Universidad de Alcalá
Escuela Politécnica Superior

GRADO EN INGENIERÍA ELECTRÓNICA

Y AUTOMÁTICA INDUSTRIAL

Trabajo Fin de Grado

APLICACIÓN DE TÉCNICAS DE DEEP REINFORCEMENT LEARNING MEDIANTE AGENTE CNN-DDPG A LA CONDUCCIÓN AUTÓNOMA

Autor: Víctor Arroyo de la Torre

Tutor/es: Rafael Barea Navarro

2021

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

Grado en Ingeniería Electrónica y Automática Industrial

Trabajo Fin de Grado

"APLICACIÓN DE TÉCNICAS DE DEEP REINFORCEMENT LEARNING MEDIANTE AGENTE CNN-DDPG A LA CONDUCCIÓN AUTÓNOMA"

Autor: Víctor Arroyo de la Torre

Tutor/es: Rafael Barea Navarro

TRIBUNAL:

Presidente: Luis Miguel Bergasa Pascual

Vocal 1º: Luciano Boquete Vázquez

Vocal 2º: Rafael Barea Navarro

FECHA: 19/09/2021

AGRADECIMIENTOS

En primer lugar, me gustaría dedicar este trabajo final del grado a mi familia, por ayudarme tanto económicamente como emocionalmente en estos cuatro años abandonando mi ciudad natal y estudiar en una ciudad desconocida.

A mi grupo de amigos de la residencia en la que he vivido estos 4 años, dar las gracias por disfrutar la mejor etapa de mi vida y por el apoyo ofrecido en todo momento.

Al grupo de investigación Robesafe por acogerme con los brazos abiertos y hacerme estos meses más llevaderos ante un proyecto el cual empecé conociendo muy poco.

Por último y no menos importante a Rafa, haciéndome aprender y disfrutar con este trabajo.

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABLAS

PALABRAS CLAVE

DRL: Deep Reinforcement Learning

DDPG: Deep Deterministic Policy Gradient

CARLA: Car Learning to Act

ROS: Robotic Operating System

LSTM: Long short-term memory.

RESUMEN

El trabajo tiene como objetivo el desarrollo de un modelo de conducción autónoma con el algoritmo DDPG-LSTM, el cual pertenece a la familia Deep Reinforcement Learning (DRL). El algoritmo parte del Deep Deterministic Policy Gradient (DDPG) al cual se le ha implementado una red neuronal de mayor memoria llamada LSTM (Long Short-Term Memory).

Con este modelo se contribuirá al desarrollo de algoritmos de navegación autónoma y en trabajos futuros se ajustará para poder implementarlo sobre el vehículo eléctrico autónomo en desarrollo del grupo Robesafe para el proyecto Techs4AgeCar.

ABSTRACT

The aim of this work is to develop an autonomous driving model with the DDPG-LSTM algorithm, which belongs to the Deep Reinforcement Learning (DRL) family. The algorithm is based on the Deep Deterministic Policy Gradient (DDPG) to which a larger memory neural network called LSTM (Long Short-Term Memory) has been implemented.

This model will contribute to the development of autonomous navigation algorithms and in future work will be adjusted to be implemented on the autonomous electric vehicle being developed by the Robesafe group for the Techs4AgeCar project.

RESUMEN EXTENDIDO

El ser humano requiere indispensablemente de la movilidad, por lo que la navegación autónoma ha supuesto un gran avance en los últimos años. Hasta ahora su control se llevaba a cabo a través del uso de diferentes dispositivos y sensores como pueden ser láser, GPS, cámaras, etc. Mediante estos sensores era posible reconocer el entorno de manera que el vehículo era capaz de circular desde un punto a otro, pero este tipo de arquitectura resultaba muy compleja debido a que se dependía de demasiada electrónica en intervalos de tiempo muy pequeños.

Para solucionar este problema se pretende usar inteligencia artificial, en concreto un campo de la tecnología que tiene mucho éxito llamado Machine Learning y Deep Learning. Actualmente, las técnicas de inteligencia artificial son utilizadas para casi todos los campos de la ciencia con numerosas finalidades, desde el campo de la medicina, a la previsión económica, conducción autónoma, etc.

Este proyecto está desarrollado dentro del grupo de investigación Robesafe de la Universidad de Alcalá para el proyecto Tech4AgeCar. Se encuentra en una etapa avanzada en la cual se ha implementado mucha inteligencia artificial para alcanzar el propósito final de conducción autónoma de vehículos.

Dentro de este proyecto se utilizan numerosos sensores como cámaras, Lídar y DGPS con los que se procesa información en la capa de percepción, y con ayuda de estos se pueden desarrollar algoritmos en el campo del Deep Reinforcement Learning (DRL) para el vehículo real, empleando redes neuronales artificiales.

Las redes neuronales se usan en detección y seguimiento de objetos, reconocimiento de señales de tráfico, segmentación de imágenes semánticas y diversos temas en los sistemas de vehículos autónomos.

En este caso se va a analizar un algoritmo Deep Reinforcement Learning con una red neuronal LSTM (Long Short-Term Memory). Esta red son una extensión de redes neuronales recurrentes con el objetivo de ampliar la memoria y tener experiencias de un mayor periodo de tiempo.

El DDPG-LSTM es un algoritmo que se establece tomando como base la memoria histórica de conducción y el estado de movimiento instantáneo de los vehículos, además de que se van obteniendo recompensas en cada una de las interacciones. Los resultados muestran que el algoritmo permite reproducir una conducción autónoma personalizada con una velocidad predeterminada. Este se comparará con la versión anterior del DDPG tanto en rendimientos en la etapa de entrenamiento como en la evaluación de modelos entrenados.

El algoritmo va a predecir una acción (aceleración y dirección) que se introducirá en el simulador y calculará una recompensa para retroalimentar la red neuronal. Con esto se construirá un entorno funcional el cual se reproducirá dentro del simulador hiperrealista CARLA (simulador que utiliza el grupo de investigación Robesafe), el cuál más adelante se explicará con detalle.

Esto permite una solución más fácil y genérica al problema de la navegación, ya que, a pesar del buen funcionamiento de los sistemas convencionales, es necesario un gran conocimiento y ajuste para lograr un correcto funcionamiento.

1. INTRODUCCIÓN

1.1. MOTIVACIÓN

Actualmente la conducción autónoma es una realidad. Un vehículo autónomo es un automóvil que obtiene información de su entorno y realiza acciones de manera segura sin intervención humana.

Para esta percepción, como ya se ha mencionado, es necesario numerosos sensores, como GPS, LiDar, cámaras, etc. Sincronizar todos estos dispositivos no es una tarea fácil, pero se ha conseguido con el objetivo de poder tomar siempre la mejor decisión posible para moverse por el entorno de forma segura.

El objetivo principal de la conducción autónoma es la navegación del vehículo partiendo de un punto inicial hasta un punto final sin ningún accidente. Actualmente, la empresa estadounidense llamada Tesla Motor, fundada en 2003, dispone de una tecnología de conducción semiautomática denominada Tesla Autopilot [1], que está cambiando la forma en que se navega y se desplaza, incorporando inteligencia artificial de última generación y mejorando la próxima generación de vehículos autónomos con actualizaciones en tiempo real.

Aparte de esta empresa está Waymo, la cual ha lanzado recientemente un conjunto de datos abiertos para desafíos sobre este tema, y de aquí surge la idea de aprender a través de Reinforcement Learning una memoria de largo a corto plazo para un modelo que imite la tecnología de esta empresa. En el paper [2] se plantea un modelo con una red LSTM, los resultados demuestran que esta red tiene un mejor desempeño que los modelos de referencia sin necesidad de utilizar modelos CNN o aquellos que dependan del tiempo en los datos.

En la figura 1 [3] se puede ver una imagen de uno de los vehículos autónomos de la empresa Waymo:

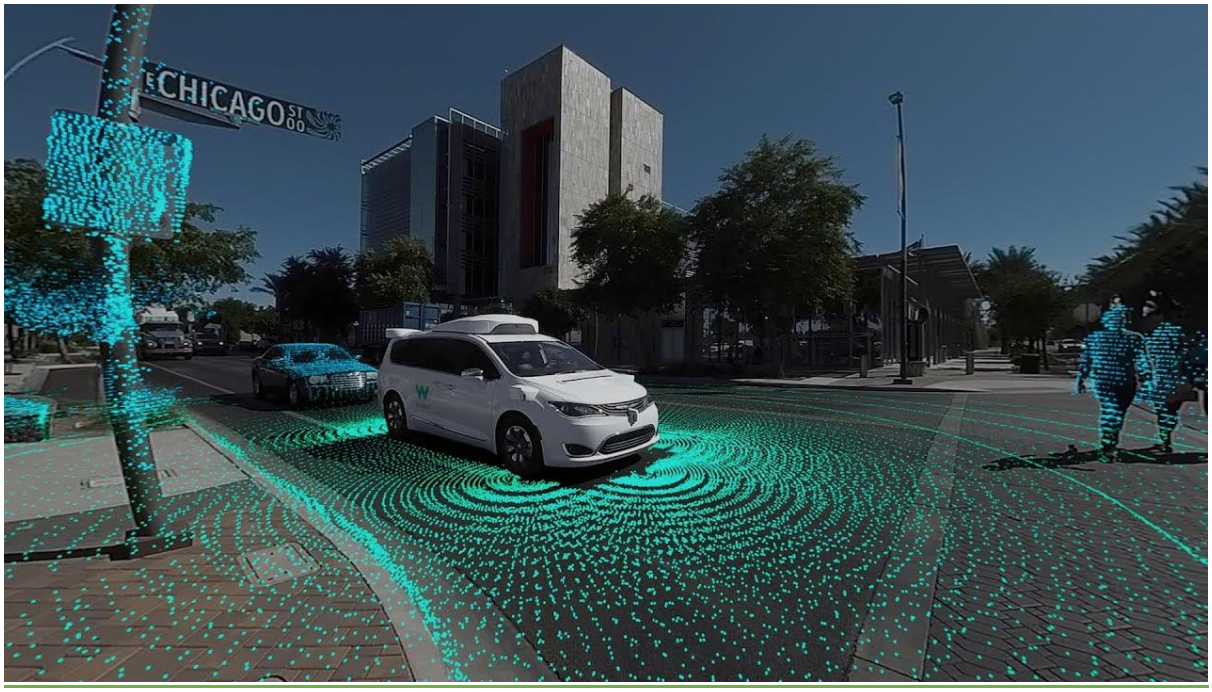

Figura 1. Vehículo autónomo Waymo [3]

Para este proyecto se abrió la oportunidad de participar dentro del grupo de investigación Robesafe y hacerlo compatible para el proyecto “Tech4AgeCar” de la Universidad de Alcalá. En este proyecto se intenta conseguir un vehículo autónomo con un presupuesto mucho menor que una empresa dedicada exclusivamente a ello.

1.2. OBJETIVOS Y CAMPO DE APLICACIÓN

Los objetivos de este trabajo son:

- Implementar una aplicación de Aprendizaje por Refuerzo Profundo (Deep Reinforcement Learning) para conducción autónoma mediante el algoritmo Deep Deterministic Policy Gradient, con una red neuronal Long Short-Term Gradient (DDPG-LSTM). Se trata de una red neuronal recurrente, de modo que tanto la conducción autónoma como la instantánea son la base para la toma de decisiones del comportamiento del vehículo.

- Conseguir un sistema de navegación estable a partir de esa aplicación. Para comprobar la efectividad se compararán los algoritmos DDPG-LSTM y DDPG tradicional en modo entrenamiento y pruebas en el simulador hiperrealista CARLA dentro de un circuito determinado.

- El objetivo principal de la conducción autónoma es conseguir eliminar las víctimas de accidentes de tráfico, ya que si todos los coches funcionasen de manera autónoma se podrían reducir estas cifras notablemente.

1.3. ESTRUCTURA DEL TRABAJO

La estructura del trabajo será:

1. Introducción.
2. Estado del arte.
3. Simulador CARLA.
4. Arquitectura general.
5. Simulación.
6. Resultados experimentales.
7. Conclusiones generales.
8. Pliego de condiciones.
9. Planos.

2. ESTADO DEL ARTE

2.1. INTRODUCCIÓN

En este apartado se comentarán los inicios de la conducción autónoma, una introducción a las redes neuronales y se explicarán diferentes técnicas de aprendizaje por refuerzo que se han ido desarrollando a lo largo del tiempo.

2.2. CONDUCCIÓN AUTÓNOMA

Como se ha comentado anteriormente, los inicios del vehículo autónomo se deben a Tesla Motor. En 2018 surge el problema de determinar qué nivel de autonomía se le da a un vehículo, por ello la Sociedad de Ingenieros Automotrices (SAE) definió el baremo de acuerdo con el estándar J3016 [4] y estableció 6 niveles. Los tres primeros el conductor es el encargado de controlar y supervisar, y el resto es el vehículo el que realiza la conducción, pero puede retornar el control al humano en cualquier momento.

Actualmente los vehículos autónomos comerciales han llegado al nivel 3, en el que el conductor puede ser requerido cuando este lo necesite. Se está desarrollando para alcanzar el nivel 4, pero se necesita una percepción muy buena del entorno para poder asegurarle una correcta posición inicial y final, así como la capacidad de realizar el trayecto de manera óptima.

Con el desarrollo tanto de la inteligencia artificial como de Big Data, los métodos de aprendizaje supervisado se han mejorado por la capacidad de datos dimensionales [5]. Además, estos métodos son más flexibles que los modelos clásicos ya que permiten la incorporación de parámetros adicionales que influyen en el comportamiento del vehículo.

2.3. REDES NEURONALES

Una red neuronal [6] es una red de entidades interconectadas (nodos), en la que cada nodo es responsable de un cálculo simple, el cual consiste en un modelo computacional que permite resolver tareas de regresión o clasificación. Esta herramienta es ampliamente utilizada en el área del aprendizaje automático y se basa en el funcionamiento de las redes biológicas.

Estas redes están formadas por neuronas, cada neurona es un pequeño modelo matemático el cual recibe una o varias entradas y proporciona un resultado. La entrada está formada por la suma de diferentes valores ponderados junto con un sesgo. El resultado es la siguiente ecuación [7]:

$$y = \sum_i (x_i \, x \, w_i) + b$$

donde:

$$y: salida\ de\ la\ neurona$$

$$x_i = salida\ de\ neurona\ i\ de\ la\ capa\ anterior$$

$$w_i = pesos\ correspondiente\ a\ la\ neurona\ i$$

$$b = sesgo\ de\ la\ neurona$$

Las neuronas están organizadas en capas dentro de las redes neuronales donde la entrada es la primera capa y la salida es la última, y cada neurona recibe las salidas de la capa anterior como entrada. Los pesos de cada neurona se pueden entrenar para que la salida se aproxime a la deseada, y se calcula el error entre la salida y la deseada, y se propaga desde la última capa a la primera.

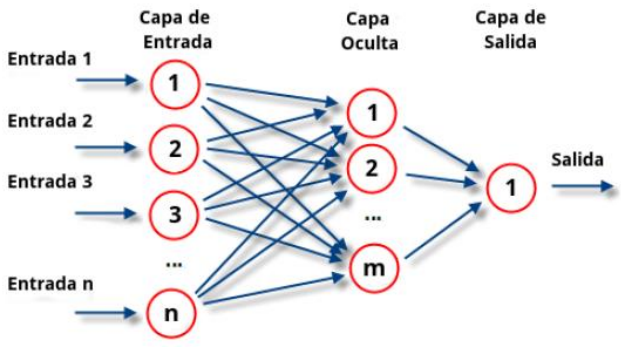

Figura 2. Esquema general de una red neuronal

2.4. REINFORCEMENT LEARNING

El Reinforcement Learning [8] es un área de aprendizaje automático cuya misión consiste en determinar qué acciones debe escoger el agente en cada para obtener una recompensa máxima.

Este optimiza los problemas en la toma de decisiones secuenciales a través del aprendizaje interactivo continuo entre el agente y el entorno. Su composición básica está representada por una tupla de 5 parámetros:

$$M = (S, A, r, P, \gamma)$$

$$donde:$$

$$S: es\ el\ conjunto\ finito\ de\ estados$$

$$A: es\ el\ conjunto\ finito\ de\ acciones$$

$$r: es\ la\ recompensa$$

$$P: es\ la\ probabilidad\ de\ transición\ del\ estado$$

$$\gamma\ \in (0,1]\ es\ un\ factor\ de\ reducción$$

Para cada estado s_i en un tiempo i, la acción a_i se determina según un plan de acción $\pi\,(s_i)$. Después, el agente pasa al siguiente estado s_{i+1} con una probabilidad $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$ y se obtiene una recompensa r_i, de manera que (s_i, a_i, r_i, s_{i+1}) son el conjunto de datos que se generan continuamente y los cuales se utilizan para optimizar el plan de acción π con fin de maximizar el rendimiento acumulado esperado.

Tras varias iteraciones, el agente aprenderá la estrategia de acción necesaria para completar la tarea correctamente. El proceso de iteraciones de Reinforcement Learning entre agente y entorno se puede visualizar en la figura 3 y la recompensa esperada sigue la siguiente expresión:

$$Q^{\pi}(s_t, a_t) = E_{\pi}[R_t | s_t] = s, a_t = a$$

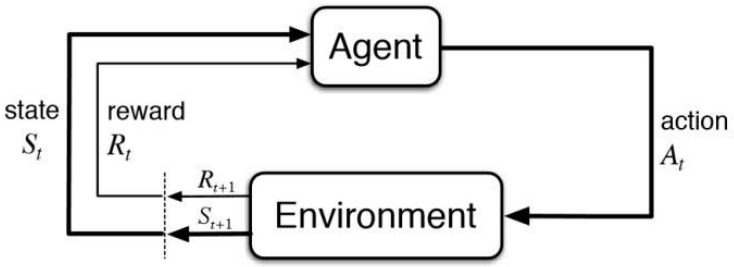

Figura 3. Arquitectura Deep Reinforcement Learning

Las tareas en las que el Reinforcement Learning quiere ayudar en la conducción autónoma son las siguientes:

- Optimización de la trayectoria.
- Planificación del movimiento.
- Optimización del controlador.
- Políticas de aprendizaje basadas en diferentes escenarios en las carreteras.

A continuación, se mostrarán diferentes ejemplos reales en los que se emplea el Reinforcement Learning:

Un ejemplo donde se utiliza es el robot AWS DeepRacer [10] el cual es un vehículo autónomo de carreras que se ha construido para probarlo dentro de un circuito. Está formado por diferentes cámaras que visualizan la pista y un modelo que controla el acelerador y la dirección. Su red neuronal está formada por 4 capas convolucionales y 3 capas completamente conectadas entre sí. En la figura 4 se puede ver una imagen del robot [11]:

Figura 4. AWS DeepRacer [11]

Otro ejemplo del uso de RL es en el ámbito de los juegos de estrategia, pues AlphaGo Zero [9] es un algoritmo el cual a través de RL pudo aprender el juego Go desde cero enfrentándose a sí mismo. El entrenamiento duró 40 años y se puso a prueba contra el número uno de este juego, Ke Jie, al cual derrotó. Únicamente se utilizaron piezas blancas y negras como entradas en una única red neuronal. En la figura 5 se puede observar visualmente el esquema del algoritmo:

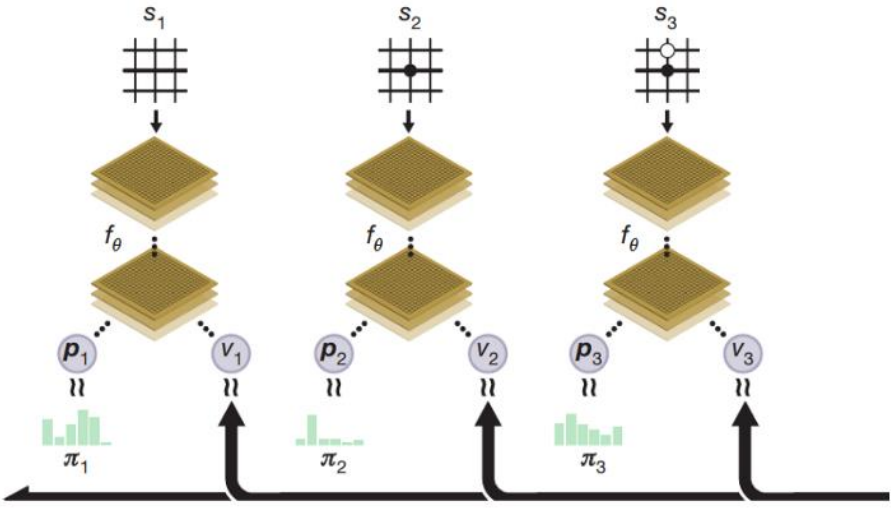

Figura 5. Algoritmo AlphaGo Zero

2.5. RL-GAN-LSTM

Una vez estudiado el tema de las redes neuronales y el Reinforcement Learning surge la idea de investigar dentro de este ámbito y encontrar un algoritmo con mayor memoria del que se tenía. Por ello el uso de las redes neuronales LSTM.

Una de las utilidades que se están probando son las redes LSTM-GAN [12] dentro de la seguridad de vehículos autónomos. El objetivo es proporcionar seguridad mantenido una distancia segura y óptima para defenderse de posibles accidentes de tráfico. La reacción del vehículo autónomo es la que se estudia con herramientas de DRL. En este ejemplo se emplea una Long Short-Term Memory (LSTM) Generative Adversial Network (GAN) lo que forman una técnica conocida como LSTM-GAN para crear un modelo.

En este algoritmo se experimentó un resultado brillante, ofreciendo una distancia eficaz en cualquier ataque de tráfico que interrumpiese el tránsito de vehículos evitando que chocasen. El modelo LSTM-GAN se utiliza para obtener características temporales e independientes entre las acciones del adversario y las reacciones de los vehículos autónomos que provocan una desviación en la distancia óptima. En la figura 6 se muestra el pseudocódigo de este algoritmo [12]:

Proposed Algorithm

Step1: Initializing the memory (MM^i) slots that will keep previous transactions of the players i.e. $\mathcal{Q}^i$

Step2: Check for the initial state for the players i.e. $s^i(0)$

Step3: Repeat:

1. Choose an action vector ω^i for each player i
2. Next selection of random action which have occurrence probability of ϵ.
3. Next Try to Maximize ω^i such that $\omega^i(A) = \max_{\omega^i}[\mathcal{Q}^i(s^i(A), \omega^i(A))]$
4. Both AVh and adversary will perform their actions ω^i at the same time.
5. Find the utility value $\mathbb{U}^i(A+1)$ and establish newer state $s^i(A+1)$
6. Saving the interactions $\{s^i(A), \omega^i(A), \mathbb{U}^i(A+1), s^i(A+1)\}$ in the replay-able memory RM^i for each entity i
7. Taking a sample randomly from the memory interactions RM^i for each entity i, where ρ shows some random point. $\{\hat{s}^i(\rho), \widehat{\mathbb{U}}^i(\rho+1), \widehat{\omega}^i(\rho), \hat{s}^i(\rho+1)\}$
8. Next find the Estimated value (T_v^i) for every player entity i
9. Now if the tested interaction is for $A=0$ then $T_v^i = \widehat{\mathbb{U}}^i(\rho+1)$
10. Else $T_v^i = \widehat{\mathbb{U}}^i(\rho+1) + \max_{\omega^i} \mathcal{Q}^i(\hat{s}^i(\rho+1), \widehat{\omega}^i(\rho))$
11. Training the model $\mathcal{Q}^i$ with every involved player as: $[T_v^i - \mathcal{Q}^i(\hat{s}^i(A), \widehat{\omega}^i(A))]^2$
12. A=A+1;
13. Till it converges to the Mixed-Strategy-Nash-Equilibrium

Step4: End

Figura 6. Algoritmo LSTM-GAN

2.6. ALGORITMOS RELACIONADOS

2.6.1. ACTOR-CRITIC

El DDPG parte de algoritmos dentro del Reinforcement Learning más antiguos como los clásicos Actor-Critic [13].

El algoritmo Actor-Critic es un paradigma clásico del Reinforcement Learning que cuenta con dos agentes. El actor que es el que controla como reacciona el agente y el critic evalúa como fue la decisión tomada.

2.6.2. ALGORITMO A2C

El algoritmo A2C significa Actor Critic con ventaja, parte del Actor-Critic comentado en el apartado anterior y sigue la misma idea, lo que cambia es la función que se utiliza que en este caso se llama función de ventaja:

$$A(s,a) = Q(s,a) - V(s)$$

donde $V(s)$ es el promedio de ese estado.

Con esto se podrá ver si mejora o no, ya que si el valor $A(s,a)$ es positivo significa que $Q(s,a)$ es mejor que el promedio, por lo que se estaría mejorando.

2.6.3. ALGORITMO A3C

El algoritmo A3C sigue las siglas de Actor Critic asíncrono con ventaja, al igual que en el caso anterior el algoritmo parte del A2C, con la diferencia de que este algoritmo usa múltiples agentes y cada agente tiene sus propios parámetros de red y una copia del entorno. Estos agentes interactúan entre sí de forma asíncrona aprendiendo con cada interacción.

Este algoritmo imita el entorno de la vida real entre humanos, ya que cada humano adquiere conocimientos de las experiencias de otros y permite que la red global se mejore.

El pseudocódigo que sigue este algoritmo se refleja en la siguiente figura [14]:

Algorithm S3 Asynchronous advantage actor-critic - pseudocode for each actor-learner thread.

// Assume global shared parameter vectors θ *and* θ_v *and global shared counter* $T = 0$
// Assume thread-specific parameter vectors θ' *and* θ'_v
Initialize thread step counter $t \leftarrow 1$
repeat
  Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$.
  Synchronize thread-specific parameters $\theta' = \theta$ and $\theta'_v = \theta_v$
  $t_{start} = t$
  Get state s_t
  repeat
    Perform a_t according to policy $\pi(a_t|s_t;\theta')$
    Receive reward r_t and new state s_{t+1}
    $t \leftarrow t + 1$
    $T \leftarrow T + 1$
  until terminal s_t **or** $t - t_{start} == t_{max}$
  $R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t, \theta'_v) & \text{for non-terminal } s_t \text{// Bootstrap from last state} \end{cases}$
  for $i \in \{t-1, \ldots, t_{start}\}$ **do**
    $R \leftarrow r_i + \gamma R$
    Accumulate gradients wrt θ': $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i|s_i;\theta')(R - V(s_i;\theta'_v))$
    Accumulate gradients wrt θ'_v: $d\theta_v \leftarrow d\theta_v + \partial \left(R - V(s_i;\theta'_v)\right)^2 / \partial \theta'_v$
  end for
  Perform asynchronous update of θ using $d\theta$ and of θ_v using $d\theta_v$.
until $T > T_{max}$

Figura 7. Pseudocódigo algoritmo A3C

3. SIMULADOR CARLA.

3.1. INTRODUCCIÓN

CARLA [15] es un simulador autónomo hiperrealista de código abierto que se usa en el proyecto Tech4AgeCar. A través de PythonAPI se pueden gestionar numerosas tareas involucradas en el problema de la conducción autónoma dentro del simulador. El simulador [16] se basa en Unreal Engine 4, que es una herramienta de creación 3D en tiempo real usada también en la creación de videojuegos. Esta herramienta le permite a CARLA tener una increíble vista realista del simulador, haciendo que los problemas sean visibles y lo más real posible gracias a su propia física. Además, CARLA junto a PythonAPI está en continuo desarrollo, por lo que la hacen una de las mejores opciones del mercado dentro de los simuladores.

El Simulador CARLA se separa en dos vertientes. Por un lado, el servidor es el responsable de todo lo relacionado con la simulación, es decir, de la representación de sensores y cámaras, cálculos de la física, etc. Y por otra parte está el lado del usuario, que controla la lógica de los actores en la escena y los escenarios, como por ejemplo las condiciones del clima, sensores, actores pasivos, etc.

Es posible añadir módulos del usuario al servidor a través de PythonAPI y crear el entorno de simulación que se quiera. Con ello, se pueden llegar a tener infinitas funcionalidades para incorporar al proyecto.

Algunas de las características del simulador son las siguientes:

- Capacidad de añadir diferentes cámaras, sensores y la configuración ser cambiada en cada episodio. También se puede modificar la posición y orientación de la cámara.
- Obtener información sobre el entorno a través de PythonAPI ya que se puede leer la información de odometría, GPS o sensores de colisión imprescindibles para la simulación.
- CARLA cuenta con integración ROS. ROS [17] (Robotic Operative System) es una plataforma de desarrollo para la comunicación entre sensores y actuadores en robots, facilitando el intercambio de información mediante protocolos Ethernet.

El desarrollo de CARLA se actualiza constantemente en el simulador y en sus funcionalidades tanto en documentación como en funciones de PythonAPI, lo que obliga al usuario a aprender constantemente nuevas técnicas. Además, PythonAPI hace que el simulador sea sencillo, por lo que la información fluye directamente de un lado a otro.

El asentamiento de CARLA en sistemas de navegación autónoma se debe a la unión creada entre ROS y CARLA a través del llamado RosBridge, el cual permite publicar y enviar información al vehículo.

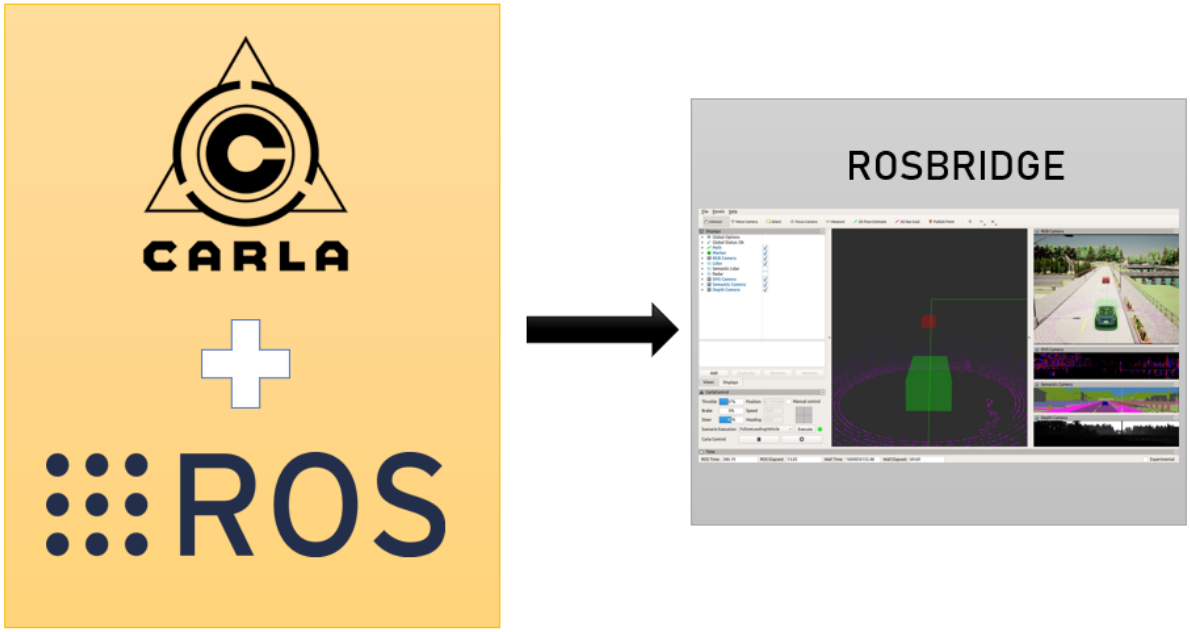

Figura 8. RosBridge

3.2. CREACIÓN DE MAPAS

Una de las características más importantes de CARLA son los mapas realistas predeterminados “Town0X”, estos mapas incluyen todo lo que se puede ver en la vida real como semáforos, señales, destellos de luz, paso de peatones, peatones, rectas, curvas, etc. En las figuras 9 y 10 se pueden observar dos mapas predeterminados:

Figura 9. Town01

Figura 10. Town03

Pero no solo eso, CARLA gracias a RoadRunner permite la creación de nuevos mapas personalizados. En la figura 11 se puede ver la secuencia de creación, exportación e importación en CARLA usando este programa.

Figura 11. Esquema general creación de mapas

En este trabajo se va a usar un mapa real llevado al simulador, el cual es una réplica al Campus Externo de la Universidad de Alcalá realizado por integrantes del proyecto Tech4AgeCar, y que servirá para llevar a cabo el entrenamiento y simulación de un tramo que consta de dos rotondas. En la figura 12 se puede ver el mapa real en vista satélite y en la figura 13 el mapa en el simulador CARLA.

Figura 12. Vista pájaro sobre el mapa real

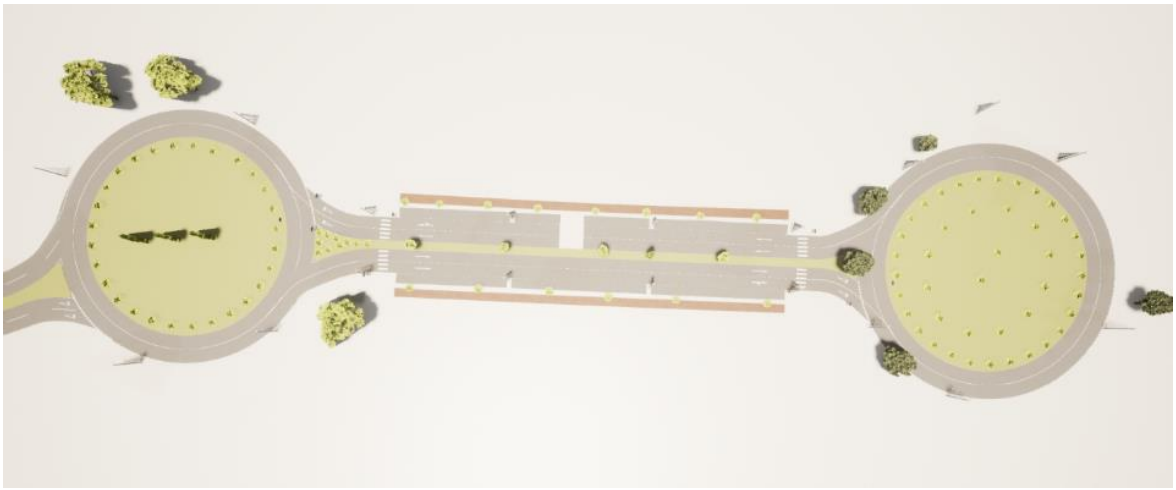

Figura 13. Vista pájaro sobre simulador CARLA

4. ARQUITECTURA GENERAL

4.1. INTRODUCCIÓN

En este apartado se va a describir la estructura de los algoritmos que se utilizarán en este trabajo. En primer lugar, el concepto de Reinforcement Learning debe explicarse dentro de Deep Reinforcement Learning (DRL), el cual es un método que utiliza conceptos de Deep Learning y Reinforcement Learning para crear algoritmos utilizados en áreas tecnológicas y científicas.

En un esquema de DRL se tiene en primer lugar un entorno, donde existe un agente que tiene que ser creado y modelado, y un observador el cual recoge datos del entorno y extrae valores de estado y recompensas que son enviados al agente. Finalmente, el agente en función de estos valores calcula una acción que utilizará para moverse al siguiente estado.

La decisión más importante en este sistema es escoger la mejor acción para obtener una buena recompensa, la cual, en este caso, consiste en que el coche realice un recorrido desde un punto inicial hasta un punto final sin salirse del carril y evitando choques con las aceras, farolas, etc. Si consigue realizar todo el recorrido sin interrupciones, se obtendrá la máxima recompensa posible.

Para este análisis, el algoritmo general empleado es el Deep Deterministic Policy Gradient (DDPG) y se va a comprobar el funcionamiento de dos versiones de este algoritmo, DDPG (clásico) y DDPG-LSTM.

4.2. DDPG

El Deep Deterministic Policy Gradient (DDPG) es un algoritmo formado por 2 módulos denominados Actor y Critic. El módulo que aprende y tomas las decisiones se conoce como agente, mientras que todo lo que interactúa con él es el entono [18].

El agente continuamente escoge acciones de un espacio de acción $A = \mathbb{R}^N$ (un estado) y una recompensa $r(s_t, a_t)$ que es la devolución del entorno. El comportamiento del agente se rige por una norma de distribución probabilística de acción $\pi: S \rightarrow P(A)$ en un entorno estocástico E.

El retorno del estado se define como la suma de todas las recompensas futuras descontadas:

$$R_t = \sum_{i=t}^{T} \gamma^{i-t} * r(s_i, a_i)$$

Donde $\gamma \in [0,1]$ es un factor de descuento. Definiendo el valor de acción como el valor esperado cuando una acción a_t se toma en un estado s_t siguiendo la norma π:

$$Q^{\pi}(s_t, a_t) = E_{r_{i \geq t}, s \geq t \sim E, a_i \sim t \sim \pi}[R_t | s_t, a_t]$$

La ecuación de Bellman se utiliza con una norma determinista μ:

$$Q^{\mu}(s_t, a_t) = E_{r_t, s_{t+1} \sim E}[r(s_t, a_t) + \gamma Q^{\mu}(s_{t+1}, \mu(s_{t+1}))]$$

La función actualizada Q, redondeada a través de una función θ^Q de parámetros que minimizan la pérdida $L(\theta^Q)$:

$$L(\theta^Q) = E_{s_t \sim \rho^{\beta}, a_t \sim \beta, r_t \sim E}[(Q(s_t, a_t | \theta^Q) - y_t)^2]$$

Beta β es cualquier norma estocástica y la distribución reducida para una π, y y_t se define como:

$$y_i = r_i(s_i, a_i) + \gamma Q(s_{t+1}, \mu(s_{t+1} | \theta^Q))$$

A través de estas ecuaciones, se encuentra la función $Q(s,a)$ del critic. Las actualizaciones del actor se basan en seguir al gradiente del retorno esperado de la distribución J, con respecto a los parámetros del aproximador de funciones del actor El gradiente es:

$$\nabla_{\theta^\mu} J \approx E_{s_t \sim \rho^\beta}\left[\nabla_{\theta^\mu} Q(s,a|\theta^Q)|_{s=s_t, a=\mu(s_t|\theta^\mu)}\right]$$
$$= E_{s_t \sim \rho^\beta}\left[\nabla_a Q(s,a|\theta^Q)|_{s=s_t, a=\mu(s_t)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|s = s_t\right]$$

Un problema en el uso de redes neuronales en Reinforcement Learning es la suposición de muchos optimizadores de muestras independientes que siguen una distribución similar, caso que no es cierto en un proceso con interacción ambiental, donde los siguientes estados son consecuencia directa, agregando la repetición de experiencia que también se implementa en DDPG.

En la siguiente figura se muestra un diagrama de bloques donde se explica la estructura del DDPG a utilizar:

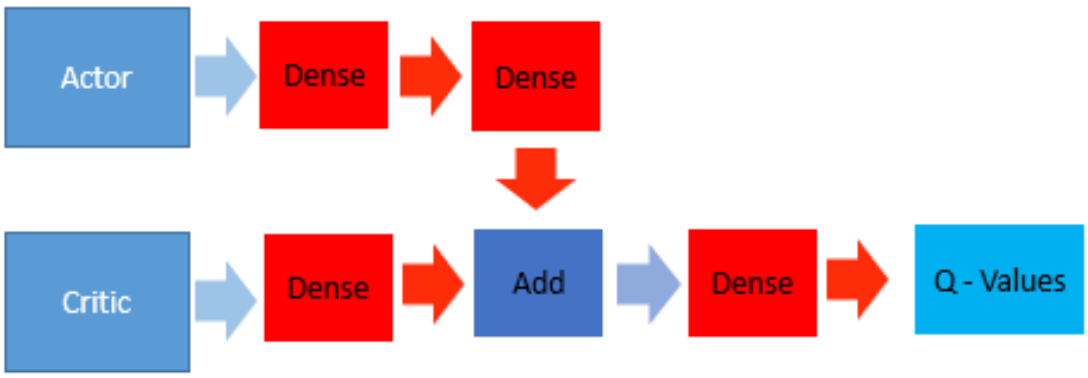

Figura 14. Estructura DDPG

Analizando el diagrama se ve que está separado la parte del agente en actor y critic, pero se concatenan pasando por unos Dense (capa en la cual se conectan todas las salidas de la capa anterior) para dar lugar a los valores de entrada que se introducirán al entorno.

En la figura 15 [18] se puede ver un esquema visual del DDPG, donde se diferencia la parte del agente con su actor y critic, y el entorno. A través del agente se mandan acciones (aceleración y dirección) a CARLA mediante PythonAPI y tras procesarse la información se devuelve una recompensa.

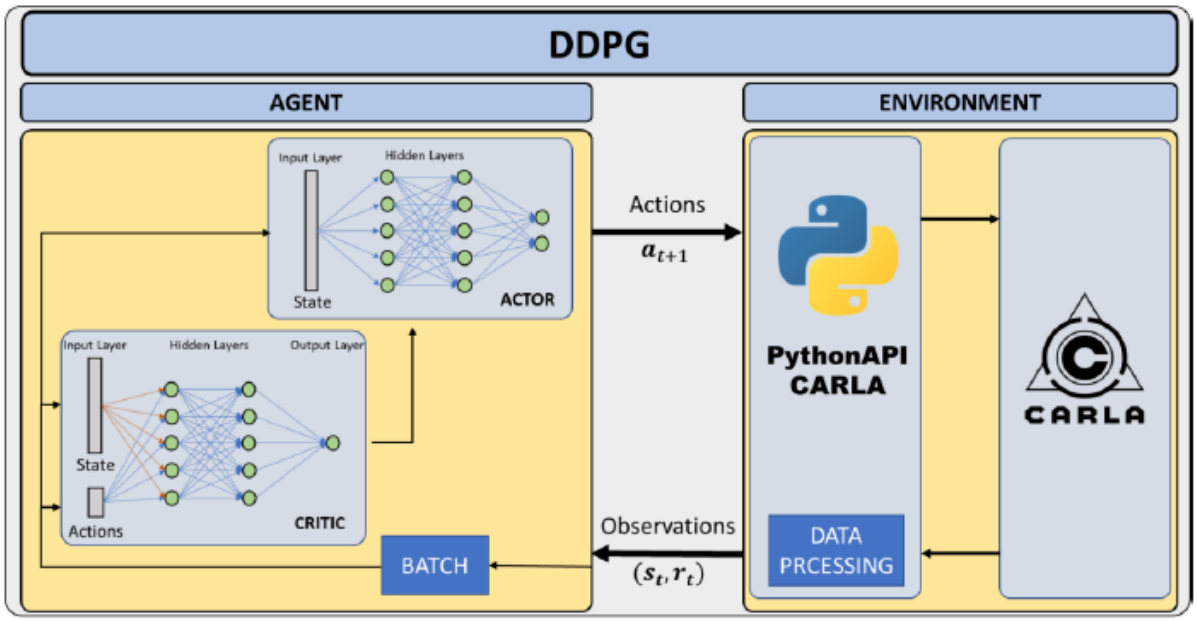

Figura 15. Esquema general DDPG [18]

Una explicación del algoritmo DDPG [19] desde un punto más conceptual se resume en estos pasos:

- Por un lado, se inicializan ambas redes neuronales. Estas redes son el actor μ y el critic Q. Por otro lado, las redes objetivo μ', Q' y el buffer también se inicializa.
- Se inicia un proceso iterativo sobre m episodios.
- Para cada episodio se obtiene el estado inicial y numerosos comandos para cada N step.
- Una acción a_t es calculada por el actor con el estado inicial s_t como entrada. Esto se aplica a la acción y devuelve un nuevo estado s_{t+1} junto a una recompensa asociada $r(a_t, s_t)$.
- Una iteración está formada por una tupla (s_t, a_t, r_t, s_{t+1}) que se almacena en el buffer R.
- El proceso de aprendizaje se inicializa muestreando un minibatch aleatorio con N iteraciones por encima del buffer R.
- El critic objetivo Q' devuelve su predicción para el siguiente estado s_{t+1} usando la salida del actor objetivo con s_{t+1} como entrada, es decir, la acción que el

actor objetivo habría tomado en el estado s_{t+1}, se almacena y la tupla de acción se evalúa por el critic objetivo. Este valor de predicción debe ser en comparación con el valor del estado real s_t válido en el minibatch, pero al ser un solo step de diferencia el valor Q futuro se reduce y su recompensa r_t que recibe el agente se aplica a la acción de ese estado s_t.

- Cuando y_i se calcula, el critic se actualiza minimizando el MSE entre los valores pronosticados por el critic para (s_t, a_t) y y_i.
- La siguiente tarea es la actualización del actor. Los pesos de la red neuronal se mueven siguiendo el gradiente ∇_{θ^μ}, que se calcula con el promedio del producto del gradiente del critic con respecto a las acciones y el gradiente del actor con respecto a sus pesos.
- Finalmente, las dos redes objetivo se actualizan aproximándolas a las ponderaciones del critic y el actor, actualizándose mediante una suma ponderada de este último y las redes objetivo de la iteración anterior.
- Itera sobre N steps.
- Itera sobre N episodios.

En la figura 15 se muestra el pseudocódigo del DDPG [19]:

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

Figura 16. Algoritmo DDPG [19]

4.3. LSTM

Las redes Long Short Term Memory surgen en 1997 con el objetivo de realizar conexiones hacia atrás entre las capas [20]. Son redes neuronales recurrentes mejoradas, que utilizan una unidad de memoria para determinar qué información debe ser retenida o transmitida.

Una celda de memoria administra el sentido de la información, la entrada controla cuando quiere que entre nueva información y la salida controla cuanto tiempo tiene la información almacenada, y si la elimina o la guarda.

LSTM es una red con función de memoria muy utilizada en la actualidad ya que se utiliza también en reconocimiento de voz y escritura. Su estructura interna se muestra en las figuras 17 y 18 [21]:

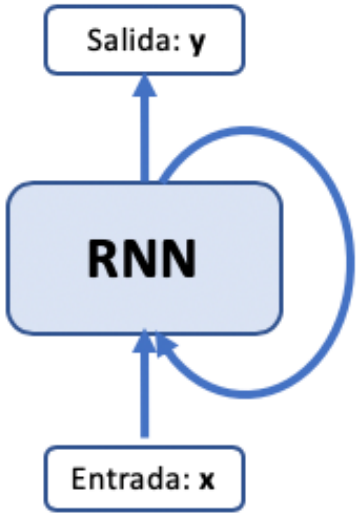

Figura 17. Esquema red recurrente [21]

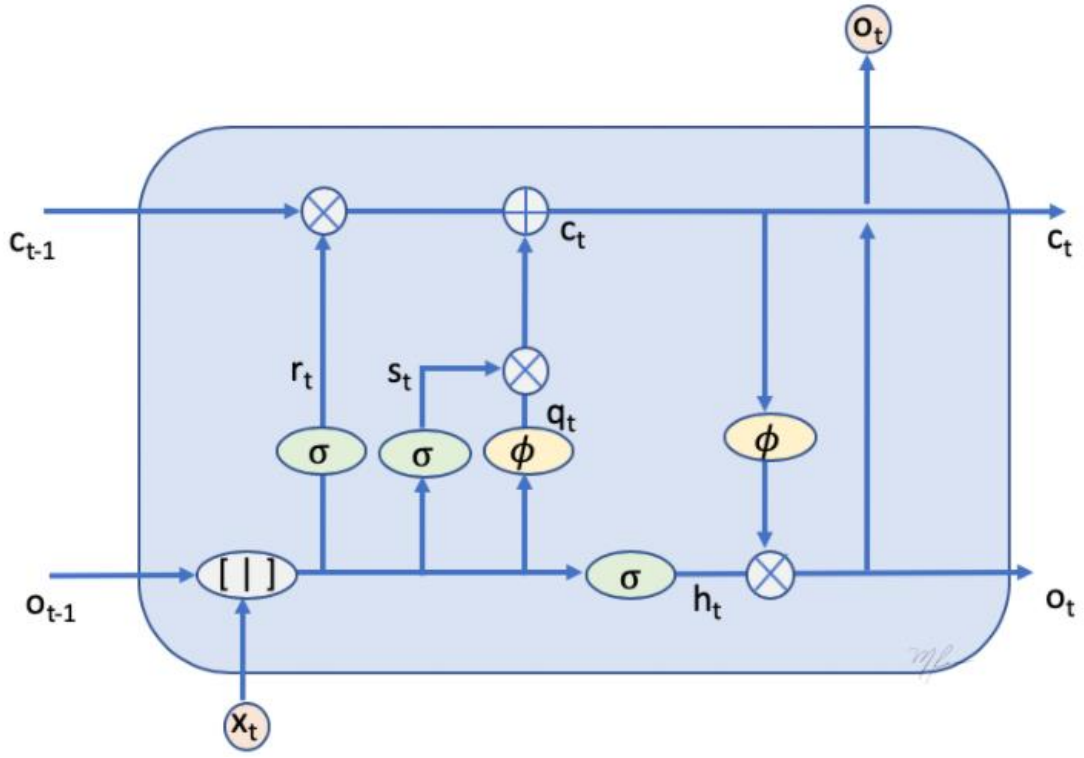

Figura 18. Esquema red Long Short-Term Memory (LSTM) [21]

En la figura 18, se ven las diferentes variables:

- x_t es la entrada en el tiempo t.
- o_{t-1} es la salida de la celda en el tiempo $t-1$, es decir, la salida anterior.
- c_{t-1} es la memoria que pasa por la etapa anterior.
- c_t es la información que pasa a la siguiente etapa.

En total en el esquema se puede ver 3 entradas y 2 salidas.

- Entradas: $[x_t, o_{t-1}, c_{t-1}]$.
- Salidas: $[o_t, c_t]$.

Las diferentes denotaciones de la celda son:

- La función ∅ es una función de activación.
- La operación [|] significa la concatenación de las entradas.
- Las funciones σ aproximan a una respuesta binaria.
- El operador $\oplus$ suma tensores.
- El operador $\otimes$ hace la función de un switch después de la función σ.

La celda del LSTM se actualiza como:

$$\boldsymbol{c_t = c_{t-1} \otimes r_t + i_t \ \otimes \ a_t}$$

El esquema que sigue el LSTM dentro del algoritmo diseñado es el siguiente:

Figura 19. Estructura LSTM

Como se puede ver se aplicará al módulo Critic del agente, y se empleará una LSTM con 128 neuronas artificiales para almacenar información.

4.4. DDPG-LSTM

El algoritmo DDPG-LSTM se forma combinando el DDPG y la red LSTM, la estructura que forma es similar a la del DDPG, pero cambia la parte del critic que se añade la red LSTM de 128 neuronas y la concatenación de ambos módulos del agente.

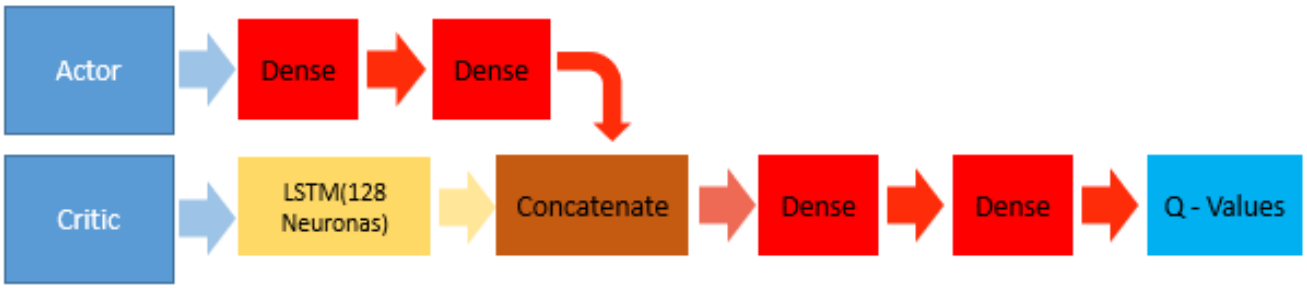

Figura 20. Estructura DDPG-LSTM

Las redes del actor y critic del DDPG adoptan una estructura de 3 capas, las cuales son una etapa de entrada, una de salida y una capa oculta. Para el caso de la capa oculta, en la red recurrente RNN se incluye la red neuronal LSTM, la cual se integra en la estructura del DDPG. Los estados y la memoria (o_{t-1}, c_{t-1}) son la entrada de la red LSTM en el tiempo t para generar una acción A_t y la memoria (información) actualizada (o_t, c_t). Entonces (o_t, c_t) será en el step $t+1$.

La explicación del algoritmo DDPG-LSTM desde un punto más conceptual se resume con estos pasos:

- Se inicializan aleatoriamente la red critic $Q((s_t, A_t), (o_{t-1}, c_{t-1}|\theta^Q))$ y la red del actor $\mu(s_t, (o_{t-1}, c_{t-1})|\theta^\mu)$.
- Se inicializan las redes objetivo Q' y μ' con sus pesos $\theta^{Q'} \leftarrow \theta^Q$ y $\theta^{\mu'} \leftarrow \theta^\mu$.
- Se inicia un proceso iterativo de episodios y el buffer R.
- Para el episodio=1:
    - Inicializa la memoria (o_{t-1}, c_{t-1}) para las redes Q, μ, Q' y μ'.
    - Recibe el estado de observación inicial s_t.
    - Para el step $t = 1$:
        - Selecciona la acción: $(A_t, (o_t, c_t))) = \mu(s_t, (o_{t-1}, c_{t-1})|\theta^Q)$.
        - Ejecuta la acción A_t, recibe la recompensa r y el nuevo estado s_{t+1}.
        - Se almacena en R: $(o_t, A_t, r_t, s_{t+1}, (o_{t-1}, c_{t-1}), (o_t, c_t))$
        - Se muestrea un minibatch aleatorio de N transiciones de R.
        - Se obtiene: $y_i = r_i + \gamma Q'(\left(s_{t+1}, \mu'\left(s_{t+1}, (o_t, c_t)\middle|\theta^{\mu'}\right)\right), (o_t, c_t)|\theta^{Q'})$
        - Se actualiza el critic para minimizar las perdidas:

$$L = \frac{1}{N}\sum_i (y_i - Q\left((s_t, A_t), (o_{t-1}, c_{t-1)}|\theta^Q\right)\right)^2$$

- Se actualiza el actor según el gradiente:

$$\nabla_{\theta^\mu} J = \frac{1}{N}\sum_i \nabla\ Q((s, A), (o, c|\ \theta^Q)\ |_{s=s_i \ldots 1=\mu(s_t,(o_{t-1},c_{t-1}))}$$

$$\nabla_{\theta^\mu}\mu(s,(o,c))|_{s_i}$$

- Se actualizan las redes objetivo:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

- Cierre de bucle: $Step\ t_n$
- Cierre de bucle: $Episodio\ n$.

4.5. MODELO DE AGENTE – WAYPOINTS

El agente va a seguir un modelo de waypoints. Los waypoints son puntos que van marcando la trayectoria y el vehículo debe seguirlos para llegar al punto final.

En la figura 21 se tiene un esquema visual de este modelo, se ve que CARLA proporciona una lista de waypoints a través de su propio planificador global junto a PythonAPI, y serán la entrada del agente junto a los parámetros de conducción para conseguir enviar los parámetros de dirección y aceleración al entorno.

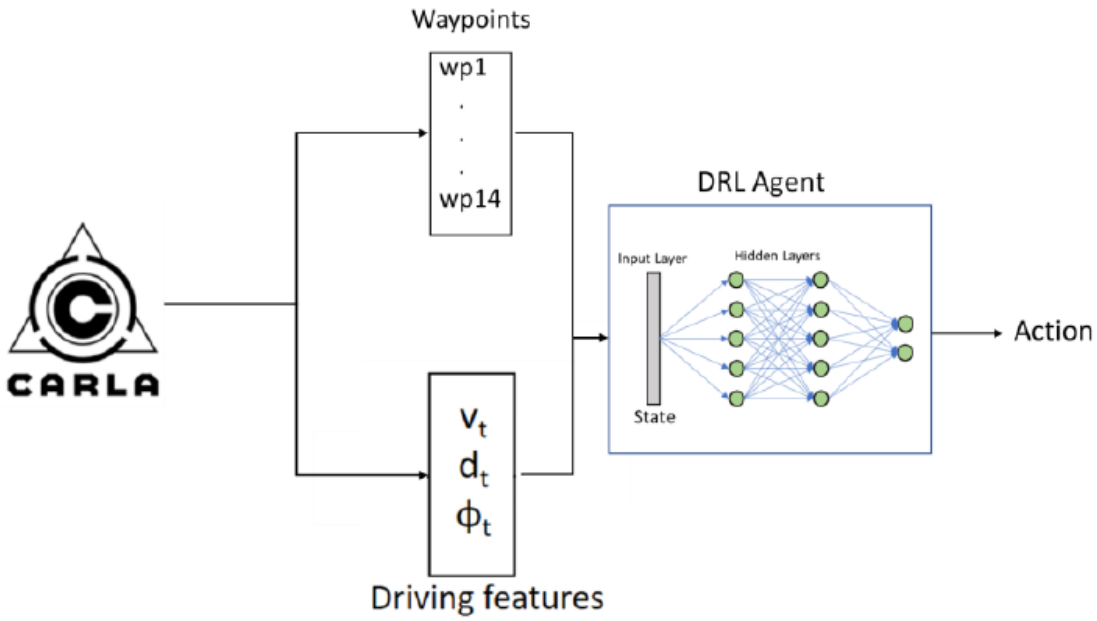

Figura 21. Modelo Waypoints CARLA [18]

Al planificador se le dan dos puntos, el inicial y el final, y a partir de ellos se genera una trayectoria dentro del mapa devolviendo una lista de waypoints que une ambos puntos.

La cantidad de elementos de esta lista de puntos depende de la distancia entre los puntos inicial y final. Como esta lista puede llegar a ser demasiado grande se cogen solo 15 puntos, siendo el primero el más cercano a la posición actual del vehículo y cuando estos se rebasan se van eliminando.

Estos puntos están referenciados globalmente al punto (0,0,0) del mapa de CARLA. Para pasar los puntos referenciados en el mapa a referenciados en el vehículo se necesita una matriz de transformación como la siguiente:

$$M = \begin{pmatrix} \cos(\gamma_c) & -\sin(\gamma_c) & 0 & X_c \\ \sin(\gamma_c) & \cos(\gamma_c) & 0 & Y_c \\ 0 & 0 & 1 & Z_c \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Para construir esta matriz se debe conocer $[X_c, Y_c, Z_c, \gamma_c]$, que indica la posición global del vehículo (X, Y, Z) y el yaw (γ) en cada instante. Para pasar los puntos referenciados del mapa P_{map} a referenciados al vehículo ($P_{vehicle}$) se hace de la siguiente manera:

$$P_{map} = M^{-1} * P_{vehicle} \rightarrow \boldsymbol{P_{vehicle} = M^{-1} * P_{car}}$$

Para formar el vector de estados además de los waypoints, se tienen dos parámetros de la ubicación del vehículo en la carretera:

- $\boldsymbol{d_t}$ indica la posición del vehículo respecto a la carretera.
- $\boldsymbol{\phi_t}$ el ángulo del vehículo respecto a la carretera.

El vector de estados que se forma es el siguiente:

$$S = ([wp_0 \dots wp_{14}], \phi_t, d_t)$$

5. SIMULACIÓN

5.1. INTRODUCCIÓN

Una vez vista la estructura general del sistema, en este apartado se va a explicar cómo se implementa dentro del simulador CARLA, para poder entrenar los diferentes modelos y realizar una evaluación posteriormente.

Se va a utilizar una estructura que permite la comunicación entre CARLA y PythonAPI. Para entrenar cada red neuronal y conseguir un modelo para la fase de evaluación se cuenta con las bibliotecas de código abierto Keras [22] y Tensorflow [23].

Dentro del simulador todo se configura desde scripts que se envían a CARLA, esta los recibe y envía datos de la observación. El aprendizaje del algoritmo es en línea, lo que supone que cuando finaliza un episodio se inicia otro automáticamente.

Dentro de los scripts, los parámetros que se pueden configurar son los siguientes:

- Vehículo: vehículo que se quiere utilizar dentro del simulador. En este caso es el Audi A2, ya que es el vehículo cuyos parámetros son los más parecidos al vehículo eléctrico del grupo de investigación.
- Posición inicial: una vez lanzado el mapa, el simulador necesita una posición inicial para lanzar el vehículo y la simulación.
- Sensores: los sensores que se utilizan en este algoritmo son el sensor de colisión, de cámara y GPS. Todos ellos tienen una ubicación relativa en el vehículo la cual se debe establecer.

- Sensor de colisión: detecta cuando el vehículo choca con otro vehículo o elementos del mapa, como puede ser aceras, farolas, semáforos, etc.

Figura 22. Sensor de colisión en CARLA

- Invasión de carril: es un sistema que proporciona CARLA y da información de que se están cruzando las líneas de la carretera, lo que conlleva que el vehículo está cambiándose de carril. En el circuito que se ha creado no es necesario el cambio de carril por lo que se utilizará en la etapa de entrenamiento para interrumpir el episodio y reiniciarlo cuando se detecte la invasión.

Figura 23. Sensor de invasión de carril en CARLA

- GNSS: da información del centro del vehículo en el mapa. En las figuras 24 y 25 se muestra un ejemplo visual de este sensor en el cual se puede ver la localización del vehículo en dos orientaciones totalmente contraritas. En la figura 24 está en una orientación de -108 º y en la figura 25 en sentido contrario a 72º.

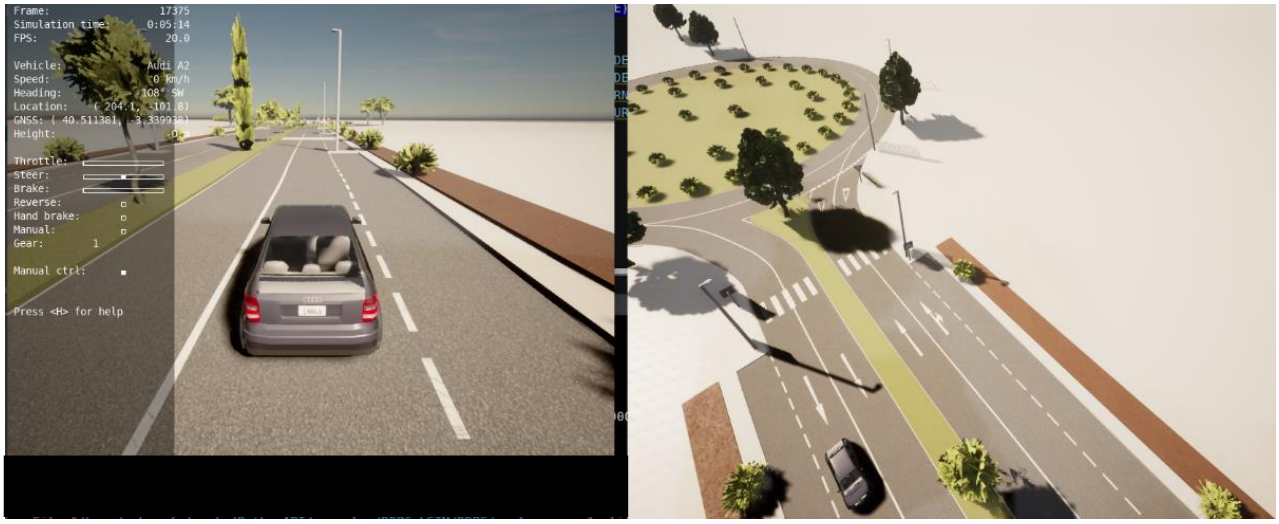

Figura 24. Orientación -108º en CARLA

Figura 25. Orientación en CARLA 72º

5.2. ACCIONES EN DDPG-LSTM

En cuanto a las acciones, el algoritmo DDPG-LSTM es continuo por lo que la salida también lo será. Los comandos de control que se aplican en el vehículo dentro del simulador son el acelerador y la dirección.

El rango de salida depende de la función de activación y en este caso se utiliza la tangente hiperbólica [22]:

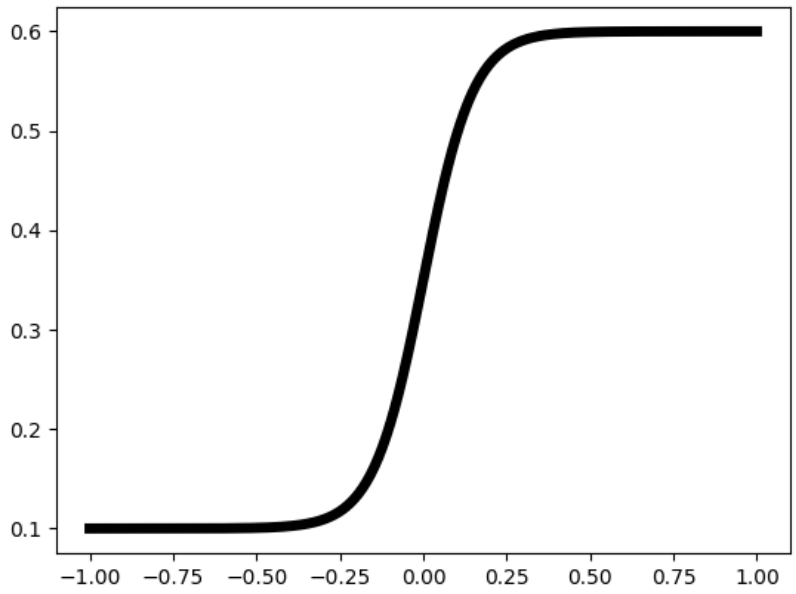

Figura 26. Función de activación tanh

CARLA tiene unos rangos para los comandos de control que para el caso del acelerador el rango es [0,1] y para la dirección [-1,1], por ello el empleo de esta función de activación.

Además, se necesita una ecuación que transforme el rango de salida de la red [-1,1] en acelerador para CARLA, ya que el acelerador real enviado al simulador es [0.2,0.6] que corresponde a 2km/h y 30km/h respectivamente.

$$acelerador = 0.125 * acción_{acelerador} + 0.4$$

En cuanto a la dirección se sigue la siguiente ecuación:

$$dirección = 0.15 * acción_{dirección}$$

6. RESULTADOS EXPERIMENTALES

6.1. INTRODUCCIÓN

En este apartado se verá cómo se obtienen los modelos entrenados para el agente y se comparará para ver cuál de los dos es más eficiente y confiable para un futuro ser probado en el vehículo eléctrico de Robesafe que se muestra en la figura 27.

Figura 27. Vehículo eléctrico del grupo Robesafe (UAH)

6.2. ESCENARIO

Las pruebas se van a realizar en un escenario creado por el proyecto “Tech4AgeCar” recreando el Campus Externo de la Universidad de Alcalá llamado “CampusUAH_RL_v1_0” dentro del simulador CARLA.

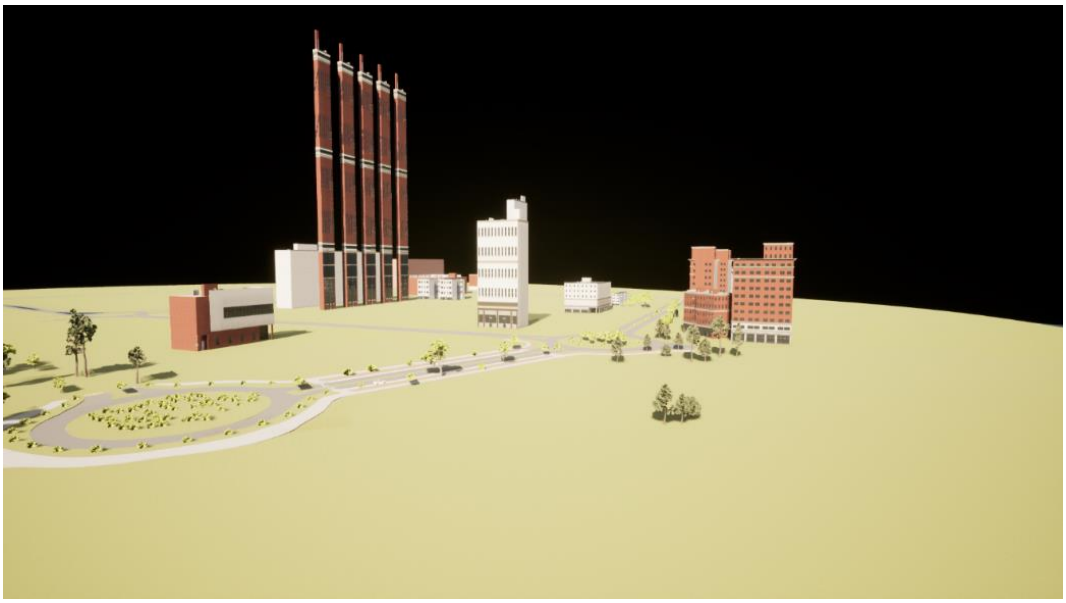

Figura 28. Mapa campus UAH en CARLA

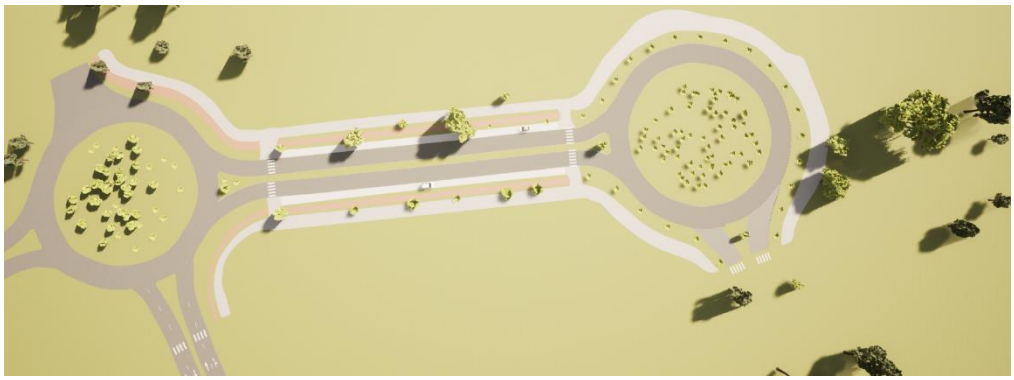

Figura 29. Escenario pruebas

6.3. MÉTRICAS

Las métricas de evaluación para comparar los modelos van a ser:

Para la etapa de entrenamiento:

- Número de episodios necesarios para entrenar el modelo.
- Recompensas obtenidas en función del número de episodios.

En cuanto al número de episodios se refiere a la cantidad de episodios que ha necesitado el modelo para obtener un peso del algoritmo correcto para ser capaz de que el coche llegue al destino final sin colisionar y sin salirse del carril.

Para ver la evolución de las recompensas a lo largo de los episodios, se utilizará una función de Tensorflow que muestrea la recompensa en cada episodio.

Para la etapa de evaluación:

- Error cuadrático medio (MSE) del error lateral y de orientación de la trayectoria realizada.

Para calcular el error lateral, se va a utilizar un algoritmo realizado por uno de los integrantes del grupo Robesafe [25] para obtener la trayectoria ideal mediante la interpolación de los puntos obtenidos por el planificador global de CARLA. Entonces el error lateral será la diferencia entre la trayectoria ideal y real, calculando finalmente el MSE con estos datos.

6.4. ETAPA DE ENTRENAMIENTO

En primer lugar, se lleva a cabo el entrenamiento de los modelos, y una vez se obtengan los pesos entrenados se toman resultados para poder realizar las comparaciones.

El proceso de entrenamiento itera un número de episodios determinados y cada vez que se predice una acción, avanza un paso. La etapa de entrenamiento se va a dividir en tramos de 8000 episodios seguidos, y el peso con mayor recompensa será utilizado como peso de partida para seguir a los siguientes 8000 episodios. El número de veces que se repite este proceso es cuando se vea que el vehículo puede ser llevado a la etapa de evaluación porque ya realiza el circuito correctamente.

La ruta que va a realizar es la circulación de dos rotondas unidas por una recta que se muestra en la figura 30:

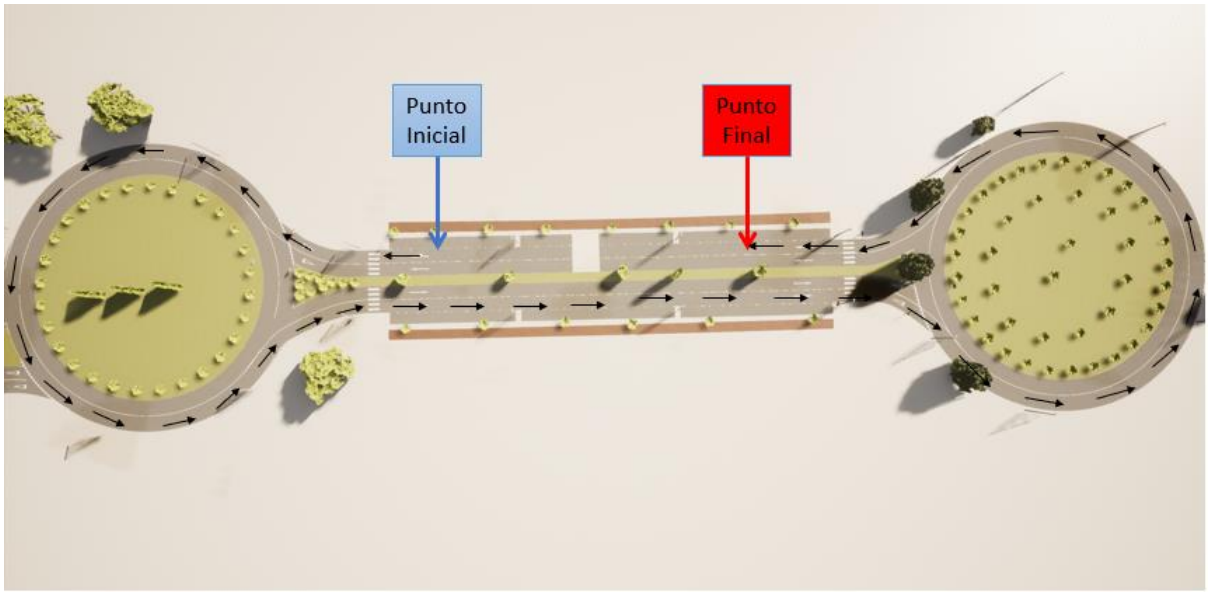

Figura 30. Ruta para la etapa de entrenamiento

El proceso que se lleva a cabo es el siguiente:

- Al inicio de cada episodio, CARLA junto a PythonAPI genera una ruta usando el planificador desde un origen a un destino dado (figura 30) utilizando el algoritmo elegido.
- En cada step se obtiene un vector correspondiente a las características visuales y de conducción para que la acción sea realizada por el vehículo.
- En cada episodio se va calculando una recompensa acumulada y se obtienen unos pesos, los cuales se van actualizando.
- Se escogerá el episodio que haya obtenido una mayor recompensa para utilizarlo en la etapa de evaluación.
- El episodio puede acabar por varios motivos: que llegue a su destino, que se produzca una colisión o que se salga del carril.

En la figura 31 se puede observar una imagen en la etapa de entrenamiento, donde la trayectoria negra son los diferentes waypoints que debe seguir el vehículo

Figura 31. Etapa de entrenamiento en CARLA

6.4.1. RESULTADOS

- EPISODIOS

Los episodios que se muestran en la tabla 1 son los que se han necesitado en la etapa de entrenamiento para obtener la mayor recompensa posible y emplear sus pesos en la etapa de evaluación. La mayor recompensa no se consigue siempre en el último episodio, es probable que después de conseguir la mayor recompensa esta disminuya.

Modelo	Episodios en etapa de entrenamiento
DDPG	24000
DDPG-LSTM	32000

Tabla 1. Episodios según el algoritmo empleado

Se puede observar que es mayor el número de episodios al utilizar la red LSTM que el DDPG clásico, esto se debe a que el modelo LSTM necesita más iteraciones para aprender y realizar el entrenamiento del circuito completo sin colisionar y sin invadir otro carril debido a su mayor capacidad de memoria.

- RECOMPENSAS

Como ya se ha comentado, las gráficas que se muestran a continuación se consiguen a través de un muestreo de la suma de recompensas de cada episodio a lo largo de la etapa de entrenamiento.

Las recompensas que se pueden recibir en cada etapa son las siguientes:

- Invasión de carril: -200
- Colisión: -200
- Llegada al objetivo: +100
- Velocidad mayor a 10 km/h: +1
- Velocidad menos a 10 km/h: -1

A continuación, en las siguientes seis figuras se va a representar la comparativa en la etapa de entrenamiento de la media de recompensas conseguidas en cada episodio, distancia recorrida y máxima recompensa.

- **DDPG**

Las gráficas que se muestran a continuación son sacadas de un entrenamiento de 8000 episodios que parte de un peso de 16000 episodios anteriores, pero solo se representan los últimos 8000, es decir, las recompensas desde el episodio 16000 al 24000 del algoritmo.

Figura 32. Media de recompensas logradas en algoritmo DDPG

En la figura 32 se puede ver como la media de recompensas logradas con el algoritmo básico Deep Deterministic Policy Gradient (DDPG) es al 95% mayor de cero, y en el resto de los episodios las recompensas el valor de la recompensa media está por encima de 10^3.

Figura 33. Distancias logradas en algoritmo DDPG

En la figura 33 se muestran las distancias recorridas con el algoritmo DDPG, se puede observar que sigue el patrón de las recompensas medias, es decir, cuanta más distancia se recorre mayor es su recompensa.

Figura 34. Máximas recompensas logradas en algoritmo DDPG

En la figura 34 se representan las máximas recompensas en el algoritmo, sigue el mismo patrón que la de recompensa media y distancia recorrida en el valle entre el episodio 2200 y 2900. La mayor recompensa se consigue en el episodio 6760 (episodio 22760 del completo) con un valor de 45740.

Con este algoritmo se consiguen recompensas muy altas porque la velocidad a la que se realiza la simulación es mayor al tener menos pasos la red en cada episodio.

- **DDPG-LSTM**

Con la red LSTM se han necesitado más episodios que el DDPG clásico para conseguir un peso que se pueda utilizar en la etapa de evaluación, en total 32000 episodios. Para que las gráficas sean comparables con el DDPG clásico las recompensas que se muestran van desde el episodio 16000 al 24000, por lo que las recompensas y las distancias logradas no serán las máximas obtenidas con este algoritmo.

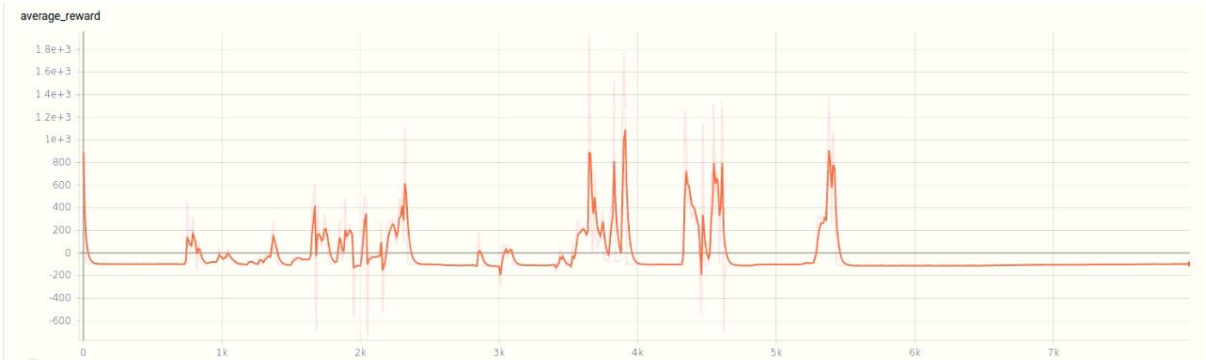

Figura 35. Media de recompensas logradas en algoritmo DDPG-LSTM

En la figura 35 se puede ver que la media de recompensas en el 70% de los casos es menor que cero, totalmente diferente al caso del DDPG clásico. Pero se obtienen intervalos de episodios con rango de valores de recompensa media de 10^3. Esto se debe a que la velocidad del vehículo es menor a 10km/h en la mayor parte del circuito y la velocidad de aprendizaje de la red es más baja.

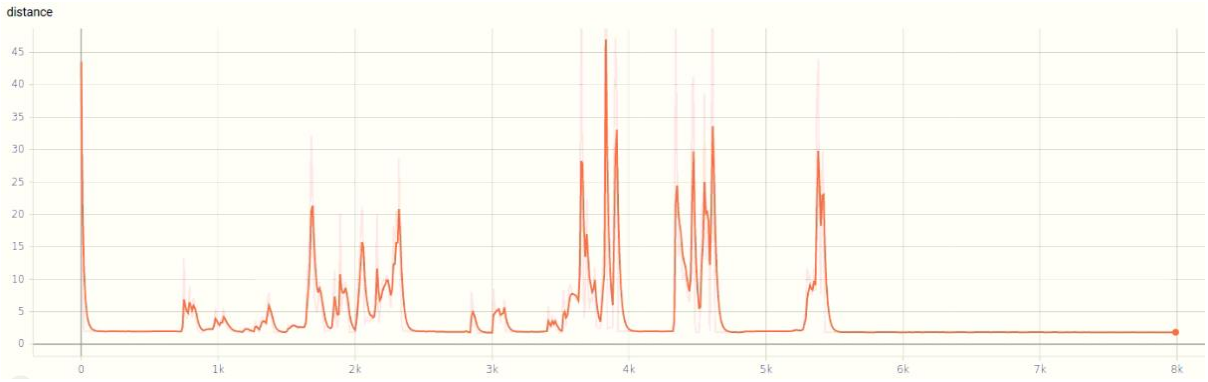

Figura 36. Distancia lograda en algoritmo DDPG-LSTM

En la figura 36 se aprecia que sigue el mismo patrón que en la figura 35, en los intervalos mencionados anteriormente las distancias recorridas son altas y cuando la recompensa es menor que cero la distancia conseguida es muy baja.

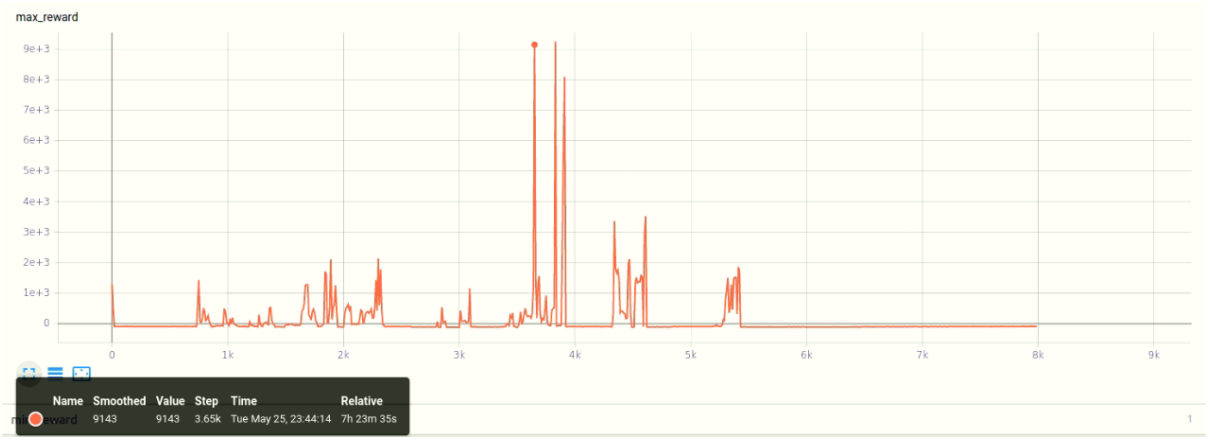

Figura 37. Máximas recompensas logradas en el intervalo con el algoritmo DDPG-LSTM

En la figura 37 se ve la máxima recompensa que se consigue cuando la distancia recorrida también es máxima, la cual se da en el episodio 3650 (episodio 19650 del total) con un valor de 9143.

6.5. ETAPA DE EVALUACIÓN

Una vez realizado el proceso de entrenamiento se avanza a la etapa de evaluación. Se compararán los errores que se cometen al realizar el mismo recorrido.

La ruta que se va a realizar es la misma que en la etapa de entrenamiento. En la figura 38 se puede ver la trayectoria pintada con la lista de waypoints completa:

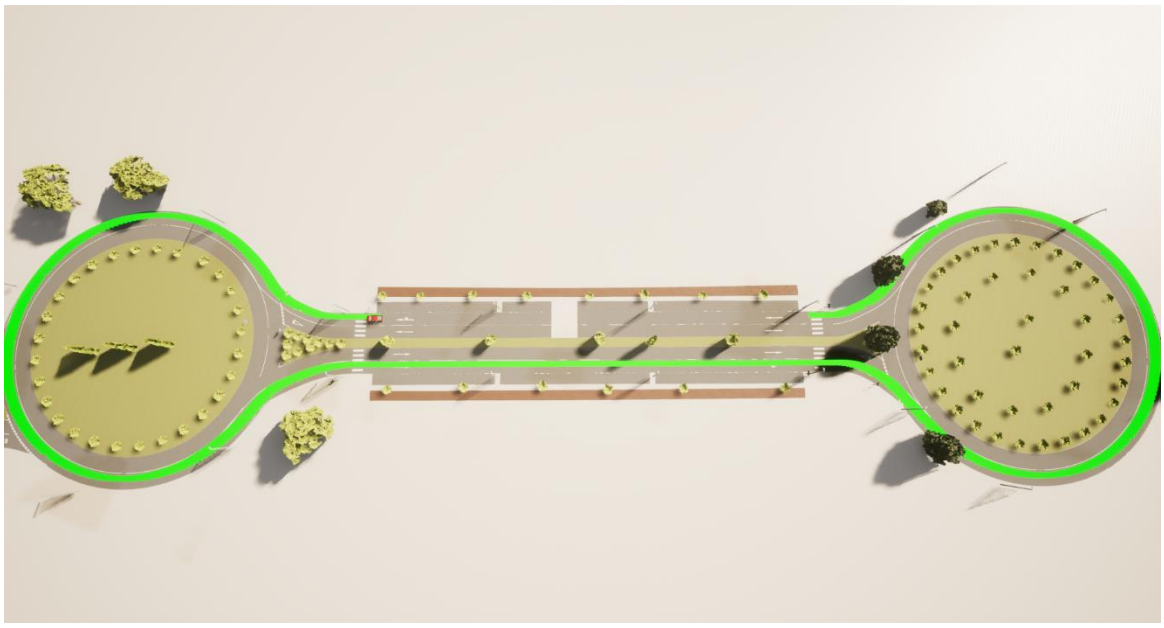

Figura 38. Ruta para la etapa de evaluación

En esta etapa se va a observar el comportamiento del vehículo dentro del simulador CARLA, dándole un punto inicial y otro final como en la etapa de entrenamiento, pero sin que se reinicie ante salidas de carril.

Al igual que en el apartado anterior, en la figura 39 se puede ver la etapa de evaluación dentro del simulador CARLA, el vehículo debe seguir la trayectoria verde formada por el modelo de Waypoints.

Figura 39. Etapa de evaluación en CARLA

El procedimiento para obtener los resultados es el siguiente:

- Escoger los pesos entrenados. Se cogerán los pesos con la mayor recompensa obtenida.

- Deshabilitar el reinicio automático ante salida de carril.

- Obtener las gráficas a través de Matlab con datos que proporciona CARLA.

6.5.1. GRÁFICAS

En la figura 40 se pueden ver las tres trayectorias obtenidas superpuestas, en color rojo está la trayectoria ideal que debería seguir, la verde es la del algoritmo DDPG clásico y la azul es la que se ha obtenido con la red LSTM-DDPG. Se puede observar que el comportamiento de ambas trayectorias es muy bueno.

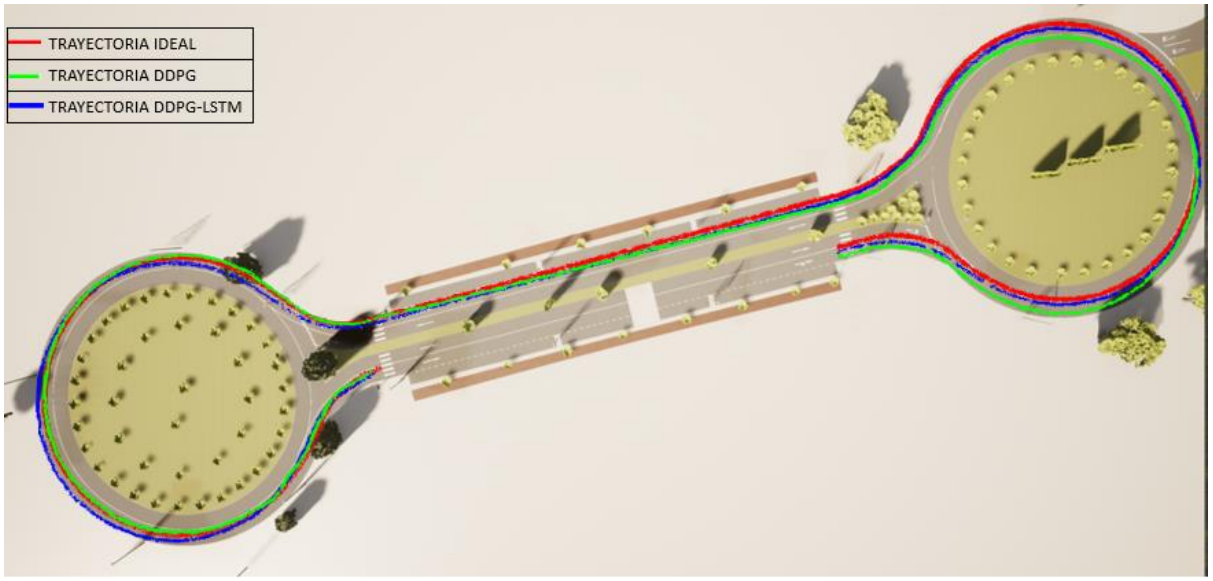

Figura 40. Trayectorias obtenidas junto al mapa

Las gráficas que se muestran en esta etapa se obtienen mediante unos archivos que generan CARLA y PythonAPI, estos se introducen dentro de un script de Matlab que analiza las trayectorias y los waypoints de manera individual, y una vez analizados se pasa a plotearlos frente a un eje de coordenadas XY.

A continuación, se van a mostrar las trayectorias por individual para poder analizarlas por partes.

En la figura 41 se muestra la trayectoria ideal obtenida mediante los waypoints que ofrece el planificador de CARLA.

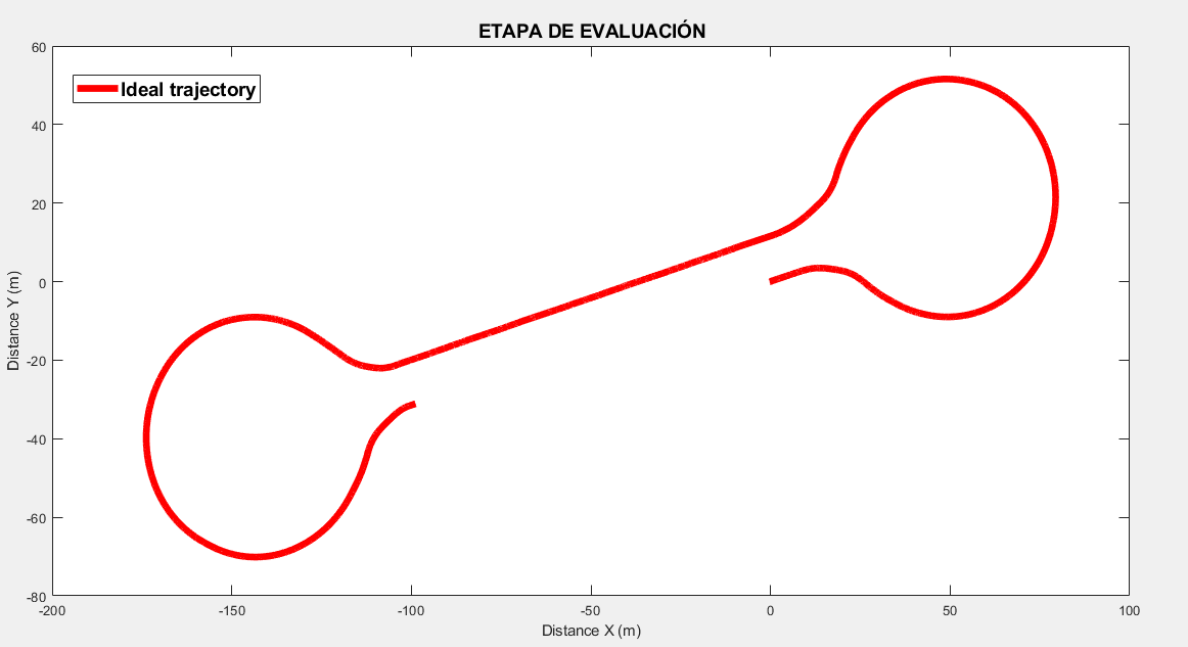

Figura 41. Trayectoria ideal

- **DDPG**

En la figura 42 se muestra la trayectoria que se ha obtenido para el algritmo DDPG clásico, donde se observan unos cambios bruscos de dirección durante toda la trayectoria lo que hacía imposible la implementación en el vehículo real.

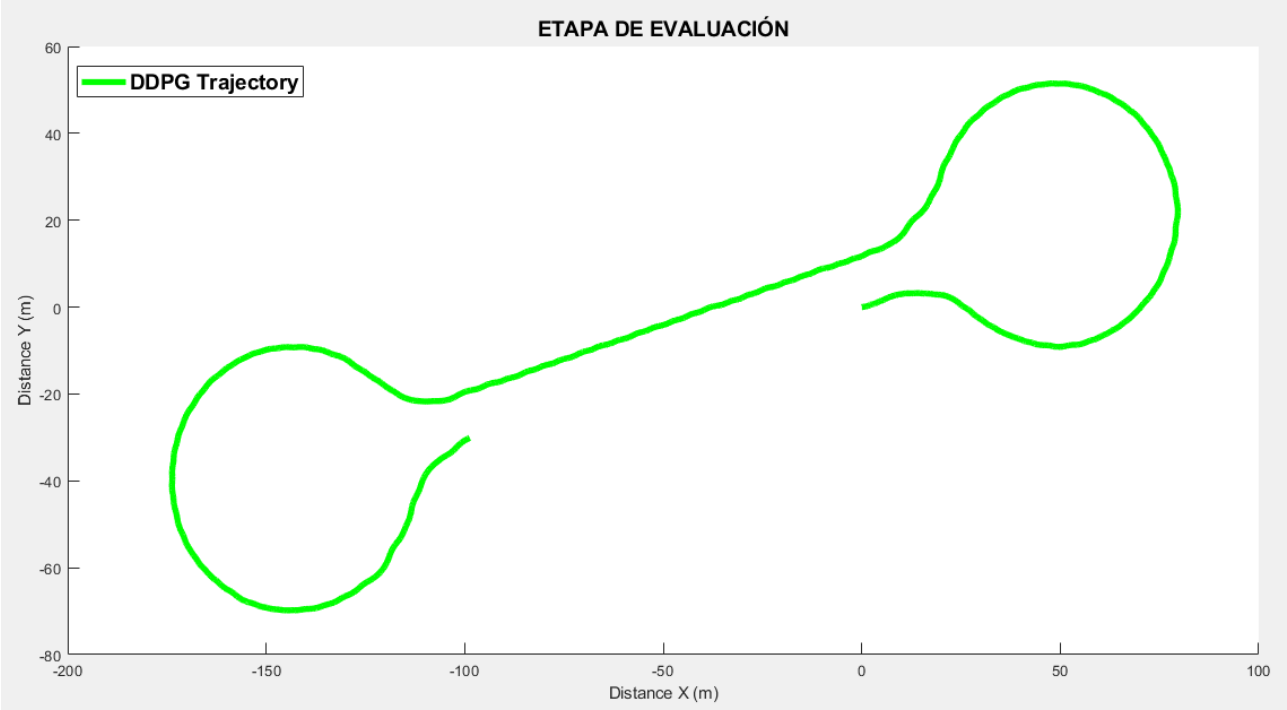

Figura 42. Trayectoria obtenida con DDPG

- **DDPG-LSTM**

En la figura 43 se ve la trayectoria del DDPG-LSTM, donde se corrigen los giros del algoritmo anterior y hace posible la futura implementación. Esto se puede observar perfectamente en el tramo de la recta donde no hay ningún cambio de dirección del volante.

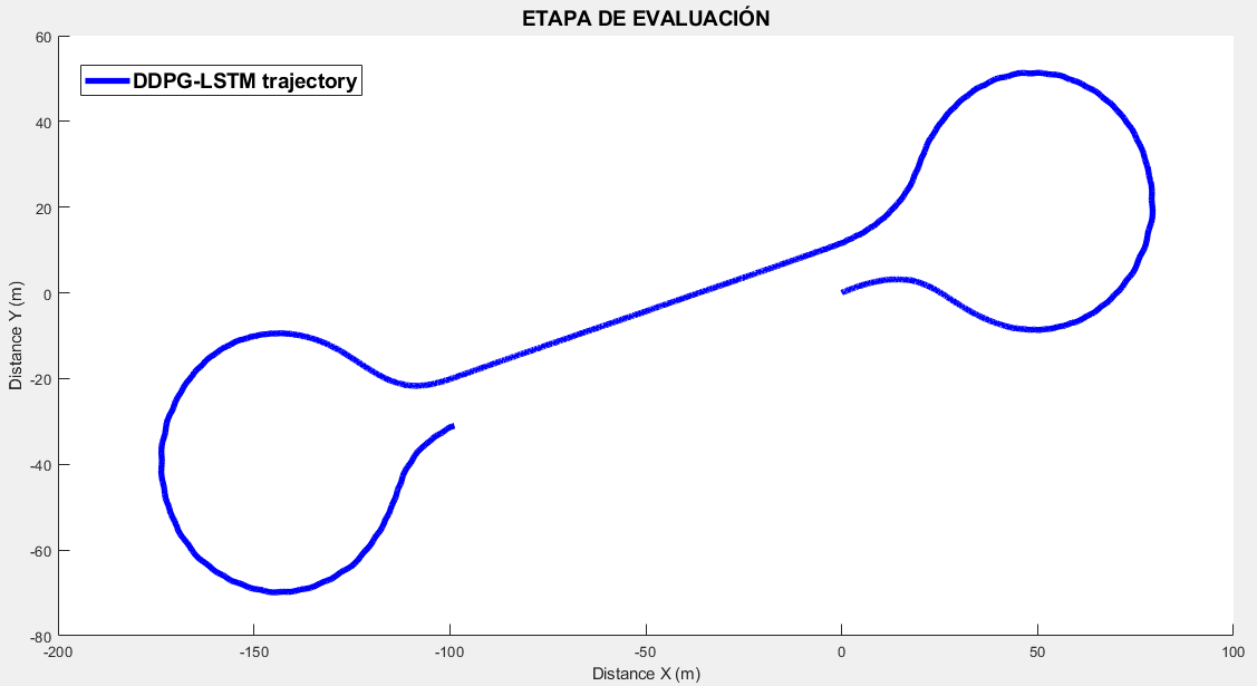

Figura 43. Trayectoria obtenida con DDPG-LSTM

- **COMPARATIVA:**

En la figura 44 se muestra una gráfica comparando las 3 trayectorias donde se observa que en términos generales las trayectorias son prácticamente idénticas, pero si se ve la figura 45, donde se analiza el tramo de salida ampliado de una de las rotondas, se ve como la oscilación del DDPG es mucho mayor a la del DDPG-LSTM.

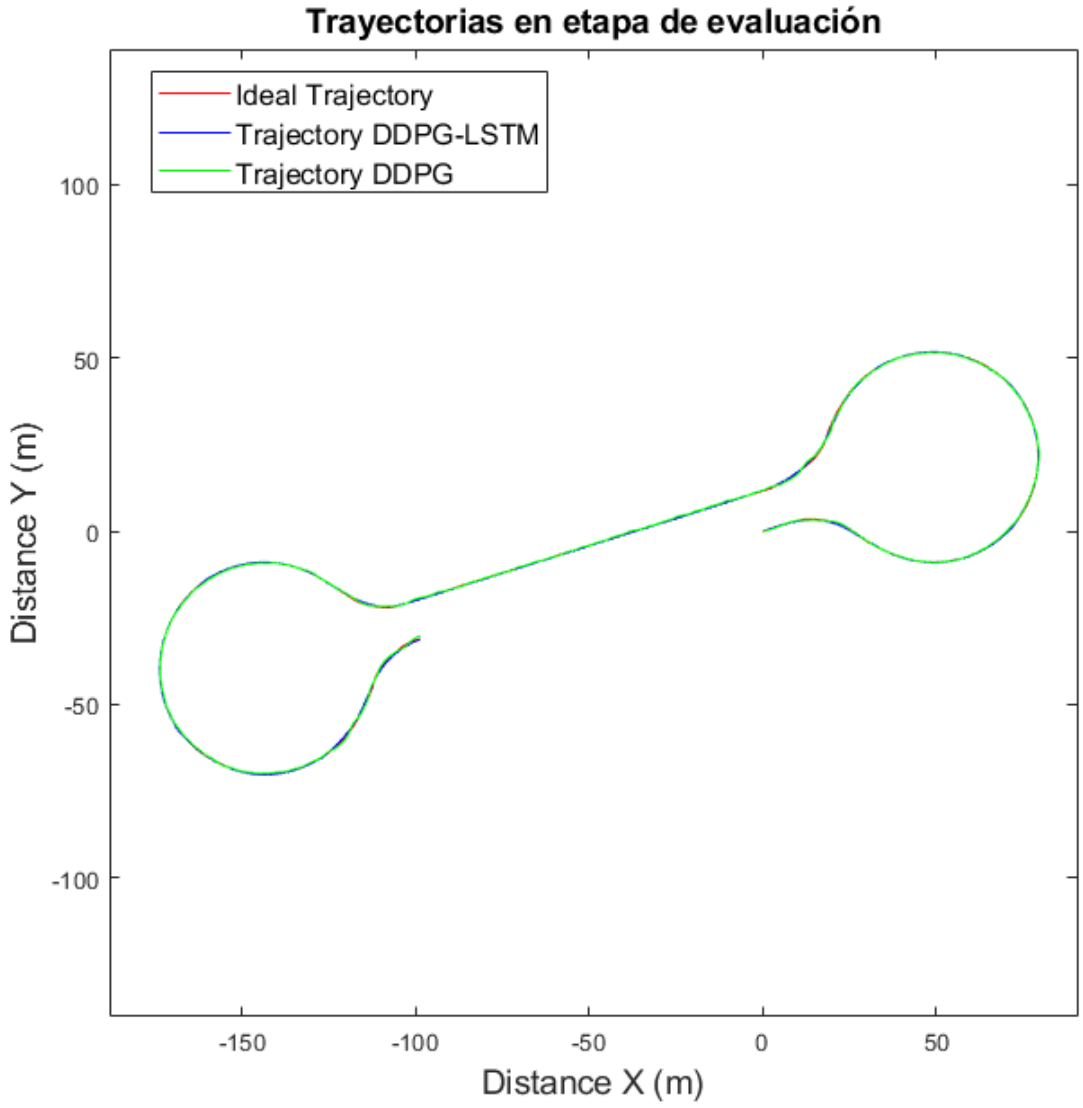

Figura 44. Trayectorias en la etapa de evaluación

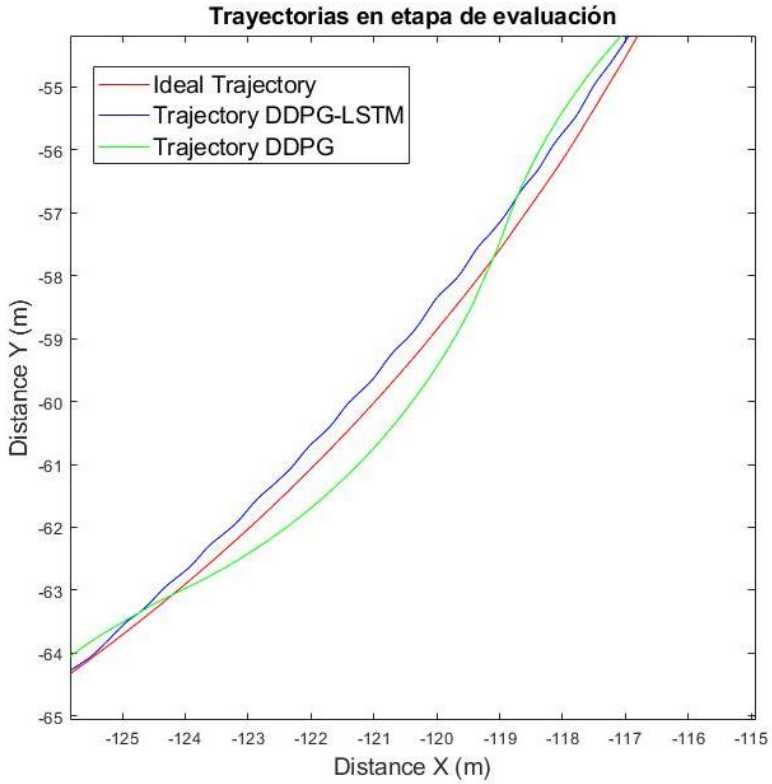

Figura 45. Trayectorias de una curva ampliada en la etapa de evaluación

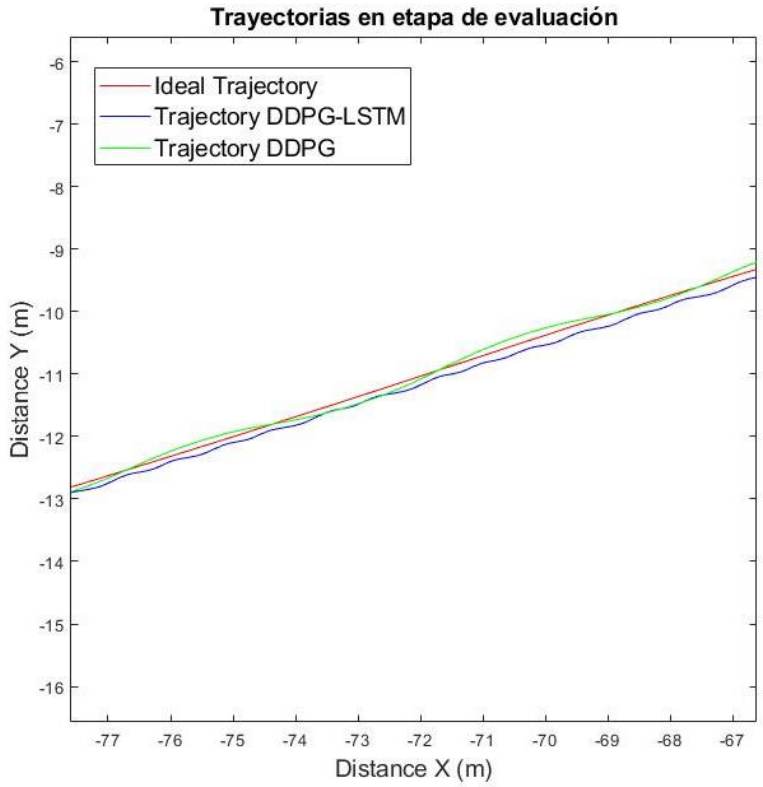

Figura 46. Trayectorias de la recta ampliada en la etapa de evaluación

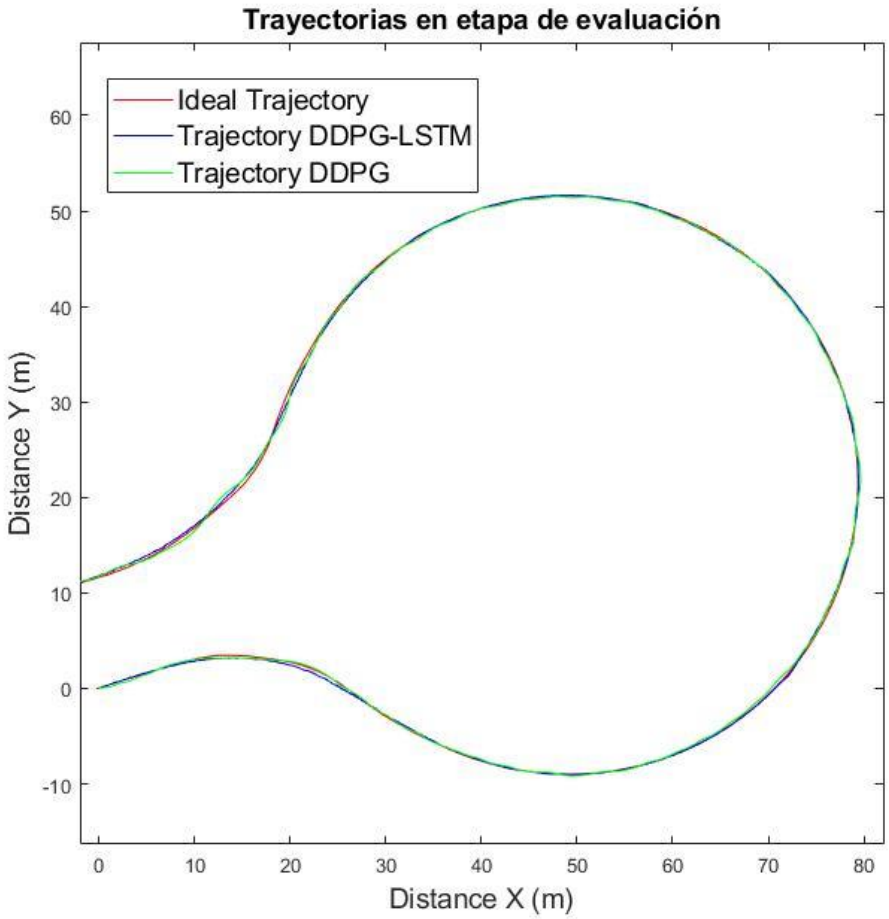

Figura 47. Trayectorias de una rotonda en la etapa de evaluación

Se observa también que la mayor variación de la trayectoria se produce en las rotondas, ya que son tramos en los que el vehículo encuentra dificultades, sin embargo, en el tramo recto, se obtiene una trayectoria con giros muy leves.

6.5.2. ERRORES

Aunque visualmente en las gráficas parece que el algoritmo DDPG-LSTM es ligeramente mejor que el DDPG, ambos algoritmos tienen una trayectoria muy parecida entre la real e ideal, por ello que se analicen los errores laterales y de orientación.

Estos se han obtenido mediante la comparación directa punto a punto entre trayectoria ideal y trayectoria del algoritmo escogido.

En la tabla 2 se puede ver como el máximo error lateral y máximo de orientación es prácticamente el mismo, pero el error medio lateral existe una diferencia en el error de 20 cm a favor del DDPG-LSTM.

Si se analiza el error máximo lateral se ve que es el mismo en ambos algoritmos, esto será el máximo valor permitido para que el vehículo no se salga del carril.

Algoritmo	Error medio lateral	Error máximo lateral	Error de orientación
DDPG	0.386 m	0.824 m	2.8353 º
DDPG-LSTM	0.12 m	0.847 m	2.9015 º

Tabla 2. Error lateral y de orientación según algoritmo empleado

7. CONCLUSIONES

El desarrollo de este trabajo ha permitido la adaptación del algoritmo Deep Deterministic Policy Gradient con una nueva red neuronal Long Short Term Memory, dando lugar a un nuevo algoritmo conocido como DDPG-LSTM. El algoritmo desarrollado ha sido entrenado y evaluado dentro del simulador CARLA que a través de Matlab y Tensorflow se han podido obtener diferentes resultados experimentales.

En primer lugar, las recompensas que se han ido generando en las etapas de entrenamiento se ha comprobado que son mayores en el algoritmo anterior (DDPG) que en el DDPG-LSTM, esto no quiere decir que el algoritmo sea peor o mejor, sino que se debe a que en la simulación este último va a una velocidad menor y hace que no se alcancen recompensas tan altas.

En segundo lugar, se ha desarrollado la etapa de evaluación donde primero se han mostrado las gráficas comparativas de ambos algoritmos y se ha visto que con el algoritmo DDPG-LSTM no existen giros tan bruscos.

Después, se ha evaluado el error lateral y de orientación en la etapa de evaluación, y se ha visto que con ambos algoritmos el error máximo es prácticamente el mismo, pero el error medio lateral muestra que el algoritmo DDPG-LSTM es bastante mejor ya que el valor es mucho más bajo.

Esto lleva a que se ha conseguido el objetivo para el proyecto Robesafe, ya que era imposible implementar los pesos del algoritmo DDPG antiguo y con esta adaptación se ha reducido el error lateral medio y en trabajos futuros se puede llegar a conseguir una navegación del vehículo real a través de Deep Reinforcement Learning.

Comentado todo esto, la conclusión general del proyecto es que tanto el algoritmo DDPG como el DDPG-LSTM son totalmente válidos en simulación, pero para el vehículo real el DDPG-LSTM hace que se pueda llegar a implementar sobre él y solucionar el problema de la navegación.

8. PLIEGO DE CONDICIONES

8.1. HARDWARE

Ordenador Z390 Gaming X (Gigabyte Technology):

- Intel Core i7-9700 CPU @ 3.60GHz.
- RAM: 32GB DDR4
- 500 GB SSD.
- Nvidia GeForce 2080 RTX.

8.2. SOFTWARE

- Ubuntu 18.04.3 LTS (Bionic Beaver)
- CARLA Simulator 0.9.6
- Matlab Rb2019b
- Microsoft Office 365 ProPlus
- Python 3.6.9
    - Keras 2.2.4
    - Keras-Application 1.0.8
    - Keras-Preprocessing 1.1.0
    - OpenCV-Python 4.1.2.30
    - Pygame 1.9.6
    - Tensorboard 1.14.0
    - Tensorflow-estimator 1.14.0
    - Tensorflow-GPU 1.14.0
    - Networkx 2.4
    - Numpy 1.18.1
    - Matplotlib 3.13
- Cuda 10.0
- NVIDIA Drivers 440.100

9. PLANOS

INSTALACIÓN CARLA

Abrir un terminal:

sudo add-apt-repository ppa:ubuntu-toolchain-r/test

sudo apt-get update

sudo apt-get install build-essential clang-6.0 lld-6.0 g++-7 cmake ninja-build python python-pip python-dev python3-dev python3-pip libpng16-dev libtiff5-dev libjpeg-dev tzdata sed curl wget unzip autoconf libtool

pip3 install --user setuptools nose2

sudo update-alternatives --install /usr/bin/clang++ clang++ /usr/lib/llvm-6.0/bin/clang++ 102

sudo update-alternatives --install /usr/bin/clang clang /usr/lib/llvm-6.0/bin/clang 102

git clone --depth=1 -b 4.21 https://github.com/EpicGames/UnrealEngine.git ~/UnrealEngine_4.21

cd ~/UnrealEngine_4.21

./Setup.sh && ./GenerateProjectFiles.sh && make

git clone https://github.com/CARLA-simulator/CARLA

./Update.sh

export UE4_ROOT=/media/robesafe/SSD500/UnrealEngine

make launch # Compiles the simulator and launches Unreal Engine's Editor.

make PythonAPI # Compiles the PythonAPI module necessary for running the Python examples.

make package # Compiles everything and creates a packaged version able to run without UE4 editor.

make help # Print all available commands

LIBRERIAS PYTHON

sudo apt-get update

sudo apt-get install python3.6

sudo apt install python3-pip

pip3 install tensorflow-gpu==1.14

pip3 install tensorboard==1.14

pip3 install numpy

pip3 install neworkx

pip3 install keras==2.2.4

pip3 install opencv-pyhton

pip3 install matplotlib

EJECUCIÓN CARLA

En un terminal:

cd CARLA/Dist/CARLA_Shipping_0.9.6-23-g89e329b7/LinuxNoEditor

sh ./CARLAUE4.sh

En otro terminal:

cd ~/CARLA/PythonAPI/examples/ /DDPG_LSTM

python3 launch_DDPG.py

CÓDIGO CRITIC CON RED NEURONAL LSTM

```
def generate_model(self):
    state_input = Input(shape=[self.state_size])
    print("Dimensiones state")
    print(state_input.shape)
    state_h1 = Dense(hidden_units[0], activation="relu")(state_input)
    state_h2 = Dense(hidden_units[1], activation="linear")(state_h1)

    action_input = Input(shape=[2,1])
    print("Dimensiones action")
    print(action_input.shape)

    lst_layer = LSTM(units=128, activation="tanh", input_shape=(hidden_units[0], hidden_units[1]))(action_input)
    print("DIMENSIONES DEL LSTM")
    print(lst_layer.shape)
    action_h1 = Dense(hidden_units[1], activation="linear")(lst_layer)
    merged = add([state_h2, action_h1])
    merged_h1 = Dense(hidden_units[1], activation="relu")(merged
    output_layer = Dense(1, activation="linear")(merged_h1)
    model = Model(input=[state_input, action_input], output=output_layer)
    model.compile(loss="mse", optimizer=Adam(lr=self.lr))
    model.summary()
    return model, state_input, action_input
```

REFERENCIAS

[1] Shantanu Ingle, M. P. (2016). Tesla Autopilot: Semi Autonomous Drivig, an Uptick for Future Autonomy. International Research Journal of Engineering and Technology (IRJET).

[2] Zhicheng Gu, Zhihao Li, Xuan Di, Rongye Shi. (2020). *An LSTM-Based Autonomous Driving Model Using Waymo Open Dataset.*

[3] Gao, R. (2018). Waymo's new 360-degree video uses incredible visuals to explain how its self-driving tech works.

[4] SAE INTERNATIONAL. (2018). J3016_201806 - Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles.

[5] Liu, H., Sun, F., Guo, D., Fang, B., & Peng, Z. (2017). Structured Output-Associated Dictionary Learning for Haptic Understanding. IEEE Transactions on Systems, Man, and Cybernetics: Systems.

[6] Piñas, J. M. (2019). Deep Reinforcement Learning aplicado a la Coducción Autónoma. Obtenido de https://e-archivo.uc3m.es/bitstream/handle/10016/30350/TFG_Javier_Moralejo_Pi%C3%B1as.pdf?sequence=1 . Visto 20/02/2021.

[7] Wikipedia. (27 de 02 de 2017). Red neuronal artificial. Obtenido de https://es.wikipedia.org/wiki/Red_neuronal_artificial Visto 05/02/2021.

[8] Richard S. Sutton y Andrew G. Barto. (2014,2015). Reinforcement Learning: An Introduction. Obtenido de https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf

[9] Mwiti, D. (16 de 07 de 2021). 10 Real-Life Applications of Reinforcement Learning. Obtenido de https://neptune.ai/blog/reinforcement-learning-applications Visto 20/05/2021.

[10] Bharathan Balaji. (05 de 11 de 2019). DeepRacer: Educational Autonomous Racing Platform for Experimentation with Sim2Real Reinforcement Learning. Obtenido de https://arxiv.org/pdf/1911.01562.pdf . Visto 20/05/2021.

[11] Amazon. (s.f.). AWS DeepRacer. Obtenido de https://aws.amazon.com/es/deepracer/ . Visto 20/05/2021.

[12] Iftikhar Rasheed, Fei Hu, Lin Zhang. (2020). Deep reinforcement learning approach for autonomous vehicle systems for maintaining security and safety using LSTM-GAN.

[13] Noriega, V. (26 de 05 de 2019). *A3C: Actor-Critic en OpenAIGym*. Obtenido de https://victornoriega.github.io/a2c-refuerzo/ . Visto 20/05/2021.

[14] Análisis de código fuente A3C. (s.f.). Obtenido de https://programmerclick.com/article/64221972146/. Visto 20/05/2021..

[15] CARLA: Open-source simulator for autonomous driving research. (s.f.). Obtenido de https://CARLA.org/ . Visto 01/02/2021.

[16] Intel Labs; Toyota Research Institute; Computer VIsion Center. (10 de 11 de 2017). CARLA: An Open Urban Driving Simulator.

[17] Computer Science Department, Stanford University, Stanford, CA. (01 de 2009). ROS: an open-source Robot Operating System.

[18] Gil, Ó. P. (2020). Application of deep reinforcement learning for self-driving vehicles

[19] Aprendizaje de refuerzo profundo combat-Tensorflow implementa DDPG. (s.f.). Obtenido de https://programmerclick.com/article/1345825640/ Visto 13/02/2021.

[20] Aprende Machine Learning. (2018). Obtenido de https://www.aprendemachinelearning.com/breve-historia-de-las-redes-neuronales-artificiales/ . Visto 15/02/2021.

[21] Rivera, M. (11 de 2018). Introducción a Redes Neuronales Recurrentes (RNN). Obtenido de http://personal.cimat.mx:8181/~mrivera/cursos/aprendizaje_profundo/RNN_LTSM/introduccion_rnn.html . Visto 15/02/2021.

[22] Keras. (s.f.). Obtenido de https://keras.io/ . Visto 12/02/2021.

[23] Tensorflow. (s.f.). Obtenido de https://www.tensorflow.org/ . Visto 12/02/2021.

[24] Función de activación de la tangente hiperbólica de la red neuronal. (s.f.). Obtenido de https://programmerclick.com/article/41221174718/ . Visto 25/06/2021.

Universidad de Alcalá

Escuela Politécnica Superior

ESCUELA POLITECNICA SUPERIOR