Universidad Internacional de La Rioja (UNIR)

ESIT

Máster Universitario en Industria 4.0

Predicción en la navegación de vehículos autónomos basada en Deep Reinforcement Learning

Trabajo Fin de Máster

presentado por: Sarango Chamba, Roger Augusto

Director/a: Rodríguez García, Luis Alberto

Resumen

Actualmente el despliegue de vehículos autónomos presenta múltiples desafíos en organizaciones empresariales, industriales, tecnológicas y legislativas. Todos estos retos deben ser revisados y solucionados antes que puedan ser usados para la movilidad en diversos entornos. Las funcionalidades de los VA ofrecen mejoras para las tareas de búsqueda y rescate debidas a desastres naturales, en donde se requiere una exploración en entornos desconocidos. Para que sea posible la conducción autónoma en vehículos y puedan cumplir con las tareas asignadas, se requiere implementar técnicas de inteligencia artificial. En este trabajo se busca integrar un algoritmo de Aprendizaje por Refuerzo Profundo (DQN) en la navegación por rutas, mediante simulación se verifica el funcionamiento del sistema propuesto de VA usando DRL y finalmente se presenta los resultados obtenidos del sistema diseñado para VA.

Palabras clave: DRL, AI, Autonomous Vehicles, Reinforcement.

Abstract

Currently the deployment of autonomous vehicles presents multiple challenges in business, industrial, technological and legislative organizations. All of these challenges must be reviewed and resolved before they can be used for mobility in various environments. VA capabilities offer enhancements for search and rescue operations due to natural disasters, where exploration into unknown environments is required. In order for autonomous driving in vehicles to be possible and to fulfill assigned tasks, it is necessary to implement artificial intelligence techniques. This work seeks to integrate a Deep Reinforcement Learning (DQN) algorithm in route navigation, by means of simulation the operation of the proposed AV system is verified using DRL and finally the results obtained from the system designed for AV are presented.

Keywords: DRL, AI, Autonomous Vehicles, Reinforcement.

Índice de contenidos

Índice de figuras

Índice de tablas

Índice de Ecuaciones

Acrónimos y Abreviaturas

A2C	Advantage Actor-Critic
A3C	Asynchronous Advantage Actor-Critic
ACKTR	Actor Critic using Kronecker Trust Region
ADAS	Advanced Driver Assistance Systems
AI	Artificial Intelligence
AirSim	Aerial Informatics and Robotics Simulation
CA	Conducción autónoma
CARLA	Car Learning to Act
CAV	Connected and Autonomous Vehicles
CH	Conductor Humano
CNN	Convolutional Neural Network
CR	Control Remoto
DDPG	Deep Deterministic Policy Gradient
DL	Deep Learning
DNN	Deep Neural Network
DQN	Deep Q-Network
DRL	Deep Reinforcement Learning
ECU	Engine Control Unit
GPS	Global Positioning System
HD	High-definition
HIL	Hardware in the Loop
HOG	Histogram of Gradients
ICP	Iterative Closest Point
IMU	Inertial Measurement Unit
LQR	Linear–Quadratic Regulator
MIL	Model in the Loop
ML	Machine Learning
MPC	Model Predictive Control
NDT	Normal Distributions Transform
PID	Proportional–Integral–Derivative
PPO-Clip	Proximal Policy Optimization
PPO-Penalty	Proximal Policy Optimization

QAC	Q-value Actor-Critic
RCNN	Regiones con características CNN
RMSE	Root-Mean-Square Error
RNA	Redes Neuronales Artificiales
RNN	Recurrent Neural Network
ROI	Region of Interest
SAC	Soft Actor-Critic
SAE	Society of Automotive Engineers
SCA	Sistemas de Conducción Autónoma
SCNN	Sequential Convolutional Neural Network
SIL	Software in the Loop
SLAM	Simultaneous Localization and Mapping
SLR	Systematic Literature Review
SO	Sistema Operativo
SSD	Single Shot MultiBox Detector
TD3	Twin Delayed DDPG
TOF	Time of Flight
TR	Tiempo Real
TRPO	Trust Region Policy Optimization
UAV	Unmanned Aerial Vehicle
UE	Unreal Engine
USAR	Urban Search and Rescue
VA	Vehículo autónomo
VAC	Vehículos Autónomos y Conectados
VSC	Visual Studio Code
YOLO	You Only Look Once

1. Introducción

En la actualidad se viven múltiples transformaciones tecnológicas en diferentes áreas de la industria; como es el caso del transporte, esto se debe a factores como la cantidad de vehículos en uso en todo el mundo. En 2015 había casi 1300 millones de vehículos en uso (OICA, 2021) lo que representa el aumento de emisión de gases de efecto invernadero, y esto obliga a reforzar la importancia de la movilidad y desarrollo sostenible. El vehículo autónomo (VA) presenta un gran potencial para la conducción eficiente mediante la utilización de vehículos ligeros y seguros (Silva et al., 2022).

Se afirma que los vehículos autónomos brindaran la solución de algunos problemas sociales existentes hoy en día, aportando con la seguridad de tráfico, excluyendo a los humanos en la tarea de conducir. Los VA tienen el potencial de eliminar la conducción en estado de ebriedad, la conducción con fatiga, también podría ayudar a personas mayores o con discapacidades físicas aumentando su movilidad y la eliminación de la dependencia de personas en la conducción para una mayor movilidad de personas en el transporte público. La movilidad futura se plantea integrar taxis robotizados y servicios de transporte más flexibles unificándolos en un único servicio de movilidad (Van Uytsel & Vasconcellos Vargas, 2021).

La investigación e inclusión de tecnologías emergentes para vehículos autónomos está haciendo evolucionar a los sistemas, políticas y redes de transporte de acuerdo a las nuevas necesidades y prioridades que exige el mercado de hoy en día. De acuerdo al nivel de autonomía de los vehículos se presentan diferentes desafíos para la planificación de movilidad urbana, en donde los sectores públicos y privados desarrollan sistemas que promueven nuevos planteamientos para el transporte actual (Legacy et al., 2019).

La tecnología VA esta aun en desarrollo y se busca obtener el máximo beneficio de la autonomía en diferentes entornos de aplicación. Dentro de VA existen diferentes niveles de automatización y está basada en la normativa de la Sociedad de Ingenieros Automotrices (SAE, 2021), se considera 6 niveles desde la no automatización (nivel cero) hasta la automatización total (nivel 5), donde el vehículo puede operar en cualquier condición ambiental o estado de infraestructura.

Existen diversos enfoques para la aplicación de conducción autónoma, uno de los importantes a considerar son los impactos que tienen los desastres naturales y provocados por el hombre debido a construcciones o asentamientos poblacionales. Debido a las consecuencias de estas catástrofes se busca implementar vehículos de búsqueda y rescate,

donde sus capacidades permitan realizar tareas de exploración, mapeo de entornos desconocidos, entregar objetos esenciales a las víctimas o brindar soporte en actividades de rescate (Habibian et al., 2021). Para realizar estas tareas se requiere incluir habilitadores tecnológicos. Tal es el caso de la inteligencia artificial, donde se busca implementar un conjunto de procesos de control, detección, predicción del entorno y navegación del vehículo en la arquitectura VA. Se busca siempre la robustez ante eventualidades del entorno, garantizando la estabilidad y proporcionar una ruta adecuada para llegar al destino (Pérez-Gil et al., 2022). El comportamiento de conducción debe ser correcto, usando políticas de control basadas en Machine Learning, tienen que superar la falta de una métrica que evalué la calidad de conducción. Los métodos de aprendizaje supervisado no aprenden la dinámica del entorno ni del agente. Mientras que el aprendizaje por refuerzo esta formulado para manejar procesos de decisión secuenciales y lo convierte en un enfoque natural para aprender políticas de control en sistemas de conducción autónoma (Talpaert et al., 2019).

En este trabajo se ocupa en la implementación de técnicas de IA basadas en algoritmos de Aprendizaje por Refuerzo Profundo (DRL en inglés) a la capa de control en la arquitectura VA para la navegación autónoma de vehículos. El objetivo es planear una ruta hacia un objetivo, evitando colisiones con el uso de sensores que permitan reconocer el entorno mediante simulación.

1.1. Motivación

En la actualidad existen múltiples retos y problemas a los cuales el área de los vehículos autónomos se debe enfrentar: congestiones de tráfico, accidentes, cambios de velocidad en zonas pobladas y urbanas, etc. Por este hecho, se están desarrollando marcos técnicos para la conducción autónoma que cambiarán la forma de transporte en zonas urbanas y rurales (Pérez-Gil et al., 2022).

La navegación autónoma se ha convertido en un activo valioso para los sistemas basados en arquitecturas de vehículos autónomos. El principal reto es la planificación de rutas, cuya tarea es encontrar el mejor camino para que el agente del VA pueda guiarse hacia su objetivo, lo que se busca es que los algoritmos de planificación de rutas intenten encontrar obtener la mejor ruta minimizando la cantidad de tiempo para recorrer dicho camino. Esto resulta importante en misiones de búsqueda y rescate, donde se requiera brindar auxilio a las víctimas, movilización de herramientas, insumos médicos, etc. También hay que considerar otras funciones de optimización como las características del terreno, las limitaciones de movilidad del vehículo autónomo (Sánchez-Ibáñez et al., 2021).

Por lo tanto, se propone la implementación de algoritmos de aprendizaje por refuerzo profundo en la capa de control de vehículos autónomos. Esto permitirá mejorar los sistemas de navegación en VA que serían operativos en zonas con terrenos irregulares como los presentes en Ecuador. En este sentido, es de especial interés su uso en zonas rurales, poblaciones que estén ubicadas en lugares remotos, terrenos muy accidentados como en la región de la sierra y similares. Los lugares donde no existe una infraestructura vial adecuada para el tránsito vehicular y que son propensos a terminar en muy malas condiciones, si se da algún desastre natural, son también de especial interés para el uso de VA específico en este contexto.

1.2. Planteamiento del trabajo

En este trabajo se pretende desarrollar e implementar un sistema de navegación basado en técnicas DRL que permita al vehículo autónomo navegar en la carretera de manera eficiente. Para evaluar el funcionamiento de este sistema de conducción autónoma se usará una plataforma de simulación cuyo mapa tendrá un entorno rural que emulará algunas condiciones del medio ambiente.

Para cumplir con el objetivo de este trabajo se realizará una investigación de las diferentes técnicas existentes en la literatura y se seleccionará los mejores algoritmos de inteligencia artificial para la navegación, se implementará en simulación para demostrar el funcionamiento del agente en un entorno desconocido. También se considera el proceso del despliegue de VA en relación con la conectividad con otros vehículos, estado de la infraestructura vial, efecto sobre la movilidad.

La finalidad de este TFM es realizar una investigación en el área de la inteligencia artificial, específicamente en la aplicación de algoritmos de Aprendizaje por Refuerzo Profundo en vehículos autónomos para la planeación de rutas en zonas desconocidas o de difícil acceso en donde se requiere que el agente tenga un óptimo rendimiento en estos entornos.

1.3. Estructura de capítulos

En este documento, la primera sección se expone una introducción breve de la tecnología de vehículos autónomos, del uso de técnicas de Inteligencia Artificial para la navegación de rutas y control de VA, se identifica las necesidades de implementación y problemas a resolver en ciertas tareas, y se presenta un esquema de trabajo en donde se muestran los principales contenidos a abordar en este tema de investigación.

En la segunda sección, se presentan y describen las metodologías usadas para la realización de las tareas asignadas implementando algoritmos de Aprendizaje por Refuerzo Profundo, además de describir la arquitectura para vehículos autónomos.

En la tercera sección, se detallan cuáles son los objetivos a alcanzar en este trabajo de investigación, y también se describe la metodología a usar para el desarrollo de cada una de las fases descritas.

En la cuarta sección, se describe los siguientes elementos: arquitectura de un sistema de VA con DRL, herramientas software existentes, estado actual del sector de VA, y los cálculos requeridos para la conducción autónoma en vehículos.

En la quinta sección, se describe el sistema VA que va a ser implementado en simulación, y se detalla el diseño que fue realizado para cumplir con el objetivo de trabajo.

En la sexta sección, se integra todos los elementos de diseño, herramientas software y simulación para demostrar el funcionamiento del sistema VA con DRL.

En la séptima sección, se presentan y describen los resultados obtenidos de las pruebas realizadas en simulación.

En la octava sección, se presentan las conclusiones obtenidas de todo el proceso de diseño e implementación en simulación y se detallan las líneas de trabajo futuras que se realizaran en base a los resultados obtenidos en esta investigación.

2. Contexto y estado del arte

2.1. Descripción general del contexto del proyecto

En este apartado se describe las técnicas, herramientas y metodologías del proyecto que se va a implementar, ofreciendo una solución tecnológica aplicando inteligencia artificial enfocada al área de conducción autónoma, concretamente a la aplicación de algoritmos de Aprendizaje por Refuerzo Profundo en la capa de control de un vehículo ofreciendo autonomía de control y mejora del rendimiento del agente en entornos desconocidos donde se haya producido algún desastre natural o provocado por el ser humano.

La importancia de integrar este habilitador tecnológico a los vehículos autónomos brindará mejorar las tareas de búsqueda y rescate, y también ampliar la aplicabilidad de esta tecnología en el sistema de transporte actual del país de Ecuador, el cual posee una geografía muy variada y también requiere optimizar varios aspectos tecnológicos y políticos.

2.1.1. Vehículos autónomos

Dentro de la comunidad académica y de automoción ha crecido la atención hacia las técnicas de conducción, esto debido a las grandes mejoras en tecnologías informáticas, aprendizaje automático y comunicaciones. La tarea principal de la conducción autónoma es hacer que el vehículo realice acciones de control correctas en tiempo real en un entorno dado. La conducción autónoma tiene un diseño constituido por varios tipos de sensores y dispositivos de computación, comunicación, almacenamiento, administración de energía, además de integrar algoritmos de aprendizaje profundo para la detección de objetos/carriles, localización y mapeo simultaneo (SLAM) (Shi & Liu, 2021).

El sistema informático es parte elemental del proceso de conducción autónoma, existen dos tipos de diseño para vehículos autónomos: modulares y de extremo a extremo. El diseño modular define como módulos separados a las tareas de localización, percepción, control del vehículo, planificación de misiones y movimiento, etc. El diseño basado en un extremo a otro está motivado por el desarrollo de la inteligencia artificial, este sistema está basado en técnica de aprendizaje automático para procesar datos de los diferentes sensores y generar comandos de control para el vehículo (Bojarski et al., 2016).

El sistema informático para vehículos de conducción autónoma está conformado por: computación, comunicación, almacenamiento, seguridad, privacidad y administración de energía. Existen cuatro capas que constituyen este sistema: sensores, sistema operativo, middleware y aplicaciones (Shi & Liu, 2021).

2.1.2. Categorización de Vehículos

Actualmente los vehículos “autónomos” comercializados en los mercados están en el nivel dos. La Sociedad de Ingenieros Automotrices (SAE) (SAE, 2021), ha establecido seis niveles de automatización para vehículos autónomos. El nivel más bajo, es el Nivel 0 (sin automatización) es donde el conductor realiza todas las tareas de conducción y la tecnología solo se limita a proporcionar advertencias. En el nivel 1, los vehículos ofrecen asistencia al conductor como el control de crucero adaptativo o el centrado de carril. En el nivel 2, existe una automatización parcial del vehículo donde hay una combinación entre control de crucero adaptativo y sistemas de frenado de emergencia. En el nivel 3, existe la automatización condicional, permite que el vehículo controle el entorno de conducción, pero se requiere que el conductor controle el sistema cuando sea solicitado. En el nivel 4, no es necesario el control por parte del conductor, debido que el automóvil es capaz de interpretar el entorno y en base a esto tomar decisiones para la conducción bajo ciertas condiciones, el conductor toma el control cuando el desee o cuando el sistema falle debido al estado de las vías. El nivel 5 ofrece una automatización completa, lo que significa que el sistema debe funcionar en todo tipo de circunstancias como en entorno cambiantes o impredecibles (Van Uytsel & Vasconcellos Vargas, 2021).

Tabla 1: Niveles de automatización de conducción.

Nivel	Descripción	Actores
0	**Sin automatización:** El humano controla de forma continua las funciones de conducción (dirección, aceleración freno).	**Niveles del 0 al 2:** El conductor humano supervisa y controla el entorno de conducción.
1	**Con asistencia:** El humano conduce el vehículo, pero se proporcionan de control de dirección, freno, aceleración y seguridad.	
2	**Automatización parcial:** Proporcionan capacidades estándar de dirección, freno o aceleración, pero el conductor todavía tiene el control y monitorea el resto de tareas de conducción.	
3	**Automatización condicional:** El vehículo puede funcionar de forma independiente. Las funciones de control están automatizadas pero el conductor aún puede intervenir en la conducción.	**Niveles del 3 al 5:** Existe un sistema de conducción autónoma que supervisa el entorno de conducción.
4	**Automatización alta:** El vehículo tiene todas las funciones de control de conducción en circunstancias específicas. El conductor tiene la opción de control en cualquier momento.	
5	**Automatización completa:** Todas las tareas de conducción están completamente automatizadas y se las realiza sin condiciones y en cualquier parte. La intervención humana no es necesaria.	

Fuente: Elaboración a partir de (SAE, 2021).

2.1.3. Aprendizaje Automático

En la conferencia de Dartmouth realizada en 1956, John McCarthy acuño por primera vez el termino de Inteligencia Artificial. A partir de este año se inició la IA como un campo de la informática en la cual se desarrollaron algoritmos para resolver problemas que pueden formularse mediante reglas matemáticas y lógicas (Dong et al., 2020).

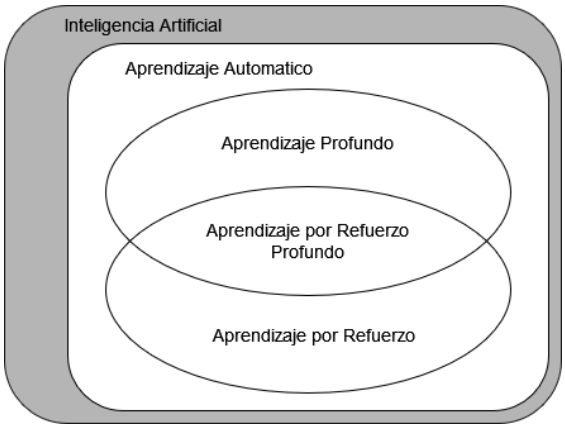


Figura 1: Relación de enfoques de inteligencia artificial.

Fuente: Elaboración propia, a partir de (Dong et al., 2020)

La inteligencia artificial es un campo general que abarca diversos enfoques, en sus inicios la IA simbólica demostró ser útil para resolver problemas lógicos (jugar al ajedrez) surgieron limitaciones a la hora de resolver problemas más complejos como clasificación de imágenes y los nuevos enfoques como aprendizaje automático facilitaron la resolución de estos nuevos retos. Con la introducción de aprendizaje automático, la maquina analiza los datos de entrada y las respuestas correspondientes, este sistema de aprendizaje se entrena en lugar de programarse (Chollet, 2021).

2.1.3.1. Aprendizaje Profundo

El aprendizaje profundo es un subcampo especifico del aprendizaje automático: un nuevo marco matemático de aprendizaje a partir de datos con énfasis en capas sucesivas de aprendizaje con representaciones más significativas. El termino profundo viene de capas sucesivas de representaciones. El aprendizaje profundo moderno implica decenas o cientos de capas sucesivas que aprenden automáticamente a partir de la exposición a los datos de entrenamiento. Los modelos para este enfoque se llaman redes neuronales, estructuradas en capas apiladas una encima de la otra (Chollet, 2021).

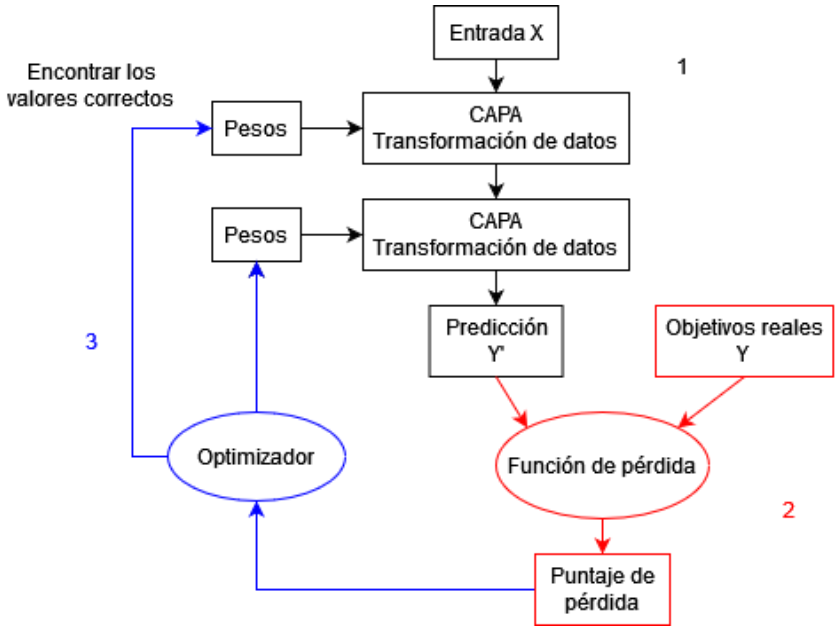


Figura 2: Red neuronal parametrizada por pesos.

Fuente: Elaboración propia, a partir de (Chollet, 2021)

En la Figura 2, se muestra el funcionamiento del aprendizaje profundo: 1) Las redes neuronales profundas hacen el mapeo de entrada a través de una secuencia de transformaciones de datos simples (capas) y estas transformaciones de datos se aprenden mediante la exposición a ejemplos. Las transformaciones que implementa una capa esta parametrizada por sus pesos, lo que significa que aprender es encontrar un conjunto de valores para los pesos de cada capa de una red. 2) Para controlar la salida de la red neuronal, se debe medir que tan lejos esta la salida de lo que se esperaba, para realizar esto entra en acción la función de pérdida de la red (función costo) quien toma las predicciones de la red y el verdadero objetivo y calcula una puntuación de distancia, capturando que tan bien le ha ido a la red. 3) Este puntaje es usado como señal de retroalimentación para el ajuste de los valores de los pesos, este ajuste es llamado como algoritmo de *Backpropagation*. Inicialmente los pesos de la red están asignados con valores aleatorios, la red implementa una serie de transformaciones aleatorias, con cada ejemplo que procesa la red, los pesos se ajustan en la dirección correcta y la puntuación de perdida disminuye. Este proceso se conoce como ciclo de entrenamiento, una red con una perdida mínima es aquella que los resultados están lo más cercano posible de los objetivos (red entrañada) (Chollet, 2021).

2.1.3.2. Aprendizaje por Refuerzo

Para construir un buen sistema IA inteligente, no solo se requiere que la IA pueda aprender de los datos proporcionados, sino también de las interacciones con el entorno real.

Reinforcement Learning (RL) se compone principalmente de dos componentes: un entorno y un agente. El agente interactúa con el entorno realizando varias acciones y recibe retroalimentación del entorno, o llamado también “recompensa”. El agente aprende a desempeñarse mejor obteniendo estas recompensas, todo este proceso forma un circuito de retroalimentación entre el agente y el entorno (Dong et al., 2020).

El aprendizaje por refuerzo permite que el agente software tome las medidas adecuadas para maximizar la recompensa en un entorno dado. El agente intenta diferentes acciones para explorar nuevos conocimientos y aprende por ensayo y error, obteniendo como resultado una recompensa o penalización. En este proceso aparece la política la cual define el comportamiento del agente, lo que busca es encontrar aquellas acciones en el presente conducirán al final con la mejor rentabilidad (Gupta et al., 2021).

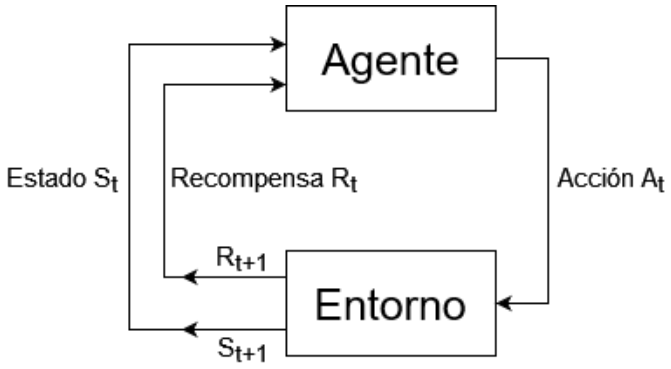


Figura 3. Proceso de aprendizaje por refuerzo.

Fuente: (Elaboración propia)

En la Figura 3, se describe el proceso de aprendizaje, esto sucede cuando el agente genera una política, inicialmente el agente se encuentra en un estado ‘s’ del entorno, en el cual se realiza una acción ‘a’. Después de actuar, el agente entra en otro estado ‘s’ del entorno y obtiene una recompensa ‘r’, este es un proceso de retroalimentación. El agente repite muchas veces el ciclo para aprender (Gupta et al., 2021). Existe otro componente importante en este proceso, el cual es el espacio de acción que define la colección de opciones disponibles que el agente puede realizar en un estado. Existen dos tipos de espacios de acción:

- Espacio de acción discreta: Se refiere a los sistemas en los que el espacio de acción es finito, pequeño y contable.
- Espacio de acción continua: se refiere al sistema en el que la acción puede tomar cualquier tipo de valor dentro de un rango en específico. Un ejemplo de esto, es la navegación, en donde se incluye el control del ángulo y la velocidad del movimiento.

Para facilitar la visión general del panorama del enfoque de aprendizaje por refuerzo, se resume en la Figura 4 la taxonomía de los algoritmos RL; la clasificación se la realiza de diferentes perspectivas que incluyen métodos basados en modelos, sin modelos, valores y políticas o la combinación de ambos, métodos de Monte Carlo, métodos de diferencia temporal, métodos dentro y fuera de la política.

En el aprendizaje profundo dada una red neuronal profunda, un "modelo" representa una función específica que contiene parámetros inicializados (modelo pre-entrenado) o parámetros aprendidos (modelo entrenado). Mientras que el aprendizaje por refuerzo basado en modelos, significa que un modelo es un conjunto de conocimientos adquiridos realizando acciones dentro de un entorno. Para los algoritmos basados en el "sin modelo", el algoritmo no se centra en el modelo, sino que buscan directamente la recompensa más alta. La diferencia entre los algoritmos basados en "modelo" y "sin modelo", es que si el agente aprenderá la dinámica del entorno como las funciones de transición y de recompensa (Dong et al., 2020).

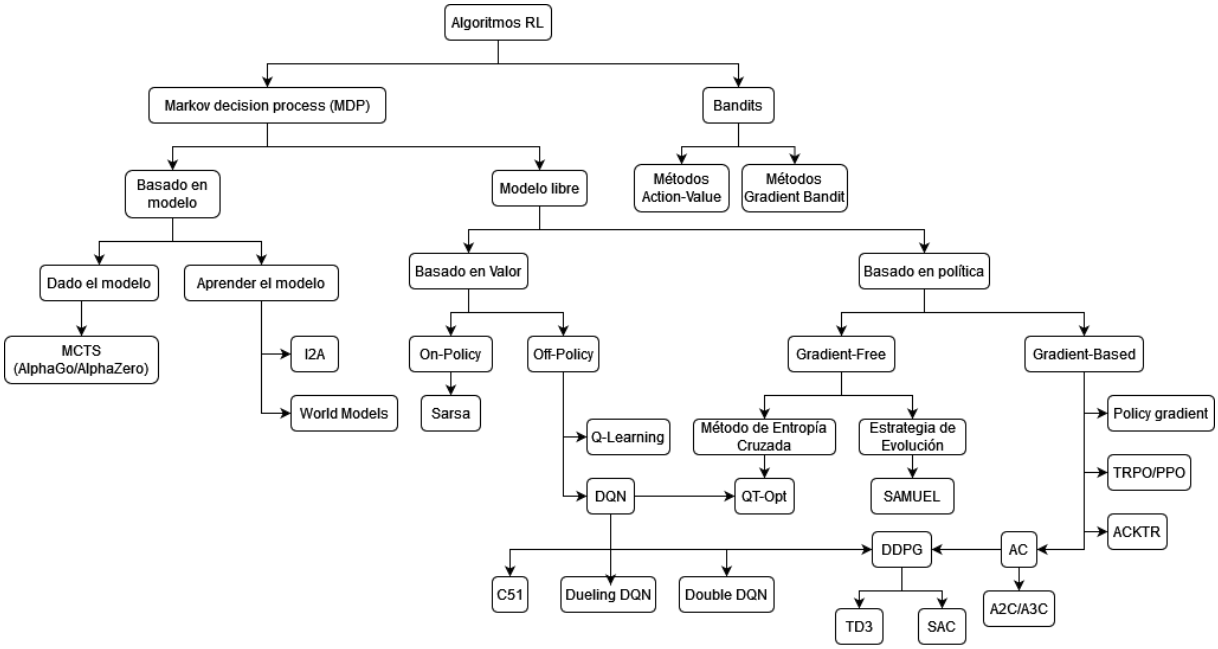


Figura 4: Mapa de algoritmos por refuerzo.

Fuente: Elaboración propia, a partir de (Dong et al., 2020).

Los métodos basados en modelos se pueden dividir en: métodos que funcionan con un modelo dado. Donde el agente puede acceder directamente a los modelos para la función de recompensa y proceso de transición. Los métodos que aprenden el modelo, no pueden adquirir el modelo directamente debido a la complejidad o la opacidad del entorno, pero el

agente aprende primero un modelo de las interacciones con el entorno y luego aplica el modelo en la mejora de políticas.

Hay dos categorías principales para la optimización de políticas en el aprendizaje de refuerzo profundo, los métodos basados en valores y los métodos basados en políticas. Una combinación de los dos genera la clase de algoritmos actor-crítico y otros algoritmos como QT-Opt. Los métodos basados en valores generalmente implican la optimización de la función acción-valor $Q\pi(s,a)$. El método basado en políticas optimiza la política directamente; actualiza la política iterativamente hasta que se maximiza el rendimiento acumulativo. En comparación con el método basado en valores, un método basado en políticas tiene las ventajas de una parametrización de políticas más simple, una mejor convergencia y es adecuado para un espacio de acción continuo o de gran dimensión. Los métodos dentro de la política intentan evaluar o mejorar la política que se utiliza para tomar decisiones, mientras que los métodos fuera de la política evalúan o mejoran una política diferente a la utilizada para generar los datos (Dong et al., 2020).

2.1.3.3. Aprendizaje por Refuerzo Profundo

El aprendizaje por refuerzo profundo es la combinación entre DL y RL para crear sistemas de IA, aprovechando la escalabilidad de DNN en un espacio de alta dimensión. En donde la arquitectura de DL actúa como un aproximador de funciones para manejar datos de alta dimensión (Gupta et al., 2021). Esta combinación permite aplicar métodos clásicos de aprendizaje por refuerzo a entornos más complejos que contienen un estado muy grande o incluso infinito y no pueden crear una política efectiva para un agente (Gridin, 2022). Las principales tareas involucradas en este sistema son:

- Exploración/Explotación: La exploración se refiere en intentar una nueva acción, mientras que la explotación hace uso del conocimiento aprendido para la toma de decisiones.
- Generalización: Se refiere a la capacidad del agente para adaptarse a un nuevo entorno.
- Búsqueda de políticas: Identifica los estados y acciones realizadas por el agente para conocer una política óptima para la toma de decisiones.
- Encontrar eventos catastróficos que podrían causar daño, evitando estos eventos y mejorando la política.

Como ya se presentó en la Figura 4, se presentó un resumen de la taxonomía y categorías de los algoritmos de aprendizaje por refuerzo. En la siguiente Tabla 2, se muestra de

manera general algunos ejemplos de algoritmos de DRL típicos junto con su espacio de acción (continuo/discreto) (Dong et al., 2020):

Tabla 2: Algoritmos para Deep Reinforcement Learning.

N	Algoritmo DRL	Espacio de acción
1	Deep Q-Network (DQN)	Discreto
2	Reinforce	Discreto/continuo
3	Reinforce con línea base.	Discreto/continuo
4	Actor-Critico	Discreto/continuo
5	Q-value Actor-Critic (QAC)	Discreto/continuo
6	Advantage Actor-Critic (A2C)	Discreto/continuo
7	Asynchronous Advantage Actor-Critic (A3C)	Discreto/continuo
8	Deep Deterministic Policy Gradient (DDPG)	Continuo
9	Twin Delayed DDPG (TD3)	Continuo
10	Soft Actor-Critic (SAC)	Discreto/continuo
11	Trust Region Policy Optimization (TRPO)	Discreto/continuo
12	Proximal Policy Optimization (PPO-Penalty)	Discreto/continuo
13	Proximal Policy Optimization (PPO-Clip)	Discreto/continuo
14	Actor Critic using Kronecker Trust Region (ACKTR)	Discreto/continuo

Fuente:(Dong et al., 2020)

2.2. Proyectos relacionados con conducción autónoma

A continuación, se presentan algunos proyectos realizados por varios autores que contienen desarrollo de sistemas de conducción autónoma y aplicaciones de búsqueda y rescate.

2.2.1. Proyecto 1

Robot de aprendizaje de refuerzo profundo para aplicaciones de búsqueda y rescate: exploración en entornos desordenados desconocidos.

En este articulo realizado por (Niroui et al., 2019) se presenta un robot de rescate que es utilizado para aplicaciones de búsqueda y rescate urbano (USAR, con siglas en inglés) con la tarea de explorar entornos desordenados y desconocidos. El aprendizaje profundo es usado para la tarea de exploración basada en frontera en combinación con el aprendizaje de refuerzo profundo para que el robot explore de manera autónoma los entornos desconocidos. Se realizaron experimentos con un robot móvil en entornos de diferentes

tamaños y diseños, de tal manera que mostraron que el enfoque de exploración expuesto puede cumplir de manera efectiva la navegación.

La arquitectura de exploración propuesta está basada en el aprendizaje que se muestra en la Figura 5. Se genera un mapa con cuadricula de ocupación 2D utilizando información de profundidad del sensor 3D y la información de edometría del robot. El módulo de exploración utiliza datos de edometría para determinar una ubicación fronteriza para explorar. Esta ubicación fronteriza se envía al módulo de navegación que calcula la ruta de navegación para el robot que se implementa el controlador de bajo nivel.

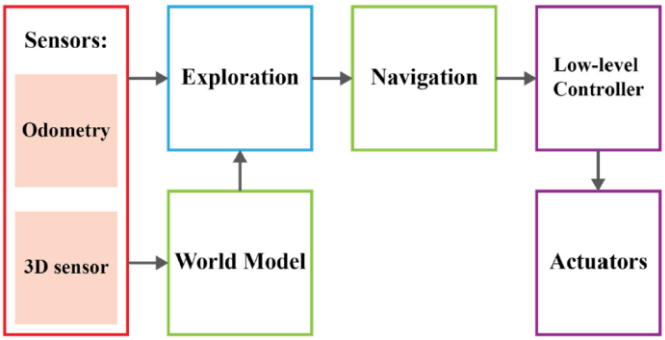


Figura 5: Arquitectura del robot de exploración para USAR.

Fuente: (Niroui et al., 2019)

Para comprobar la viabilidad de la arquitectura propuesta por los autores se llevó a cabo tres conjuntos de experimentos: 1) prueba de rendimiento y escalabilidad para el enfoque de exploración de fronteras, 2) comparación del aprendizaje propuesto de varias técnicas de exploración de fronteras y 3) despliegue del enfoque en un robot móvil físico que realiza la tarea de exploración en un entorno desordenado y desconocido.

2.2.2. Proyecto 2

Control basado en aprendizaje de refuerzo profundo para Vehículos Autónomos en CARLA.

En este artículos los autores (Pérez-Gil et al., 2022), implementan algoritmos DRL como Deep Q-Network (DQN) y Deep Deterministic Policy Gradiente (DDPG) en la capa de control de un vehículo autónomo y comprueban el resultado en entre ellos. Para la demostrar la navegación del vehículo con comandos de control utilizan el simulador de código abierto CARLA, que le permite realizar una gran variedad de pruebas sin riesgo en un entorno de simulación. Los resultados muestran que DQN y DDPG alcanza la meta, pero este último presenta mejor desempeño. En ambos casos, el valor de RMSE es inferior a 0.1 siguiendo trayectorias con un rango de 180 a 700m.

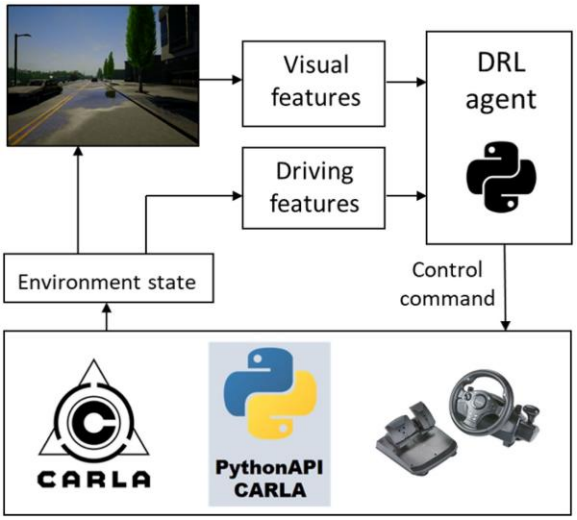


Figura 6: Descripción general del marco de trabajo.

Fuente: (Pérez-Gil et al., 2022)

En la Figura 6, se muestra la descripción general del marco que fue desarrollado, el objetivo es seguir una ruta predeterminada lo más rápido posible evitando colisiones y salidas de carretera en un entorno urbano. Fueron implementados en el simulador de CARLA y se realizó la comparación de los algoritmos DQN y DDPG, posteriormente se analizaron las limitaciones para la navegación de VA y luego decidir cuál puede transferirse a un vehículo real.

2.2.3. Proyecto 3

Algoritmo de agente DQN basado en el aprendizaje de refuerzo profundo para el seguimiento visual de objetos en una simulación ambiental virtual

En este articulo (Park et al., 2022), se presenta un marco de seguimiento de objetos mediante la simulación de un entorno virtual (AirSim, City Environ) usando uno de los modelos de aprendizaje por refuerzo profundo denominado Algoritmo de Aprendizaje Profundo Q. La red propuesta en este trabajo examina el entorno utilizando un modelo DRL para regular las actividades en el entorno de simulación virtual y utiliza imágenes secuenciales del modelo (Virtual City Environ) como entradas. El modelo se entrenó usando múltiples conjuntos de imágenes de entrenamiento secuencial y fue adaptado durante el seguimiento del tiempo de ejecución. En la Figura 7, se muestra el marco general de la propuesta del esquema integrado de la plataforma AirSim con un entorno virtual.

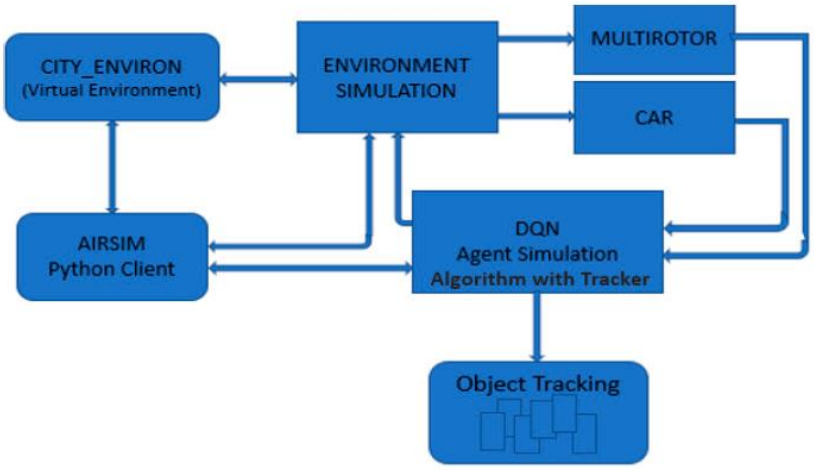


Figura 7: Marco general de implementación de AirSim con modelo DQN.

Fuente: (Park et al., 2022)

La red propuesta está conectada con un entorno virtual que permite obtener una secuencia de fotogramas de video en tiempo de ejecución y ubicar objetos en cada imagen del episodio, el algoritmo implementado utiliza Recurrent Neural Network (RNN) para aprender y predecir a partir de secuencias temporales. La plataforma de simulación incluye dos tipos de opciones Multirotor y Vehículo para probar el enfoque conectando AirSim Python Client.

2.3. Tecnologías y sistemas aplicadas a conducción autónoma

A continuación, se resume las tecnologías clave del estado de arte para automóviles con conducción autónoma.

2.3.1. Sensores

En la capa de sensores encontramos varios sensores que pueden ser integrados en un vehículo autónomo: cámara, LIDAR, radar, GPS/GNSS, ultrasónico, acelerómetros, etc. Dichos sensores capturan datos del entorno en tiempo real para que puedan ser procesados dentro del sistema de conducción (Shi & Liu, 2021).

- **Cámara:** Este tipo de sensores son los más usados en los vehículos de conducción autónoma, el alcance de la cámara varia desde unos cuantos centímetros hasta varios metros, el costo es relativamente bajo y contribuye al despliegue en un vehículo de conducción autónomo. Este sensor puede verse afectado por la mala iluminación, malas condiciones climáticas como la lluvia y nieve. Un dato importante es que la cámara puede producir entre 20-40 Mb de datos en cada segundo.
- **Radar:** La técnica que utiliza este sensor es el medir el tiempo de vuelo y calcula la distancia y velocidad, la frecuencia de trabajo del sistema de radar del vehículo es de

24 GHz o 77 GHz. Para radares con frecuencia de 24 GHz el alcance máximo de detección es de 70 m; mientras que, para radares de 77GHz aumenta el alcance a 200 m, tiene menos interferencia, mayor precisión en la detección y su tamaño es más pequeño. A diferencia de una cámara, el clima y la ilumina afectan muy poco al radar, por lo que lo hace muy útil en detección de objetos y estimación de distancia. El radar produce datos con un tamaño de 10 a 100 KB por segundo.

- **LIDAR:** Su funcionamiento se basa en Time of Flight (TOF) para la obtención de información de distancia, este sensor consta de un generador laser y un receptor laser de alta precisión. Este sensor con la información obtenida genera una imagen tridimensional de los objetos estáticos y en movimiento escaneados a su alrededor. Trabaja dentro de un rango de varios centímetros hasta 200 m, y es usado en aplicación de detección de objetos, estimación de distancias, detección de bordes y generación de mapas de alta definición. Este sensor es capaz de generar 10-70 MB de datos por segundo.
- **Sensor ultrasónico:** Este sensor se basa en el fenómeno del ultrasonido para la detección y medición de distancias. El cálculo del parámetro de la distancia se lo realiza mediante la medición TOF. El rendimiento de este sensor es bueno en condiciones climáticas adversas y entornos con poca iluminación, además de ser un dispositivo mucho más económico. Una de las desventajas del uso de sensores ultrasónicos es que el alcance máximo es de 20 m. El tamaño de datos que genera este sensor es de 10 – 100 KB.

2.3.2. Sistemas operativos

En el sistema operativo, existen controladores que funcionan como puentes entre el software y dispositivos del hardware; proporcionando interfaz de comunicación, gestión de todos los recursos y satisfaces requisitos de seguridad. Los sistemas operativos en tiempo real son un tema de interés en el diseño e implementación de sistemas de conducción continúa debido que se requiere que el vehículo pueda preceptuar el entorno y tome la decisión correcta y en tiempo real para trazar rutas.

El RTOS se usa en el sistema integrado de la Unidad de Control del Motor para controlar el acelerador, el freno, etc. del vehículo. Dentro del mercado existen dos RTOS comercializados, QNX y VxWorks, el primero contiene programación de CPU, comunicación entre procesos, redirección de interrupciones y temporizadores y el segundo está diseñado para sistemas integrados que requieren un rendimiento determinista en tiempo real y permiten algunas arquitecturas como INTEL, POWER y ARM (Shi & Liu, 2021).

2.3.3. Sistemas intermedios

La capa intermedia ofrece la facilidad de uso y programabilidad para el desarrollo y mejora de sistemas, como comunicación, sincronización de tiempo y colaboración de múltiples sensores. En sistemas de vehículos autónomos están involucrados múltiples servicios lo cual se requiere un sistema intermedio que permita la comunicación entre los diferentes servicios. ROS es una de las soluciones de sistemas intermedios que facilita la comunicación entre los diferentes módulos que contiene un sistema de vehículo autónomo, el cual admite cuatro métodos: tema, servicio, acción y parámetro. Algunos ejemplos de sistemas intermedios son: Autoware, software de código abierto para tecnología de conducción autónoma. Apollo Cyber, también de código abierto y cuyo objetivo es acelerar el desarrollo, pruebas y despliegue de vehículos autónomos (Shi & Liu, 2021).

2.3.4. Aplicaciones de conducción autónoma

En esta capa se implementan aplicaciones como la detección de objetos/carriles, SLAM, predicción, planificación y control de vehículos, generación de comandos. En esta parte se presentan algunas aplicaciones destacadas en sistemas de vehículos autónomos.

2.3.4.1. Detección de objetos

La detección es una parte muy importante para aplicaciones de aprendizaje profundo en el mundo real de vehículos autónomos. Existen dos fases que intervienen en desarrollo de algoritmos de detección de objetos, fase de detección convencional y fase de detección basada en aprendizaje profundo. Para la primera fase se encuentran los algoritmos tradicionales de detección de objetos: detectores VIOLA Jones, histograma de gradientes orientados (HOG) y el modelo basado en piezas deformables (DPM).

En cuanto al enfoque de la segunda fase usando aprendizaje profundo se incluyen: Regiones con características CNN (RCNN), Single Shot MultiBox Detector (SSD) y You Only Look Once (YOLO). En 2014, Girshick introdujo el aprendizaje profundo en el campo de la detección de objetos al proponer Fast RCNN y Faster RCNN para acelerar la velocidad de detección.

2.3.4.2. Detección de carril

Esta función es importante para los sistemas avanzados de asistencia al conductor (ADAS) ya que les permite a los vehículos autónomos conducir dentro de los carriles de las carreteras, permitiéndoles evitar colisiones, respaldar las decisiones de planificación de trayectorias y salida de carril. Los principales enfoques para la detección de carrieles implementados están en función de características de colores, tensores de estructura, filtro

de barra y de cresta. La información obtenida se combina con la Transformada de Hough y filtros de partículas o de Kalman, posteriormente se utilizan métodos de posprocesamiento para filtrar detecciones erróneas y generar los resultados finales de detección de carriles.

Recientemente, varios enfoques han surgido en el campo de detección de carril basados en el aprendizaje profundo. Por ejemplo, VPGNet propone una red multitarea para la detección de marcas de carril, SCNN utiliza la información visual aplicando una operación de convolución que agrega información de diversas dimensiones mediante el procesamiento de características divididas y luego las suma.

2.3.4.3. Localización y mapeo

La localización permite encontrar la posición del vehículo autónomo en relación con el mapa, mientras que el mapeo construye mapas multicapa en alta definición para la planificación de rutas. Para mejorar la precisión requerida en la conducción autónoma, actualmente se utiliza mapas HD prefabricados que son más prácticos y precisos, existen tres tipos:

- Basados en puntos de referencia como postes, bordillos, señales y marcadores de carreteras, que pueden ser detectados con sensores como LIDAR o cámaras.
- Basados en nubes de puntos, contiene información detallada sobre el entorno con miles de puntos, en este enfoque se usa los algoritmos de Punto más cercano iterativo (ICP) y la Transformación de distribuciones normales (NDT) para la generación de mapas HD.
- Basados en visión, es una de las técnicas más populares pero la sobrecarga computacional es muy elevada para sistemas reales, por lo que se propone métodos para hacer coincidir los mapas con la cámara 2D para realizar el mapeo.

2.3.4.4. Predicción y Planificación

En el módulo de la predicción se realiza la evaluación del comportamiento del VA y de los elementos existentes en el entorno para medir los riesgos existentes. Con la planificación su espera encontrar rutas factibles en el mapa desde el origen hasta el destino. Existen los planificadores globales como los sistemas de navegación GPS y los planificadores locales que se dividen en tres grupos:

- Planificadores basados en gráficos que ofrecen la mejor ruta hacia el destino.
- Planificadores basados en muestreo que escanean aleatoriamente todo el entorno.
- Planificadores de curvas de interpolación que se proponen para suavizar el camino.

2.3.4.5. Control de vehículos

En este módulo se integra la interconexión del sistema software de conducción autónoma con el resto de sistemas del vehículo. Aquí es donde se realiza el ajuste del ángulo de dirección y control de velocidad para las trayectorias encontradas en el módulo de planificación; sin embargo, el planificador no considera el estado en el que se encuentran las rutas, los caminos pueden estar accidentados, con curvas y el tipo de suelo puede tener grava, arena y charcos de lodo. Los comandos de salida de control se calculan a partir del estado del vehículo y la trayectoria por ley de control, como controladores difusos, PID, Stanley y el control predictivo de modelo (MPC).

2.3.5. Almacenamiento

Con la integración de nuevos sistemas hardware y software a la conducción autónoma se generan una gran cantidad de datos, como ya se mencionó los sensores serían las principales fuentes de datos. El manejo de datos eficiente y segura mejora el rendimiento general del sistema. Existen varios problemas relacionados con el almacenamiento para vehículos conectados y autónomos (CAV), la plataforma de análisis de datos OpenVDAP, consta de un diseño jerárquico del sistema de almacenamiento denominado integrador de datos de conducción (DDI), el cual proporciona almacenamiento y procesamiento de datos con reconocimiento de sensores y aplicaciones (Zhang et al., 2018).

2.3.6. Comunicación vehicular

Recientemente se han integrado varios mecanismos a los sistemas de vehículos de conducción autónoma que permiten la comunicación y obtención de información de otros vehículos, por ejemplo:

- DSRC: Protocolo de comunicación V2X diseñado para vehículos conectados, basado en el estándar IEEE 802.11p, cuya frecuencia de trabajo es de 5.9 GHz. Se definen algunos mensajes DSRC en el estándar SAE J2735 que ofrecen información: posición del vehículo, información del mapa, advertencia de emergencia, etc.
- C-V2X: Es una combinación de la red V2X tradicional con la red celular, ofrece asistencia de red y servicios comerciales de 4G/5G, la frecuencia de operación también es de 5.9 GHz, este mecanismo de comunicación es el más adecuado en lugares donde las redes celulares están ampliamente implementadas.
- LTE 4G y 5G: Las comunicaciones de cuarta generación (4G) cumplen con 1 Gbit/s para recepción estacionaria y 100 Mbit/s para móvil. 5G todavía tiene varios desafíos, sistema complejo, altos costos y poca capacidad para evitar obstáculos.

2.3.7. Seguridad y privacidad

Con la integración de una gran variedad de tecnologías de comunicación, almacenamiento de datos, técnicas de inteligencia artificial, sistemas operativos, se requiere que la seguridad sea integral en los sistemas de los vehículos autónomos. Algunos desafíos que se consideran en los VA son los siguientes: seguridad de los sensores, ataques de interferencia, ataques de suplantación de identidad. Existen problemas de ataques falsificados para los sistemas de GPS, donde se debe autentificar las fuentes de los datos y protección para los sensores LIDAR y Radar. La seguridad en las comunicaciones, incluye dos aspectos la comunicación interna y la externa. La tecnología de la criptografía es usada para mantener los datos transmitidos confidenciales, integrados y autentificados, sin embargo, está limitado por el alto costo computacional de las ECU con recursos limitados.

Se ha propuesto un marco de control de acceso para que los VA protejan los datos de manera sistemática en tiempo real y los datos históricos, el AC4AV, proporciona una API para ajustar dinámicamente los modelos de control de acceso, también proponen un método de abstracción de datos para identificar los datos de las aplicaciones y operación de acceso en CAV, configurando los permisos de los datos y aplicación en las políticas de control (Zhang et al., 2020).

2.4. Conclusiones sobre el estado del arte

En el estado de arte se ha presentado varios puntos de interés y relevantes del estado actual del desarrollo de sistemas de conducción vehicular, se presentó los pro y contra de la integración de tecnologías emergentes para modernizar los sistemas de transporte en la actualidad y el surgimiento de varios retos que se deben enfrentar para realizar un despliegue eficiente de vehículos autónomos.

Se describe la definición y explica el funcionamiento de la técnica de Aprendizaje por Refuerzo Profundo (DRL) para ser integrada en la capa de control de un sistema de conducción autónoma para vehículos que serán utilizados para misiones de búsqueda, rescate y transporte y en los que se requiere que sea capaz de interactuar con entornos desconocidos y difícil acceso para que lleguen en el menor tiempo posible a su destino.

Se detalla a profundidad las tecnologías y sistemas involucrados en los procesos de vehículos autónomos, que permitirán aumentar el nivel de automatización en la tarea conducción de acuerdo a la normativa SAE que establece seis niveles. Así mismo se describen las principales aplicaciones de VA que ayudan de punto de partida para la realización del diseño del sistema de conducción autónoma.

Con lo expuesto en el estado del arte, se puede tener claro que es lo que se requiere para implementar de manera correcta un sistema de conducción autónoma en una herramienta software para la simulación que permita demostrar la aplicabilidad y el funcionamiento de algoritmos DRL.

3. Descripción general de la contribución del TFM

3.1. Objetivos

Objetivo general

- El objetivo de este proyecto es la implementación en software de un algoritmo de Aprendizaje por Refuerzo Profundo en la capa de control de un vehículo de conducción autónoma con el fin que este sistema aporte a futuros sistemas de conducción autónoma en las tareas de rescate o apoyo en situaciones de desastres naturales. De esta manera se pretende dar solución a los retos de funcionalidad de control como la navegación, otorgando un sistema que sea capaz de realizar envíos de equipos, herramientas, alimentos, medicinas a lugares remotos o sirva como medio de transporte inteligente. Como tarea de aplicación y demostración se implementa un agente con inteligencia artificial, usando técnicas DRL (redes neuronales convolucionales y algoritmo DQN) en el entorno software de simulación AirSim.

Objetivos específicos

En este proyecto se pretende cumplir con los siguientes objetivos específicos:

- Realizar un estudio sobre vehículos de conducción autónoma, detallando las tecnologías implementadas, así como librerías que aplican técnicas de inteligencia artificial y plataformas de simulación que son usadas para estos sistemas.
- Diseñar un sistema de conducción autónoma que sea capaz de visualizar y comprender el entorno en el que encuentra y realice las acciones de control necesarias para navegar correctamente en una carretera.
- Evaluar el funcionamiento del modelo desarrollado en una plataforma de simulación para la ejecución de la tarea propuesta.
- Presentar un sistema de conducción autónoma que permita mejorar los sistemas de transporte existentes, e integrando nuevas tecnologías a los vehículos que realizan tareas de búsqueda y rescate.

3.2. Metodología del trabajo

La metodología propuesta para la gestión de este proyecto de investigación es usar el marco tradicional Cascada (Waterfall) (UNIR, 2020), esta desarrollada en fases de manera secuencial que permitirá sistematizar todo el proceso de desarrollo, de esta manera se busca cumplir con los objetivos planteados en cada etapa. Este método se divide en los siguientes procesos:

- **Inicio:** Se definió y propuso un tema de TFM de investigación en el área de vehículos autónomos aplicando técnicas de inteligencia artificial, estableciendo objetivos a cumplir, limitaciones y alcance del proyecto, finalmente siendo aprobado el tema.
- **Planificación:** Se estableció un plan de gestión para la presentación de tres avances del proyecto (borradores del TFM), en los cuales se mostraba los resultados del desarrollo del informe y software del proyecto. Así mismo, en cada reunión con el tutor se determinaron las tareas que deben realizarse en el desarrollo del proyecto.
- **Ejecución:** Para completar las entregas establecidas en las fechas dadas, se realizaron las tareas asignadas como la investigación de DRL, herramientas software disponibles, así como el desarrollo del sistema de VA e informe.
- **Monitorización y control:** En las reuniones establecidas con el tutor se supervisan, verifican las tareas ejecutadas y se revisa la planificación establecida, estableciendo correcciones y recomendaciones para mejorar la realización del proyecto.
- **Cierre:** Cumplir con las tareas asignadas en cada fase del proyecto, en donde se presenta el informe y los resultados de la implementación del sistema.

Dentro de las fases de ejecución, monitorización y control se realizaron un conjunto de tareas que fueron establecidas para las entregas de las fechas dadas. Se propuso el siguiente esquema de trabajo para el desarrollo de investigación.

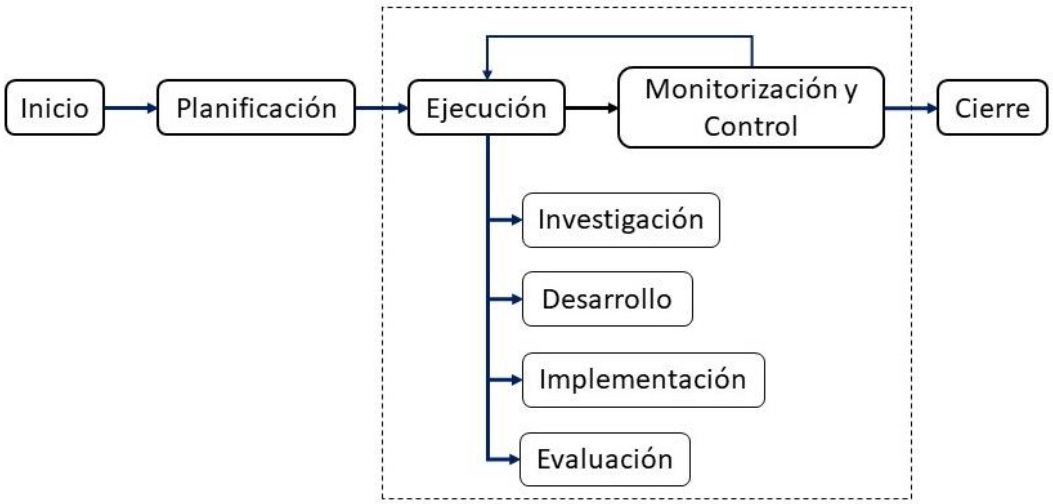


Figura 8: Metodología en cascada Waterfall.

Fuente: Elaboración Propia.

Tal como se muestra en la Figura 8, el trabajo fue realizado de acuerdo a la metodología propuesta. Hubo un proceso de retroalimentación en la fase de monitorización y control a fin de verificar los avances realizados en la fase anterior. En la etapa de ejecución se realizaron otro conjunto de etapas que permitieron desarrollar la investigación (SLR), desarrollo, implementación y evaluación del proyecto, a continuación, se presentan estas subetapas realizadas dentro de la fase de ejecución.

Primero se realizó la investigación utilizando la metodología conocida como revisión sistemática de literatura científica (SLR) para buscar, identificar, extraer información importante de cada elemento contenido en el área de investigación de este proyecto en las principales bases de datos académicas. En la fase preliminar del planteamiento de investigación se extrae preguntas de investigación y se realiza un “mentefacto conceptual” que facilita el desarrollo del tesauro para búsquedas y criterios de inclusión y exclusión. Los resultados obtenidos de este enfoque sistemático es conocer los autores más destacados de esta área en cuestión, cuáles son sus propuestas, cuáles son las bases de datos más relevantes y que publicaciones usar para presentar en nuestro trabajo, de esta forma obtendremos información actualizada y diversa de varios laboratorios, centros de investigación e instituciones de educación superior (Torres-Carrión et al., 2018).

El procedimiento para la utilización de la revisión sistemática tiene las siguientes etapas:

- Planificación
    - Identificación de la necesidad de revisión
        - Investigación del Estado Actual del Problema.

        - Preguntas de investigación
        - "Mentefacto Conceptual"
        - Revisiones sistemáticas relacionadas
    - Desarrollo de un protocolo de revisión.
        - Definición de criterios de inclusión y exclusión
        - Preparación de un formulario de extracción de datos
        - Selección de Revistas
- Realización de la revisión
    - Identificación de la investigación
    - Selección de estudios primarios
    - Evaluación de la calidad del estudio
    - Extracción y monitorización de datos
    - Síntesis y seguimiento de datos
- Informe de la revisión.

Aplicando la metodología SLR, se establecen las palabras correctas para la semántica de búsqueda de artículos. La estructura de búsqueda está adaptada para la base de datos Scopus. Realizada la búsqueda, se seleccionan los artículos que proporcionen información importante a la investigación. Finalmente, cada artículo es almacenado por el gestor bibliográfico EndNote. La cantidad de artículos encontrados y la semántica de búsqueda se presentan en la Figura 9:

Figura 9: Semántica de búsqueda para VA.

Fuente: Elaboración propia a partir de la búsqueda de artículos.

Realizada la investigación, se procede a realizar la ejecución del proyecto que consta de varias etapas la cuales son desarrollo, implementación y evaluación. Al finalizar, a partir de los resultados se obtuvo conclusiones, recomendaciones y algunos comentarios sobre los trabajos a futuro que se podría desarrollar. A continuación, se muestra un resumen de las tareas que se realizaron para la búsqueda de información científica de técnicas DRL y desarrollo referente a sistemas de conducción autónoma de vehículos.

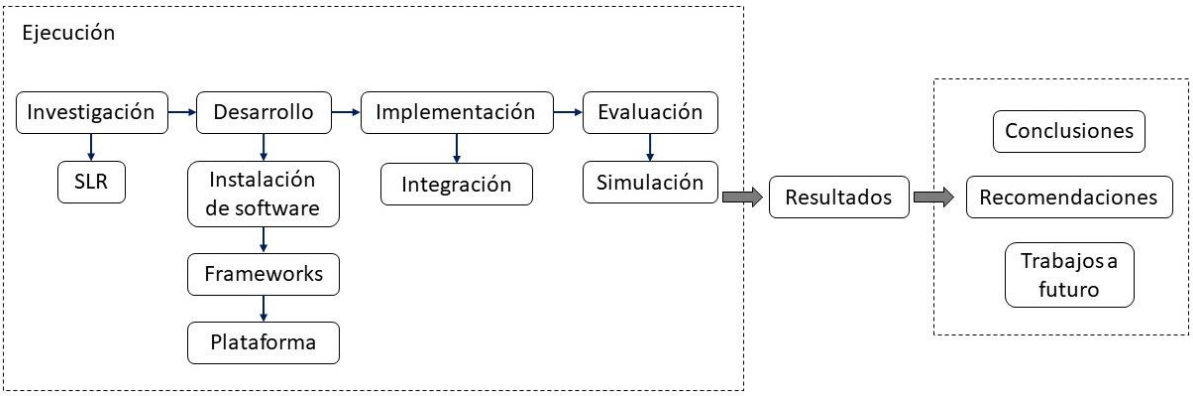


Figura 10. Esquema de trabajo para DRL en VA.

Fuente: Elaboración propia

En la Figura 10, se muestra el flujo de trabajo a realizar durante este proyecto, La etapa de Ejecución se divide en fases a fin de facilitar el estudio y la implementación del sistema DRL para vehículos autónomos.

3.2.1. Investigación

Se realiza una investigación literaria aplicando SRL sobre conducción autónoma, aprendizaje por refuerzo profundo, frameworks disponibles para la implementación de algoritmos DRL y plataformas existentes para la simulación de vehículos autónomos. Esto nos permitió encontrar información con relevancia sobre algoritmos, aplicaciones, software y metodologías para el desarrollo de sistemas inteligentes basados en DRL para el sector de conducción autónoma. Este método propone realizar tres tareas: planificación, realización e informe de resultados y permite identificar preguntas de investigación y justifica futuras investigaciones en nuestra área.

A partir de dicha investigación tenemos un gran abanico de opciones para la ejecución del proyecto. A demás, se tiene información variada y actual sobre este tema de investigación y se presenta una visión general del estado actual de vehículos de conducción autónoma, dando a conocer sus ventajas y desventajas, y como estos nuevos sistemas de transporte mejorarían la movilidad en los diferentes entornos de transporte existentes en la actualidad. Se ofrece una nueva aplicabilidad de VA para situaciones de emergencia, en donde se busca precautelar la vida humana y brindar sistemas de apoyo y transporte. Se muestran algunos retos que existen dentro del área de conducción autónoma y como los algoritmos de DRL tendrían la capacidad de solucionar los problemas expuestos. Finalmente, se plantea el desarrollo del sistema de VA, con la finalidad de cumplir con los objetivos propuestos en el trabajo de investigación.

3.2.2. Desarrollo

En esta fase se describe los elementos, frameworks, algoritmos, DRL, simuladores y librerías existentes para la implementación en simulación de sistemas de vehículos de conducción autónoma. En base a todas herramientas software disponibles, se realizó la selección e instalación de librerías y plataformas, así mismo, se escogió el lenguaje de programación que mejor se adaptaría a nuestro proyecto. Se realiza la descripción de los algoritmos DQN y DDPG, técnicas de Aprendizaje por Refuerzo Profundo y se escoge el algoritmo DQN para la implementación de nuestro agente en una simulación.

3.2.3. Implementación

Se realizó la integración de todos los componentes software que hemos seleccionado y nos permitieron realizar una simulación de conducción autónoma aplicando el algoritmo DQN. En esta fase, los paquetes software fueron librerías para la implementación de algoritmos de aprendizaje por refuerzo profundo, frameworks para el desarrollo de programas de aprendizaje automático, plataformas disponibles para la simulación de vehículos autónomos y UAV, para facilitar la integración de todos estos componentes se usó el lenguaje de programación Python, el cual nos permitió la comunicación servidor-cliente con el uso de la API de AirSim, también se realizó el envío de comandos de control al vehículo desde el script del cliente que contiene el modelo DRL.

3.2.4. Evaluación

En la etapa de evaluación, se realiza simulaciones del agente diseñado con la técnica DRL en un entorno de simulación que tiene cargado un mapa fotorrealista, este agente realiza el proceso de navegación y va tomando acciones de acuerdo a las predicciones dadas por el modelo obtenido en la fase de entrenamiento del Agente DQN. Dadas las opciones de clima que tiene el simulador se procede a realizar pruebas de navegación con varios tipos de climas.

3.2.5. Resultados

Concluidas las pruebas de simulación del sistema de conducción vehicular se procede a realizar un análisis de los resultados obtenidos. Se procede a verificar las acciones que toma el agente DQN cuando navega por la carretera de un entorno de simulación con clima variable. Con esto podemos sacar diversas conclusiones, y recomendaciones que nos servirán para mejorar o modificar ciertas partes del proyecto de conducción autónoma para futuros trabajos.

3.2.6. Alcance y limitaciones

El alcance de desarrollo de este trabajo de investigación es la implementación en simulación de un agente DQN (vehículo) que le permita navegar en la carretera de un entorno desconocido con clima variable. La principal limitación identificada en este trabajo es que no se va a desarrollar una configuración experimental para la implementación física del vehículo de conducción autónoma.

3.2.7. Tecnologías implicadas

En esta parte se identifican las tecnologías utilizadas en la propuesta del TFM, especialmente las relacionadas con la Industria 4.0, las cuales son: librerías para aprendizaje por refuerzo profundo, frameworks para aprendizaje automático, lenguaje de programación Python, librerías complementarias para el manejo de datos, plataformas de simulación de vehículos autónomos y motor gráfico para mapas fotorrealistas.

4. Desarrollo específico de la contribución

En este trabajo de investigación se propone el uso de algoritmos de Aprendizaje por Refuerzo Profundo en la capa de control de un vehículo autónomo. El objetivo es desarrollar un modelo entrenado que emplee algoritmos DRL y tenga la capacidad de enviar comandos de control al vehículo para que pueda navegar en la carretera de manera eficiente. Se presenta un agente que pueda recibir los datos de un sensor como la cámara frontal del vehículo.

4.1. ALGORITMOS DRL

El aprendizaje por Refuerzo Profundo combina la arquitectura de aprendizaje por refuerzo con redes neuronales artificiales, lo que permite que agentes definidos por software aprender a realizar mejores acciones en entornos virtuales (Pérez-Gil et al., 2022). En esta sección se presentan dos algoritmos típicos en DRL.

4.1.1. Algoritmo DQN

El algoritmo Deep Q-Network (DQN) fue presentado en el juego de Atari con Aprendizaje por Refuerzo profundo con entradas de imagen. La función Q se representó mediante una red neuronal convolucional (red neuronal profunda), cuyas entradas son imágenes preprocesadas (estado actual + estado anterior) y las salidas son valores Q previstos para cada acción posible, el parámetro de Q (pesos de la red neuronal) es la función de pérdida a optimizar en DQN. Durante el proceso de entrenamiento, las muestras se almacenan en un búfer y las actualizaciones del modelo (red neuronal) se realizan con mini lotes muestreados aleatoriamente en el búfer (Carton, 2021).

Se sabe que el agente hace una transición de un estado s al siguiente realizando alguna acción a y luego recibe una recompensa r. Esta información de transición se guarda en un búfer llamado búfer de repetición. Esta información es básicamente la experiencia del agente que se almacena durante varios episodios que nos servirá para entrenar el agente DQN (Ravichandiran, 2020). Esta información de transición se guarda de la siguiente manera en el búfer de transición:

1. Inicializar el búfer de transición.
2. Para cada episodio:
    a. Hacer una transición, en otras palabras, realizar una acción a en el estado s, pasa al siguiente estado y recibir la recompensa r.
    b. Guardar la información de transición en el búfer de repetición.

Este algoritmo es una extensión de Q-learning para casos de dimensión con una red neuronal profunda para la función de valor. Emplea una red de valor de acción objetivo y un búfer de reproducción para la actualización (Dong et al., 2020). El Algoritmo Deep Q-learning se muestra en la Tabla 3.

Tabla 3: Pseudocodigo DQN.

Pasos	DQN
1	**Hiperparámetros:** capacidad del búfer de reproducción N, factor de descuento de recompensa γ, pasos de retraso C para la actualización de la función de valor de acción,
2	**Entrada:** búfer de reproducción vacía D, parámetros iniciales θ de la función de valor de acción Q.
3	Inicializar la función objetivo de valor de acción Q con parámetro $\hat{\theta} \leftarrow \theta$
4	**Para** episodio = 0, 1, 2, … **hacer**
5	Inicializar entorno y obtener la observación O_0
6	Inicializar secuencia $S_0 = (O_0)$ y preprocesar la secuencia $\emptyset_0 = \emptyset(S_0)$
7	**Para** t = 0,1,2, … **hacer**
8	Con probabilidad ∈ seleccionar una acción aleatoria A_t, de lo contrario seleccionar $A_t = \arg max_a Q(\emptyset(S_t), a; \theta)$
9	Ejecute la acción A_t y observe O_{t+1} y recompense R_t
10	Si el episodio ha terminado, configure $D_t = 1$. De lo contrario, configure $D_t = 0$
11	Establecer $S_{t+1} = (S_t, A_{t,} O_{t+1})$ y preprocesar $\emptyset_{t+1} = \emptyset(S_{t+1})$
12	Guardar transición $(\emptyset_t, A_t, R_t, D_t, \emptyset_{t+1})$ en D
13	Minilote aleatorio de transiciones $(\emptyset_i, A_i, R_i, D_i, \emptyset_i')$ de D
14	Si $D_i = 0$, establezca $Y_i = R_i + \gamma\, max_{a'} Q'(\emptyset_i', a'; \hat{\theta})$. De lo contrario, establecer $Y_i = R_i$
15	Realizar un paso de descenso de gradiente en $(Y_i - Q(\emptyset_i, A_i;\, \theta))^2$ con respecto a θ
16	Sincronice el objetivo $\hat{Q}$ cada paso de C.
17	Si el episodio ha terminado, romper el ciclo
18	**FinPara**
19	**FinPara**

Fuente: (Dong et al., 2020)

El objetivo de Deep Q-Learning es adaptar el algoritmo de diferencia temporal (1) para que la fórmula de actualización sea equivalente al paso de descenso de gradiente para entrenar una red neuronal artificial que resuelva una determinada tarea de regresión (Ivanov & D'Yakonov, 2019). La función de valor de estado-acción $Q(s, a)$ se usa como función de evaluación. La función de valor (s, a) es el valor estimado en lugar del valor real. El algoritmo selecciona la función de valor Q máximo de los valores estimados de diferentes acciones. El método de actualización de la función de evaluación es (Sun et al., 2021):

$$Q^*(s,a) = Q^*(s,a) + a_t[r' + (1 - done)\gamma max_{a'} Q^*(s',a') - Q^*(s,a)] \quad \text{(1) Diferencia temporal Q}$$

Donde Q* es la función de valor optimo, r' es la recompensa actual, a es el factor de paso y γ es el factor de rebaja. Esta expresión se refiere a la diferencia temporal que representa la diferencia entre el valor $Q^*(s,a)$ y el paso de aproximación $r_{t+1} + \gamma Q_t^*(s_{t+1}, a')$, puede ser cero en respuesta para una función Q optima.

4.1.2. Algoritmo DDPG

Los algoritmos de gradiente de política determinística se calculan de más eficiente que la función Q que solo deriva de las acciones, este algoritmo aprende una política determinista utilizando una política estocástica para la exploración. El algoritmo Deep Deterministic Policy Gradiente (DDPG) es una combinación de DPG y DQN, el cual se basa en el actor-crítico para una política determinista como DPG, pero usa técnicas de aprendizaje Q profundo para estimar valores Q críticos (Carton, 2021). El pseudocódigo de implementación del algoritmo se muestra en la Tabla 4.

Tabla 4: Pseudocódigo DDPG.

Pasos	DDPG
1	**Hiperparámetros:** factor de actualización ρ, factor de descuento de recompensa γ
2	**Entrada:** búfer de reproducción vacía D, parámetros iniciales θ^Q de la red critica $Q(s,a\|\theta^Q)$ y parámetros θ^π de la red del actor $\pi(s\|\theta^\pi)$, objetivo de la red Q' y π'
3	Inicializar el objetivo de la red Q' y π' con pesos $\theta^{Q'} \leftarrow \theta^Q, \theta^{\pi'} \leftarrow \theta^\pi$
4	**Para** episodio = 1 **hacer**
5	Inicializar un proceso aleatorio N para la acción de exploración
6	Recibir el estado inicial de observación s_1
7	**Para** t = 1 **hacer**
8	Seleccionar la acción $A_t = \pi(S_t,\|\theta^\pi) + N_t$
9	Ejecute la acción A_t, observe la recompensa R_t y observe el nuevo estado S_{t+1}
10	Guardar transición $(S_t, A_t, R_t, D_t, S_{t+1})$ en D
11	Establecer $Y_i = R_i + (1 - D_t)Q'(S_{t+1}, \pi'(S_{t+1}\|\theta^{\pi'})\|\theta^{Q'})$
12	Actualización critica por minimización de perdida: $L = \frac{1}{N}\sum_i (Y_i - Q(S_i, A_i\|\,\theta^Q))^2$
13	Actualizar la política de actor usando la política de gradiente muestreada: $\nabla_{\theta^\pi} J \approx \frac{1}{N}\sum_i \nabla_a Q(s,a\|\theta^Q)\|_{s=S_i, a=\pi(S_i)} \nabla_{\theta^\pi}\pi(s\|\theta^\pi)\|S_i$
14	Actualizar el objetivo de la red: $\theta^{Q'} \leftarrow \rho\theta^Q + (1-\rho)\theta^{Q'}$ $\theta^{\pi'} \leftarrow \theta^{\pi'} + (1-\rho)\,\theta^{\pi'}$
18	**FinPara**
19	**FinPara**

Fuente: (Dong et al., 2020)

Este algoritmo es muy sensible a los hiperparámetros y menos estable que los algoritmos por lotes, por lo que DDPG actualiza la política en cada paso. Estos dos algoritmos suelen ser utilizados para aplicación de vehículos autónomos, en este trabajo se implementará el algoritmo DQN para demostración del funcionamiento en simulación.

4.2. ARQUITECTURA PROPUESTA

Los sistemas de conducción autónoma (SCA) integran funcionalidades que permiten a los VA operar en entornos reales sin la necesidad de conductores humanos (CH), dependiente del nivel de automatización en el que desarrolle el sistema. Existen dos componentes para SCA: sensores, controladores como freno, acelerador, etc. (hardware) y grupos funcionales (software) (Fan et al., 2019).

4.2.1. Arquitectura de Conducción Autónoma de Vehículos

En conducción autónoma existen dos arquitecturas vitales para el funcionamiento de varios sistemas, estas son la arquitectura hardware y software. La arquitectura basada en software (Zong et al., 2018) está desarrollada para vehículos de conducción autónoma, la cual está dividida en cinco módulos: percepción, localización y mapeo, predicción, planificación y control del vehículo, esto se la muestra en la Figura 11.

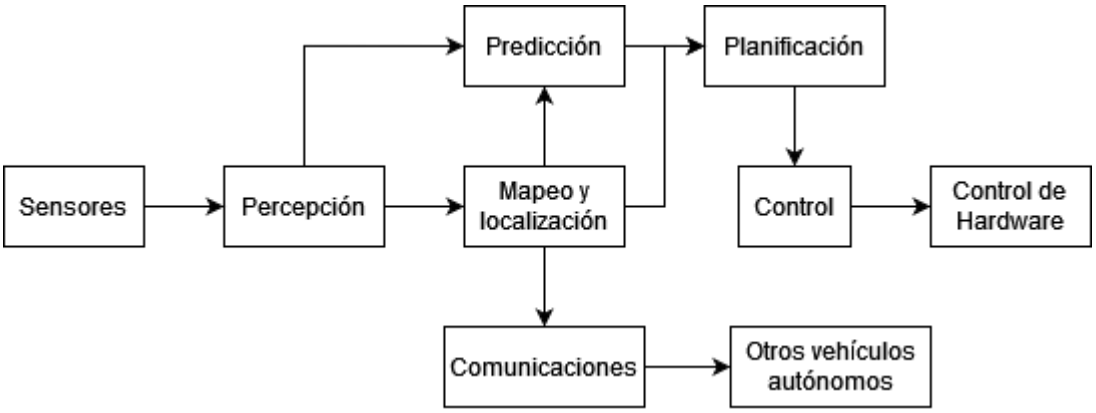


Figura 11. Arquitectura software de la navegación autónoma.

Fuente: Elaboración propia, a partir de (Fan et al., 2019)

La tarea de la percepción es brindar al vehículo la capacidad de detectar diferentes elementos que se encuentran en el entorno, esto lo hace con la adquisición de datos de sensores, lo que incluye la detección y seguimiento de objetos (carril, peatón, danos en la carretera, etc.) y las tecnologías aplicadas a esta tarea se dividen en dos categorías: basadas en visión artificial y basadas en inteligencia artificial. En el módulo de localización y mapeo se puede estimar la ubicación del VA, construir y actualizar un mapa mundial en 3D (Maurer et al., 2016).

El módulo de predicción permite que el VA tome decisiones adecuadas para la conducción así como analiza los patrones de movimiento y predice la trayectoria futura y existen dos enfoques de predicción: basados en modelos, el cual calcula el movimiento futuro del VA al propagar su estado cinemático (posición, velocidad y aceleración) a lo largo del tiempo y basados en datos con la integración de inteligencia artificial y computación de alto rendimiento usando modelos ocultos de Markov, redes bayesianas, aprendizaje por refuerzo inverso, etc.

En el módulo de planificación se determina las posibles rutas de navegación del VA en función de la percepción, localización, mapeo y la información de predicción, se genera una trayectoria que sigue la mejor ruta y maniobra de conducción garantizando la seguridad. Finalmente, el módulo de control combina varios tipos de controladores (MPC, PID, LQR, etc.) y envía comandos para aceleración, freno, par de dirección al vehículo que sigue la trayectoria planificada (Van Brummelen et al., 2018).

4.2.2. Arquitectura Software de simulación

Para la realización del diseño de la arquitectura basada en software se considera algunos elementos clave para la implementación del sistema de VA: una plataforma que permita la integración de algoritmos de IA, simulador que integre API's para la adquisición de datos, control de vehículos y mapas fotorrealistas con entornos rurales. En la Figura 12, se resaltan los principales módulos que se encuentran dentro de la arquitectura del simulador seleccionado: AirSim.

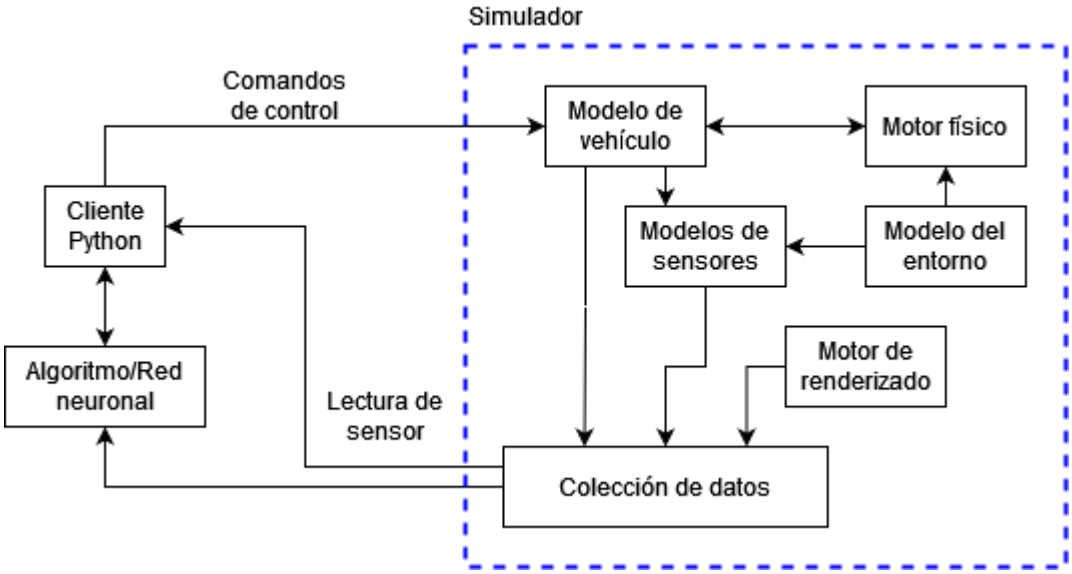


Figura 12: Estructura modular de Airsim y Unreal Engine.

Fuente: Elaboración propia, a partir de (Shah et al., 2017)

AirSim es una plataforma con estructura modular, en la capa de abstracción del simulador consta del modelo del vehículo, motor de física y entorno de simulación que permite el cálculo del movimiento realista del vehículo. Posee el módulo de sensores donde se implementan algunos tipos de sensores como IMU, LIDAR, cámara, GPS. Para la representación fotorrealista de los entornos a simular se utiliza el componente Unreal Engine, necesario para implementar algoritmos por visión de computadora, como ejemplo, para realizar un preprocesamiento en las capturas de imágenes de las escenas que se serán utilizadas para entrenar una red neuronal artificial. Toda la información recolectada por los sensores es registrada en el componente de recopilación de datos. Este simulador también permite la simulación de Hardware in the Loop (HIL) y Software in the Loop (SIL) , donde el controlador de hardware puede interactuar con otros módulos de toma de decisiones para el control de acciones (Mertens, 2018).

4.2.3. Agente de vehículo con modelo DL

En esta sección se presenta la realización de un modelo DL en la conducción autónoma de un vehículo a través de una parte del mapa de Landscape Mountain en AirSim. Este modelo de aprendizaje profundo realiza predicciones de ángulo de dirección, esta estrategia de modelado requiere una gran cantidad de datos de entrenamiento y los parámetros del estado actual del vehículo (velocidad, ángulos de dirección, aceleración, etc.). La entrada del modelo se compone de imágenes de la cámara frontal del vehículo. Para minimizar el tiempo de entrenamiento del modelo se extrae el Region of Interest (ROI) de las imágenes, tal como se muestra en la Figura 13, reduciendo el número de características irrelevantes del entorno. El ROI consta de un rectángulo cuya área de interés está dada en los puntos [76, 135, 0, 255]. Para el entrenamiento no se considera otros vehículos u objetos dinámicos o estáticos en el entorno de simulación y el modelo solo se enfocará en la carretera.

Figura 13: ROI de entrada del modelo DL.

Fuente: Elaboración propia.

El modelo de aprendizaje profundo cuando se encuentre con situaciones nuevas, recurrirá a lo ya visto en los datos de entrenamiento y hará predicciones de acuerdo al conjunto de datos. Se realiza una combinación de dos estrategias de conducción: en la primera se considera a aquella conducción que produce ángulos pequeños de dirección que se producen en una carretera y la segunda se centra en giros cerrados con ángulos de giro más altos, esto ayudara a la red neuronal a reaccionar cuando el vehículo se salga de la carretera.

Con el generador de datos de conducción se extrae el rectángulo inferior de la imagen, con este recorte de imagen se disminuirá el tiempo del procesamiento y la carga de entrenamiento. El modelo definido por DL es muy limitado debido a que solo usa imágenes para percibir el entorno, pues no considera otros elementos como entradas de más sensores o el estado del vehículo.

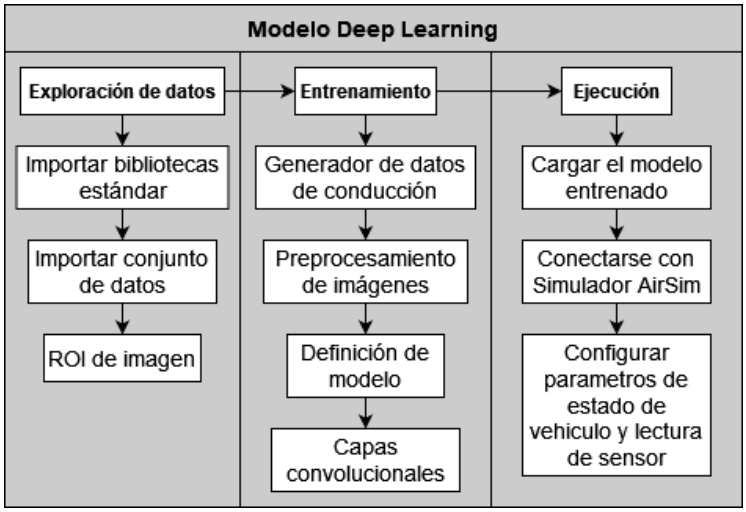


Figura 14: Modelo de aprendizaje profundo.

Fuente: Elaboración propia.

La red neuronal usada está constituida por tres capas convolucionales con 16, 32, 32 filtros y están fusionadas con una capa de entrada que proporciona el estado anterior del automóvil, y luego se conecta con cos capas completamente conectadas con 64 y 10 neuronas ocultas, la función de activación que se utiliza es ReLU. La salida de la red es el ángulo de dirección cuyo valor numérico es de punto flotante (Koul et al., 2019).

4.2.4. Agente de vehículo con modelo DRL

El agente propuesto se basa en el algoritmo DQN. Para simplificar las tareas de desarrollo del modelo DRL, la función de recompensa está basada en el proyecto de (Microsoft, 2018), cuya definición está dada en la posición del automóvil. El centro de la carretera es la mejor

posición del automóvil, la recompensa tiene un valor alto cuando el automóvil se encuentra en el centro y baja cuando está cerca del borde de la carretera, cuyo valor esta dado en el rango [0,1].

El modelo DRL utiliza imágenes de la cámara frontal del vehículo como entrada. Como en el modelo DL, solo se utiliza la porción inferior de la imagen a fin de minimizar el procesamiento y la cantidad de parámetros de entrenamiento. La arquitectura de red neuronal consta de tres capas de convolución. El enfoque de entrenamiento que se adopta es usar la Técnica de transferencia de aprendizaje, utilizando ya el modelo que se entrenó previamente y aplicarlo al modelo actual. En este caso, se usará el modelo DL entrenado que aprendió a conducir solo en el camino.

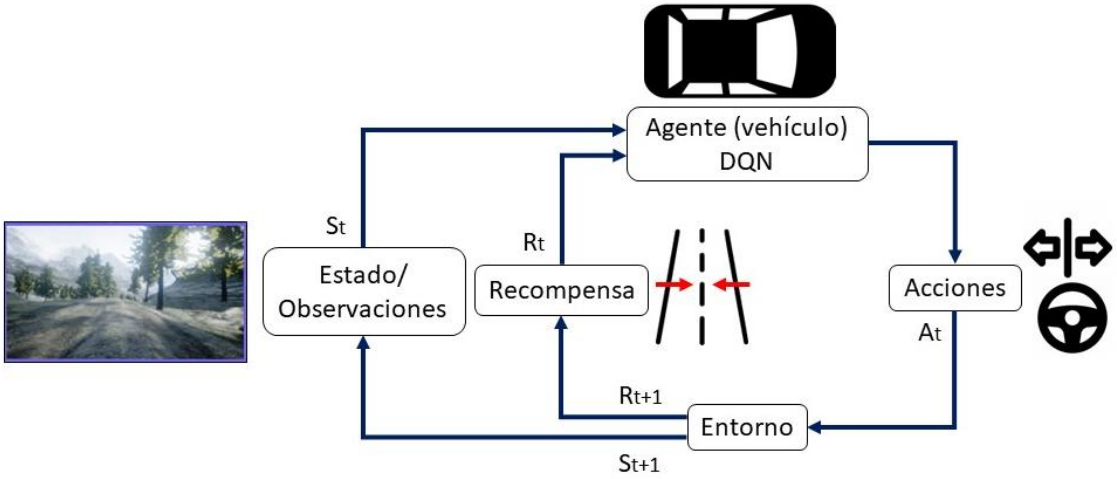


Figura 15: Modelo DQN para VA en AirSim.

Fuente: Elaboración propia.

El proceso de aprendizaje para el agente basado en el algoritmo DQN se basa en conceptos de observación, acción y recompensa. En cada paso, a partir de la observación actual, el modelo DQN dicta la acción que debe tomar el vehículo (política), con esto se realiza el cálculo de la recompensa por cada acción tomada, el modelo DQN actualiza el conocimiento con la observación, acción, recompensa actual. Para cada observación (S_t) el agente realiza una acción (A_t) dada por el modelo DQN, se evalúa la recompensa (R_t) dada por la acción y el modelo actualiza la información (S_t, A_t, R_t) y finalmente se observa un nuevo estado y se realiza de nuevo el procedimiento.

El desafío de este proyecto es establecer que el vehículo autónomo sea capaz de realizar el seguimiento de la carretera, siguiendo curvas sin salirse al costado de la vía. Se toma en cuenta algunas consideraciones:

- El vehículo autónomo navegara por la carretera tomando en cuenta dos acciones de control para la dirección y aceleración.
- No existe un destino en particular que el vehículo deba llegar, lo que se busca es darle la capacidad de moverse por carretera.
- No se agregan obstáculos como personas u otros vehículos, se pretende que el vehículo sea capaz de navegar en un entorno rural, con terreno irregular y variaciones en el clima.

Algunos elementos que comprenden en general nuestro agente DQN con el entorno virtual mostrado en la Figura 15 son:

Agente: En este caso es el propio vehículo, el cual está integrado por Microsoft AirSim para la simulación con el mapa.

Entorno virtual: Definimos entorno a todo con lo que el vehículo se va a encontrar mientras navega por el mapa Landscape Mountains: arboles, puentes, clima, terreno irregular, obstáculos que le permitirá al agente aprender a esquivar.

Estado: También llamadas observaciones, son las lecturas del sensor, una cámara que se ubica en la parte frontal del vehículo, estas imágenes son parciales pues no se observa todo el entorno sino solo la porción inferior de estas como entradas.

Acción: Son los comandos de control que se dará para que el vehículo navegue, estas son las señales de dirección y aceleración, para los giros a la izquierda se toma valores entre (-1, 0) y para la derecha valores entre (0,1), por lo que el ángulo de giro se dará en un rango de valores discretos (-1.0, -0.9, -0.8, ..., 0.0, …, 0.8, 0.9, 1.0) y la aceleración

Recompensa: El criterio de recompensa está basada en la posición del vehículo cuando navega por la carretera, se da mayores puntos de recompensa cuando el vehículo está centrado en la carretera y una recompensa baja cuando se conduce lejos del centro. La función de recompensa tiene un rango de valores [0,1] de esta manera se facilita a la red aprender.

5. Desarrollo del sistema

Existen muchos métodos de simulación que permiten cubrir de forma parcial o todo el proceso de desarrollo de sistemas de conducción autónoma. Con la variedad que nos ofrece cada herramienta, nos brindan varias opciones de implementación, tras un estudio exhaustivo y teniendo en cuenta algunos requisitos idóneos para este trabajo, se toma en cuenta algunas características como código abierto, multiplataforma, personalización y documentación. Algunas soluciones ofrecen herramientas para abordar procesos de

extremo a extremo, subsistemas o funcionalidades (ADAS, SLAM, tráfico, conducción) de un VA y se puede realizar diferentes pruebas de enfoque de diseño como: Modelo en ciclo (MIL), software en el ciclo (SIL) y Hardware en ciclo (HIL) (Fadaie, 2019). La mayoría de simuladores son compatibles con lenguajes de programación como C/C++, Perl, Python, Java, LabVIEW, URBI, MATLAB. (Rosique et al., 2019). A continuación, se presentan diferentes enfoques que se considera para seleccionar el simulador de vehículos autónomos enfocado en el desarrollo del presente trabajo.

5.1. SIMULADORES PARA CONDUCCIÓN AUTÓNOMA

Las pruebas a través de simulación son de vital importancia para complementar y acelerar las pruebas del mundo real. Los simuladores permiten probar escenarios que en el mundo real estarían regulados debido a varios problemas de seguridad, además son reproducibles, escalables y reducen el costo de desarrollo. Existen muchos simuladores disponibles para probar el software para vehículos autónomos, algunas son herramientas propietarias como CarCraft, SurfelGAN, Webviz y The Matrix y también hay simuladores disponibles de código abierto (Shi & Liu, 2021).

Se realizan pruebas de simulación con la finalidad de asegurar el funcionamiento de los algoritmos y software en vehículos autónomos para que cumplan con las exigencias del diseño. Para probar la funcionalidad y automatización de VA se identifica un conjunto de criterios que sirven como métrica para seleccionar al mejor simulador, para realizar esta tarea se considera: 1) el enfoque centrado en los requisitos impulsados por la arquitectura funcional de vehículos autónomos (Fan et al., 2019), 2) los requisitos que deben cumplirse para soportar la infraestructura para la conducción del VA, 3) se define los requisitos que permiten el uso del simulador para tareas secundarias y 4) se enumera algunos requisitos genéricos para determinar un buen simulador de VA (Shi & Liu, 2021).

5.1.1. Requisitos para la arquitectura funcional de VA

Percepción: Uno de los componentes principales de sistemas de VA es su capacidad de percibir el entorno que lo rodea. La percepción del VA se compone de hardware integrada con varios sensores y software que es el encargado de procesar e interpretar los datos de los sensores. Algunos de los sensores comunes para VA incluyen cámara, LIDAR, sensor ultrasónico, radar, sistema de posicionamiento global (GPS) y unidades de medición inercial (IMU) (Van Brummelen et al., 2018). El entorno de simulación debe tener modelos de sensores que le permitan probar el sistema de percepción.

Geometría Multivista: Para la construcción del mapa de entornos desconocidos y rastrear la ubicación del sistema de conducción autónoma (CA) es necesario el uso del componente

de localización y mapeo simultaneo (SLAM). Para integrar aplicaciones SLAM es necesario proporcionar la calibración de las cámaras y así estos algoritmos SLAM pueden ejecutar la geometría de vista múltiple y localizar el sistema CA dentro del mapa global.

Planeación de rutas: La colisión con elementos en el entorno es uno de los problemas para la planificación de rutas, la información del entorno es recibida mediante la infraestructura como sensores y sistemas de redes de comunicaciones. La panificación se subdivide en dos: global y local, el planificador global se crea en función de un mapa estático del entorno y el planificador local se genera en función del entorno inmediato del agente móvil. Existen varios algoritmos de planificación de rutas inteligentes como: A*, D*, Dijkstr, Optimización de trayectoria, Lattices, Trayectoria optima, Planificación de turno, Planificación con curvas polinómicas, Comportamiento Spline, evaluación de curvas B'ezier y RRT (González et al., 2015). Para la implementación de estos algoritmos, el simulador debe integrar funciones para la creación/importación de mapas e interfaces para programar algoritmo personalizados.

Control de vehículo: Para ejecutar la trayectoria planificada en una ruta establecida se utiliza sistemas de control en lazo cerrado para las entradas de control como el acelerador, freno y dirección (Fan et al., 2019). Entre los algoritmos de control usados para la industria tenemos a los algoritmos Proporcional, Integral y Derivativo (PID) y Control predictivo del modelo (MPC). Para implementar estos algoritmos de control el simulador debe ser capaz de construir modelos dinámicos de vehículos y tener la capacidad de programar los algoritmos con formulación matemática (Shi & Liu, 2021).

5.1.2. Requisitos para soportar infraestructura de conducción de VA

Entorno virtual 3D: Dentro del entorno virtual 3D se requiere que se pruebe todos los elementos funcionales para VA, este entorno debe incluir objetos estáticos como edificios, arboles, etc. y objetos dinámicos como peatones, otros vehículos, animales, etc. para lograr una correcta creación de entornos virtuales, los simuladores se basan en motores de juegos o utilizan mapas de alta definición (HD) de un entorno real. También se requiere que el entorno virtual tenga soporte de diferentes terrenos y condiciones climáticas, características típicas de entornos reales (Fadaie, 2019). Se recalca que el nivel de detalle de un entorno virtual 3D depende del enfoque de simulación que se requiera implementar.

Infraestructura de tráfico: Dentro de entorno virtuales 3D es requerido que la simulación sea compatible con infraestructura de tráfico como semáforos, señalización vial, etc., se considera clave que un sistema de visión artificial integrado en VA sea capaz de detectar y reconocer todo tipo de señales de tráfico (Lafuente-Arroyo et al., 2005). Con la aparición y

desarrollo de VA se espera que evolucione la infraestructura de tráfico y cumplan con las normas establecidas en los sistemas de tránsito. Las infraestructuras físicas como facilitadoras de conducción de sistemas VAC, aún se encuentran en fase de planificación/estudio y buscan obtener los impactos de VA en las infraestructuras de tráfico (Rana & Hossain, 2021).

Simulación de escenarios de tráfico: uno de los puntos que identifican a un simulador como valioso o no es su capacidad de recrear/reproducir diversos escenarios del mundo real y también probar escenarios hipotéticos que por temas de seguridad no podrían ser probados en el mundo real. Para generar estos escenarios se requiere que el simulador sea capaz de admitir diversos agentes dinámicos como humanos, animales, vehículos livianos, ambulancias, autobuses. Y también debe proporcionar al usuario la capacidad de controlar el comportamiento de objetos dinámicos, condiciones climáticas, tipo de sensores en VA, señales de tráfico, etc (Shi & Liu, 2021).

5.1.3. Requisitos para tareas secundarias

Suelo 2D/3D: El simulador debe proporcionar etiquetas de objetos y cuadros delimitadores de los objetos que aparecen en el escenario que se genere (Kaur et al., 2021).

5.1.4. Requisitos genéricos

Estabilidad y buen mantenimiento: El simulador debe poseer una documentación completa de información de su uso y funcionamiento. Esta documentación de proporcionar el mapeo preciso entre las API en desuso y las API recién agregadas que permitan mejorar el simulador.

Flexibilidad/Modular: El principio de división de concepto ayuda a desarrolladores a desarrollar y ampliar diferentes escenarios para simuladores de código abierto en menos tiempo, así también permite generar nuevos entornos y agregar diferentes agentes que puedan interactuar en la simulación.

Portabilidad: Se espera que los simuladores puedan ser ejecutados en diferentes sistemas operativos, debe existir diferentes versiones para los usuarios que usan diferentes tipos de SO.

Escalabilidad: La integración de una arquitectura escalable a través de servidores, permite el control de varios agentes al mismo tiempo por medio del uso de nodos, esto es útil cuando se requiere simular escenarios de congestión vehicular y/o escenas complejas de múltiples agentes.

Fuente abierta: Esta característica permite la colaboración colectiva de múltiples desarrolladores, lo que se facilita el desarrollo de nuevas y novedosas funcionalidades en el simulador.

Con el objetivo de estudiar las funcionalidades como percepción, localización, control de vehículos y creación de entornos virtuales 3D dinámicos se comparan los simuladores MATLAB/SIMULINK, CarSim, PreScan, CARLA, Gazebo y LGSVL y de esta manera se selecciona el simulador que preste mejores ventajas para el trabajo de investigación.

Tabla 5: Comparación de simuladores para VA.

Requerimientos	MATLAB	CarSim	PreScan	CARLA	Gazebo	LCSVL
Percepción (modelos de sensores)	Si	Si	Si	Si	Si	Si
Percepción (diferentes condiciones climáticas)	No	No	Si	Si	No	Si
Calibración de cámara	Si	No	Si	Si	No	No
Planificación de ruta	Si	Si	Si	Si	Si	Si
Control de vehículo	Si	Si	Si	Si	Si	Si
Entorno virtual 3D	Urbano	Si	Si	Exterior (Urbano)	Interior y exterior	Exterior (Urbano)
Infraestructura de trafico	Con semáforos	Si	Si	Semáforos, intersecciones, señales de alto, carriles	Si	Si
Simulación de escenarios de tráfico (objetos dinámicos)	Si	Si	Si	Si	No	Si
Suelo 2D/3D	Si	No	No	Si	Urbano	Si
Interfaz para otro software	Con CarSim, PreScan, ROS	Con Simulink	Con Simulink	Con ROS, Autoware	Con ROS	Con Autoware, Apollo, ROS
Escalabilidad	U	U	U	Si	Si	Si
Fuente abierta	No	No	No	Si	Si	Si
Portabilidad	Si	Si	Si	Si	Si	Si
API flexible	Si	Si	U	Si	Si	Si

Fuente: (Shi & Liu, 2021)

De acuerdo a la Tabla 5, se observa las diferentes características que presentan las diversas herramientas para la simulación de sistemas de conducción autónoma, la letra “U”

en la tabla nos indica que el componente es desconocido si existe o no en la plataforma; sin embargo, debido a la naturaleza de implementación en simulación del presente proyecto se optó por otra herramienta. El simulador AirSim (Aerial Informatics and Robotics Simulation) es el software más adecuado para realizar pruebas integrales de conducción autónoma de vehículos pues integra funcionalidades como percepción, mapeo, localización, control del vehículo e integración de técnicas de Machine Learning (DL y DRL), además de tener mapas con entornos rurales.

Tabla 6: Resumen de características del simulador seleccionado AirSim.

Licencia	Motor grafico	Lenguaje	Sensores	Pruebas	Plataforma
GPL/ Código abierto	Unreal Engine	C++, Python, C#, Java	GPS, IMU, LIDAR, Cámara	Conducción, HIL, SIL	Linux, Windows

Fuente: (Rosique et al., 2019)

En la Tabla 6, se muestra las principales características por las cuales fue seleccionado el simulador AirSim. Esta plataforma de simulación se ejecuta como un complemento para Unreal Engine, el cual es un motor grafico de video juegos con capacidades de renderizado fotorrealista, lo que permite usar vehículos en varios tipos de entornos visualmente realistas.

La simulación de entornos virtuales permite recolectar una gran cantidad de datos con una variedad de escenarios con diferentes condiciones climáticas, elementos de tráfico como semáforos, vías peatonales, carriles, etc. y proporciona una plataforma de pruebas segura para diferentes tipos de vehículos aéreos y terrestres. Con Airsim se puede trabajar con escenarios variados de simulación integrando modelos construidos con técnicas de inteligencia artificial como Aprendizaje Profundo y Aprendizaje por Refuerzo.

5.2. CONJUNTO DE DATOS

Actualmente un número creciente de empresas emergentes están desarrollando soluciones de conducción autónoma de vehículos con funcionalidades SAE de nivel 5. Para desarrollar la capacidad de autonomía y seguridad ante cualquier tipo de entorno y situación de tráfico sin la necesidad de un humano se requiere ejecutar una evaluación del software de un vehículo autónomo mediante el entrenamiento y pruebas de algoritmos basados en aprendizaje automático. Para realizar estas tareas se usan entornos de prueba virtuales que permitan validar el aprendizaje de estos algoritmos de extremo a extremo (Kang et al., 2019).

Existen muchos conjuntos de datos disponibles públicamente para la evaluación de algoritmos de vehículos autónomos (Bojarski et al., 2016). Los autores (Kang et al., 2019),

presentan un resumen de 37 conjuntos de datos de conducción existentes con información: hora y lugar de la recopilación de datos, condiciones de tráfico, sensores usados, formato y tamaño de datos, recursos proporcionados, recursos proporcionados (datos sin procesar, anotaciones, puntos de referencia, código fuente, soporte de herramientas), licencia y accesibilidad.

Existen páginas web que contribuyen con información de conjuntos de datos para aprendizaje automático en vehículos autónomos como es el caso de (Gutta, 2021) que presenta una colección de 21 conjuntos de datos con una descripción detallada cada uno como: nombre del conjunto de datos, año de publicación, tipo de sensor, lugar de grabación y enlace de descarga del archivo y articulo del proyecto.

Para la implementación del entrenamiento del modelo DL en AirSim, se seleccionó el conjunto de datos, imágenes recolectadas del mapa Landscape Mountain, con un tamaño sin comprimir de 3.25 GB (Koul et al., 2019).

5.3. MARCOS DE TRABAJO PARA APRENDIZAJE POR REFUERZO PROFUNDO

Programar una implementación para aplicaciones de Aprendizaje por Refuerzo Profundo desde cero, no es la mejor opción para iniciar el desarrollo del proyecto propuesto. Existen algunos marcos que integran librerías estándar y fáciles de implementar algoritmos DRL y RL que están basados en Tensorflow. A continuación, se presenta una descripción general de algunos marcos de RL disponibles.

5.3.1. Keras-RL

Permite una implementación de código sencilla y es fácil de entender que permite personalizar las funcionalidades de código de acuerdo a la necesidad del proyecto, tiene integrado varios algoritmos RL como: DQN, Double DQN, Deep Deterministic Policy Gradient (DDPG), continuos DQN (CDQN), Cross-Entropy Method (CEM) y SARSA. En GitHub contiene algunos ejemplos demostrativos que permitirán al usuario entender el funcionamiento y su programación en Python.

5.3.2. TensorLayer

Es una librería para DL y DRL que está diseñada específicamente para investigadores e ingenieros pensanda en ofrecer simplicidad, flexibilidad y alto rendimiento pudiendo realizar tareas complejas de IA, es un software de código abierto. TensorLayer admite TensorFlow como backend computacional (Dong, 2022). Algunos conceptos, expresiones matemáticas y

algoritmos DRL están implementadas en esta librería están descritas a detalle en (Dong et al., 2020).

5.3.3. Stable-Baselines3

Este marco de trabajo de código abierto contiene un conjunto de siete implementaciones de algoritmos sin modelo de aprendizaje por refuerzo profundo en Pytorch que se basa en la interfaz OpenAI Gym y TensrFlow. Proporciona una interfaz limpia y simple para el acceso a algoritmos de RL, la documentación completa se encuentra disponible en (Raffin, 2021). Las implementaciones de funciones o algoritmos dada por la API está completamente documentada y el código puede ser modificable por los usuarios que deseen personalizar su proyecto.

De los marcos de trabajo expuestos, se optó por trabajar con Keras-RL. Esta librería ofreció mayor facilidad y flexibilidad para la solución a los retos en la implementación de un sistema de conducción autónoma de vehículos con algoritmos DRL en la plataforma de AirSim.

6. Integración del sistema

Se presenta una configuración experimental de la arquitectura software para la implementación en simulación del entorno virtual con un sistema de conducción autónoma de vehículo, utilizando un modelo de aprendizaje por refuerzo profundo con el objetivo de desarrollar un algoritmo de navegación en carreteras. El plan es aprovechar herramientas de código abierto (Keras, TensorFlow, Keras-RL), la tecnología comercial de Microsoft (AirSim) y el motor de videojuegos Unreal Engine para el mapa con entorno rural. En los siguientes apartados se describe la implementación de técnicas de inteligencia artificial para realizar pruebas de simulación de conducción autónoma en vehículos con entornos virtuales realistas.

6.1. CONFIGURACIÓN Y REQUERIMIENTOS

El sistema software para la simulación del agente DQN para conducción autónoma consta del paquete del simulador AirSim, se usó la librería Keras para implementar la red neuronal artificial, conjuntamente con la librería de Keras-RL que permitió integrar el algoritmo DQN, se utilizó el conjunto de datos creado por conducción en Airsim con el mapa Landscape Mountain, en adición se instaló algunas dependencias Python requeridas para correr el código del proyecto. Cada elemento del proyecto como librerías, simulador y API's están desarrolladas en Python, lo que nos facilita y simplifica la implementación en un solo lenguaje de programación.

Para el desarrollo de la configuración experimental se usa tres componentes principales, estos son:

AirSim: Es una plataforma que es utilizada para la investigación de IA que permite experimentar con algoritmos de aprendizaje profundo, aprendizaje por refuerzo y visión artificial para vehículos autónomos y UAV's y funciona como un complemento que se ejecuta en el motor de juegos Unreal Engine (UE).

OpenAI Gym: Es una biblioteca Python de código abierto que proporciona una API estándar para la comunicación entre una gran variedad de entornos de juegos que permiten evaluar algoritmos de aprendizaje por refuerzo.

Keras-RL: Es una biblioteca de Python que implementa varios algoritmos de aprendizaje por refuerzo profundo en Python, se integra con Keras front-end con TensorFlow back-end para diseñar y entrenar la red neuronal profunda y funciona con OpenAI Gym.

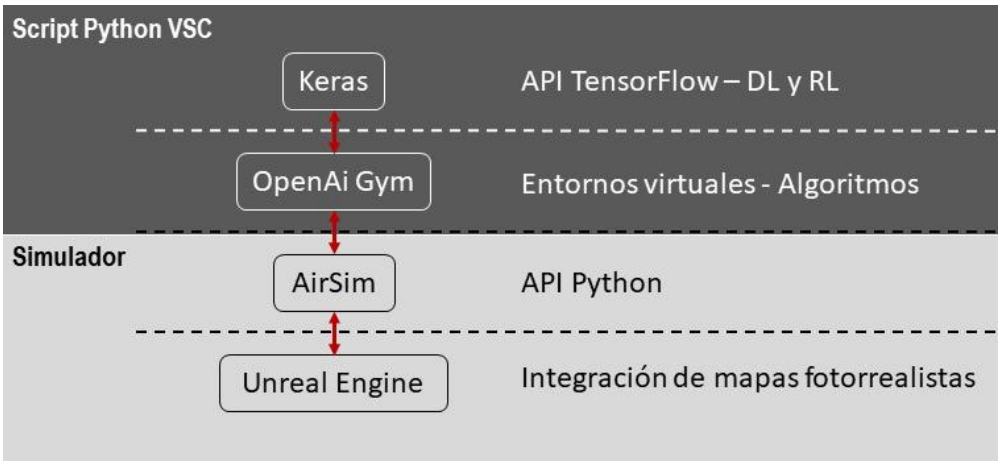


Figura 16: Componentes software para la configuración experimental.

Fuente: Elaboración propia.

En la Figura 16, se presenta los principales componentes del sistema software, las librerías Keras y Keras-RL nos permitieron desarrollar la red neuronal y la implementación del agente DQN para nuestro modelo DRL, OpenAI Gym nos ofreció la capacidad de crear un entorno personalizado para Airsim que permitió el entrenamiento del agente DQN (vehiculo). AirSim ofrece un conjunto de API disponibles para la obtención y control de datos desde y hacia el simulador con el entorno fotorrealista, de esta forma se controló las acciones del vehículo mientras navega en la carretera del mapa.

En la siguiente Tabla 7, se resume el software requerido para la implementación en simulación del proyecto.

Tabla 7: Resumen de requerimientos.

Ítem	Comentarios	Enlace
Simulador AirSim	Paquete del instalador	https://microsoft.github.io/AirSim/
Unreal Engine	Motor físico	https://www.unrealengine.com/en-US/download?sessionInvalidated=true
DataSet	Conjunto de datos del mapa Landscape.	https://github.com/Microsoft/AutonomousDrivingCookbook
Tensorflow	Plataforma para aprendizaje automático	https://www.tensorflow.org/install/pip#windows
Keras	Librerías para crear redes neuronales	https://keras.io/#installation
Microsoft Visual Studio	Cliente Python	https://visualstudio.microsoft.com/es/downloads/
OpenAI Gym	API para entornos y algoritmos	https://github.com/openai/gym
Keras-RL	Algoritmos DRL	https://github.com/keras-rl/keras-rl

Fuente: Elaboración propia.

Se debe configurar el entorno, instalando las siguientes dependencias de Python:

- Keras: Nos permitirá definir, entrenar y usar modelos, para la creación de un modelo CNN para el procesamiento de las imágenes de entrada y también se puede utilizar Keras en combinación con módulos de TensorFlow (Chollet, 2022).
- Tensorflow: Esta plataforma de aprendizaje profundo que permite construir y entrenar redes neuronales (Team, 2022).

El paquete del simulador y el conjunto de datos para el modelo se encuentran disponible en los enlaces de la Tabla 7.

6.2. ESTRUCTURA DEL SISTEMA SOFTWARE

La configuración experimental se construyó de la siguiente manera: en la primera etapa, se usó el motor de juego Unreal Engine con el mapa Landscape Mountains, el cual es un mapa con entorno rural. La plataforma AirSim proporciona al agente las API necesarias para el control del vehículo (dirección, aceleración), la conexión con el entorno, captura y procesamiento de las imágenes de la cámara frontal del vehículo.

La segunda etapa consiste en integrar los componentes software de las herramientas de Aprendizaje por Refuerzo Profundo (Keras, Keras-RL, OpenAI Gym) a la plataforma AirSim. Una vez instaladas las dependencias de Python necesarias para su uso, se crea un entorno

personalizado que llama a las API de AirSim para realizar las tareas de conexión, captura de imágenes y control del vehículo. Keras-RL funciona con las herramientas del entorno de Gym, lo que facilita la implementación del algoritmo DRL y su interacción con el entorno personalizado de AirSim. De esta manera, podremos realizar el entrenamiento del agente DQN para la conducción autónoma.

La red propuesta examina el entorno de simulación virtual (mapa Landscape Mountains), el modelo utiliza imágenes secuenciales como entradas que son tomadas de la cámara frontal del vehículo, el modelo CNN fue construido con las funcionalidades de Keras cuya arquitectura de red es una combinación de capas de convolucionales que procesan las imágenes, en la última capa densa se integra el ultimo estado del vehículo.

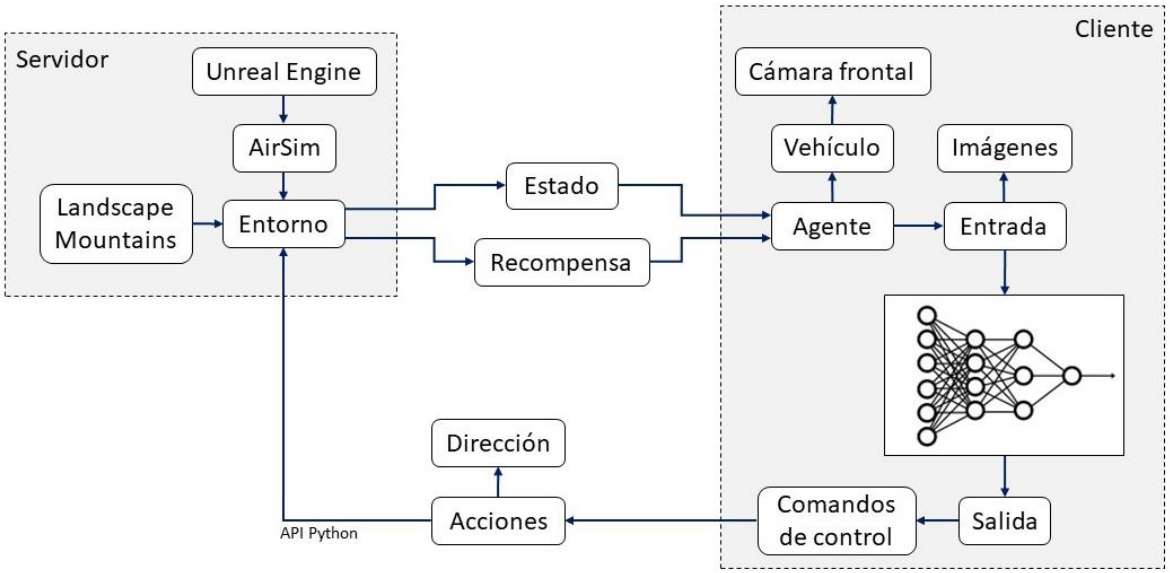


Figura 17: Marco general del proyecto.

Fuente: Elaboración propia.

Una vez instalados los paquetes software e integrados los componentes ya mencionados, se tiene un marco general de la propuesta de implementación de DRL en vehículos autónomos. La Figura 17 muestra el marco general de la propuesta con el esquema integrado de la plataforma de simulación AirSim con un entorno virtual del mapa rural y la integración de algoritmos DRL para la conducción autónoma.

En primer lugar, tenemos el lado del cliente que consta del agente que va a interactuar con el entorno, este posee una cámara frontal con la cual se adquirirá las imágenes para que el modelo DRL pueda realizar las predicciones necesarias para la navegación. Dichas imágenes serán la entrada para el modelo convolucional de tres capas desarrollado en Keras, en la última capa de esta red neuronal se tiene una única salida, cuyos datos serán la

entrada para el algoritmo DQN. Este agente DQN estará configurado con algunos parámetros como: política, memoria, etc. La salida del modelo DRL se ejecutarán el comando de control, en este caso de dirección del vehículo.

En el lado del servidor, tenemos nuestro entorno rural que consta de un mapa fotorrealista implementado en plataforma de simulación AirSim y el motor grafico Unreal Engine. Las acciones tomadas por nuestro agente, serán efectuadas con la API integrada en el simulador. A partir de esto, el modelo DRL toma las observaciones del entorno y calcula la recompensa dada por las acciones realizadas por el vehículo cuando navega en la carretera.

El proyecto está dividido en dos etapas software: la primera es la etapa del entrenamiento del agente, en donde se realiza la configuración de componentes, creación de entorno AirSim, entrenamiento y obtención del modelo DQN. La segunda etapa consta de la evaluación del modelo DRL en simulación en el entorno virtual con el mapa Landscape Mountains, en la cual se establece la conexión cliente-entorno y visualización de los comandos de control realizados en simulación. Estas etapas de desarrollo de software se detallan en los dos siguientes apartados.

6.2.1. Entrenamiento del Agente

Para realizar el entramiento del agente se debe configurar algunos componentes, para el objetivo de este trabajo se requiere resolver algunas tareas específicas en nuestro script de Python en VSC, las cuales están detalladas a continuación:

Entorno de AirSim: Gym proporciona una variedad de entornos para entrenar a un agente de aprendizaje por refuerzo. También se puede crear nuestro propio entorno para entrenar a un agente, tomando en cuenta que debe incluir algunos métodos como (Ravichandiran, 2020):

- Step(): Ejecuta una acción al devolver lo siguiente:
  - next_state: Nuevo estado del entorno como resultado del paso.
  - reward: Recompensa por realizar una acción.
  - info: Información relacionada con el sistema.
- Reset(): Reestablece el proceso devolviendo un nuevo estado.
- Render(): Visualiza cada paso.

Para el propósito de este proyecto se requiere crear un entorno personalizado, se usó un entorno basado en Unreal Engine como motor grafico para el mapa Landscape Mountains, AirSim proporciona las API necesarias para el control del vehículo y la adquisición de imágenes.

Modelo de la red neuronal: Se define un modelo de Keras simple que tiene una arquitectura de dos capas densas con funciones de activación 'Relu'.

Terminada esta etapa, se procede a crear el agente DQN usando el modelo anterior y configuramos lo siguiente:

Memoria Keras-RL: La librería posee una clase llamada rl.memory que proporciona una estructura de datos en la que se puede almacenar de manera secuencial varios estados, acciones y recompensas del agente. El hiperparámetro que se necesita especificar es un tamaño máximo de memoria para este objeto. Cuando se agregan nuevas experiencias, la memoria se llena y se "olvidan" las viejas experiencias. El código es el siguiente:

$$\text{memory} = \text{SequentialMemory}(\text{limit}=50000, \text{window\_length}=1)$$

Política Keras-RL: Es una clase de Política Q llamada rl.policy que pueden ser usadas para la exploración y explotación. El código es el siguiente:

$$\text{policy} = \text{LinearAnnealedPolicy}(\text{EpsGreedyQPolicy}(), \text{attr}='\text{eps}', \text{value\_max}=1., \text{value\_min}=.1, \text{value\_test}=.05, \text{nb\_steps}=10000)$$

Agente: existe una clase denominada rl.agents.dqn con la que podemos definir nuestro agente DQN habiendo ya configurado el modelo, memoria y política. El código es el siguiente:

$$\text{dqn} = \text{DQNAgent}(\text{model}=\text{model}, \text{nb\_actions}=\text{num\_actions}, \text{memory}=\text{memory}, \text{nb\_steps\_warmup}=10, \text{target\_model\_update}=1\text{e-}2, \text{policy}=\text{policy})$$

Entrenamiento Keras-RL: La librería proporciona varias devoluciones de llamadas que permiten el registro y control de los modelos a entrenar. El entrenamiento se realiza con el siguiente código:

$$\text{dqn} = \text{build\_agent}(\text{model}, \text{actions})$$

$$\text{dqn.compile}(\text{Adam}(\text{lr}=1\text{e-}3), \text{metrics}=['\text{mae}'])$$

$$\text{dqn.fit}(\text{env}, \text{nb\_steps}=60000, \text{visualize}=\text{False}, \text{verbose}=1)$$

La calidad de los resultados del agente DQN depende del tiempo de entrenamiento del modelo, el parámetro nb_steps puede ser modificado para generar mejores respuestas.

En la Figura 18, se presenta de manera resumida la creación de un Agente DQN usando Keras-RL y OpenAI-Gym. Primero se presenta la arquitectura del modelo, luego se define la memoria de red, la política de exploración y finalmente se entrena al agente.

El agente utilizara una red neuronal sencilla cuya arquitectura consta de tres capas de 16, 32, 32 respectivamente. La entrada será una imagen RGB y tendrá una neurona como salida que predecirá el valor Q de la acción en cada paso. Keras-RL proporciona las clases necesarias para crear el agente. Primero se crea la variable “memory”, la cual proporcionara la estructura de datos para almacenar las experiencias del agente. Luego añadimos la política Q “EpsGreedyQPolicy()” que permitirá equilibrar la exploración y explotación, la cual selecciona una acción aleatoria con Eps=0.1 de probabilidad, esto hace que el agente explore el entorno y luego se realice las acciones de acuerdo a lo que conoce.

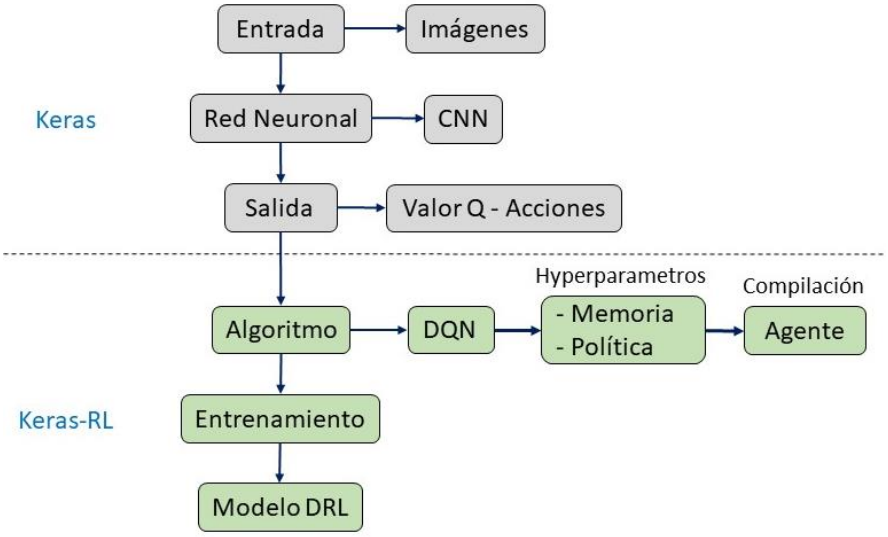


Figura 18: Creación del Agente DQN.

Fuente: Elaboración propia.

Definidos el modelo, la memoria y política, estos objetos se deben enviar al agente DQN. Keras-RL proporciona la clase “rl.agents.dqn”, con la cual se configura los parámetros como:

- “nb_steps_warmup” = determina el tiempo que se debe esperar, para producir la experiencia.
- “target_model_update” = En este algoritmo, la función Q es recursiva y cuando el agente actualiza su red para Q(s, a), esa actualización también afecta a la predicción que hará para el siguiente estado.

Se procede a construir las devoluciones de llamada “callback” que permiten el registro y control del modelo. Finalmente se realiza el proceso de entrenamiento del modelo con el método “fit()”. Con esto realizamos que el agente interactúe con el entorno virtual a medida que realiza acciones de control.

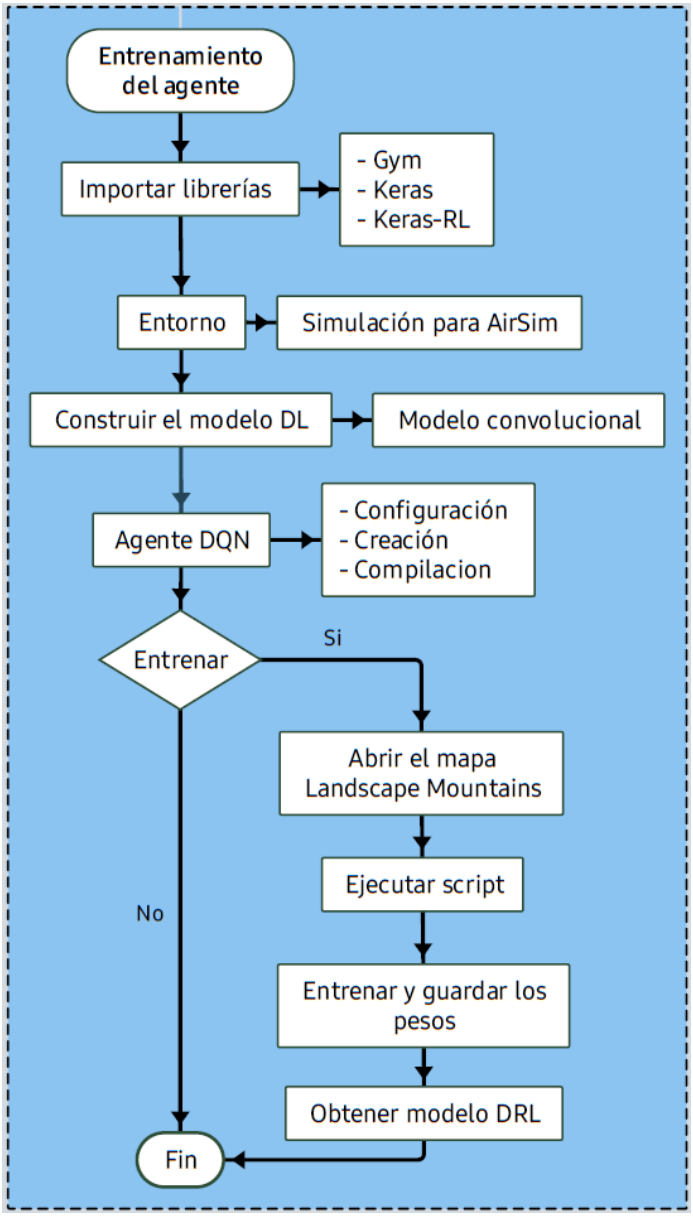


Figura 19: Proceso de entrenamiento del Agente DQN.

Fuente: Elaboración propia.

En la Figura 19 se presenta de forma resumida los pasos de la configuración de componentes software para la creación del agente DRL (DQN) usando Keras, Keras-RL y Gym. Como resultado de esta etapa de entrenamiento, obtendremos un archivo con formato .h5f en el cual tendremos nuestro modelo DQN y podremos evaluar en simulación.

6.2.2. Evaluación del Agente

Para la evaluación del agente se ejecuta otro script Python en Visual Studio Code, en este script se crea un "cliente" con las API de AirSim y se usa algunas funciones que permitan interactuar con el vehículo en la simulación mediante la programación en Python. Algunos

de estos procesos se presentar en el diagrama de flujo de la Figura 20 y se detallan cada uno de estos procesos.

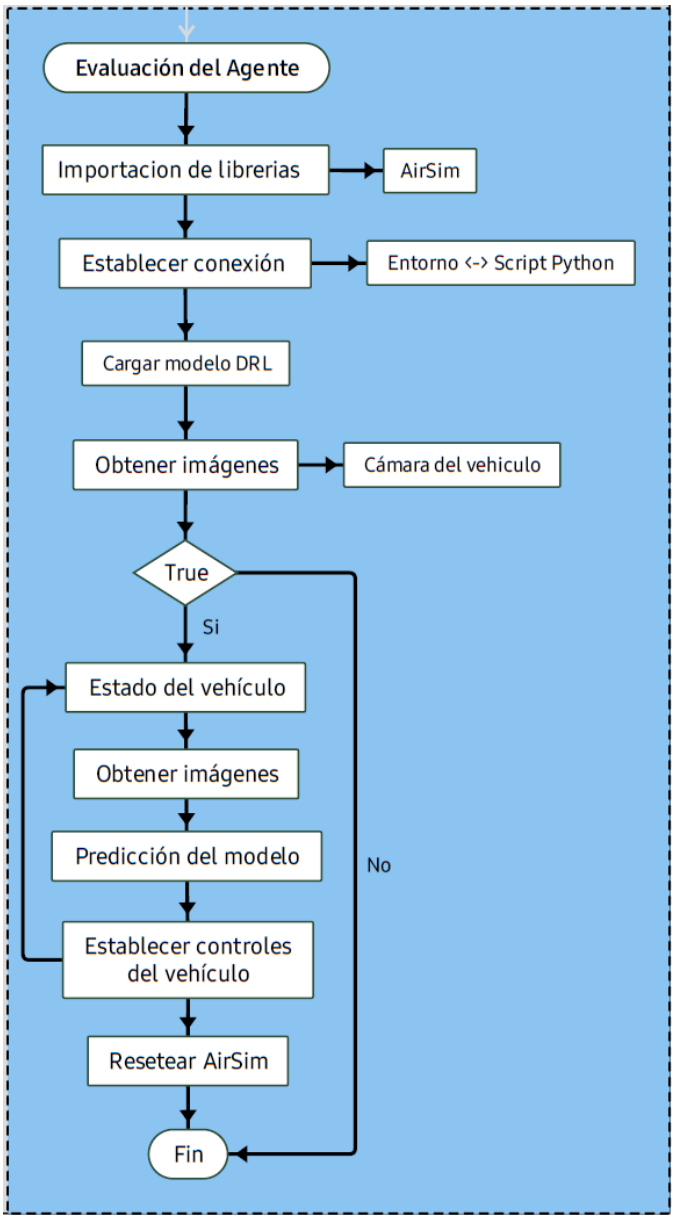


Figura 20: Proceso de evaluación del Agente DQN en simulación.

Fuente: Elaboración propia.

Antes de ejecutar este programa se requiere abrir el mapa Landscape Mountains, en el Script Python del cliente, primero se importa las librerías necesarias para la ejecución del programa, luego se usa la función que permita establecer la conexión del simulador con el cliente, a continuación se carga el modelo DRL con el agente DQN entrenado, se crea una función para obtener las imágenes del entorno de simulación por medio de la cámara frontal del vehículo, dentro un ciclo infinito se realiza lo siguiente:

1) Se verifica el estado del vehículo,
2) Se lea las imágenes del simulador,
3) Se realiza las predicciones del ángulo de giro acuerdo al modelo DRL, y
4) Se establece los comandos de control del vehículo mientras navega en la carretera del mapa.

Durante la simulación se abrirá un terminal de consola en la cual podremos observar los valores de los comandos de control enviados al vehículo.

6.3. PRUEBAS DE SIMULACIÓN

Antes de mostrar los resultados finales de la implementación del sistema, primero se aborda algunos detalles de la plataforma y fases que se realizó. Ya se mencionó que Airsim es un simulador para drones y automóviles construido como un complemento de Unreal Engine, es una plataforma de código abierto y admite software para simulaciones realistas tanto física como visualmente. A demás permite realizar simulación con control remoto (CR), controlando el dron o vehículo. Para interactuar con el vehículo en la simulación, AirSim dispone de API disponibles en los varios lenguajes (C++, Python, C# y Java), funciones con las cuales podemos recuperar imágenes, obtener el estado, control del vehículo, etc. Estas API están disponible como parte de una biblioteca multiplataforma, por lo que se puede probar códigos del programa en el simulador y luego ejecutarlo en vehículos reales (Microsoft, 2022).

Para realizar el entrenamiento de un modelo DL, existen dos formas de adquirir imágenes para la red neuronal, en la primera los datos son generados desde AirSim y capturados con el botón de grabación en la esquina inferior derecha de la pantalla, tal como se muestra en la Figura 21, y se comenzará a escribir las imágenes para cada fotograma.

En la segunda, los datos de entrenamiento son generados mediante el uso de las API, de esta forma se tiene el control completo de donde y cuando registrar estos datos. AirSim posee el modo de visión artificial, en el cual se usa el teclado para moverse por la escena y colocar las cámaras en el cualquier lugar del vehículo y recopilar imágenes con características de profundidad, disparidad, superficies normales o segmentación de objetos. Landscape Mountains es un entorno montañoso y nevado, cuyo terreno tiene un tamaño aproximado de 5 km por 5 km y consta de valles, picos altos, lagos congelados y bosques de cientos árboles

Figura 21: Escena del entorno Landscape Mountain simulado en AirSim.

Fuente: Elaboración a partir de AirSim.

El simulador también tiene opciones para controlar los efectos del clima, el control se lo puede realizar usando las API disponibles o accediendo a las opciones mediante el teclado (F1). Para acceder a las opciones del clima mediante la tecla F10, en la Figura 22 se muestra la configuración disponible.

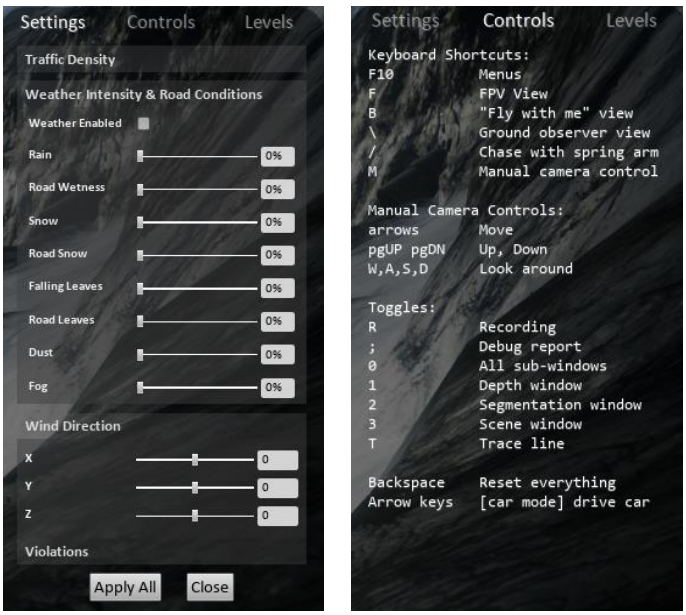


Figura 22: Configuración/control del entorno.

Fuente: Elaboracion a partir de AirSim.

El agente se comunica con la escena simulada a través de la API de AirSim que proporciona comandos para el control como dirección y aceleración del vehículo autónomo para que se

mueva en el camino. En la Figura 23, se presenta los datos que son mostrados durante la simulación.

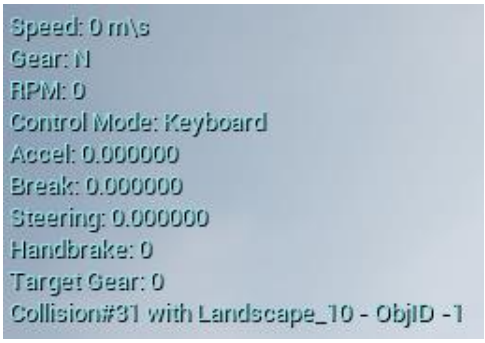


Figura 23: Estado/Control del vehículo.

Fuente: Elaboración a partir de AirSim.

Antes de realizar la simulación de conducción autónoma con el agente DQN, se procede a realizar una prueba de conexión del cliente con el simulador. Se abre el archivo con el mapa de Landscape Mountains y el script Python “HelloCar”, al ejecutar el simulador, se iniciará el control en modo manual y el usuario puede controlar la dirección de giro del vehículo mediante el teclado.

Para realizar el entrenamiento del modelo DRL, el agente interactúa directamente en el entorno y procesa todos los datos en tiempo real (TR) usando la red neuronal CNN y el algoritmo DQN y así obtener un agente para la conducción autónoma.

7. Resultados finales

Terminadas las etapas de diseño, desarrollo y simulación en la plataforma AirSim del agente para la conducción autónoma, se presenta los resultados obtenidos de la implementación, mencionando algunas características importantes de los modelos.

Modelo DL

- Este modelo tiene una arquitectura de red cuya combinación de capas de agrupación convolucionales que procesan las imágenes de entrada, en la capa densa de la red se inyecta el ultimo estado conocido del vehículo.
- La Región de Interés (ROI) de la imagen está establecida de acuerdo para el caso de uso este problema en particular, navegar en una carretera capturando imágenes con la cámara frontal del vehículo y que solo se toma en cuenta la parte inferior de la imagen que corresponde a la carretera. Por lo tanto, este modelo solo se centrará en la carretera y no tomará en cuenta lo suceda con el resto del entorno. Reduciendo el

tamaño de la imagen, facilita el entrenamiento de la red neuronal a partir del conjunto de datos.

- En el procesamiento de datos para el entrenamiento del modelo de red neuronal profundo con capas convolucionales, existen dos estrategias de conducción para el modelo DL, el primero se denomina conducción normal en que casi no se produce ángulos de dirección debido a la conducción en línea recta. La segunda estrategia de conducción "con desvió" se centra en la realización de giros cerrados con ángulos más grandes. Esto lo podemos resumir tal como se muestra en la Figura 24.

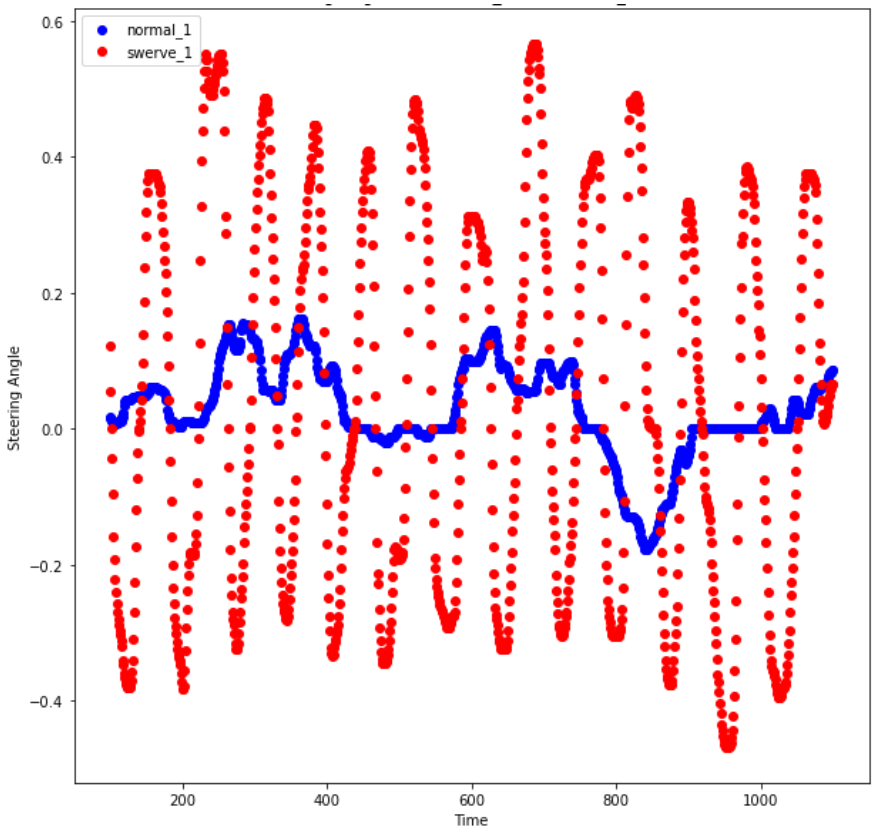


Figura 24: Comparación de las estrategias de conducción debidos a sus ángulos.

Fuente: Elaboración a partir de (Koul et al., 2019).

- En resumen, el modelo DL: se centra solo en el ROI de cada imagen, se puede modificar el conjunto de datos modificando algunos parámetros de las imágenes como el brillo, simulando diferentes condiciones de iluminación y girando las imágenes horizontalmente e invirtiendo así el ángulo de giro para el procesamiento de las imágenes.
- La arquitectura de la red neuronal consta de tres capas convolucionales con 16, 32, 32 filtros y una ventana convolucional (3, 3). La salida de las capas convolucionales consta de las características de las imágenes y la capa de entrada proporciona el

estado anterior del automóvil. La función de activación de la red es ReLu. La salida de la red es el ángulo de dirección. Un valor con punto flotante.

Modelo DRL

Los resultados obtenidos están basados en la implementación de una red neuronal convolucional y la aplicación del algoritmo DQN. El objetivo del agente DQN es minimizar la distancia desde el centro de la carretera y así maximizar la recompensa.

Figura 25: Simulación del modelo en AirSim.

Fuente: Elaboración a partir de la simulación.

En la Figura 25, se observa la simulación realizada en AirSim, con el terminal en donde se envían los comandos de control del vehículo. Se observa tanto en la simulación como en los comandos de control mostrados en el terminal que el movimiento del automóvil no es uniforme y esto se debe que el modelo realiza la predicción del ángulo de giro para cada cuadro de imagen que obtiene el cliente.

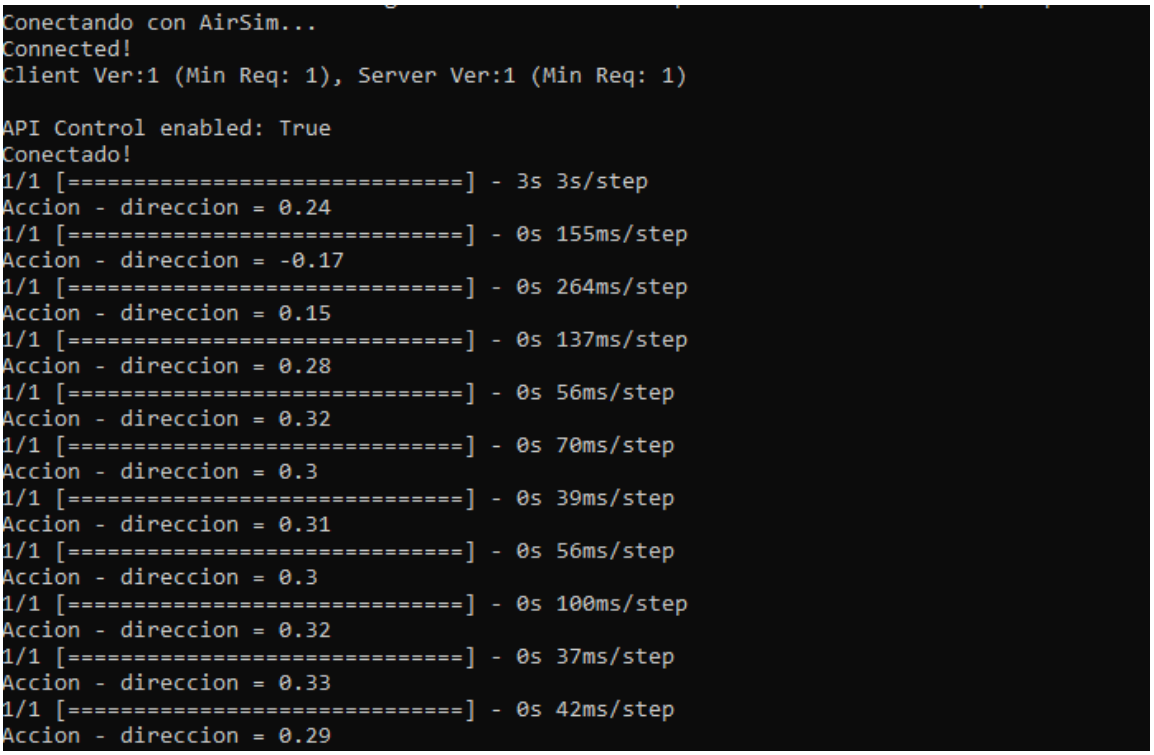


Figura 26: Salida de terminal de comandos.

Fuente: Elaboración a partir de la simulación.

En la Figura 26, se presenta la salida con los datos de las acciones enviadas al agente para que se movilice en la carretera, dichas acciones son predichas por el modelo DRL compuesto por redes neuronales y algoritmo DQN.

8. Conclusiones y trabajos futuros

Este capítulo presenta algunas conclusiones obtenidas a partir del diseño, desarrollo, simulación y resultados de implementación del modelo DRL en conducción autónoma de vehículos en la plataforma AirSim.

8.1. CONCLUSIONES

En esta investigación, se ha propuesto la comparación de técnicas de DRL con DL para la navegación en carretera en entornos rurales, la implementación fue realizada en la plataforma de simulación virtual Landscape Mountain usando el cliente desarrollado las API Python dadas por AirSim que se desempeña en un entorno virtual realista como un agente de vehículo. La plataforma de simulación AirSim permitió probar el modelo en un entorno virtual, recopilando información por medio de imágenes tomadas del camino con la cámara central.

Respecto al objetivo principal, este se ha alcanzado tras la implementación en software de un sistema de conducción autónoma con algoritmo de Aprendizaje por Refuerzo Profundo usando DQN para la creación de un agente realizado con la librería Keras-RL, este interactúa con un entorno de simulación en la plataforma AirSim. En lo relativo a las tareas de apoyo en rescate, se ha presentado una nueva propuesta para innovar los procesos que son realizados por los equipos de rescate integrando una tecnología que aún no es implementada en el país para este tipo de casos.

Los objetivos secundarios se alcanzaron, dado que se realizó el estudio pertinente al área de conducción autónoma de vehículos, utilizando la metodología SLR que permitió obtener información relevante sobre el estado del arte de VA, algoritmos, frameworks, librerías y plataformas de simulación obteniendo de este proceso con las siguientes conclusiones, el área de conducción autónoma es bastante extensa en cuanto a aplicabilidad, enfoques de diseño, herramientas disponibles para el desarrollo y retos que existen en el terreno de trabajo que deben ser solucionados. Se diseñó un sistema de conducción autónoma basado en la arquitectura de software que ofrece la plataforma AirSim y se usó el algoritmo de DQN que integra las funcionalidades de una red neuronal profunda y la capacidad de entrenar al agente directamente en el entorno simulado sin la necesidad de un conjunto de datos. Pudo evaluarse satisfactoriamente el funcionamiento del modelo, no sin encontrar diversas dificultades debido a limitaciones en el hardware, inconvenientes típicos de instalación de paquetes software como versión de paquetes de software, uso de recursos de PC, etc., donde la limitación del hardware para este trabajo jugó un papel determinante, pero pudo resolverse. Por lo tanto, se allanó el camino para alcanzar un sistema de conducción autónoma con un diseño específico y estudio al efecto para vehículos de rescate, tal como se planteó en este trabajo.

Los vehículos autónomos deben dar buenas respuestas para los cambios en el terreno por el que se desplazan, en este caso en una carretera, pero también deben estar preparados para los cambios en las condiciones externas como el clima y variación de iluminación. AirSim ofrece crear varias condiciones climáticas para los entornos en los que requieran recopilar imágenes para un conjunto de datos de entrenamiento o la simulación de un modelo DL o DRL en un entorno fotorrealista.

Para el entrenamiento del modelo DL se requiere que el conjunto de datos contenga datos para una estrategia de conducción normal en línea recta sino también datos para una estrategia de conducción para realizar giros cuando se desplaza en la carretera.

El sistema propuesto está desarrollado en una arquitectura basada en software, cuyas funcionalidades tienen soporte en la estructura modular de AirSim, esta plataforma nos

brindó las capacidades necesarias para simular el entorno rural e integrar un agente DRL de conducción autónoma para la navegación de carreteras.

Con la aplicación de algoritmos DRL en específico DQN se resuelve el desafío de navegación autónoma, dando así un ejemplo de conducción autónoma para un vehículo, desarrollando un modelo predictivo que permita tomar acciones de control de dirección en un entorno rural con variaciones de clima.

A diferencia de un modelo de aprendizaje profundo que necesita un conjunto de datos para entrenar la red neuronal, el modelo DQN de aprendizaje por refuerzo profundo requiere implementar un entorno virtual para que nuestro agente pueda entrenarse realizando acciones y aprendiendo de la experiencia; sin embargo, existen otros métodos de aprendizaje que podemos aplicar para generar el modelo DRL, uno de estos consiste en el método de Transferencia de Aprendizaje en el cual se aprovecha ya un modelo DL entrenado y luego se entrena solo la red DQN con dos capas densas para la predicción del ángulo de giro, posteriormente se transfiere los pesos de las capas convolucionales a la red DQN. Esto reduce en gran medida el tiempo de entrenamiento y recurso hardware a usar durante este proceso.

Durante la evaluación del agente DQN, se pudo observar que el modelo de aprendizaje por refuerzo profundo durante el proceso de entrenamiento, mientras más iteraciones realiza mejores resultados se obtiene, esto debido que el agente tendrá más tiempo para ajustar parámetros como la recompensa por la ejecución de las acciones.

Actualmente las tareas de búsqueda y rescate en Ecuador, las realizan instituciones como Policía Nacional, Ejercito y Cuerpo de Bomberos bajo la tutela de Servicio Nacional de Gestión de Riesgos y Emergencias (Ecuador, 2022), que utilizan como recursos a personas y canes entrenados, sin embargo, debido a la modernización de las tecnologías existentes se están incluyendo drones para las tareas de búsqueda. Como uno de los objetivos de este trabajo es ofrecer un sistema de conducción autónoma que mejore los sistemas de transporte, ya finalizado este trabajo se da por sentada la base para continuar con la línea de investigación amplia y poco desarrollada en el país.

8.2. LÍNEAS DE TRABAJO FUTURAS

Se plantea mejorar el rendimiento del algoritmo DRL modificando la arquitectura del sistema de conducción autónoma, en específico el modelo de red neuronal convolucional (CNN), de igual manera realizar un análisis comparativo con otros algoritmos DRL y verificar que modelo presenta mejores resultados de simulación.

Los sistemas de conducción autónoma poseen una arquitectura compleja debido a todos los componentes involucrados, se pretende integrar a nuestro sistema un sensor LIDAR y poder realizar más experimentos de simulación con diferentes condiciones climáticas y diferentes entornos y obstáculos en la carretera.

El alcance de este trabajo fue implementar un agente de aprendizaje por refuerzo profundo en la plataforma de simulación AirSim. Sin embargo, la línea de investigación de vehículos autónomos no se limita solo a entornos simulados. Podemos aprovechar la escalabilidad y flexibilidad de AirSim implementando dichos agentes en sistemas físicos reales, alcanzando este punto tendríamos tres opciones:

1) Logrando un sistema de conducción autónoma muy eficiente, se puede mejorar las capacidades de nuestros modelos DRL con la integración de protocolos de comunicación con otros vehículos autónomos, CAV (vehículos autónomos y conectados). De esta manera, se prevé mejorar la coordinación en las tareas de búsqueda y rescate en terrenos extensos, donde un solo vehículo tendría limitaciones para realizar dichas tareas.
2) Además de algoritmos de navegación, para mejorar las funcionalidades de conducción del vehículo, se pueden integrar algoritmos de planificación de rutas y reconocimiento de objetos mientras el vehículo este navegando en el terreno a fin de brindar un mejor soporte a las unidades de socorro.
3) Agregar un sistema de monitoreo en tiempo real, para que el operador tenga información sobre: ubicación y estado del sistema de conducción.

Microsoft Research creo la plataforma AirSim para la investigación y experimentación de modelos de conducción autónoma y desde su lanzamiento en 2017, se ha desarrollado muchos avances de en el área de IA y debido que van a lanzar una nueva plataforma de simulación, Microsoft Project AirSim (Research, 2022) en el 2023, se espera continuar con esta línea de investigación mejorando los algoritmos de desarrollo software para la conducción autónoma.

Referencias bibliográficas

Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L. D., Monfort, M., Muller, U., & Zhang, J. (2016). End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*.

Carton, F. (2021). *Exploration of reinforcement learning algorithms for autonomous vehicle visual perception and control* Institut polytechnique de Paris].

Chollet, F. (2021). *Deep learning with Python*. Simon and Schuster.

Chollet, F. (2022). *API Keras*. https://keras.io/

Dong, H., Dong, H., Ding, Z., Zhang, S., & Chang. (2020). *Deep Reinforcement Learning*. Springer.

Dong, H. Z., Shanghang. (2022). *TensorLayer*. https://tensorlayer.readthedocs.io/en/latest/#

Ecuador, G. d. (2022). *Servicio Nacional de Gestión de Riesgos y Emergencias*. https://www.gestionderiesgos.gob.ec/

Fadaie, J. (2019). The state of modeling, simulation, and data utilization within industry: An autonomous vehicles perspective. *arXiv preprint arXiv:1910.06075*.

Fan, R., Jiao, J., Ye, H., Yu, Y., Pitas, I., & Liu, M. (2019). Key ingredients of self-driving cars. *arXiv preprint arXiv:1906.02939*.

González, D., Pérez, J., Milanés, V., & Nashashibi, F. (2015). A review of motion planning techniques for automated vehicles. *IEEE Transactions on intelligent transportation systems*, *17*(4), 1135-1145 %@ 1524-9050.

Gridin, I. (2022). *Practical Deep Reinforcement Learning with Python - Concise Implementation of Algorithms, Simplified Maths, and Effective Use of TensorFlow and PyTorch*.

Gupta, S., Singal, G., & Garg, D. (2021). Deep reinforcement learning techniques in diversified domains: a survey. *Archives of Computational Methods in Engineering*, *28*(7), 4715-4754 %@ 1886-1784.

Gutta, S. (2021). *Datasets for Machine Learning in Autonomous Vehicles*. https://medium.com/analytics-vidhya/datasets-for-machine-learning-in-autonomous-vehicles-dd13bae5925b

Habibian, S., Dadvar, M., Peykari, B., Hosseini, A., Salehzadeh, M. H., Hosseini, A. H. M., & Najafi, F. (2021). Design and implementation of a maxi-sized mobile robot (Karo) for rescue missions. *Robomech Journal*, *8*(1), 1-33 %@ 2197-4225.

Ivanov, S., & D'Yakonov, A. (2019). Modern deep reinforcement learning algorithms. *arXiv preprint arXiv:1906.10025*.

Kang, Y., Yin, H., & Berger, C. (2019). Test your self-driving algorithm: An overview of publicly available driving datasets and virtual testing environments. *IEEE Transactions on Intelligent Vehicles*, *4*(2), 171-185 %@ 2379-8904.

Kaur, P., Taghavi, S., Tian, Z., & Shi, W. (2021). A survey on simulators for testing self-driving cars.

Koul, A., Ganju, S., & Kasam, M. (2019). *Practical Deep Learning for Cloud, Mobile, and Edge: Real-World AI & Computer-Vision Projects Using Python, Keras & TensorFlow*. O'Reilly Media.

Lafuente-Arroyo, S., Gil-Jimenez, P., Maldonado-Bascon, R., López-Ferreras, F., & Maldonado-Bascon, S. (2005). Traffic sign shape classification evaluation I: SVM using distance to borders.

Legacy, C., Ashmore, D., Scheurer, J., Stone, J., & Curtis, C. (2019). Planning the driverless city. *Transport reviews*, *39*(1), 84-102 %@ 0144-1647.

Maurer, M., Gerdes, J. C., Lenz, B., & Winner, H. (2016). *Autonomous driving: technical, legal and social aspects*. Springer Nature.

Mertens, J. (2018). *Generating Data to Train a Deep Neural Network End-To-End within a Simulated Environment* Freie Universität Berlin].

Microsoft. (2018). *The Autonomous Driving Cookbook (Preview)*. https://github.com/microsoft/AutonomousDrivingCookbook#the-autonomous-driving-cookbook-preview

Microsoft. (2022). *AirSim Simulator*. https://microsoft.github.io/AirSim/

Niroui, F., Zhang, K., Kashino, Z., & Nejat, G. (2019). Deep reinforcement learning robot for search and rescue applications: Exploration in unknown cluttered environments. *IEEE Robotics and Automation Letters*, *4*(2), 610-617 %@ 2377-3766.

OICA. (2021). *World vehicles in use by country/region and type, 2005-2015. The International Organization of Motor Vehicle Manufacturers*. http://www.oica.net/category/vehicles-in-use/

Park, J.-H., Farkhodov, K., Lee, S.-H., & Kwon, K.-R. (2022). Deep reinforcement learning-based DQN agent algorithm for visual object tracking in a virtual environmental simulation. *Applied Sciences*, *12*(7), 3220 %@ 2076-3417.

Pérez-Gil, O., Barea, R., López-Guillén, E., Bergasa, L. M., Gómez-Huelamo, C., Gutiérrez, R., & Díaz-Díaz, A. (2022). Deep reinforcement learning based control for Autonomous Vehicles in CARLA. *Multimedia Tools and Applications*, 1-24 %@ 1573-7721. https://doi.org/10.1007/s11042-021-11437-3

Raffin, A. (2021). *Stable-Baselines3: Reliable Reinforcement Learning Implementations*. https://araffin.github.io/post/sb3/

Rana, M., & Hossain, K. (2021). Connected and Autonomous Vehicles and Infrastructures: A Literature Review. *International Journal of Pavement Research and Technology*, 1-21 %@ 1997-1400.

Ravichandiran, S. (2020). *Deep Reinforcement Learning with Python*. Packt Publishing.

Research, M. (2022). *Project AirSim*. https://www.microsoft.com/en-us/ai/autonomous-systems-project-airsim?activetab=pivot1:primaryr3

Rosique, F., Navarro, P. J., Fernández, C., & Padilla, A. (2019). A systematic review of perception system and simulators for autonomous vehicles research. *Sensors*, *19*(3), 648 %@ 1424-8220.

SAE. (2021). Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles. Technical report J3216_202107 : SAE Internacional 2021. In.

Sánchez-Ibáñez, J. R., Pérez-del-Pulgar, C. J., & García-Cerezo, A. (2021). Path Planning for Autonomous Mobile Robots: A Review. *Sensors*, *21*(23), 7898. https://doi.org/10.3390/s21237898

Shah, S., Dey, D., Lovett, C., & Kapoor, A. (2017). Aerial informatics and robotics platform. *Washigton: Microsoft Research*.

Shi, W., & Liu, L. (2021). *Computing Systems for Autonomous Driving*. Springer. https://doi.org/10.1007/978-3-030-81564-6

Silva, Ó., Cordera, R., González-González, E., & Nogués, S. (2022). Environmental impacts of autonomous vehicles: A review of the scientific literature. *Science of The Total Environment*, 154615 %@ 150048-159697. https://doi.org/10.1016/j.scitotenv.2022.154615

Sun, H., Zhang, W., Yu, R., & Zhang, Y. (2021). Motion planning for mobile robots—Focusing on deep reinforcement learning: A systematic review. *IEEE access*, *9*, 69061-69081 %@ 62169-63536.

Talpaert, V., Sobh, I., Kiran, B. R., Mannion, P., Yogamani, S., El-Sallab, A., & Perez, P. (2019). Exploring applications of deep reinforcement learning for real-world autonomous driving systems. *arXiv preprint arXiv:1901.01536*.

Team, G. B. (2022). *TensorFlow*. https://www.tensorflow.org/

Torres-Carrión, P. V., González-González, C. S., Aciar, S., & Rodríguez-Morales, G. (2018). Methodology for systematic literature review applied to engineering and education.

UNIR, L. U. d. I. (2020). *4 metodologías para la gestión de proyectos que debes conocer*. https://www.unir.net/empresa/revista/metodologias-gestion-proyectos/

Van Brummelen, J., O'Brien, M., Gruyer, D., & Najjaran, H. (2018). Autonomous vehicle perception: The technology of today and tomorrow. *Transportation research part C: emerging technologies*, *89*, 384-406 %@ 0968-0090X.

Van Uytsel, S., & Vasconcellos Vargas, D. (2021). Autonomous Vehicles. *Springer, Singapore*, *1*, 5.

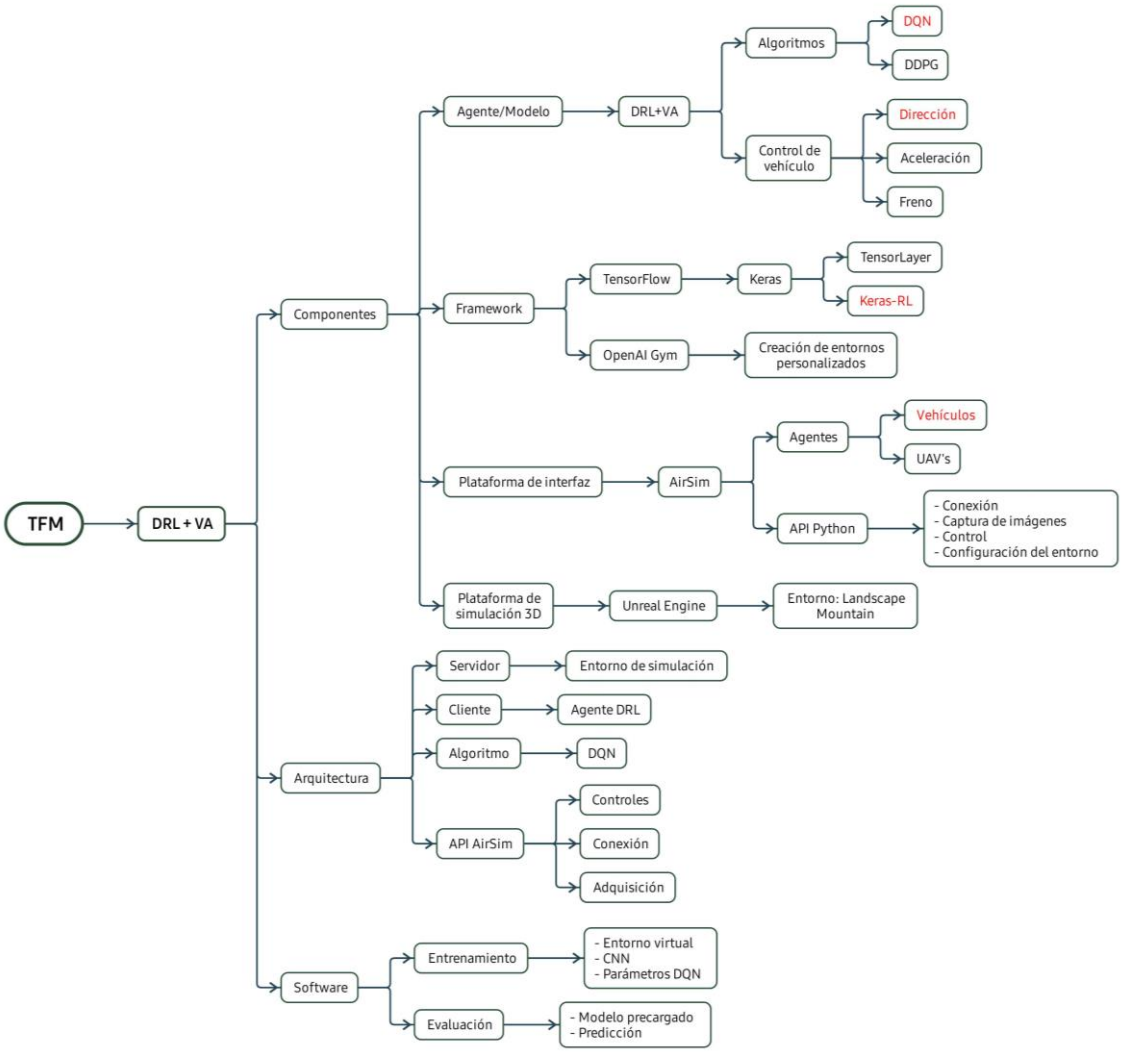
Zhang, Q., Wang, Y., Zhang, X., Liu, L., Wu, X., Shi, W., & Zhong, H. (2018). OpenVDAP: An open vehicular data analytics platform for CAVs.

Zhang, Q., Zhong, H., Cui, J., Ren, L., & Shi, W. (2020). AC4AV: a flexible and dynamic access control framework for connected and autonomous vehicles. *IEEE Internet of Things Journal*, *8*(3), 1946-1958 %@ 2327-4662.

Zong, W., Zhang, C., Wang, Z., Zhu, J., & Chen, Q. (2018). Architecture design and implementation of an autonomous vehicle. *IEEE access*, *6*, 21956-21970 %@ 22169-23536.

Anexo A.

Diagrama resumen de los componentes del Trabajo de Fin de Master:



Anexo B.

Código fuente implementado en el proceso de evaluación del Agente:

```
# DRL - VA
# Proceso de Evaluacion
# Autor: Roger Sarango
#--------------------------------------------------------------------------------
from multiprocessing.connection import Client
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
import sys
import glob
import airsim
import tensorlayer as tl
import cv2
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from keras.models import load_model

tf.keras.backend.clear_session()  # Para restablecer facilmente el estado del portatil.

#----- Cargar archivo modelo h5 -----------------------------------------------------
if ('C:/Program Files (x86)/AirSim/PythonClient/' not in sys.path):
    sys.path.insert(0, 'C:/Program Files (x86)/AirSim/PythonClient/')

MODEL_PATH = None
if (MODEL_PATH == None):
    models = glob.glob('C:/Users/RodgerS/Downloads/LandscapeMountains/modelo1.h5f')
    best_model = max(models, key=os.path.getctime)
    MODEL_PATH = best_model
print('Usando el modelo {0} para prueba'.format(MODEL_PATH))
model = load_model(MODEL_PATH)

#----- Conexion AirSim-Entorno (ejecutar primero el entorno y despues el script) ------
print("Conectando con AirSim...")
client = airsim.CarClient()
client.confirmConnection()
client.enableApiControl(True)
print("API Control enabled: %s" % client.isApiControlEnabled())
car_controls = airsim.CarControls()
print("Conectado!")

#----- Controles del carro ----------------------------------------------------------
car_controls.steering = 0
car_controls.throttle = 0
car_controls.brake = 0

image_buf = np.zeros((1, 59, 255, 3))
state_buf = np.zeros((1,4))

#----- Tomar imagenes del simulador con numpy ---------------------------------------
def get_image(client):
    image_response = client.simGetImages([airsim.ImageRequest("0", airsim.ImageType.Scene, False
    image_res = image_response[0]
    image1d = np.frombuffer(image_res.image_data_uint8, dtype=np.uint8)
    image_rgba = image1d.reshape(image_res.height, image_res.width, 3)

    return image_rgba[76:135,0:255,0:3]

```

```
image = get_image(client)
image1 = plt.imshow(image)

while (True):
    car_state = client.getCarState()

    if (car_state.speed < 5):
        car_controls.throttle = 1.0
    else:
        car_controls.throttle = 0.0

    image_buf[0] = get_image(client)
    state_buf[0] = np.array([car_controls.steering, car_controls.throttle, car_controls.brake, c
    model_output = model.predict([image_buf, state_buf])
    car_controls.steering = round(0.5 * float(model_output[0][0]), 2)

   # print('Enviando direccion = {0}, aceleracion = {1}'.format(car_controls.steering, car_contr
    print('Accion - direccion = {0}'.format(car_controls.steering))

#---- Restablecer al estador original ---------------------------------------------------------
client.reset()
client.enableApiControl(False)
```