Sistema de Navegación Autónoma para un agente(AV) en Unity usando ML-Agents y Deep Q Learning

Luis Coronell lcoronell@unal.edu.co*, Juan Barragán jbarragana@unal.edu.co†
*ingeniería mecatrónica, Universidad Nacional de Colombia
†ingeniería mecatrónica, Universidad Nacional de Colombia

***Resumen*—This project explores the use of reinforcement learning algorithms for autonomous driving in a simulated 3D environment developed with Unity. Three algorithms were implemented and compared: Proximal Policy Optimization (PPO), SARSA, and Q-Learning, with a primary focus on the first two due to the practical limitations of Q-Learning in high-dimensional state spaces. The agent was trained using Unity's ML-Agents toolkit in a closed-circuit track with raycast sensors to perceive the environment. PPO, being a policy-based approach, exhibited more stable and efficient behavior, while SARSA achieved a more conservative yet adaptable learning performance. The results demonstrate the feasibility of reinforcement learning in complex simulated environments and highlight key differences between policy-based and value-based methods for autonomous navigation tasks.**

***Index Terms*—Reinforcement Learning, Autonomous Driving, Simulated Environment, Unity ML-Agents, Proximal Policy Optimization (PPO), SARSA, Q-Learning, Deep Reinforcement Learning, Policy-Based Methods, Value-Based Methods.**

I. Introducción

En los últimos años, la detección de objetos ha tenido avances significativos para los vehículos de conducción autónoma (AV), ya que permite identificar y localizar diversos objetos en el entorno circundante en tiempo real, como peatones, ciclistas, señales de tráfico, etc., lo que proporciona una base importante para la toma de decisiones en la conducción autónoma y representa una mayor seguridad en el sector del transporte [1]. El gran interés en la innovación de algoritmos optimizados radica en su capacidad para detectar objetos pequeños en entornos dinámicos y cambiantes [2]. Sin embargo, es importante reconocer las limitaciones en precisión y velocidad de estos modelos [3]. Algunos modelos, como el iS-YOLOv5, aumentan la precisión media (mean Average Precision, mAP) en un 3,35 % con respecto a su version estandar YOLOv5, optimizando el flujo de información [4].

A partir de estos avances, las mejoras en la detección de objetos, desarrolló un algoritmo ligero basado en YOLOv5 con el objetivo de reducir la carga de procesamiento y aumentar la velocidad de detección [5]. Este modelo demostró ser más eficiente en términos de precisión y velocidad, seguido de cerca por el YOLOv8, que obtuvo una mayor precisión, aunque con un desempeño más lento que YOLOv5. En contraste, YOLOv7 presentó un rendimiento notablemente inferior [3], lo que resalta la importancia de seleccionar una arquitectura adecuada para los sistemas de detección. Asimismo, se han implementado modelos como SO-YOLOv8, que incorpora técnicas de aumento de datos y optimizaciones arquitectónicas para mejorar la detección de objetos pequeños [4] [6].

Para mejorar la robustez en la detección, se han desarrollado metodologías especializadas basadas en aprendizaje profundo [7]. Los algoritmos basados en este enfoque superan a los tradicionales en términos de velocidad y precisión de detección [8]. Uno de los métodos más eficientes es el DQN (Deep Reinforcement Learning), que permite al agente explorar entornos dinámicos y tomar decisiones óptimas mediante redes neuronales profundas que estiman los valores Q sin requerir datos de entrenamiento previos. Como mejora, el DDQN (Double Deep Q-Learning Network) resuelve la sobreestimación del valor Q utilizando dos redes —la Deep Q-Network y la Target Network—, lo que permite un aprendizaje más estable, rápido y preciso [9]. Además, la integración de técnicas como el aprendizaje por transferencia, redes neuronales convolucionales (CNN), redes profundas (DNN) y la fusión de sensores mejora significativamente la precisión y adaptabilidad del sistema [10].

Dentro de estos algoritmos de detección de objetos, existen modelos de aprendizaje profundo como AttenRetina, el cual mejora el enfoque en entornos de IoT y aprovecha de manera eficiente los datos multimodales, lo que permite sólidas capacidades de extracción de características, vitales para detectar objetos de todos los tamaños [1]. Sin embargo, este modelo utiliza una arquitectura modular de redes neuronales, mientras que existen arquitecturas más eficientes, como las de extremo a extremo. Esta última es atractiva debido a su estructura sencilla, que reduce la carga de diseñar módulos complejos [11]. En general, las arquitecturas de extremo a extremo son más sencillas y presentan menos componentes que las modulares [12].

El aprendizaje por refuerzo (RL), resuelve principalmente dos desafíos: la necesidad de grandes cantidades de datos etiquetados para el entrenamiento y la propagación del error desde el entorno. Debido a que este descompone el problema en varias partes, como la percepción del entorno a partir de los sensores, la planificación de rutas —por ejemplo, mantenerse en el carril— y el control del movimiento [11].

Uno de los modelos propuestos por Jiang et al. (2023) en-

frenta la toma de desiciones del agente de manera prometedora mediante redes convolucionales de grafos (GCN), utilizan la representación gráfica de alto nivel de las interacciones de la multitud y se formula la toma de decisiones de los agentes en multitud como un problema de aprendizaje por refuerzo (RL) [13]

En contraste, el trabajo de Lee, Choi y Park (2021) sobre el diseño e implementación de un robot de "palets" autónomo utilizando el Robot Operating System (ROS) detalla el uso de algoritmos de localización y mapeo, destacando cómo el robot puede trazar rutas para navegar de manera autónoma utilizando SLAM con Cartographer, demostrando que pueden integrarse de manera efectiva en sistemas robóticos personalizados, permitiendo una navegación precisa y eficiente en diferentes entornos. [14] [15]

Además de lo mencionado anteriormente, el Robot Operating System (ROS) es neutral en cuanto al lenguaje de programación y admite cuatro lenguajes principales: C++, Python, Octave y LISP. Esto lo hace más eficiente en términos de tiempo de programación, facilidad de depuración, eficiencia en tiempo de ejecución, entre otros beneficios [16].

En el contexto del control autónomo basado en aprendizaje por refuerzo profundo (Deep Reinforcement Learning, DRL), diversas investigaciones han explorado aplicaciones tanto en vehículos autónomos como en robótica móvil. Un enfoque notable propone el uso del algoritmo Soft Actor-Critic (SAC) para la toma de decisiones en entornos urbanos complejos, introduciendo un vector de destino concatenado con características extraídas mediante redes convolucionales, lo cual permite una navegación más fluida y segura en escenarios de tráfico denso, con validación en el simulador CARLA [17]. Otro trabajo aborda las limitaciones del algoritmo TD3 mediante una estructura de triple crítico y un promedio de pasos temporales múltiples, lo que mejora la estabilidad y precisión en decisiones de conducción autónoma, especialmente en giros sin protección y trayectorias en centros de carril [18]. Por otro lado, un tercer estudio realiza un análisis comparativo del desempeño de algoritmos de última generación (PPO, TD3 y SAC) en tareas de locomoción de robots cuadrúpedos, destacando cómo el tipo de entrada sensorial y el diseño del entorno influyen significativamente en la efectividad del aprendizaje y la robustez del control [19].

En línea con estos avances en aprendizaje por refuerzo profundo, también se ha considerado el algoritmo SARSA (State-Action-Reward-State-Action) para tareas de control en entornos dinámicos, particularmente por su enfoque on-policy que actualiza la política basada en la acción realmente tomada, lo cual resulta valioso en escenarios donde la retroalimentación inmediata y la adaptación continua son críticas [20].

En este marco, el proyecto se centra en la creación de un sistema que utiliza una cámara como sensor principal y aplica métodos de aprendizaje profundo para identificar obstáculos y diseñar rutas de manera eficaz, mientras reconoce y supera obstáculos en su entorno. Mediante el uso de avances en visión por computador, identificación de objetos y aprendizaje reforzado, se busca desarrollar, en una primera etapa, un entorno virtual simulado que permita validar el desempeño del sistema en navegación autónoma y evasión de obstáculos. Esta sólida solución integra percepción, organización y gestión en tiempo real, ya que la información en tiempo real sobre las trayectorias de objetos en movimiento es imprescindible para una navegación segura [21]. Este método no solo facilita una navegación adaptable en ambientes cambiantes, sino que también establece las bases para aplicaciones futuras como la robótica móvil y el transporte inteligente.

II. Metodología

Para la construcción del sistema de navegació autónoma, se utilizó el motor de juego Unity 2022.3 LTS como entorno de simulación 3D. Se diseñó un mapa tipo circuito cerrado, compuesto por una pista delimitada por colisiones físicas, múltiples puntos de control (checkpoints) ordenados y una zona de partida definida como estado inicial del agente. El entorno simulado representa una pista de conducción virtual tomada de [22] y diseñada por RCC Design sobre la cual un vehículo autónomo debe aprender a desplazarse de forma eficiente. El controlador del vehiculo se basa en "Simple Car Controller"tomado de [23], el cual esta adaptado para poder realizar el entrenamieto. El agente es un modelo de aprendizaje por refuerzo que se conecta con Unity a través del paquete ML-Agents v0.30.0 con una arquitetura inicial de proximal policy optimization, y tiene como objetivo alcanzar correctamente todos los checkpoints sin colisionar, recibiendo información del entorno a través de sensores tipo raycast 3D, así como datos del propio vehículo, incluyendo su posición, rotación y velocidad.

En este proyecto se implementarán tres algoritmos de aprendizaje por refuerzo: PPO, SARSA y Q-Learning. Sin embargo, el enfoque principal se centrará en los dos primeros, debido a que el último, según lo trabajado en clase, presenta limitaciones significativas en entornos de alta dimensionalidad. Q-Learning requiere explorar una cantidad muy grande (prácticamente infinita) de estados, lo cual implica un consumo elevado de memoria y dificulta el aprendizaje del agente. Por esta razón, se ha decidido priorizar PPO y SARSA, sobre los cuales se realizarán comparativas detalladas para evaluar su rendimiento y eficacia dentro del entorno de simulación planteado.

II-A. Configuración Física del Entorno

El entorno está delimitado por muros y barreras físicas mediante `Mesh Colliders` y `Box Colliders`, todos correctamente asignados a la capa "Wall" para ser detectables por los raycasts. Los checkpoints están representados por objetos con `Box Collider` no trigger (es decir son simplemente muros imaginarios en el espacio), asignados a la capa "Checkpoints". Estos elementos actúan como metas intermedias obligatorias, cada una de ellas cuenta con una posicion determinada el espacio y cuentan con un indice, para un total de 67 puntos de control.

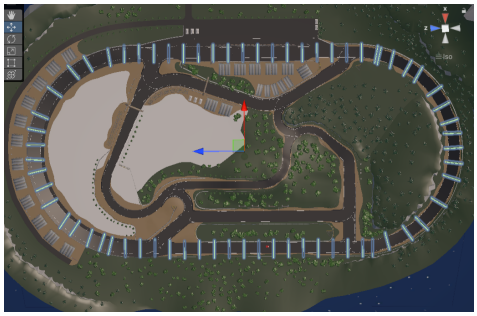

Figura 1. Pista de aprendizaje y Sistema de Checkpoint.

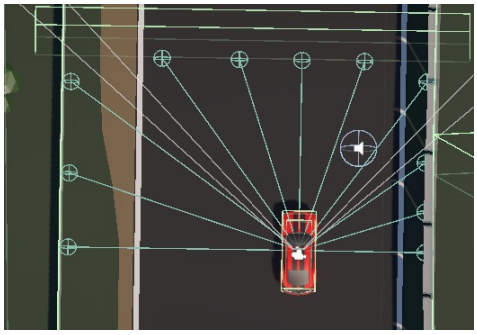

Figura 2. Sensor Ray Car.

II-B. Configuración Física del Agente

El vehículo principal cuenta con un componente `Rigidbody` que posee una masa de 1000 unidades. La amortiguación lineal de esta masa se encuentra en 0 y la amortiguación angular se fija en 0.05, este utiliza gravedad (`Use Gravity`) para que sea afectado por la física del motor y la detección de colisiones está configurada como `Discrete`. Además, incorpora un `Box Collider` ajustado a las dimensiones del vehículo que representa el agente (1.4, 0.5, 3.8) con un centro elevado en el eje Y (0.7) y un desplazamiento hacia atrás en Z (-0.2). Este collider está habilitado como `Is Trigger` para registrar eventos de colisiones físicas.

II-C. Configuración del Ray Car

El agente autónomo está equipado con un sensor de percepción tridimensional basado en rayos (similar a una cámara), conocido como Ray Perception Sensor 3D, el cual le permite detectar elementos específicos del entorno. Este sensor ha sido configurado para identificar dos tipos de objetos etiquetados: “Checkpoints” y “Wall”, que corresponden a los puntos de control y los muros delimitadores del circuito, respectivamente.

El sensor emite un total de cinco rayos por dirección, cubriendo un ángulo de visión máximo de 180°, lo cual proporciona una cobertura frontal amplia. Cada rayo se proyecta utilizando un radio de esfera de 0.45 unidades y una longitud máxima de 50 unidades dentro del entorno 3D. Además, se activa la opción de orden alternante en la emisión de rayos, lo que mejora la exploración simétrica del entorno.

La configuración vertical del sensor incluye un desplazamiento inicial desde el centro del agente de 1.0 unidades hacia arriba, y un desplazamiento final de 0.53 unidades, generando una inclinación progresiva de los rayos hacia el suelo. Se establece un sistema de raycasting simple (no apilado) y el sensor se asigna a una máscara de capa combinada (Mixed Layer Mask), lo que permite detectar múltiples capas físicas simultáneamente.

Gracias a esta configuración, el agente puede percibir la geometría circundante y reaccionar con precisión ante obstáculos o cambios de trayectoria, facilitando una navegación inteligente y adaptable.

II-D. Configuración Física del Controlador del Vehículo

El comportamiento físico del agente está gobernado por un componente de tipo Simple Car Controller. Este configura tanto la física de conducción como las restricciones de dirección, tracción y velocidad del vehículo.

Se establece un ángulo máximo de giro de 40°, permitiendo giros amplios durante la navegación. La fuerza del motor es de 4000 unidades, y la fuerza de frenado máxima alcanza los 125000. La velocidad máxima del vehículo es de 160 km/h, medida en el sistema KPH. Para mejorar la estabilidad, se asigna un valor de anti-roll de 1000 unidades, lo que contrarresta el balanceo lateral durante curvas o cambios bruscos de dirección. El control de tracción está habilitado, junto con un límite de deslizamiento (Slip Limit) de 0.3, restringiendo la pérdida de tracción en curvas cerradas o aceleraciones abruptas.

Se activa la asistencia de dirección con un coeficiente de 0.5 para suavizar los giros del agente. El controlador utiliza 6 marchas, aunque ni estas ni el sonido del motor —proporcionado por una fuente de audio— son relevantes para el entrenamiento. Además, se asignan Wheel Colliders y sus respectivas mallas a cada rueda para garantizar una simulación visual y física coherente.

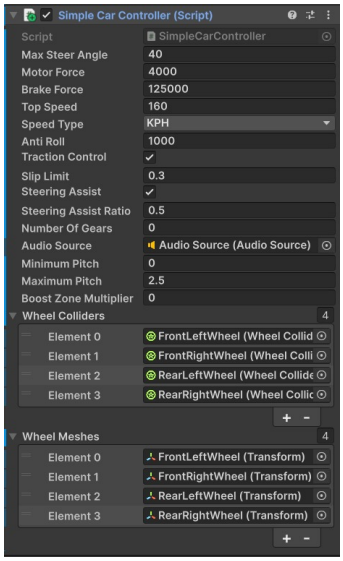

Figura 3. Configuracón del Controlador del Agente.

II-E. Sistema de Recompensas y Penalizaciones

El agente es guiado por una función de recompensa basada principalmente en el enfoque propuesto en el proyecto de [18], el cual representó un avance significativo en el comportamiento autónomo en entornos de navegación controlada. Sin embargo, para adaptarla mejor a los objetivos y características específicas de nuestro entorno, se realizaron ajustes manuales a sus componentes. Estos fueron cuidadosamente seleccionados para equilibrar el aprendizaje entre exploración eficiente, progreso continuo y penalización de comportamientos no deseados:

- **+1.0** por cada *checkpoint* cruzado en el orden correcto.
- **-1.0** por colisión con muros o salida del circuito.
- **-0.02** por rozar un muro.
- **-0.01** por cada paso sin movimiento significativo (anti-estancamiento).

Estos valores fueron determinados mediante un proceso iterativo de prueba y error, observando el comportamiento emergente del agente en distintos episodios. Se buscó mantener un balance entre incentivar el progreso constante y corregir comportamientos erráticos o ineficientes, sin inducir aprendizaje inestable debido a penalizaciones excesivas.

El sistema de colisiones se gestiona mediante eventos `OnTriggerEnter` o `OnCollisionEnter`, dependiendo del tipo de objeto involucrado. Asimismo, las zonas fuera de pista están delimitadas y provocan penalizaciones inmediatas al ser invadidas.

Para optimizar el proceso de entrenamiento, se ha implementado una condición de reinicio automático: si el agente acumula una recompensa total inferior a **-25**, el episodio se reinicia. Esta medida evita que el agente continúe explorando políticas ineficaces, reduciendo así el tiempo de entrenamiento perdido en trayectorias no productivas, esta lógica se puede visualizar de forma más clara en el Pseudocódigo 1.

Además, los *checkpoints* están secuenciados mediante índices numéricos, lo que obliga al agente a cruzarlos en un orden predefinido. Esta restricción impide que el agente aprenda comportamientos engañosos, como saltarse puntos de control intermedios y aun así recibir recompensa. De este modo, se refuerza el aprendizaje de trayectorias válidas y completas dentro del circuito.

II-F. Entrenamiento del agente a través de Mlagents

El entrenamiento del agente se llevó a cabo utilizando la plataforma Unity ML-Agents, la cual permite simular entornos 3D complejos y realizar aprendizaje por refuerzo dentro de ellos. Se diseñó un entorno de conducción tipo circuito, donde el agente (vehículo) debe aprender a desplazarse evitando obstáculos y cumpliendo objetivos. Para el entrenamiento se utilizó un modelo personalizado en Python, conectado con Unity mediante la API de ML-Agents, permitiendo aplicar el algoritmo PPO desde el entorno de Python y controlar los parámetros, episodios y comportamiento de la política de forma externa.

Figura 4. Entorno Python- Unity.

El entrenamiento se basa en un ciclo de interacción continuo entre un vehículo simulado (agente) y su entorno virtual. Este entorno está diseñado con una pista de conducción que incluye obstáculos, puntos de control (checkpoints) y límites, y es capaz de proporcionar información sensorial en cada paso temporal. Para fomentar un aprendizaje más robusto, el agente no inicia siempre desde un punto fijo, como se puede apreciar en la Figura 5, sino que aparece en una posición aleatoria dentro de los límites de la pista, manteniendo una distancia constante respecto al centro. Esta variación en la posición inicial se implementa con el fin de comparar el desempeño del agente frente a entrenamientos con posiciones iniciales fijas, y de favorecer que aprenda a generalizar su comportamiento en distintos escenarios.

Figura 5. Puntos de partidas del Agente.

***Ciclo de interacción**:*

- El agente observa el estado actual del entorno s_t a partir de la percepción visual y de sensores (distancias a obstáculos, posición relativa a checkpoints, etc.).
- Utilizando su política actual $\pi_\theta(a \mid s_t)$, el agente selecciona una acción a_t, representada por un par de valores discretos: uno para la aceleración $(-1, 0, 1)$ y otro para la dirección $(-1, 0, 1)$, dando lugar a 9 combinaciones posibles.
- Esta acción se aplica al entorno, produciendo un nuevo estado s_{t+1} y una recompensa r_t.

- La tupla (s_t, a_t, r_t, s_{t+1}) se almacena en un *buffer* de experiencia para su posterior uso durante el proceso de actualización.

Este flujo se encuentra ilustrado en la Figura 6, donde se visualiza el paso del estado por la red neuronal (*policy*) y su efecto en el entorno, retornando datos sensoriales relevantes.

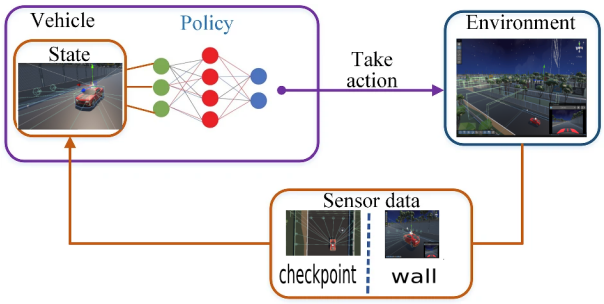

Figura 6. Arquitectura general de conducción autónoma [11].

Arquitectura y actualización con PPO

La Figura 7 representa el marco de aprendizaje del agente con PPO, el cual está basado en una arquitectura de actor-crítico. Esta incluye:

- Una **red de política (actor)** que es la encargada de tomar decisiones, modelando una política $\pi_\theta(a \mid s_t)$ que indica qué acción tomar dado un estado. Su objetivo es aprender a seleccionar acciones que maximicen la recompensa esperada [24].
- Una **red de valor (crítico)** la cual evalúa qué tan buenas son las acciones tomadas por el actor, estimando el valor del estado $V_\phi(s)$ o de la acción $Q_\phi(s, a)$. Su retroalimentación guía al actor durante el entrenamiento para mejorar su desempeño [24].

Función objetivo de PPO: El algoritmo PPO busca maximizar una función objetivo modificada para evitar actualizaciones bruscas de la política. En concreto, utiliza una función de pérdida con *clip*, definida de la siguiente manera [25]:

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

donde:

- $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ es el ratio de probabilidad entre la nueva y antigua política.
- $\hat{A}_t$ es la ventaja estimada (*advantage*) para la acción seleccionada
- ϵ es un hiperparámetro que controla el grado de actualización permitida ($\epsilon = 0{,}0167$s).
- La función *clip* restringe el ratio a un rango seguro para evitar cambios extremos.

Cálculo de la ventaja (Advantage Estimation): La ventaja $\hat{A}_t$ se calcula como la diferencia entre el retorno observado y el valor esperado, de la siguiente forma [25]:

$$\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + (\gamma\lambda)^2\delta_{t+2} + \cdots$$

donde $\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$ y $\lambda \in [0, 1]$ es un parámetro que controla el balance entre sesgo y varianza (utilizado en GAE - Generalized Advantage Estimation).

Pérdida total: La función de pérdida total utilizada en PPO incluye términos adicionales para regularizar el entrenamiento [25]:

$$L(\theta, \phi) = L^{CLIP}(\theta) - c_1 E_t\left[\left(V_\phi(s_t) - R_t\right)^2\right] + c_2 E_t\left[\text{entropy}[\pi_\theta](s_t)\right]$$

El segundo término penaliza errores en la estimación del valor, y el tercer término fomenta la exploración manteniendo alta la entropía de la política.

Algorithm 1 Agente de conducción entrenado con PPO en Unity

```
Inicializar parámetros de la red del actor y del crítico
Inicializar entorno, controlador del vehículo y estado inicial
Definir umbrales de finalización (tiempo, recompensa, pasos)
for cada episodio do
    Reposicionar agente en el entorno
    Reiniciar velocidad, recompensa y estado del vehículo
    Reiniciar estado del entorno y vector de observaciones
    for cada paso hasta cumplir condición de fin do
        Recoger observaciones del entorno:
        Dirección al siguiente checkpoint
        Velocidad actual del agente
        Seleccionar acción a_t desde la política π_θ(s_t)
        Ejecutar acción en el entorno y observar:
        Nuevo estado s_{t+1}, recompensa r_t, y done
        Almacenar transición (s_t, a_t, r_t, s_{t+1}) en el buffer
        if velocidad ¡umbral de inactividad then
            Penalizar por estar inactivo
        end if
        if colisión con muro then
            Penalizar proporcional a la fuerza del impacto
        end if
        if checkpoint correcto then
            Recompensar positivamente +1
        else if checkpoint incorrecto then
            Penalizar negativamente −1
        end if
        if se alcanzó recompensa objetivo O tiempo máximo O pasos máximos O recompensa < −25 then
            Terminar episodio
        end if
        if paso múltiplo del intervalo de actualización then
            Actualizar red crítica minimizando L_critic
            Actualizar red actor maximizando L_actor
        end if
    end for
end for
```

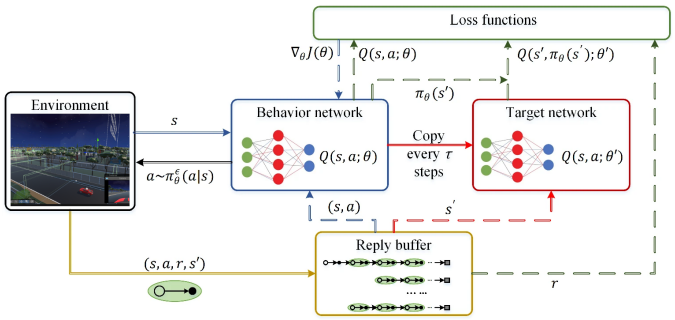

Figura 7. Diagrama de flujo del algoritmo PPO [11].

Arquitectura y actualización con SARSA

La Figura 8 ilustra el esquema de aprendizaje del agente con SARSA, el cual esta basado en una arquitectura:

- Una **tabla o red Q** almacena los valores $Q(s, a)$, representando la recompensa esperada al tomar acción a en estado s, y seguir la política actual [20].
- Se utiliza una política ϵ-greedy para seleccionar acciones: la mayor parte del tiempo se elige la acción con mayor valor de Q, pero ocasionalmente se explora otra acción [26].

Regla de actualización de SARSA: La actualización de SARSA está dada por la siguiente ecuación [26]:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\big[\, r_{t+1} + \gamma\, Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\,\big]$$

donde:

- α es la tasa de aprendizaje.
- γ es el factor de descuento.
- La estimación de $Q(s_{t+1}, a_{t+1})$ corresponde a la acción que el agente realmente tomará, no a la óptima hipotética.

Este mecanismo promueve un aprendizaje conservador y coherente con la política real del agente, reduciendo el riesgo de comportamientos peligrosos durante la conducción [26].

Ventajas en entornos de conducción simulada: SARSA presenta varias características que lo hacen especialmente adecuado para simulaciones de conducción segura:

- Su naturaleza *on-policy* disminuye la varianza en las actualizaciones, lo que favorece una convergencia más estable, factor crucial en entornos con penalizaciones definidas como rozaduras, estancamientos o checkpoints.
- Es sencillo de implementar cuando el espacio de acciones es discreto (como girar izquierda/derecha o avanzar), como es el caso de tu simulador. También permite extensiones como SARSA(λ) con trazas de elegibilidad para acelerar el aprendizaje [26].

Integración con la función de recompensa definida: En nuestra simulación, la función de recompensa incorpora componentes como:

- Recompensas positivas por cruzar checkpoints.
- Penalizaciones por colisiones, rozaduras o estancamiento.

SARSA se adapta naturalmente a este entorno, ya que actualiza sus valores en base a la acción tomada, reflejando exactamente las consecuencias definidas por la función de recompensa en cada paso real del agente. Esto permite que el agente aprenda una política que equilibra progreso y seguridad.

Algorithm 2 Entrenamiento con SARSA + Aproximación Fourier

Parámetros: α: tasa de aprendizaje, γ: factor de descuento, ϵ: probabilidad de exploración
Hiperparámetros: n: orden de Fourier, N: número de episodios, M: pasos por episodio
Inicializar: pesos $\theta \leftarrow \vec{0}$
Generar: conjunto de coeficientes C de Fourier para cada dimensión del estado
for episodio $= 1$ **hasta** N **do**
    Reiniciar entorno $\Rightarrow$ obtener estado s
    Calcular $\phi(s)$ mediante funciones base de Fourier
    Elegir acción a con política ϵ-greedy basada en $Q(s, a; \theta)$
    for paso $= 1$ **hasta** M **do**
        Ejecutar acción a, obtener recompensa r y nuevo estado s'
        Calcular $\phi(s')$
        Elegir a' con política ϵ-greedy usando $Q(s', a'; \theta)$
        Estimar: $Q(s, a) = \theta^\top \phi(s, a)$
        Estimar: $Q(s', a') = \theta^\top \phi(s', a')$
        Calcular TD-error: $\delta = r + \gamma Q(s', a') - Q(s, a)$
        Actualizar pesos: $\theta \leftarrow \theta + \alpha\delta \cdot \phi(s, a)$
        $s \leftarrow s'$
        $a \leftarrow a'$
        if recompensa acumulada > 110 **o** recompensa acumulada < 25 **then**
            break ▷ Terminar episodio por recompensa extrema
        end if
        if tiempo del episodio $> 90s$ **o** número de pasos ≥ 3000 **then**
            break ▷ Terminar episodio por límite de tiempo o pasos
        end if
    end for
    Decaer $\epsilon \leftarrow \max(\epsilon \cdot \text{decaimiento}, \epsilon_{\min})$
end for

II-G. Métricas de Evaluación

Para medir el desempeño del agente durante el proceso de entrenamiento y validación, se definieron un conjunto de métricas generales que permiten evaluar su comportamiento en el entorno simulado. Estas métricas proporcionan información cuantitativa sobre el progreso, eficiencia y estabilidad del aprendizaje.

- **Pasos por episodio**: número total de pasos ejecutados antes de finalizar un episodio.

- **Recompensa total**: suma acumulada de recompensas obtenidas por el agente durante el episodio.
- **Checkpoints alcanzados**: cantidad de puntos de control superados correctamente en el entorno.
- **Colisiones detectadas**: número de impactos con obstáculos o bordes del circuito.
- **Tiempo promedio por vuelta**: tiempo promedio requerido por el agente para completar una vuelta al circuito.

II-H. HUD y Visualización de Variables

Como parte final de la metodología, se diseñó un *HUD* (Head-Up Display) que permite visualizar en tiempo real el desempeño del agente durante la simulación. Esta interfaz facilita el seguimiento del aprendizaje y el análisis del comportamiento del agente en el entorno. En pantalla se muestran las siguientes variables:

- Velocidad del vehículo.
- Tiempo de la simulación.
- Recompensa acumulada.
- Pasos realizados por el agente.

Figura 8. HUD y Visualización de Variables.

III. Resultados

A continuación, se presentan los resultados obtenidos, donde se realiza una comparativa del rendimiento de los algoritmos SARSA (State-Action-Reward-State-Action) y PPO (Proximal Policy Optimization) en el entorno de navegación autónoma. Para SARSA, se evaluaron tres configuraciones diferentes utilizando funciones de aproximación basadas en series de Fourier: (i) orden 3 con 1000 aproximaciones, (ii) orden 5 con 5 aproximaciones, y (iii) orden 7 con 5 aproximaciones. Por otro lado, el algoritmo PPO fue analizado bajo dos condiciones: (i) entrenamiento con reinicio del episodio al alcanzar una recompensa acumulada de -25, y (ii) entrenamiento sin opción de reinicio anticipado al alcanzar recompensas negativas por mal desempeño. Las métricas consideradas incluyen la recompensa acumulada, la evolución temporal del aprendizaje, la tasa de éxito en la tarea, y la estabilidad del comportamiento del agente, permitiendo así una evaluación integral de cada enfoque.

III-A. PPO: Estrategia con reinicio de episodio al alcanzar recompensa acumulada de -25

En la Figura [9] se puede evidenciar que el agente converge eficientemente hacia la recompensa esperada alrededor de los 2.5 millones de pasos. A partir de ese punto, no solo alcanza de forma sostenida una recompensa cercana a los 110, sino que también mantiene dicha recompensa con baja variabilidad, lo que indica una política estable y efectiva. Este comportamiento sugiere que el agente ha logrado aprender una estrategia óptima para la tarea de navegación, minimizando colisiones y optimizando su recorrido a lo largo del circuito.

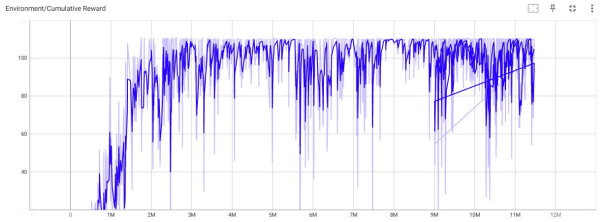

Figura 9. PPO: Estrategia con reinicio de episodio al alcanzar recompensa acumulada de -25: Recompesa vs pasos totales

Se puede observar que, durante los primeros episodios, el agente tiende a explorar extensivamente el entorno, alcanzando en muchos casos el número máximo de pasos permitidos por episodio sin lograr una recompensa significativa. Sin embargo, a medida que el entrenamiento progresa y el agente comienza a aprender una política más efectiva, se evidencia una reducción progresiva en la cantidad promedio de pasos por episodio. Este comportamiento refleja una mejora sustancial en la eficiencia del agente, indicando que logra completar la tarea en menos tiempo y con decisiones más acertadas. En consecuencia, el agente no solo se aproxima a la recompensa esperada, sino que también optimiza su ruta y reduce acciones innecesarias, lo cual es un claro indicador de aprendizaje exitoso y convergencia de la política.

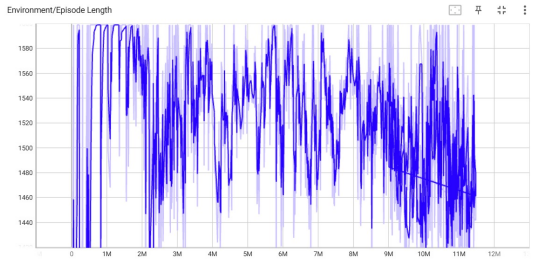

Figura 10. PPO: Estrategia con reinicio de episodio al alcanzar recompensa acumulada de -25: Pasos totales vs Episodios.

III-B. PPO: Estrategia sin opción de reinicio anticipado al alcanzar recompensas negativas

Barra aqui van los resultados del ppo sin opcion de reinicio al canzar recompensa negativas, colocas las dos graficas asi como lo llevo.

III-C. SARSA con optimización de tiempo: orden de Fourier 3 y 1000 aproximaciones

El agente no logra una adecuada generalización de los estados, lo que limita su capacidad para completar satisfactoriamente los episodios. Como resultado, su mejor desempeño

alcanzó únicamente una recompensa de -1 sobre un máximo posible de 110, con un promedio general de recompensas de -23.89, lo que refleja un aprendizaje deficiente y una falta de convergencia efectiva de la política.

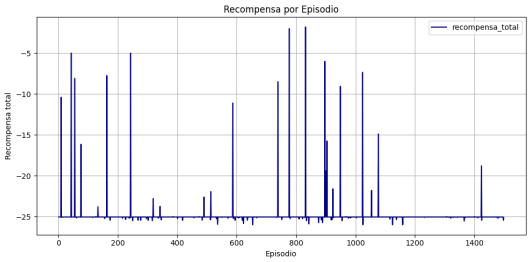

Figura 11. SARSA con la optimzaicon de tiempo, orden de fourier 3 y 1000 aproximaciones. Recompesa vs Episodios

El comportamiento inestable del agente impide la estabilización del número de pasos por episodio a lo largo del entrenamiento. Esta métrica presenta una alta variabilidad conforme el agente interactúa con el entorno, lo que indica una falta de consistencia en la política aprendida y una exploración poco eficiente del espacio de estados.

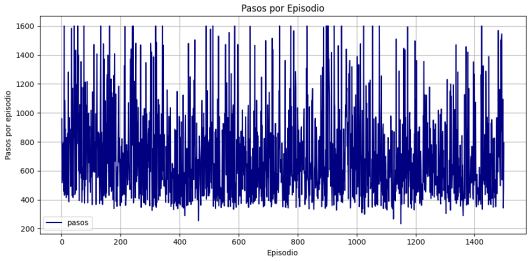

Figura 12. SARSA con la optimzaicon de tiempo, orden de fourier 3 y 1000 aproximaciones.: Pasos alcanzados en cada episodio vs episodios totales.

III-D. SARSA sin optimización de tiempo: orden de Fourier 5 y 5 aproximaciones

En la figura [13] se presentan los resultados obtenidos utilizando SARSA con representación de Fourier de orden 5 y 5 funciones base. Se observa una alta variabilidad en las recompensas por episodio, con una tendencia oscilatoria persistente durante los 3000 episodios. El agente no logra estabilizar su comportamiento ni converger hacia una política efectiva, lo que sugiere que esta configuración no permite una aproximación adecuada de la función de valor. La recompensa media se mantiene en valores negativos, y el proceso de aprendizaje parece estancado en una fase de exploración sin mejoras significativas.

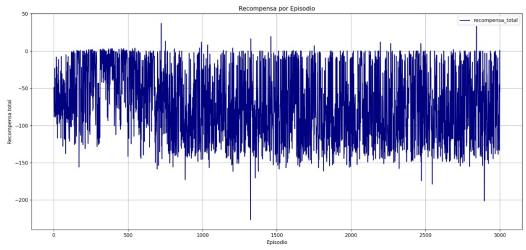

Figura 13. SARSA sin optimización de tiempo: orden de Fourier 5 y 5 aproximaciones: Recompensa vs episodios totales.

El número total de pasos alcanzado por episodio se mantuvo constantemente en el valor máximo permitido (3000), lo que evidencia que el agente no logró aprender una política efectiva. Este comportamiento refleja un desempeño deficiente a lo largo del entrenamiento, sin señales claras de mejora ni convergencia hacia una solución óptima.

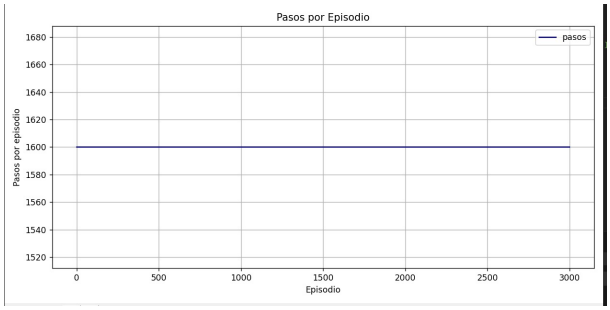

Figura 14. SARSA sin optimización de tiempo: orden de Fourier 5 y 5 aproximaciones: Pasos vs Episodios totales.

III-E. SARSA sin optimización de tiempo: orden de Fourier 7 y 5 aproximaciones

Barragan aqui escribe los resultados dependiendo de los resultados obtenidos del sarsa

IV. Conclusiones

Los resultados obtenidos a lo largo del desarrollo del proyecto permiten afirmar que el algoritmo que evidenció el mejor desempeño en el aprendizaje de la pista fue el Proximal Policy Optimization (PPO) con reinicio al alcanzar recompensas negativas. Esta configuración permitió que el agente no solo completara satisfactoriamente una vuelta a la pista objetivo principal del entrenamiento, sino que incluso lograra recorrer vuelta y media. Esto refleja una política sólida y generalizada, capaz de adaptarse eficientemente al entorno.

En contraste, la versión de PPO sin optimización de tiempo mostró un comportamiento menos eficiente. Aunque exploró una mayor cantidad de estados, la duración excesiva de los episodios y la alta cantidad de iteraciones requeridas hicieron que el proceso de aprendizaje fuese considerablemente más lento. Este enfoque, al no contar con una penalización clara ante trayectorias ineficientes, tendía a prolongar la exploración sin alcanzar mejoras consistentes en menor tiempo.

Por otra parte, el algoritmo SARSA con optimización de tiempo, representación de Fourier de orden 3 y 1000 aproximaciones logró un avance parcial. El agente consiguió aprender a recorrer aproximadamente un cuarto de la pista antes de colisionar con los muros, lo que recurrentemente lo llevaba a obtener la recompensa negativa máxima de -25. Aunque se evidencian ciertos patrones de aprendizaje, estos no fueron suficientes para completar el circuito.

En cuanto a las configuraciones de SARSA sin optimización de tiempo y representaciones de Fourier de orden 5 con 5 aproximaciones, y orden 7 con 5 aproximaciones, el desempeño fue aún más limitado. A pesar de que el agente ejecutaba el número máximo de pasos por episodio, las trayectorias eran poco eficientes y las recompensas altamente negativas eran frecuentes. Esto indica que dichas configuraciones no permitieron una adecuada aproximación de la función de valor ni una política efectiva para resolver la tarea.

En definitiva, el objetivo del proyecto fue alcanzado al identificar y validar que el algoritmo de aprendizaje profundo más eficiente para resolver el entorno propuesto fue el PPO con reinicio basado en recompensa negativa. Esta configuración demostró ser la más robusta en términos de exploración, aprendizaje y desempeño final del agente.

V. Mejoras Futuras

Como línea de trabajo futuro, se considera pertinente continuar explorando y optimizando el algoritmo SARSA, dado que con las configuraciones implementadas en este proyecto particularmente con funciones de base de Fourier el agente solo logró avanzar hasta un cuarto del recorrido total de la pista. Esto sugiere que, aunque SARSA mostró cierto grado de aprendizaje, su rendimiento podría mejorar significativamente ajustando parámetros como el orden de la base de Fourier, el número de aproximaciones, la tasa de aprendizaje o incluso incorporando técnicas de optimización temporal más robustas.

Adicionalmente, se propone evaluar el desempeño del agente utilizando otros algoritmos de aprendizaje por refuerzo más avanzados, como Twin Delayed Deep Deterministic Policy Gradient (TD3) o Soft Actor-Critic (SAC). Estos métodos, al estar diseñados para entornos continuos y tener mecanismos más eficientes de exploración y actualización de políticas, podrían ofrecer mejores resultados en términos de estabilidad, velocidad de convergencia y capacidad para completar trayectorias más complejas como la pista abordada en este estudio.

Estas futuras exploraciones permitirían no solo comparar diferentes enfoques de aprendizaje por refuerzo profundo, sino también fortalecer la robustez del agente frente a diversos escenarios y mejorar su rendimiento general en tareas de navegación complejas.

Referencias

[1] G. Liu, W. Jiang, C. Sun, N. Ning, R. Wang, and A. Buhari, "Object detection algorithm for autonomous driving: Design and real-time performance analysis of AttenRetina model," Alexandria Engineering Journal, vol. 123, pp. 392–402, 2025. https://doi.org/10.1016/j.aej.2025.02.063

[2] S. S. A. Zaidi, M. S. Ansari, A. Aslam, N. Kanwal, M. Asghar, and B. Lee, "A survey of modern deep learning based object detection models," Digital Signal Processing, vol. 126, Art. no. 103514, 2022. https://doi.org/10.1016/j.dsp.2022.103514

[3] N. M. Alahdal, F. Abukhodair, L. Haj Meftah, and A. Cherif, "Real-time object detection in autonomous vehicles with YOLO," Procedia Computer Science, vol. 246, pp. 2792–2801, 2024. https://doi.org/10.1016/j.procs.2024.09.392

[4] B. Mahaur and K. K. Mishra, "Small-object detection based on YOLOv5 in autonomous driving systems," Pattern Recognition Letters, vol. 168, pp. 115–122, 2023. https://doi.org/10.1016/j.patrec.2023.03.009

[5] G. Liu, Y. Hu, Z. Chen, J. Guo, and P. Ni, "Lightweight object detection algorithm for robots with improved YOLOv5," Engineering Applications of Artificial Intelligence, vol. 123, Art. no. 106217, 2023. https://doi.org/10.1016/j.engappai.2023.106217

[6] Iqra and K. J. Giri, "SO-YOLOv8: A novel deep learning-based approach for small object detection with YOLO beyond COCO," Expert Systems with Applications, vol. 280, Art. no. 127447, 2025. https://doi.org/10.1016/j.eswa.2025.127447

[7] P. Thottempudi, A. B. B. Jambek, V. Kumar, B. Acharya, and F. Moreira, "Resilient object detection for autonomous vehicles: Integrating deep learning and sensor fusion in adverse conditions," Engineering Applications of Artificial Intelligence, vol. 151, Art. no. 110563, 2025. https://doi.org/10.1016/j.engappai.2025.110563

[8] M. Haris and A. Glowacz, "RETRACTED: Road Object Detection: A Comparative Study of Deep Learning-Based Algorithms," Electronics, vol. 10, no. 16, Art. no. 1932, 2021. https://doi.org/10.3390/electronics10161932

[9] R. Bin Issa, M. Das, M. S. Rahman, M. Barua, M. K. Rhaman, K. S. N. Ripon, and M. G. R. Alam, "Double Deep Q-Learning and Faster R-CNN-Based Autonomous Vehicle Navigation and Obstacle Avoidance in Dynamic Environment," Sensors, vol. 21, no. 4, Art. no. 1468, 2021. https://doi.org/10.3390/s21041468

[10] P. Thottempudi, A. B. B. Jambek, V. Kumar, B. Acharya, and F. Moreira, "Resilient object detection for autonomous vehicles: Integrating deep learning and sensor fusion in adverse conditions," Engineering Applications of Artificial Intelligence, vol. 151, Art. no. 110563, 2025. https://doi.org/10.1016/j.engappai.2025.110563

[11] B. Peng, Q. Sun, S. E. Li, D. Kum, Y. Yin, J. Wei, and T. Gu, "End-to-end autonomous driving through Dueling Double Deep Q-network," Automotive Innovation, vol. 4, no. 3, pp. 328–337, 2021. https://doi.org/10.1007/s42154-021-00151-3

[12] D. Coelho and M. Oliveira, "A review of end-to-end autonomous driving in urban environments," IEEE Access, vol. 10, pp. 75296–75311, 2022. https://doi.org/10.1109/access.2022.3192019

[13] H. Jiang, N. Bhujel, Z. Lin, K.-W. Wan, J. Li, and S. Jayavelu, "Learning Relation in Crowd Using Gated Graph Convolutional Networks for DRL-Based Robot Navigation," IEEE Transactions on Intelligent Transportation Systems, vol. 25, no. 6, pp. 5085–5095, 2023. https://doi.org/10.1109/TITS.2023.3343923

[14] U.-G. Lee, K.-J. Choi, and S.-Y. Park, "The design and implementation of autonomous driving pallet robot system using ROS,in Proceedings of the 2021 12th International Conference on Ubiquitous and Future Networks (ICUFN), Jeju Island, South Korea, 2021, pp. 1-3. https://doi.org/10.1109/ICUFN49451.2021.9528735

[15] M. Alberri, S. Hegazy, M. Badra, M. Nasr, O. M. Shehata, and E. I. Morgan, "Generic ROS-based Architecture for Heterogeneous Multi-Autonomous Systems Development," in *Proc. 2018 IEEE Int. Conf. Vehicular Electronics and Safety (ICVES)*, Madrid, Spain, 2018, pp. 1–6. https://doi.org/10.1109/ICVES.2018.8519589

[16] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source Robot Operating System," in *Proc. Open-Source Software Workshop of the Int. Conf. on Robotics and Automation (ICRA)*, Kobe, Japan, 2009, pp. 1–6. https://ai.stanford.edu/~ang/papers/icraoss09-ROS.pdf

[17] B. Elallid, N. Benamar, M. Bagaa, and Y. Hadjadj-Aoul, "Enhancing autonomous driving navigation using Soft Actor-Critic," *Future Internet*, vol. 16, no. 7, p. 238, 2024. https://doi.org/10.3390/fi16070238

[18] T. Xu, Z. Meng, W. Lu, and Z. Tong, "End-to-end autonomous driving decision method based on improved TD3 algorithm in complex scenarios," *Sensors (Basel)*, vol. 24, no. 15, p. 4962, 2024. https://doi.org/10.3390/s24154962

[19] J. W. Mock and S. S. Muknahallipatna, "A comparison of PPO, TD3 and SAC reinforcement algorithms for quadruped walking gait

generation," *J. Intell. Learn. Syst. Appl.*, vol. 15, no. 1, pp. 36–56, 2023. https://doi.org/10.4236/jilsa.2023.151003

[20] G. Yao, N. Zhang, Z. Duan, and C. Tian, "Improved SARSA and DQN algorithms for reinforcement learning," *Theor. Comput. Sci.*, vol. 1027, no. 115025, p. 115025, 2025. https://doi.org/10.1016/j.tcs.2024.115025

[21] C. R. Udumula, V. P. Ancha, M. K. Ramayanam, P. Venkata Sai Manoj Teja, and V. Rachpudi, "An enhanced real-time object detection method using liquid neural network and echo state network architecture," in *Proc. 2024 Int. Conf. Inventive Comput. Technol. (ICICT)*, Coimbatore, India, 2024. https://doi.org/10.1109/ICICT60155.2024.10544664

[22] Unity Technologies, "Cartoon Race Track - Oval," Unity Asset Store, San Francisco, CA, USA. https://assetstore.unity.com/packages/3d/environments/roadways/cartoon-race-track-oval-175061 [Accessed: Jul. 5, 2025].

[23] M. Mubshar, "GitHub profile - mubshar1," GitHub. https://github.com/mubshar1 [Accessed: Jul. 5, 2025].

[24] K. Miyamoto, "Training PPO — Unity ML-Agents Guide," GitHub repository, 2023. [Online]. Available: https://github.com/miyamotok0105/unity-ml-agents/blob/master/docs/Training-PPO.md

[25] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," in *arXiv preprint arXiv:1707.06347*, 2017. https://doi.org/10.48550/arXiv.1707.06347

[26] G. A. Jiménez, A. de la Escalera Hueso, and M. J. Gómez-Silva, "Reinforcement learning algorithms for autonomous mission accomplishment by unmanned aerial vehicles: A comparative view with DQN, SARSA and A2C," Sensors, vol. 23, no. 21, Art. no. 9013, 2023. https://doi.org/10.3390/s23219013