aprendizaje por refuerzo es incluir todas las mejoras recientes en técnicas de aprendizaje profundo (*Deep Learning*). Concretamente, el algoritmo Q-Learning utilizado en este TFM tiene su equivalente usando técnicas de aprendizaje profundo con el algoritmo DQN. Esta mejora no necesitaría un procesamiento

manual de la imagen para extraer de la línea valores como el centro sino que es a través de la imagen de entrada donde se extraen las características aprendidas que desembocan en acciones y recompensas útiles para el robot.

Hubo dos grandes hitos alcanzados durante la realización este trabajo: el ensamblaje de todas las piezas para tener el nuevo entorno disponible en el que realizar los entrenamientos con las distintas configuraciones de parámetros; y la creación del ejercicio en sí, que ha necesitado en bastantes ocasiones el análisis de los resultados y la interpretación del comportamiento del robot para entender las necesidades para completar su entrenamiento. Esta labor «didáctica» aporta mucho valor a los algoritmos de aprendizaje por refuerzo que consiguen, en algunas ocasiones, resultados sorprendentes.

A nivel personal este Trabajo de Fin de Máster ha supuesto un reto en varios sentidos. El campo de la robótica siempre había sido una asignatura pendiente y este trabajo me ha ayudado a finalmente dar el paso y asomarme a su inmensidad. La gran cantidad de piezas que componen ROS y Gazebo, así como las librerías de Python que permiten la comunicación con los componentes, genera impresión en un primer momento pero la focalización en un problema en concreto hace que paulatinamente vayan cobrando sentido. Por otro lado, siempre he sentido curiosidad por las técnicas del aprendizaje por refuerzo, por su comportamiento más «humano» y sus resultados sorprendentes y, a veces, impredecibles.

Juntar ambos mundos en este trabajo ha supuesto un enriquecimiento personal que ha aportado valor, experiencia y otro punto de vista para conocer nuevas tecnologías, ver su relación entre sí y su aplicación (en pequeña escala) a la vida real como es el coche autónomo.

Bibliografía

[1] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, Anil Anthony Bharath. .A Brief Survey of Deep Reinforcement Learning (BMVC)"(2017).

[2] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy P Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. .Asynchronous Methods for Deep Reinforcement Learning"(2016).

[3] Richard S. Sutton and Andrew G. Barto. Reinforcement Learning: An Introduction"(2014, 2015, 2016, 2017).

[4] Christopher JCH Watkins and Peter Dayan. "Machine Learning"(1992).

[5] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, Martin Riedmiller. "Playing Atari with Deep Reinforcement Learning"(2018).

[6] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. "Human-Level Control through Deep"(2015).

[7] Miles Brundage Anil Anthony Bharath. Kai Arulkumaran, Marc Peter Deisenroth. .A brief survey of Deep Reinforcement Learning"(2017).

[8] Gavin A Rummery and Mahesan Niranjan. .On-line q-learning using *connectionistsystems*"(1994).

[9] Jan Peters Daan Wierstra, Alexander Forster and Jurgen Schmidhuber. Recurrentpolicy gradients"(2010).

[10] Sebastian Ruder. .An overview of gradient descent optimization algorithms"(2017).

[11] Faustino Gomez and Jurgen Schmidhuber. .Evolving modular fast-weight networks forcontrol"(2005).

[12] Satinder Singh Yishay Mansour Richard S. Sutton, David McAllester. Policy gradient methods for reinforcement learning with function approximation"(1999).

[13] Pieter Abbeel Justin Fu, Sergey Levine. .One-shot learning of manipulation skills withonline dynamics adaptation and neural network priors"(2016).

[14] Naveen Venkat. The curse of dimensionality. Inside out"(2018).

[15] Eric Kolve Joseph J. Lim Abhinav Gupta Li Fei-Fei Ali Farhadi Yuke Zhu,Roozbeh Mottaghi. "Target-driven visual navigation in indoor scenes using deep reinforcement learning"(2016).

[16] Mehdi Mirza Alex Graves Timothy P Lillicrap Tim Harley David SilverVolodymyr Mnih, Adria Puigdomenech Badia and Koray Kavukcuoglu. .Asynchronous methods for deep reinforcement learning"(2016).

[17] Varaiya, P. and Walrand, J. C. "Multi-armed bandit problems and resource sharing systems. In Computer Performance and Reliability, Proceedings of the International Workshop"(1983).

[18] J. N. Tsitsiklis and B. V. Roy, “Average cost temporal-difference learning"(1999).

[19] Bellman, R. “A Markovian decision process.” Technical report, DTIC Document (1957).

[20] Bertsekas, D. and Tsitsiklis, J. Neuro-Dynamic Programming. Belmond, MA: Athena Scientific (1996).

[21] J.-C. Walter and G. T. Barkema. .An introduction to Monte Carlo methods"(1999).

[22] Thomas B Sch on Niklas Wahlstrom and Marc P Deisenroth. Learning deep dynamicalmodels from image pixels (2015).

[23] David Silver Andrei A Rusu Joel Veness Marc G Bellemare Alex Graves Mar-tin Riedmiller Andreas K Fidjeland Georg Ostrovski et al. Volodymyr Mnih, Ko-ray Kavukcuoglu. "Human-level control through deep"(2015).

[24] Mohamed Medhat Gaber, Ahmed Hussein and Eyad Elyan. "Deep active learning forautonomous navigation"(2016).

[25] Ronan Collobert Yoshua Bengio, Jerome Louradour and Jason Weston. Çurriculum learning"(2009).

[26] Nando de Freitas Jakob Foerster, Yannis M Assael and Shimon Whiteson. "Learning tocommunicate with deep multi-agent reinforcement learning"(2016).

[27] Eric Tzeng, Coline Devin, Judy Hoffman, Chelsea Finn, Xingchao Peng, Sergey Levine, Kate Saenko, and Trevor Darrell. "Towards Adapting Deep Visuomotor Representations from Simulated to Real Environments"(2016).

[28] Tejas D Kulkarni Tom Erez Peter Battaglia Misha Denil, Pulkit Agrawal and Nandode Freitas. "Learning to perform physics experiments via deep reinforcement learning"(2017).

[29] Dagui Chen1, Qi Yan1, Shangqi Guo1, Zhile Yang1, Xin Su1 and Feng Chen. "Learning Effective Subgoals with Multi-Task Hierarchical. Reinforcement Learning"(2019).

[30] Oliver Cameron. .ªdvancing Self-Driving Cars with Reinforcement and Imitation Learning"(2020).

[31] Łukasz Kaiser, Mohammad Babaeizadeh, Piotr Miłos, Błażej Osiński, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, Afroz Mohiuddin, Ryan Sepassi, George Tucker, Henryk Michalewski. "Model Based Reinforcement Learning for Atari"(2020).

[32] David Ha, Jürgen Schmidhuber. "World Models"(2018).

[33] 'Crecimiento de la robótica en la última década'.

[34] Librería Gym Gazebo (erlerobot): `https://github.com/erlerobot/gym-gazebo`.

[35] Extending the OpenAI Gym for robotics: a toolkit for reinforcement learning using ROS and Gazebo (Paper del software Gym Gazebo (2016).)

[36] Gym, de OpenAI. Librería para entrenamiento de entornos utilizando el aprendizaje por refuerzo.

[37] Aurélien Géron. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition"(2019) `https://www.oreilly.com/library/view/hands-on-machine-learning/9781492032632/`.

[38] Maxim Lapan. "Deep Reinforcement Learning Hands-On - Second Edition."(2020) https://www.packtpub.com/product/deep-reinforcement-learning-hands-on-second-edition/9781838826994

A
Anexos

A.1. Resultados de los entrenamientos

En esta sección se completan los resultados mostrados en el apartado 5.2 en relación a los resultados que devuelven las diferentes combinaciones de entrenamientos.

A.1.1. Circuito simple, conjunto de acciones medio y un punto de percepción

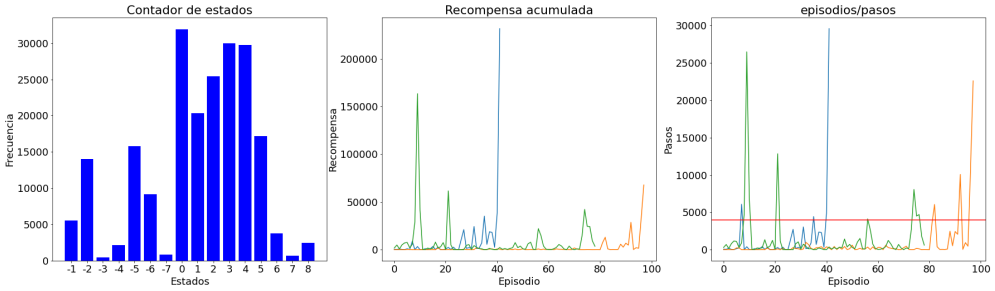


Figura A.1: Gráfica de resultados del entrenamiento en el «Circuito Simple» con un conjunto de acciones medio y 1 punto de nivel de percepción.

Tabla de entrenamiento en el Circuito Simple			
Entrenamiento	**1**	**2**	**3**
Vuelta completada	Sí	Sí	Sí
Tiempo hasta completar	05:19 min	21:32 min	10:33 min
Épocas hasta completar	7	82	8
Valor de ϵ final	0.81	0.76	0.79
Tamaño de la Tabla-Q	53	55	33
Nº total de épocas	42	98	79

Tabla A.1: Resultados del entrenamiento en el «Circuito Simple» con un conjunto de acciones medio y 1 punto de nivel de percepción.

A.1.2. Circuito simple, conjunto de acciones difícil y un punto de percepción

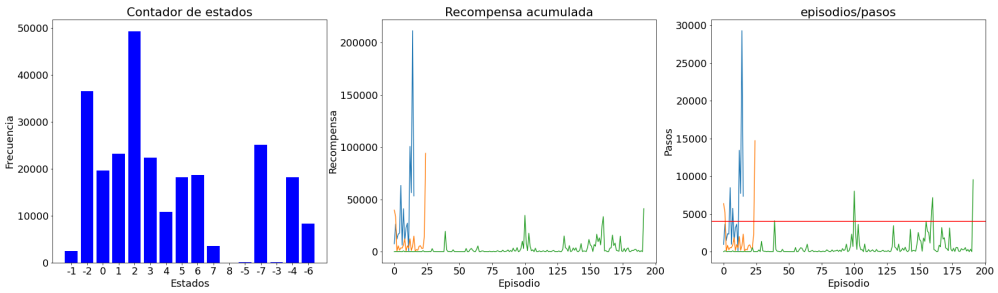


Figura A.2: Gráfica de resultados del entrenamiento en el «Circuito Simple» con un conjunto de acciones difícil y 1 punto de nivel de percepción.

Tabla de entrenamiento en el Circuito Simple			
Entrenamiento	**1**	**2**	**3**
Vuelta completada	Sí	Sí	Sí
Tiempo hasta completar	16:28 min	04:28 min	08:07 min
Épocas hasta completar	5	1	39
Valor de ϵ final	0.84	0.80	0.68
Tamaño de la Tabla-Q	27	29	73
Nº total de épocas	16	25	192

Tabla A.2: Resultados del entrenamiento en el «Circuito Simple» con un conjunto de acciones difícil y 1 punto de nivel de percepción.

A.1.3. Circuito simple, conjunto de acciones simple y dos puntos de percepción

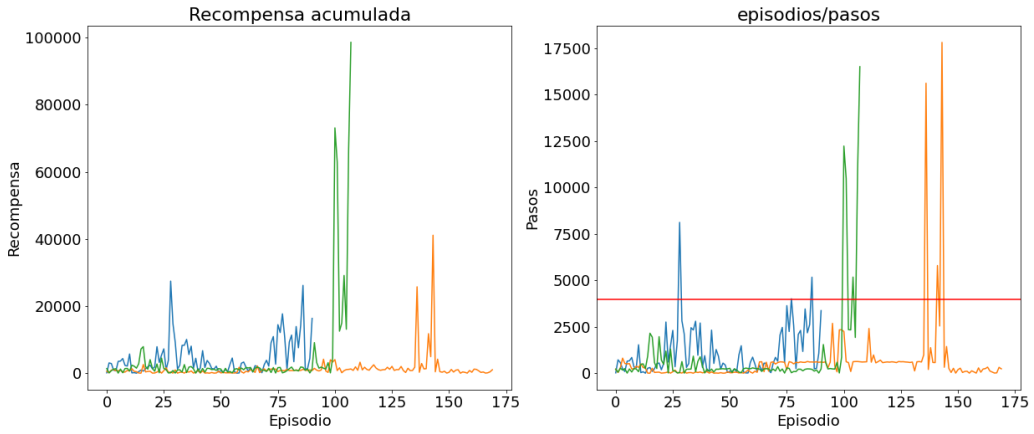


Figura A.3: Gráfica de resultados del entrenamiento en el «Circuito Simple» con un conjunto de acciones simple y 2 puntos de nivel de percepción.

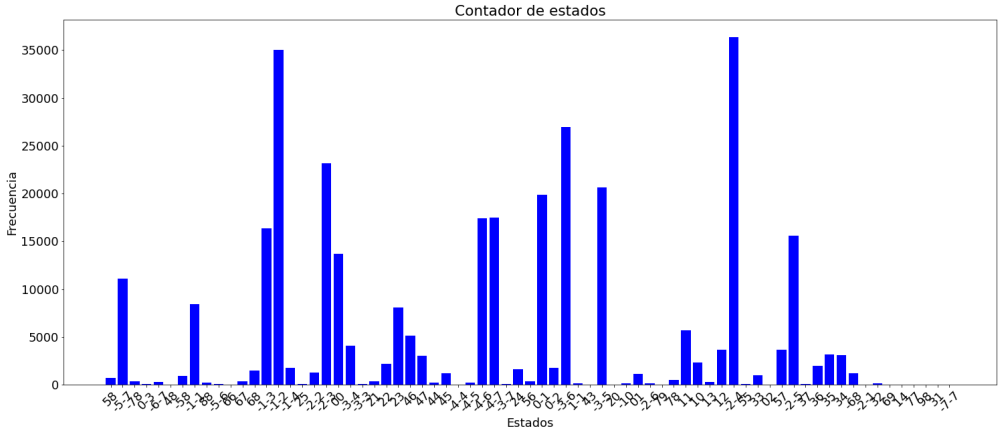


Figura A.4: Gráfica de resultados de la distribución de los estados.

Tabla de entrenamiento en el Circuito Simple			
Entrenamiento	**1**	**2**	**3**
Vuelta completada	Sí	Sí	Sí
Tiempo hasta completar	1:23:35 horas	1:09:11 horas min	41:55 min
Épocas hasta completar	77	136	100
Valor de ϵ final	0.79	0.73	0.76
Tamaño de la Tabla-Q	106	126	93
Nº total de épocas	91	170	108

Tabla A.3: Resultados del entrenamiento en el «Circuito Simple» con un conjunto de acciones simple y 2 puntos de nivel de percepción.

A.1.4. Circuito simple, conjunto de acciones medio y dos puntos de percepción

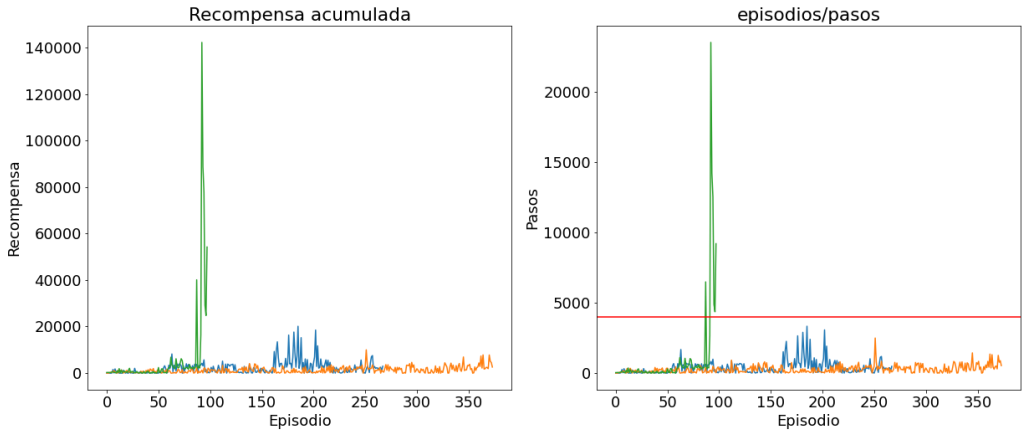


Figura A.5: Gráfica de resultados del entrenamiento en el «Circuito Simple» con un conjunto de acciones medio y 2 puntos de nivel de percepción.

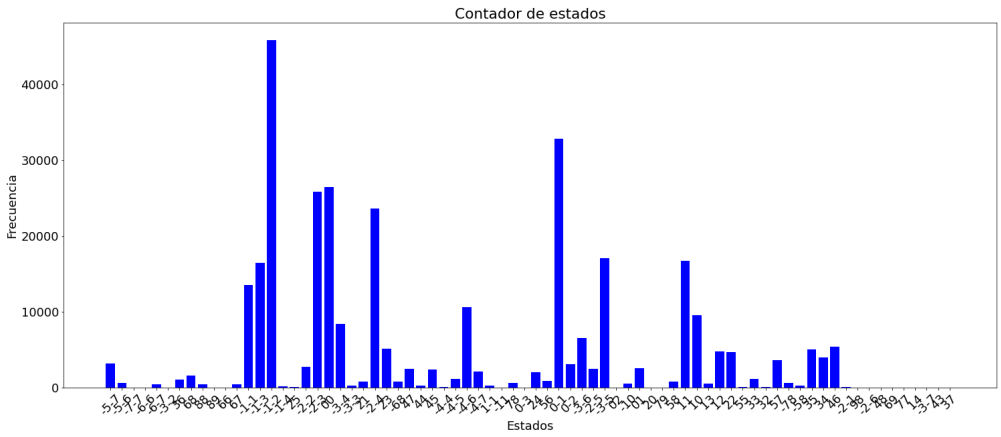


Figura A.6: Gráfica de resultados de la distribución de los estados.

Tabla de entrenamiento en el Circuito Simple			
Entrenamiento	**1**	**2**	**3**
Vuelta completada	No	No	Sí
Tiempo hasta completar	∞	∞	26:55 min
Épocas hasta completar	-	-	87
Valor de ϵ final	0.65	0.58	0.67
Tamaño de la Tabla-Q	163	168	115
Nº total de épocas	268	374	97

Tabla A.4: Resultados del entrenamiento en el «Circuito Simple» con un conjunto de acciones medio y 2 puntos de nivel de percepción.

A.1.5. Circuito simple, conjunto de acciones difícil y dos puntos de percepción

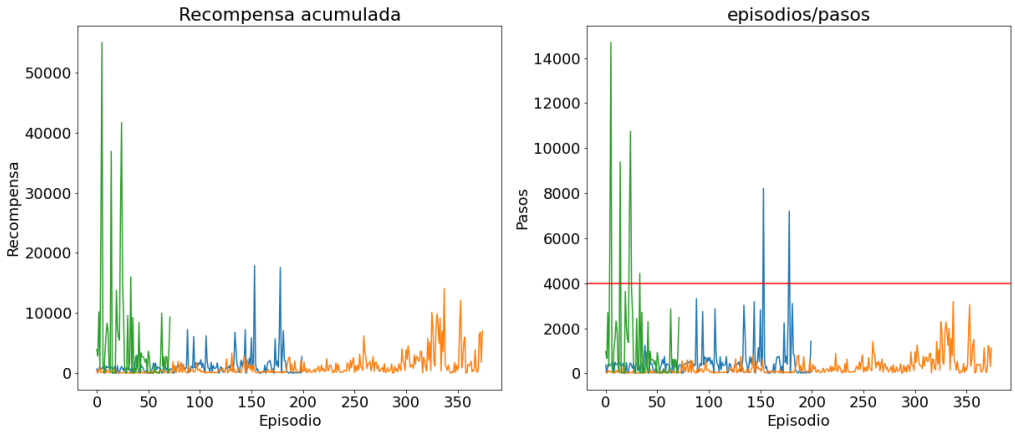


Figura A.7: Gráfica de resultados del entrenamiento en el «Circuito Simple» con un conjunto de acciones difícil y 2 puntos de nivel de percepción.

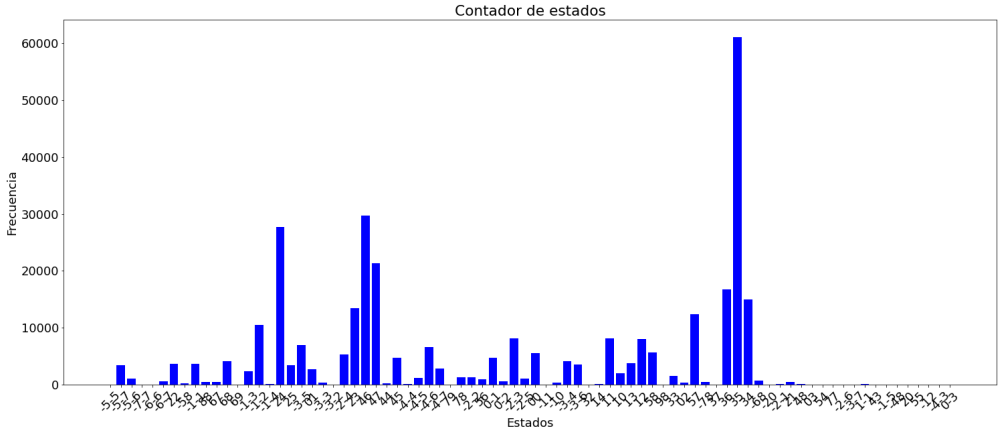


Figura A.8: Gráfica de resultados de la distribución de los estados.

Tabla de entrenamiento en el Circuito Simple			
Entrenamiento	**1**	**2**	**3**
Vuelta completada	Sí	Sí	Sí
Tiempo hasta completar	1:25:03 horas	1:58:26 horas	09:30 min
Épocas hasta completar	153	375	4
Valor de ϵ final	0.70	0.56	0.80
Tamaño de la Tabla-Q	184	214	123
Nº total de épocas	200	375	72

Tabla A.5: Resultados del entrenamiento en el «Circuito Simple» con un conjunto de acciones difícil y 2 puntos de nivel de percepción.

A.1.6. Circuito simple, conjunto de acciones simple y tres puntos de percepción

El estado más frecuente es el (137) con un valor de 32382 veces.

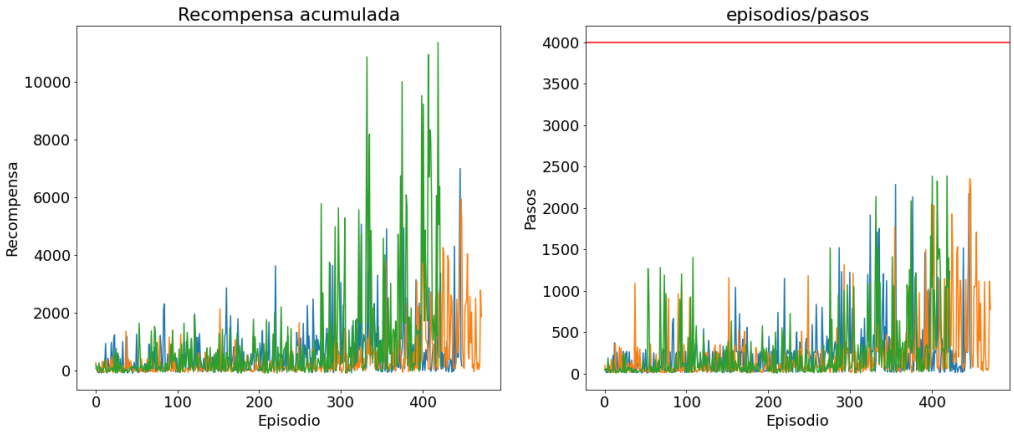


Figura A.9: Gráfica de resultados del entrenamiento en el «Circuito Simple» con un conjunto de acciones simple y 3 puntos de nivel de percepción.

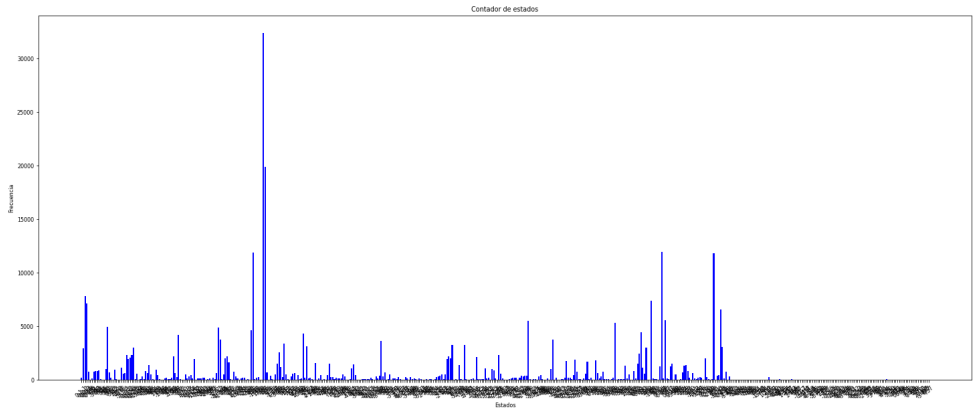


Figura A.10: Gráfica de resultados de la distribución de los estados.

Tabla de entrenamiento en el Circuito Simple			
Entrenamiento	**1**	**2**	**3**
Vuelta completada	No	No	No
Tiempo hasta completar	∞	∞	∞
Épocas hasta completar	-	-	-
Valor de ϵ final	0.51	0.48	0.52
Tamaño de la Tabla-Q	590	702	626
Nº total de épocas	449	473	424

Tabla A.6: Resultados del entrenamiento en el «Circuito Simple» con un conjunto de acciones simple y 3 puntos de nivel de percepción.

A.1.7. Circuito simple, conjunto de acciones medio y tres puntos de percepción

El estado más frecuente es el $(1, 3, 7)$ con un valor de 10698 veces.

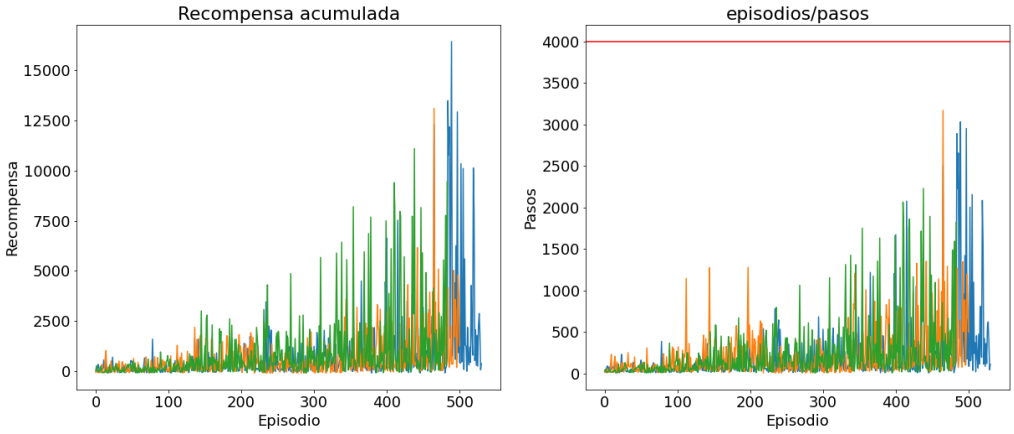


Figura A.11: Gráfica de resultados del entrenamiento en el «Circuito Simple» con un conjunto de acciones medio y 3 puntos de nivel de percepción.

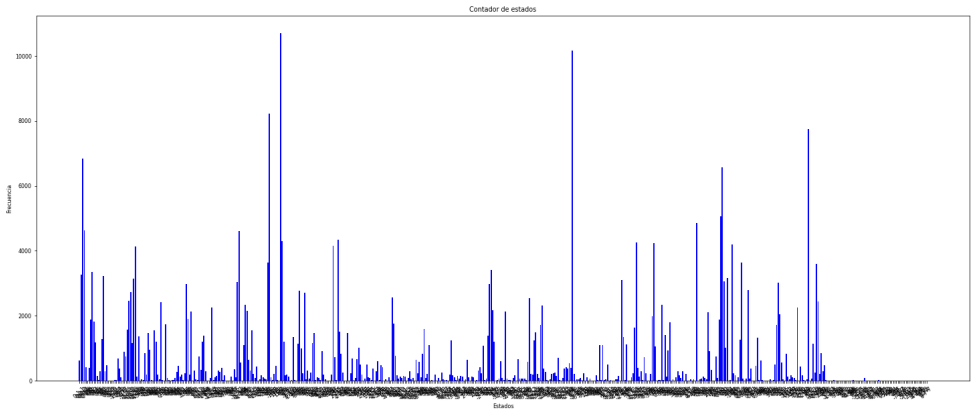


Figura A.12: Gráfica de resultados de la distribución de los estados.

Tabla de entrenamiento en el Circuito Simple			
Entrenamiento	**1**	**2**	**3**
Vuelta completada	No	No	No
Tiempo hasta completar	∞	∞	∞
Épocas hasta completar	-	-	-
Valor de ϵ final	0.45	0.48	0.48
Tamaño de la Tabla-Q	857	745	792
Nº total de épocas	531	501	484

Tabla A.7: Resultados del entrenamiento en el «Circuito Simple» con un conjunto de acciones medio y 3 puntos de nivel de percepción.

A.1.8. Circuito simple, conjunto de acciones difícil y tres puntos de percepción

El estado más frecuente es el $(0, -2, -6)$ con un valor de 12814 veces.

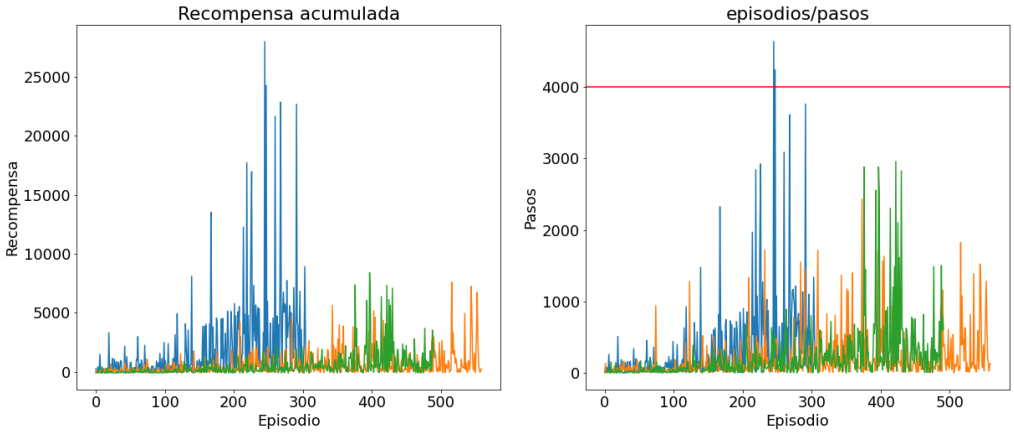


Figura A.13: Gráfica de resultados del entrenamiento en el «Circuito Simple» con un conjunto de acciones difícil y 3 puntos de nivel de percepción.

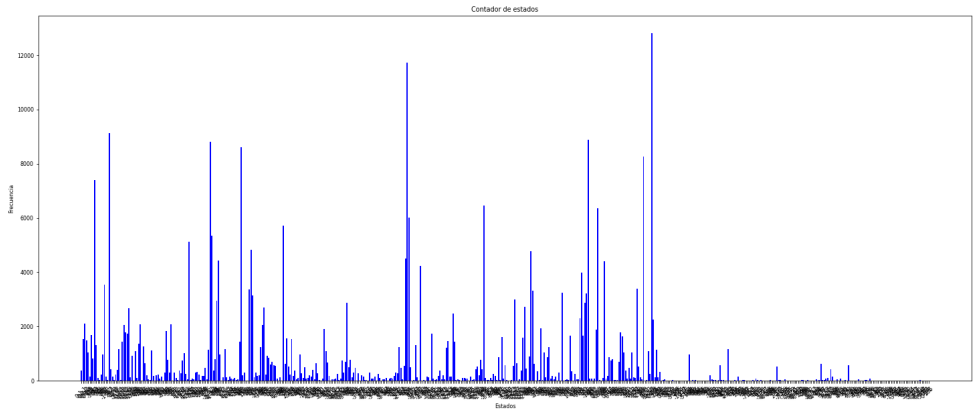


Figura A.14: Gráfica de resultados de la distribución de los estados.

Tabla de entrenamiento en el Circuito Simple			
Entrenamiento	**1**	**2**	**3**
Vuelta completada	Sí	No	No
Tiempo hasta completar	1:12:30 horas	∞	∞
Épocas hasta completar	245	-	-
Valor de ϵ final	0.61	0.44	0.48
Tamaño de la Tabla-Q	648	869	804
Nº total de épocas	306	560	491

Tabla A.8: Resultados del entrenamiento en el «Circuito Simple» con un conjunto de acciones difícil y 3 puntos de nivel de percepción.

A.1.9. Circuito de Nürburgring, conjunto de acciones simple y un punto de percepción

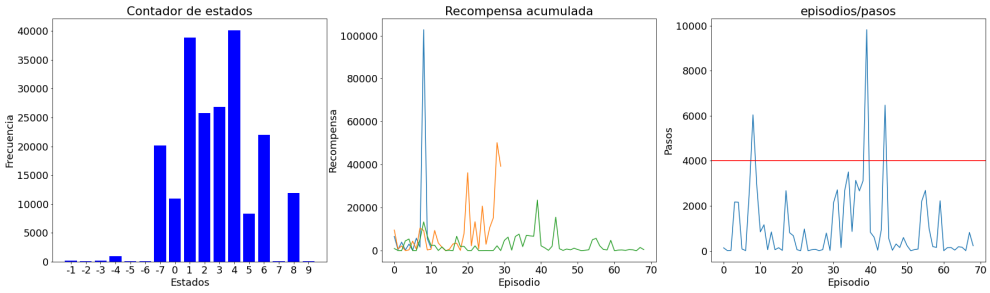


Figura A.15: Gráfica de resultados del entrenamiento en el Nürburgring con un conjunto de acciones simple y 1 punto de nivel de percepción.

Tabla de entrenamiento en Nürburgring			
Entrenamiento	**1**	**2**	**3**
Vuelta completada	Sí	Sí	Sí
Tiempo hasta completar	07:43 min	10:34 min	11:57 min
Épocas hasta completar	8	8	8
Valor de ϵ final	0.84	0.83	0.80
Tamaño de la Tabla-Q	26	34	41
Nº total de épocas	11	30	69

Tabla A.9: Resultados del entrenamiento en el Nürburgring con un conjunto de acciones simple y 1 punto de nivel de percepción.

A.1.10. Circuito de Nürburgring, conjunto de acciones medio y un punto de percepción

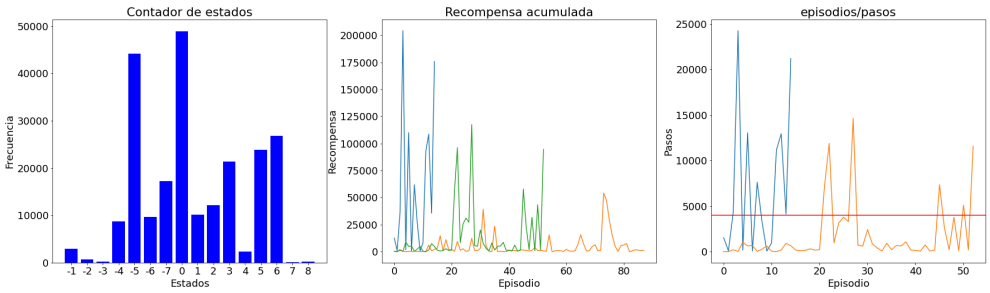


Figura A.16: Gráfica de resultados del entrenamiento en el Nürburgring con un conjunto de acciones medio y 1 punto de nivel de percepción.

Tabla de entrenamiento en Nürburgring			
Entrenamiento	**1**	**2**	**3**
Vuelta completada	Sí	Sí	Sí
Tiempo hasta completar	10:34 min	24:53 min	11:25 min
Épocas hasta completar	3	31	21
Recompensa en esa época	18803	3646	7846
Valor de ϵ final	0.84	0.78	0.81
Tamaño de la Tabla-Q	31	53	41
Nº total de épocas	15	88	53

Tabla A.10: Resultados del entrenamiento en el Nürburgring con un conjunto de acciones medio y 1 punto de nivel de percepción.

A.1.11. Circuito de Nürburgring, conjunto de acciones difícil y un punto de percepción

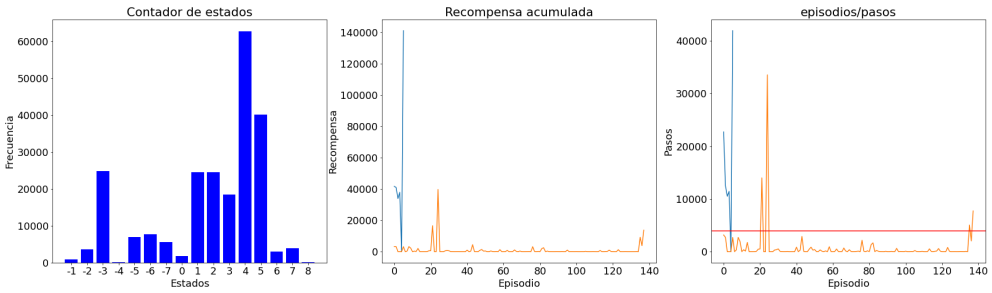


Figura A.17: Gráfica de resultados del entrenamiento en el Nürburgring con un conjunto de acciones difícil y 1 punto de nivel de percepción.

Tabla de entrenamiento en Nürburgring			
Entrenamiento	**1**	**2**	**3**
Vuelta completada	Sí	Sí	Sí
Tiempo hasta completar	29:13 min	22:47 min	3:56 min
Épocas hasta completar	2	21	1
Valor de ϵ final	0.85	0.72	0.85
Tamaño de la Tabla-Q	27	71	10
Nº total de épocas	6	138	1

Tabla A.11: Resultados del entrenamiento en el Nürburgring con un conjunto de acciones difícil y 1 punto de nivel de percepción.

A.1.12. Circuito de Nürburgring, conjunto de acciones simple y dos puntos de percepción

Tabla de entrenamiento en Nürburgring			
Entrenamiento	**1**	**2**	**3**
Vuelta completada	Sí	Sí	Sí
Tiempo hasta completar	29:31 min	52:30 min	1:15:52 horas
Épocas hasta completar	224	285	357
Valor de ϵ final	0.61	0.51	0.53
Tamaño de la Tabla-Q	171	149	175
Nº total de épocas	281	429	401

Tabla A.12: Resultados del entrenamiento en el Nürburgring con un conjunto de acciones simple y 2 puntos de nivel de percepción.

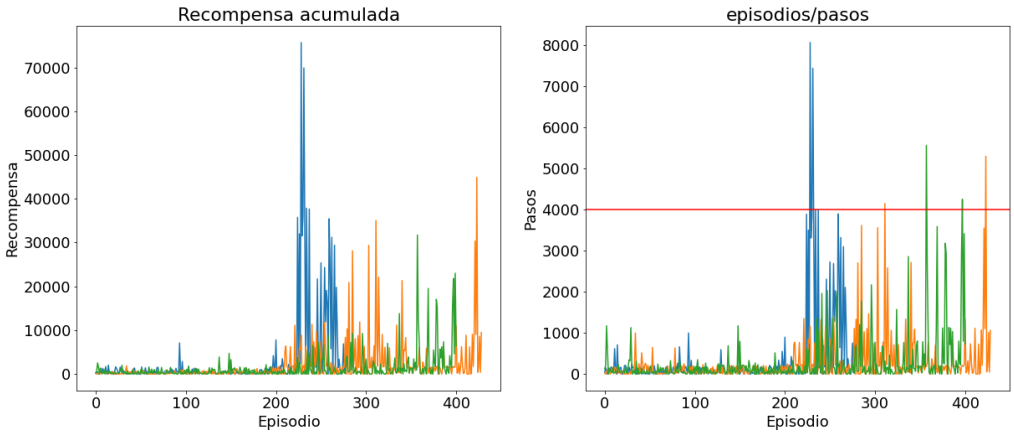


Figura A.18: Gráfica de resultados del entrenamiento en el Nürburgring con un conjunto de acciones simple y 2 puntos de nivel de percepción.

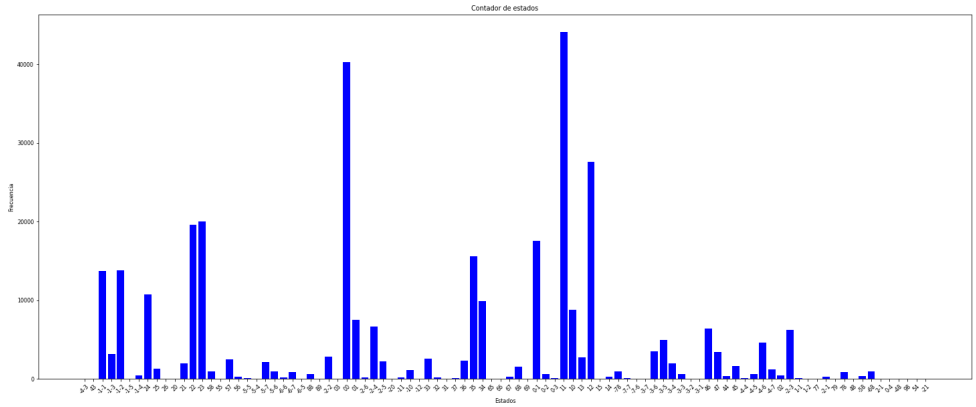


Figura A.19: Gráfica de resultados de la distribución de los estados.

A.1.13. Circuito de Nürburgring, conjunto de acciones difícil y dos puntos de percepción

Tabla de entrenamiento en Nürburgring			
Entrenamiento	**1**	**2**	**3**
Vuelta completada	Sí	Sí	Sí
Tiempo hasta completar	44:21 min	50:59 min	1:38 min
Épocas hasta completar	301	291	495
Valor de ϵ final	0.52	0.43	0.45
Tamaño de la Tabla-Q	239	248	262
Nº total de épocas	415	574	528

Tabla A.13: Resultados del entrenamiento en el Nürburgring con un conjunto de acciones difícil y 2 puntos de nivel de percepción.

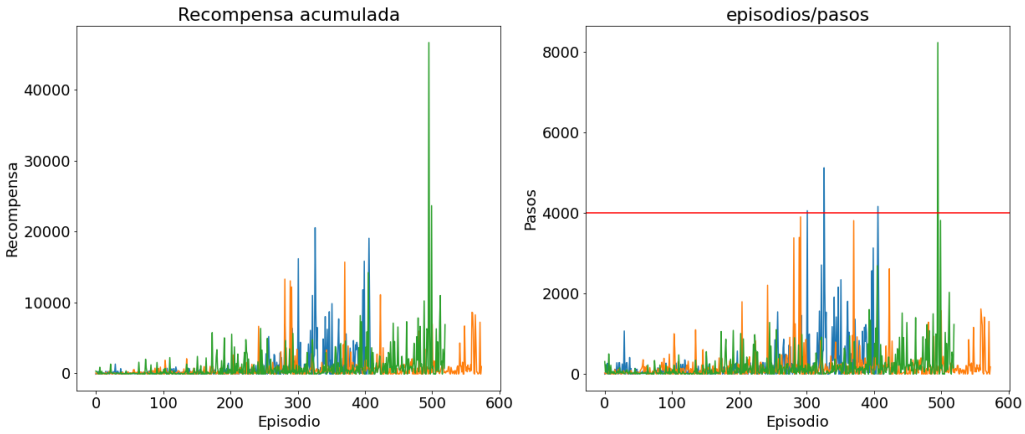


Figura A.20: Gráfica de resultados del entrenamiento en el Nürburgring con un conjunto de acciones difícil y 2 puntos de nivel de percepción.

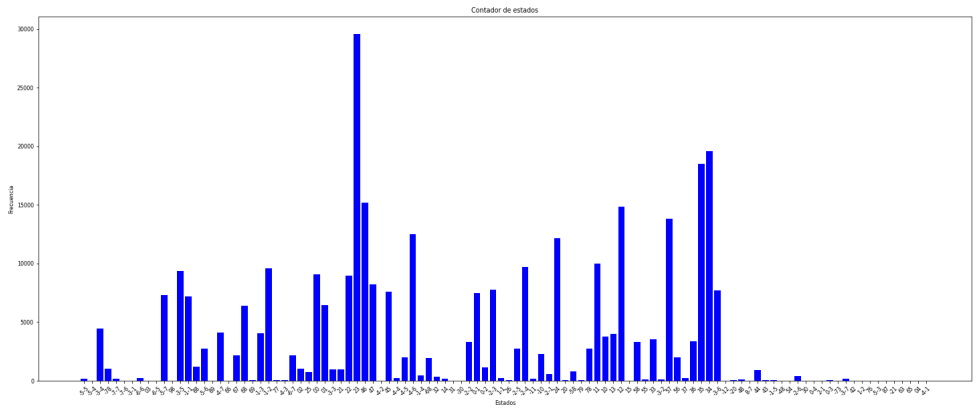


Figura A.21: Gráfica de resultados de la distribución de los estados.

A.1.14. Circuito de Nürburgring, conjunto de acciones simple y tres puntos de percepción

El estado más frecuente es el $(0, -1, -4)$ con un valor de 16114 veces.

Tabla de entrenamiento en el Nürburgring			
Entrenamiento	**1**	**2**	**3**
Vuelta completada	No	No	No
Tiempo hasta completar	∞	∞	∞
Épocas hasta completar	-	-	-
Valor de ϵ final	0.28	0.25	0.29
Tamaño de la Tabla-Q	1199	1194	1191
Nº total de épocas	911	892	771

Tabla A.14: Resultados del entrenamiento en el Nürburgring con un conjunto de acciones simple y 3 puntos de nivel de percepción.

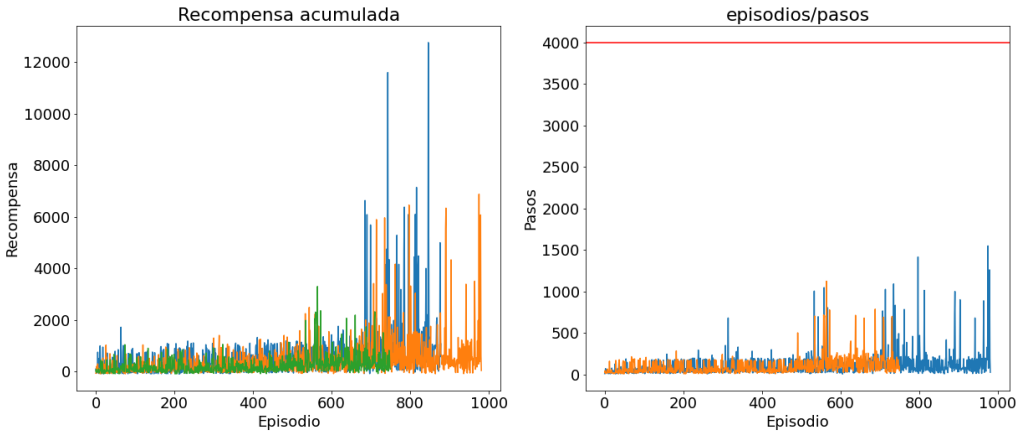


Figura A.22: Gráfica de resultados del entrenamiento en el Nürburgring con un conjunto de acciones fácil y 3 puntos de nivel de percepción.

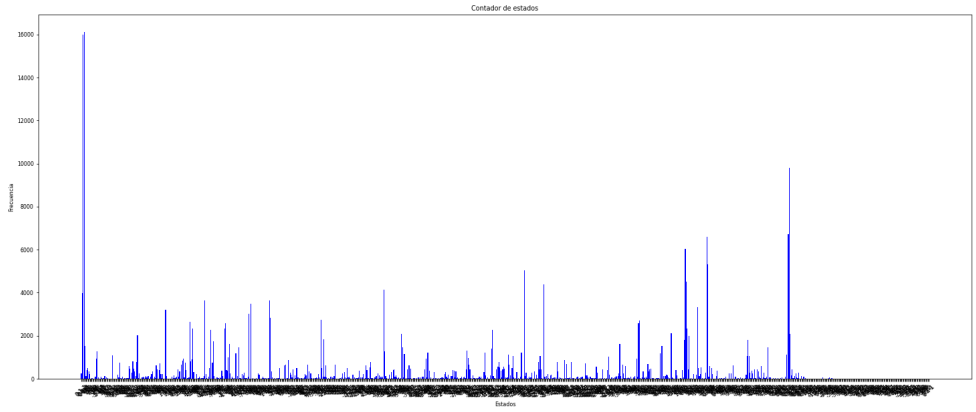


Figura A.23: Gráfica de resultados de la distribución de los estados.

A.1.15. Circuito de Nürburgring, conjunto de acciones medio y tres puntos de percepción

El estado más frecuente es el $(1, 3, 6)$ con un valor de 9161 veces.

Tabla de entrenamiento en el Nürburgring			
Entrenamiento	**1**	**2**	**3**
Vuelta completada	No	No	No
Tiempo hasta completar	∞	∞	∞
Épocas hasta completar	-	-	-
Valor de ϵ final	0.24	0.25	0.34
Tamaño de la Tabla-Q	1003	976	755
Nº total de épocas	303	1824	1064

Tabla A.15: Resultados del entrenamiento en el Nürburgring con un conjunto de acciones medio y 3 puntos de nivel de percepción.

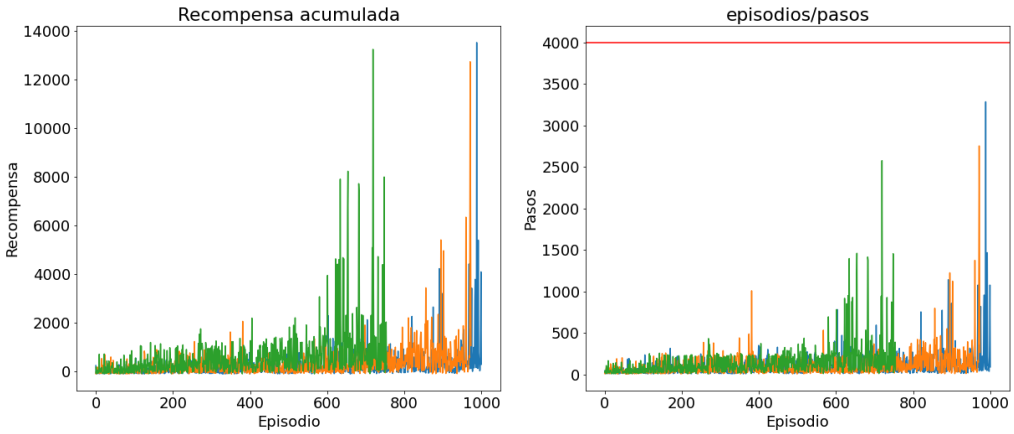


Figura A.24: Gráfica de resultados del entrenamiento en el Nürburgring con un conjunto de acciones medio y 3 puntos de nivel de percepción.

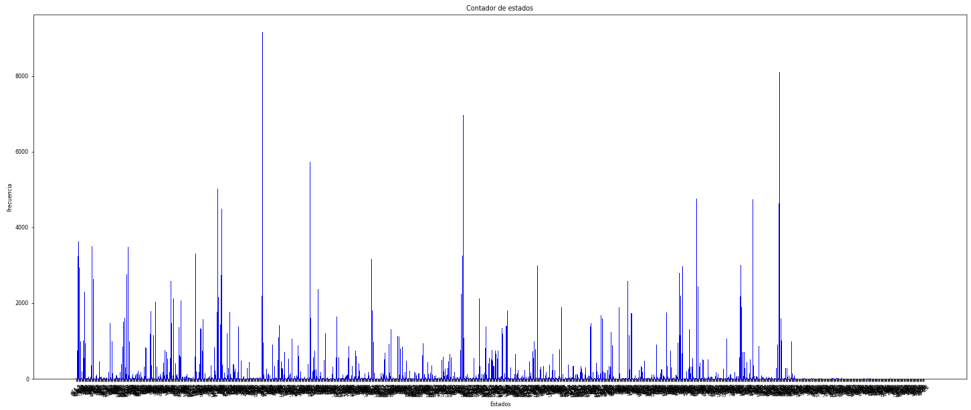


Figura A.25: Gráfica de resultados de la distribución de los estados.

A.1.16. Circuito de Nürburgring, conjunto de acciones difícil y tres puntos de percepción

El estado más frecuente es el $(2, 4, 7)$ con un valor de 8707 veces.

Tabla de entrenamiento en el Nürburgring			
Entrenamiento	**1**	**2**	**3**
Vuelta completada	No	No	No
Tiempo hasta completar	∞	∞	∞
Épocas hasta completar	-	-	-
Valor de ϵ final	0.26	0.29	0.31
Tamaño de la Tabla-Q	1577	1506	1466
Nº total de épocas	958	863	816

Tabla A.16: Resultados del entrenamiento en el Nürburgring con un conjunto de acciones difícil y 3 puntos de nivel de percepción.

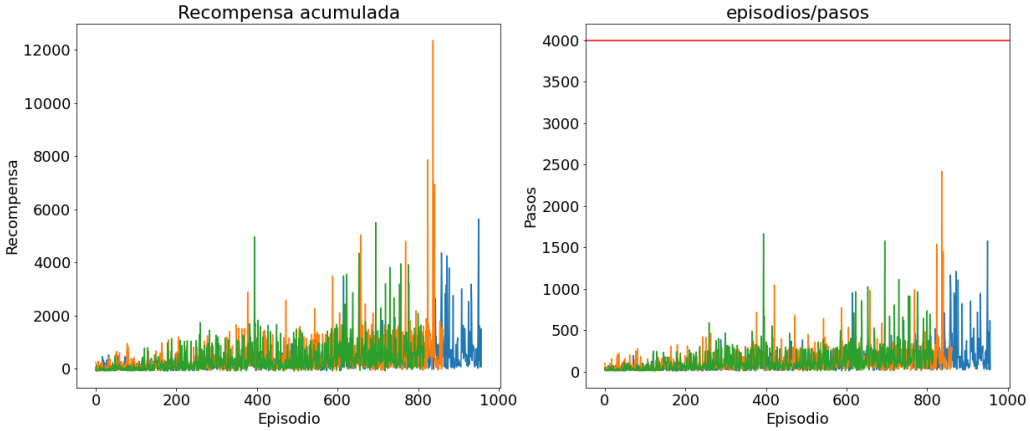


Figura A.26: Gráfica de resultados del entrenamiento en el Nürburgring con un conjunto de acciones difícil y 3 puntos de nivel de percepción.

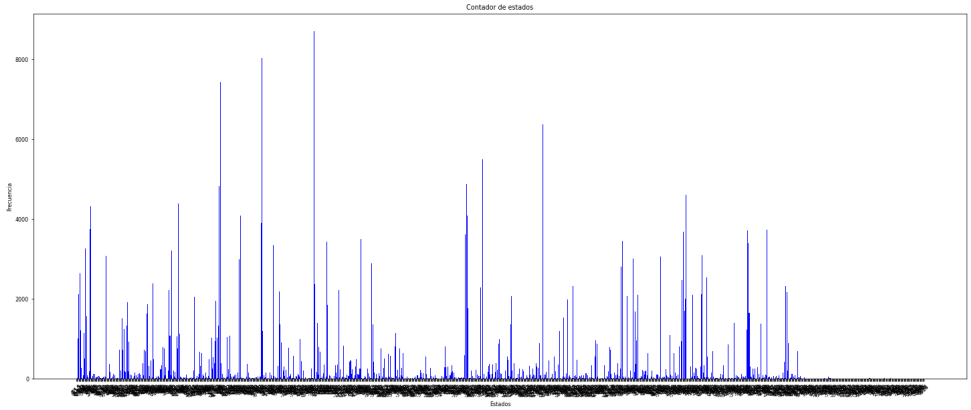


Figura A.27: Gráfica de resultados de la distribución de los estados.