PROYECTO:

Resolución del entorno "Car Racing"

Aprendizaje por Refuerzo

Integrantes del grupo:

- Unai de Leon (unaide.leon@opendeusto.es)
- Antón García (anton.garcia@opendeusto.es)
- Xabier Losa (xabier.losa@opendeusto.es)

Enero 2026

Resumen

Este documento presenta el desarrollo completo de un agente de aprendizaje por refuerzo profundo (Deep Reinforcement Learning) entrenado para resolver el entorno *CarRacing-v3* de Gymnasium. El proyecto implementa el algoritmo Proximal Policy Optimization (PPO) con observaciones visuales (píxeles), utilizando redes neuronales convolucionales para el procesamiento de imágenes y la extracción de características.

El objetivo principal es entrenar un agente capaz de conducir un vehículo de forma autónoma en circuitos generados aleatoriamente, maximizando la recompensa acumulada mediante el aprendizaje de políticas de control basadas exclusivamente en información visual. La arquitectura desarrollada incluye preprocesamiento de imágenes, frame stacking, paralelización de entornos y callbacks personalizados para evaluación y seguimiento.

Los resultados demuestran que el agente aprende políticas efectivas de conducción, alcanzando recompensas medias superiores a 800 puntos con episodios de longitud cercana a 470 pasos (equivalente a $\sim$940 pasos del entorno base con frame skip de 2). Este documento detalla la motivación del proyecto, la implementación técnica, los objetivos de entrenamiento, el proceso experimental y las conclusiones obtenidas.

Índice

1. Introducción

El aprendizaje por refuerzo (*Reinforcement Learning*, RL) es una rama de la inteligencia artificial en la que un agente aprende a tomar decisiones a través de la interacción con un entorno. En lugar de aprender a partir de ejemplos etiquetados (como ocurre en el aprendizaje supervisado), el agente observa las consecuencias de sus acciones y ajusta su comportamiento para maximizar una señal de recompensa.

En la práctica, esto convierte a RL en una herramienta especialmente atractiva para problemas donde las reglas son difíciles de programar a mano, pero es posible simular la tarea y evaluar el desempeño de manera objetiva.

1.1. Contexto del Proyecto

Este proyecto se enmarca en el *Deep Reinforcement Learning* (DRL), donde se combinan técnicas de RL con redes neuronales profundas capaces de procesar entradas de alta dimensionalidad, como imágenes. En concreto, abordamos la conducción autónoma en el entorno `CarRacing-v3` de Gymnasium, un benchmark muy utilizado para evaluar algoritmos de control basados en visión.

El reto es deliberadamente exigente: el agente debe aprender a conducir **solo con píxeles**, sin acceso directo a variables como la velocidad, la orientación o la posición del vehículo.

1.2. Objetivos Generales

Los objetivos principales del trabajo son:

- Implementar un sistema completo de entrenamiento de agentes de RL con observaciones visuales.
- Aplicar el algoritmo Proximal Policy Optimization (PPO), uno de los métodos más efectivos en RL moderno.
- Desarrollar una arquitectura de preprocesamiento y entrenamiento escalable y reproducible.
- Evaluar el rendimiento del agente mediante métricas cuantitativas y análisis cualitativo.
- Documentar el proceso completo para facilitar la reproducibilidad y extensión del trabajo.

1.3. Contribuciones del Equipo

El proyecto ha sido desarrollado de forma colaborativa por un equipo de tres personas, repartiendo responsabilidades entre la implementación, la experimentación, el análisis de resultados y la documentación. Las contribuciones concretas se reflejan en las secciones correspondientes.

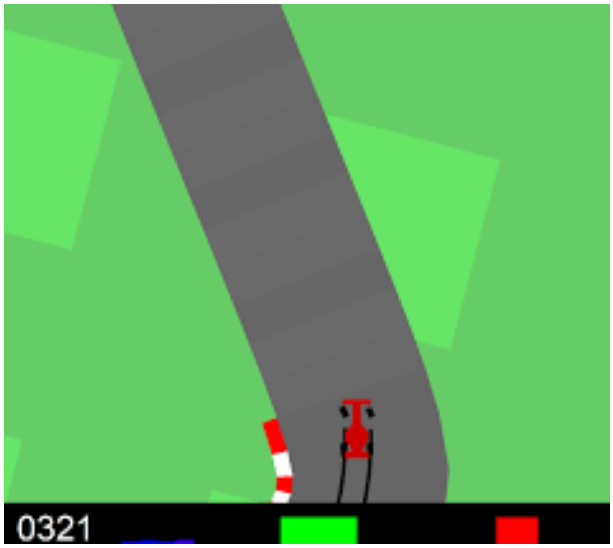

Figura 1: Ejemplo de observación del entorno `CarRacing-v3`. De izquierda a derecha: velocidad (números), sensores ABS (barras verticales azules), posición del volante (barra horizontal verde) y giroscopio (barra horizontal roja).

2. Descripción del Entorno CarRacing-v3

2.1. Características Generales

`CarRacing-v3` es un entorno de Gymnasium basado en el motor de física Box2D que simula la conducción de un vehículo en una pista de carreras vista desde arriba. Se trata de un problema de **control continuo** con **observaciones visuales** de alta dimensionalidad, lo que lo convierte en un benchmark especialmente exigente para algoritmos de aprendizaje por refuerzo profundo.

En cada episodio, la pista se genera de forma procedimental a partir de una semilla, por lo que el agente no puede memorizar un circuito fijo: debe aprender una estrategia general de conducción.

2.2. Espacio de Observaciones

- **Tipo**: Imagen RGB.
- **Dimensiones originales**: $96 \times 96 \times 3$ (alto $\times$ ancho $\times$ canales RGB).
- **Formato**: Array de enteros sin signo de 8 bits (`uint8`), valores en el rango $[0, 255]$.
- **Contenido**: Vista aérea del vehículo y la pista, incluyendo el coche (verde), la carretera (gris), el césped circundante (verde oscuro) y marcadores de progreso (tiles de colores en la pista).

El agente debe aprender a interpretar esta información visual para tomar decisiones de control efectivas, sin acceso a información privilegiada como posición, velocidad o ángulo del vehículo.

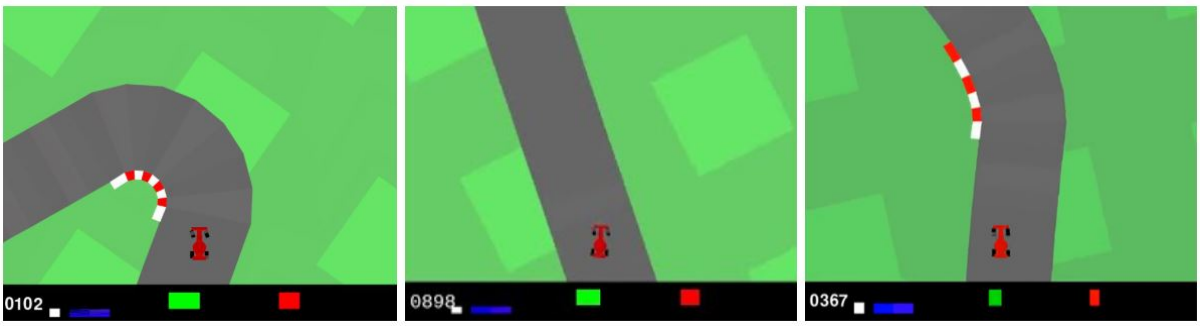

Figura 2: Ejemplos de observaciones visuales representativas del entorno. De derecha a izquierda: curva cerrada, recta y curva suave.

2.3. Espacio de Acciones

El espacio de acciones es **continuo** y tridimensional:

$$\mathbf{a} = [\text{steering}, \text{gas}, \text{brake}] \in [-1, 1]^3 \tag{1}$$

- **Steering** (dirección): $\in [-1, 1]$ — valores negativos giran a la izquierda, positivos a la derecha.
- **Gas** (aceleración): $\in [0, 1]$ — intensidad de aceleración.
- **Brake** (freno): $\in [0, 1]$ — intensidad de frenado.

2.4. Función de Recompensa

La recompensa está diseñada para incentivar el progreso a lo largo de la pista:

- $+1000/N$ puntos por cada tile de pista visitado por primera vez, donde N es el número total de tiles en la pista
- Penalización de $-0{,}1$ por cada frame/paso (incentiva completar rápido)
- Recompensa total máxima teórica: aproximadamente +900 puntos al completar el circuito

Un episodio termina cuando:

- El vehículo sale completamente de la pista (fuera del área gris)
- Se completa el circuito (se visitan todos los tiles)
- Se alcanza el límite de pasos por episodio (típicamente 1000 pasos en el entorno base)

2.5. Dinámica del Entorno

- **Física**: Simulación realista con Box2D que incluye fricción, inercia, y deslizamiento
- **Generación de pistas**: Cada episodio puede generar un circuito aleatorio diferente basado en la semilla proporcionada. Las pistas incluyen curvas suaves, curvas cerradas y rectas de longitud variable
- **Dificultad**: El control requiere anticipación y suavidad — acciones bruscas (especialmente steering + gas simultáneos) pueden causar derrapes y salidas de pista

2.6. Desafíos del Entorno.

CarRacing presenta varios desafíos significativos para el aprendizaje:

1. **Alta dimensionalidad de observaciones**: Las imágenes $96 \times 96 \times 3$ contienen $27,648$ valores por frame.
2. **Dependencia temporal**: Una sola imagen no proporciona información de velocidad o dirección de movimiento.
3. **Control continuo**: A diferencia de acciones discretas, el espacio de acciones continuo requiere exploración más sofisticada.

4. **Horizonte largo**: Episodios de hasta 1000 pasos requieren planificación a largo plazo.
5. **Sparse rewards**: La mayoría de la recompensa proviene de visitar nuevos tiles, no hay señales densas de çonducción correcta".
6. **Generalización**: El agente debe aprender políticas que funcionen en circuitos no vistos durante el entrenamiento.

2.7. Comparación con Otros Entornos

CarRacing se sitúa en un punto intermedio de complejidad:

- Más complejo que entornos clásicos de control (CartPole, MountainCar) por usar visión
- Menos complejo que simuladores realistas (CARLA, TORCS) pero suficientemente desafiante para evaluación de algoritmos
- Ampliamente utilizado en investigación como benchmark estándar para RL visual

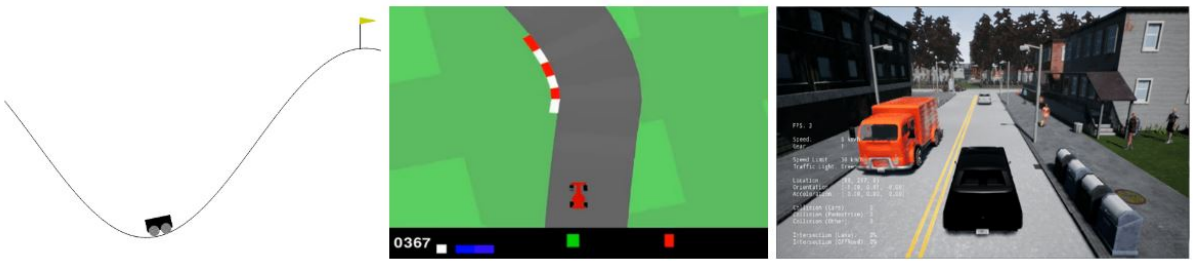

Figura 3: Tres entornos de distinta complejidad. De derecha a izquierda: MountainCar, CarRacing y CARLA.

3. Motivación y Justificación del Proyecto

3.1. ¿Por Qué Conducción Autónoma con Visión?

La conducción autónoma es uno de los retos más relevantes y complejos de la inteligencia artificial moderna. En este proyecto la abordamos desde aprendizaje por refuerzo visual porque obliga a aprender una política de control **end-to-end**: desde píxeles hasta acciones, sin atajos ni información privilegiada.

3.1.1. Relevancia Práctica

Los sistemas de conducción autónoma del mundo real dependen fundamentalmente de sensores visuales (cámaras) como principal fuente de información. Desarrollar agentes capaces de aprender políticas de control directamente desde píxeles tiene aplicaciones directas en:

- Vehículos autónomos comerciales.
- Sistemas de asistencia al conductor (ADAS).
- Simulación y entrenamiento de conductores.
- Robótica móvil y navegación autónoma.

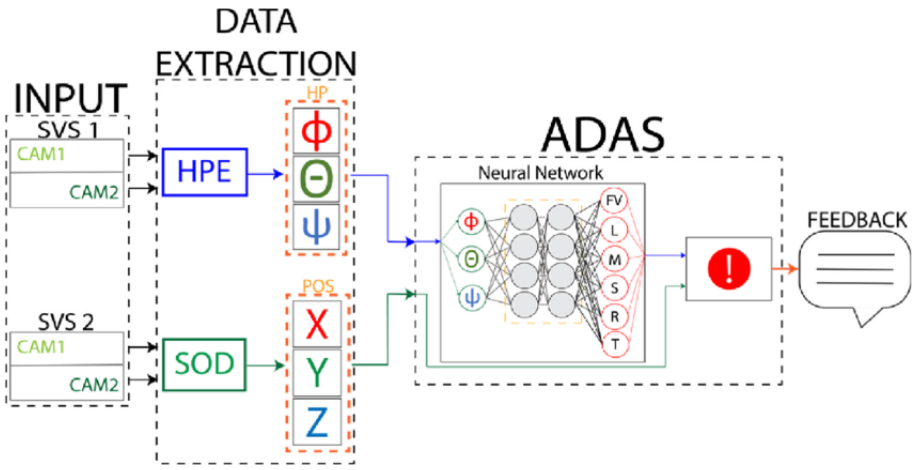

Figura 4: Diagrama procedimental de los ADAS

3.1.2. Desafío Técnico

Aprender desde observaciones visuales introduce retos técnicos interesantes y realistas que nos permiten aplicar métodos modernos de deep learning:

- **Procesamiento de imágenes**: Extracción de características relevantes de datos de alta dimensionalidad.
- **Redes neuronales convolucionales**: Aplicación de CNNs para visión artificial.

- **Aprendizaje end-to-end**: Mapeo directo de píxeles a acciones sin ingeniería manual de características.
- **Inferencia temporal**: Incorporación de historia temporal mediante frame stacking.

4. Implementación Técnica

En esta sección se describe la arquitectura del sistema, las decisiones de diseño más relevantes y los componentes que forman el pipeline completo de entrenamiento y evaluación. El objetivo es que el lector pueda entender *qué hace cada pieza*, por qué existe y cómo encaja con el resto.

4.1. Arquitectura General del Sistema

El repositorio está organizado de forma modular, separando de manera clara la configuración, los entornos, el entrenamiento, la evaluación y las utilidades comunes:

```
ProyectoRL-main/
 src/                    # Scripts
 configs/                # Archivos de configuración YAML
 optuna/                 # Optimización de hiperparámetros
 results/                # Resultados de entrenamientos
 doc.pdf                 # Documentación (este documento)
 requirements.txt        # Dependencias del proyecto
 README.md               # Guía de uso rápida
```

4.2. Preprocesamiento de Observaciones

El preprocesamiento de imágenes es clave para entrenar de forma estable y eficiente. Para ello, implementamos wrappers de Gymnasium que transforman las observaciones originales en un formato más adecuado para redes convolucionales y aprendizaje por refuerzo.

4.2.1. FrameSkip Wrapper

```python
class FrameSkip(gym.Wrapper):
    """Repite la misma acci n N pasos y acumula recompensa."""
    def __init__(self, env: gym.Env, skip: int = 2):
        super().__init__(env)
        self.skip = skip

    def step(self, action):
        total_reward = 0.0
        for _ in range(self.skip):
            obs, reward, terminated, truncated, info = self.env
                .step(action)
            total_reward += float(reward)
            if terminated or truncated:
                break
        return obs, total_reward, terminated, truncated, info
```

Listing 1: Implementación del wrapper FrameSkip

extbfPropósito: reduce la frecuencia de decisión, repitiendo cada acción durante N frames consecutivos. En la práctica, esto:

- Acelera el entrenamiento al reducir el número de decisiones por episodio
- Suaviza el control y reduce acciones erráticas
- Es común en RL visual (valor típico: $N = 2$ o $N = 4$)

4.2.2. PreprocessCarRacing Wrapper

```
class PreprocessCarRacing(gym.ObservationWrapper):
    def __init__(self, env: gym.Env, grayscale: bool = True,
       resize: int = 84):
        super().__init__(env)
        self.grayscale = grayscale
        self.resize = resize
        c = 1 if grayscale else 3
        self.observation_space = gym.spaces.Box(
            low=0.0, high=1.0, shape=(c, resize, resize), dtype
               =np.float32
        )

    def observation(self, obs):
        if self.grayscale:
            img = cv2.cvtColor(obs, cv2.COLOR_RGB2GRAY)
            img = cv2.resize(img, (self.resize, self.resize))
            img = img.astype(np.float32) / 255.0
            img = np.expand_dims(img, axis=0)  # (1, H, W)
        else:
            img = cv2.resize(obs, (self.resize, self.resize))
            img = img.astype(np.float32) / 255.0
            img = np.transpose(img, (2, 0, 1))  # (3, H, W)
        return img
```

Listing 2: Preprocesamiento de imágenes

Transformaciones aplicadas:

1. **Conversión a escala de grises** (opcional): Reduce de 3 canales RGB a 1 canal, disminuyendo dimensionalidad en 3× sin perder información esencial para la tarea
2. **Redimensionado**: De 96×96 a 84×84 (estándar en DRL visual, balance entre resolución y coste computacional)
3. **Normalización**: De $[0, 255]$ (uint8) a $[0, 1]$ (float32), facilitando el entrenamiento de redes neuronales

4. **Reordenamiento de canales**: De formato HWC (Height, Width, Channels) a CHW (Channels, Height, Width), formato esperado por PyTorch

Resultado: Observación transformada de dimensión $(1, 84, 84)$ para grayscale o $(3, 84, 84)$ para color.

extit[Poner imagen aquí: ejemplo visual del preprocesado (RGB original vs grayscale vs resize 84×84) en 3 subfiguras]

Figura 5: Ejemplo del preprocesamiento aplicado a una observación del entorno.

4.2.3. Frame Stacking

Para proporcionar información temporal (velocidad, dirección de movimiento), apilamos los últimos K frames consecutivos:

- **Configuración típica**: $K = 4$ frames
- **Dimensión final**: $(4 \times 1, 84, 84) = (4, 84, 84)$ para grayscale
- **Implementación**: Usamos `VecFrameStack` de Stable-Baselines3 para entornos vectorizados

Esto permite al agente inferir velocidad y dirección de movimiento sin necesidad de variables de estado explícitas.

4.2.4. Pipeline Completo de Preprocesamiento

Figura 6: Pipeline de transformación de observaciones (de izquierda a derecha): imagen original RGB $96 \times 96 \times 3 \rightarrow$ grayscale $96 \times 96 \times 1 \rightarrow$ resize $84 \times 84 \times 1 \rightarrow$ normalización $\rightarrow$ frame stack $\rightarrow$ entrada CNN $(4, 84, 84)$

extit[Poner imagen aquí: diagrama más visual del pipeline (con flechas y tamaños de tensores).]

4.3. Vectorización de Entornos

Para acelerar la recolección de experiencias, ejecutamos múltiples instancias del entorno en paralelo:

```
1 def make_vec_env(env_id, seed, n_envs=4, ...):
2     env_fns = [make_env_fn(env_id, seed + i, ...) for i in
          range(n_envs)]
3
4     if n_envs > 1:
```

```
        vec = SubprocVecEnv(env_fns)  # Paralelizacion por
            procesos
    else:
        vec = DummyVecEnv(env_fns)

    vec = VecFrameStack(vec, n_stack=4, channels_order="first")
    return vec
```

Listing 3: Creación de entornos vectorizados

Ventajas de la paralelización:

- **Throughput**: Con $n = 8$ entornos, recolectamos 8× más experiencia por unidad de tiempo
- **Diversidad**: Diferentes entornos pueden estar en distintas fases del episodio, aumentando diversidad de datos
- **Eficiencia GPU**: Batches más grandes aprovechan mejor la GPU durante el forward pass

4.4. Algoritmo PPO

Utilizamos Proximal Policy Optimization (PPO) [**?**], un algoritmo de *policy gradient* que ofrece un buen equilibrio entre estabilidad y rendimiento en problemas de control continuo.

4.4.1. Características de PPO

- **On-policy**: Aprende de experiencia reciente, mejorando estabilidad
- **Clipped objective**: Limita cambios grandes en la política, evitando colapsos
- **Sample efficiency**: Balance entre eficiencia muestral y estabilidad
- **Simplicidad**: Menos hiperparámetros críticos que TRPO, más robusto que A3C

4.4.2. Función Objetivo (Clipped Surrogate)

$$L^{CLIP}(\theta) = \mathbb{E}_t \left[\text{mín} \left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right] \tag{2}$$

donde:

- $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ es la ratio de probabilidades
- $\hat{A}_t$ es la ventaja estimada (usando GAE)
- ϵ es el parámetro de clipping (típicamente 0.2)

4.4.3. Arquitectura de Red Neuronal

Usamos `NatureCNN` de Stable-Baselines3, basada en la arquitectura original de DQN [?]:

```
policy_kwargs = dict(
    features_extractor_class=NatureCNN,
    features_extractor_kwargs=dict(features_dim=512),
    normalize_images=False  # Ya normalizamos en el wrapper
)

model = PPO(
    policy="CnnPolicy",
    env=train_env,
    policy_kwargs=policy_kwargs,
    learning_rate=2.5e-4,
    n_steps=2048,
    batch_size=256,
    n_epochs=10,
    gamma=0.99,
    gae_lambda=0.95,
    clip_range=0.2,
    ...
)
```

Listing 4: Configuración de la política CNN

Arquitectura NatureCNN:

- Conv1: 32 filtros 8×8, stride 4
- Conv2: 64 filtros 4×4, stride 2
- Conv3: 64 filtros 3×3, stride 1
- Fully Connected: 512 unidades
- Salidas: Actor (política) y Critic (función de valor)

4.5. Hiperparámetros Principales

Parámetro	Valor
Learning rate	$2{,}5 \times 10^{-4}$
Discount factor (γ)	0.99
GAE lambda (λ)	0.95
Clip range (ϵ)	0.2
Steps per update (`n_steps`)	2048
Mini-batch size	256
Epochs per update	10
Entropy coefficient	0.01
Value function coefficient	0.5
Max gradient norm	0.5
Frame skip	2
Frame stack	4
Image resolution	84×84
Grayscale	Sí
Number of parallel envs	4-8

Tabla 1: Hiperparámetros utilizados en el entrenamiento

4.6. Sistema de Configuración

Para facilitar la experimentación y reproducibilidad, implementamos un sistema de configuración centralizado:

```python
@dataclass(frozen=True)
class TrainConfig:
    env_id: str = "CarRacing-v3"
    seed: int = 42
    total_timesteps: int = 2_000_000
    learning_rate: float = 2.5e-4
    n_steps: int = 2048
    batch_size: int = 256
    # ... mas parametros
```

Listing 5: Dataclass de configuración en `src/config.py`

extbfVentajas:

- Configuración versionable en archivos YAML/JSON
- Cada run guarda automáticamente su configuración en `results/<run>/config.json`
- Overrides por línea de comandos: `-n-envs 8 -total-timesteps 1000000`

4.7. Callbacks y Evaluación

Implementamos callbacks personalizados para seguimiento y evaluación periódica:

```
eval_callback = EvalCallback(
    eval_env,
    n_eval_episodes=10,
    eval_freq=50000,  # Evaluar cada 50k steps
    best_model_save_path=model_dir,
    deterministic=True
)
```

Listing 6: Callback de evaluación en `src/callbacks.py`

extbfFuncionalidades:

- Evaluación periódica en entorno separado (sin exploración)
- Guardado automático del mejor modelo según recompensa media
- Logging de métricas a TensorBoard
- Checkpoints periódicos cada N pasos
- Modo visual opcional para debugging (`-visual`)

4.8. Logging y Reproducibilidad

```
# Estructura generada automaticamente:
results/
          ppo_carracing/
              20260111-230209/
                  config.json          # Configuracion del run
                  logs/
                        ppo_carracing_1/  # TensorBoard logs
                  models/
                        best_model.zip    # Mejor modelo segun
            eval
                        final_model.zip   # Modelo al final
                  videos/               # Videos de evaluacion
            (opcional)
```

Listing 7: Sistema de logging y organización de runs

Medidas de reproducibilidad:

- Seeds fijas para Python, NumPy, PyTorch y Gymnasium.
- Configuración completa guardada en JSON.
- Versiones de dependencias especificadas en `requirements.txt`.
- Logs detallados con timestamps.

4.9. Optimizaciones de Rendimiento

1. **Paralelización multinúcleo**: `SubprocVecEnv` distribuye entornos en procesos separados
2. **Control de threads**: `-num-threads` limita hilos por proceso, evitando oversubscription
3. **Aceleración GPU**: Automática si disponible, forzable con `-device cuda`
4. **Frame skip**: Reduce decisiones por episodio manteniendo calidad de control

4.10. Herramientas de Debugging

Modo visual (`-visual`):

- Muestra render original y observación preprocesada en tiempo real
- Útil para verificar correctitud del pipeline de preprocesamiento
- Permite inspección cualitativa del comportamiento del agente

Scripts de evaluación:

- `src/eval.py`: Evalúa modelos guardados, genera métricas y vídeos
- `src/debug_obs.py`: Herramientas de inspección de observaciones

En conjunto, esta arquitectura modular y reproducible facilita tanto la experimentación como la extensión del proyecto (por ejemplo, para probar *reward shaping*, nuevas arquitecturas o algoritmos alternativos).

5. Objetivos del Entrenamiento

En esta sección se recogen los objetivos cuantitativos y cualitativos que guiaron el diseño del entrenamiento y el protocolo de evaluación. La idea es definir criterios claros de éxito para poder comparar experimentos de forma objetiva.

5.1. Objetivos Cuantitativos

5.1.1. Recompensa Acumulada

Objetivo principal: alcanzar una recompensa media por episodio ≥ 800 puntos en evaluación con semillas no vistas durante el entrenamiento.

Justificación:

- La recompensa teórica máxima en CarRacing es aproximadamente +900 puntos (completar el circuito)
- Valores ≥ 800 indican que el agente completa la mayoría del circuito consistentemente
- Es un umbral reportado en la literatura como indicativo de políticas efectivas [**?**]

Métricas asociadas:

$$R_{mean} = \frac{1}{N} \sum_{i=1}^{N} R_i \geq 800 \tag{3}$$

donde R_i es la recompensa acumulada del episodio i y $N \geq 10$ episodios de evaluación.

5.2. Objetivos Cualitativos

5.2.1. Suavidad de Control

Objetivo: El agente debe demostrar control suave y anticipatorio, no reactivo o errático.

Indicadores:

- Steering gradual en curvas (sin oscilaciones bruscas)
- Anticipación: comenzar a girar antes de la curva, no dentro de ella
- Uso apropiado del freno en curvas cerradas
- Aceleración controlada en rectas

Evaluación: inspección visual de episodios grabados y análisis de la distribución de acciones (dirección, gas y freno).

5.2.2. Generalización a Circuitos No Vistos

Objetivo: La política debe funcionar en circuitos con seeds diferentes a las del entrenamiento.

Protocolo de evaluación:

- Entrenar con seeds en rango $[42, 42 + N)$ donde N es el número de seeds de entrenamiento
- Evaluar con seeds en rango $[10000, 10100)$ (no vistas)
- Comparar degradación de rendimiento: $\Delta R = R_{train\_seeds} - R_{eval\_seeds} < 100$

5.2.3. Robustez ante Perturbaciones

Objetivo: Recuperación de situaciones desfavorables (ligeros derrapes, posiciones subóptimas).

Indicadores:

- Capacidad de corregir trayectoria tras pequeños derrapes
- No colapsar tras un error menor (no efecto dominó)
- Exploración razonable en fases tempranas sin colapsar completamente

5.3. Objetivos de Reproducibilidad y Documentación

5.3.1. Reproducibilidad Exacta

Objetivo: Cualquier miembro del equipo (o evaluador externo) debe poder reproducir los resultados exactamente.

Requisitos:

- Seeds fijas documentadas
- Versiones específicas de dependencias
- Configuraciones guardadas automáticamente
- Instrucciones claras de ejecución

5.3.2. Trazabilidad de Experimentos

Objetivo: Cada experimento debe ser trazable y comparable.

Implementación:

- Estructura de directorios con timestamps
- Logs de TensorBoard completos

- Guardado de checkpoints periódicos
- Vídeos de evaluación (opcional pero recomendado)

5.4. Criterios de Éxito del Proyecto

Consideramos el proyecto exitoso si se cumplen los siguientes criterios mínimos:

Criterio	**Umbral Mínimo**	**Objetivo Ideal**
Recompensa media	≥ 700	≥ 850
Desviación estándar	≤ 200	≤ 150
Longitud media (post-skip)	≥ 350	≥ 450
Steps de entrenamiento	$\leq 3M$	$\leq 2M$
Tiempo de entrenamiento	$\leq 8h$	$\leq 6h$
Generalización (ΔR)	≤ 150	≤ 100

Tabla 2: Criterios de éxito cuantitativos del proyecto

6. Entrenamiento

6.1. Optimización de Hiperparámetros con Optuna y Entrenamiento Final

Tras el experimento baseline con recompensas originales (`envs.py`), se observó que el rendimiento y la estabilidad del agente eran altamente sensibles a la elección de los hiperparámetros del algoritmo PPO. Con el objetivo de sistematizar este proceso y evitar ajustes manuales poco reproducibles, se empleó la librería Optuna para la optimización automática de hiperparámetros.

Configuración del estudio de Optuna La optimización se formuló como un problema de maximización de la recompensa media en evaluación. El estudio se compuso de un total de **30 trials**, cada uno correspondiente a un entrenamiento independiente de **200 000 pasos**, seguido de fases de evaluación periódicas sobre un entorno de validación separado del entorno de entrenamiento. Este diseño permitió explorar de forma eficiente el espacio de hiperparámetros manteniendo un coste computacional asumible.

Para guiar la búsqueda se utilizó un muestreador TPE (*Tree-structured Parzen Estimator*) junto con un mecanismo de *pruning* basado en la mediana, lo que permitió descartar de forma temprana configuraciones con bajo rendimiento y concentrar el presupuesto computacional en regiones prometedoras del espacio de búsqueda. El número de trials y la duración de cada uno se fijaron de acuerdo con las limitaciones computacionales disponibles, priorizando la exploración de configuraciones prometedoras frente a una búsqueda exhaustiva.

Los hiperparámetros optimizados durante el estudio fueron los siguientes:

- Tasa de aprendizaje (`learning_rate`): 0.00030386087977161573
- Factor de descuento (γ): 0.9650797105054421
- Parámetro de trazas GAE (`gae_lambda`): 0.936123066469635
- Rango de *clipping* (`clip_range`): 0.2407139477519057
- Coeficiente de entropía (`ent_coef`): 0.0015320467336528416
- Coeficiente de la función de valor (`vf_coef`): 0.9096897152894949
- Número de épocas por actualización (`n_epochs`): 11
- Número de pasos por rollout (`n_steps`): 2048
- Tamaño de batch (`batch_size`): 256

Mejor configuración encontrada Una vez finalizado el estudio, Optuna identificó como mejor configuración de hiperparámetros la correspondiente al *trial* con mayor recompensa media en evaluación:

```
learning_rate: 3.04e-4
gamma: 0.965
gae_lambda: 0.936
```

```
clip_range: 0.241
ent_coef: 1.53e-3
vf_coef: 0.91
max_grad_norm: 0.58
n_epochs: 11
n_steps: 2048
batch_size: 256
```

Listing 8: Mejores hiperparámetros encontrados mediante Optuna

Esta configuración mostró un equilibrio adecuado entre estabilidad del entrenamiento y rendimiento final, siendo consistente a lo largo de las evaluaciones intermedias realizadas durante el estudio.

Entrenamiento final con la configuración seleccionada Los modelos entrenados durante el proceso de optimización no se utilizaron directamente como agentes finales. Con el fin de evitar sesgos derivados de entrenamientos parciales, el agente final se entrenó desde cero utilizando la configuración seleccionada. En concreto, la mejor configuración encontrada se empleó para entrenar un nuevo agente en una ejecución prolongada de aproximadamente **10 millones de pasos**, que se considera el modelo final del sistema.

El entrenamiento final se realizó utilizando exactamente el mismo pipeline descrito en la Sección 4, manteniendo fijos los hiperparámetros seleccionados por Optuna. Este procedimiento permitió evaluar el comportamiento del agente en un régimen prolongado de aprendizaje y sirvió como base para el análisis de resultados presentado en la Sección 7, garantizando una separación clara entre el proceso de selección de hiperparámetros y la evaluación final del rendimiento del agente.

7. Resultados y Evaluación

En esta sección se presentan los resultados obtenidos tras la optimización de hiperparámetros con Optuna y el entrenamiento final del agente utilizando la configuración seleccionada. El análisis se centra en la evolución del aprendizaje durante el entrenamiento y en una evaluación cualitativa del comportamiento del agente, evitando métricas no medidas explícitamente.

7.1. Resultados de la Optimización con Optuna

La optimización de hiperparámetros mediante Optuna permitió analizar de forma sistemática el impacto de distintas configuraciones del algoritmo PPO sobre el rendimiento del agente. A lo largo de los distintos *trials*, se observó una mejora progresiva del valor objetivo, lo que indica que el proceso de búsqueda fue capaz de identificar regiones del espacio de hiperparámetros más prometedoras.

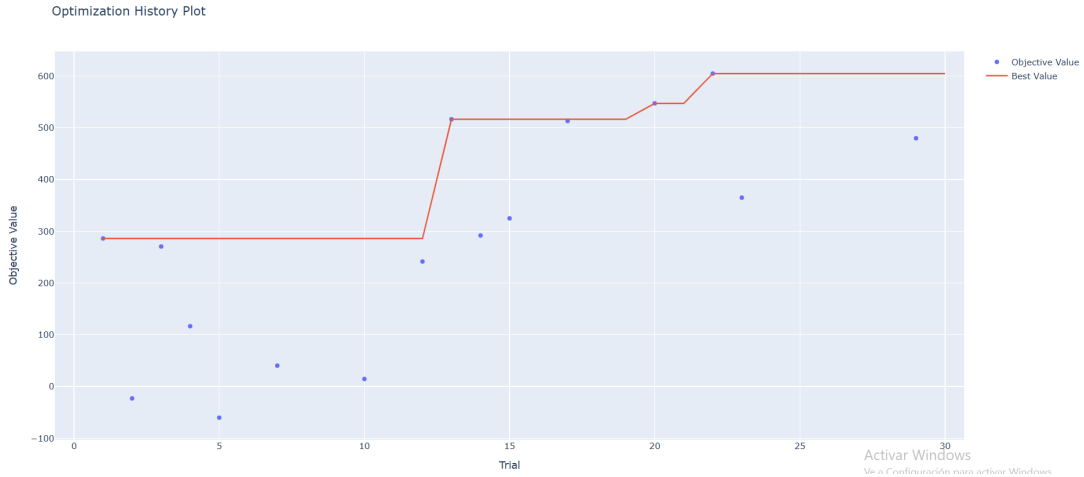

Figura 7: Evolución de la recompensa objetivo a lo largo de los trials de Optuna.

El análisis de importancia de hiperparámetros sugiere que parámetros como la tasa de aprendizaje, el factor de descuento y los coeficientes asociados a la función de pérdida tienen una influencia significativa sobre el rendimiento alcanzado, lo que justifica su inclusión en el proceso de optimización automática.

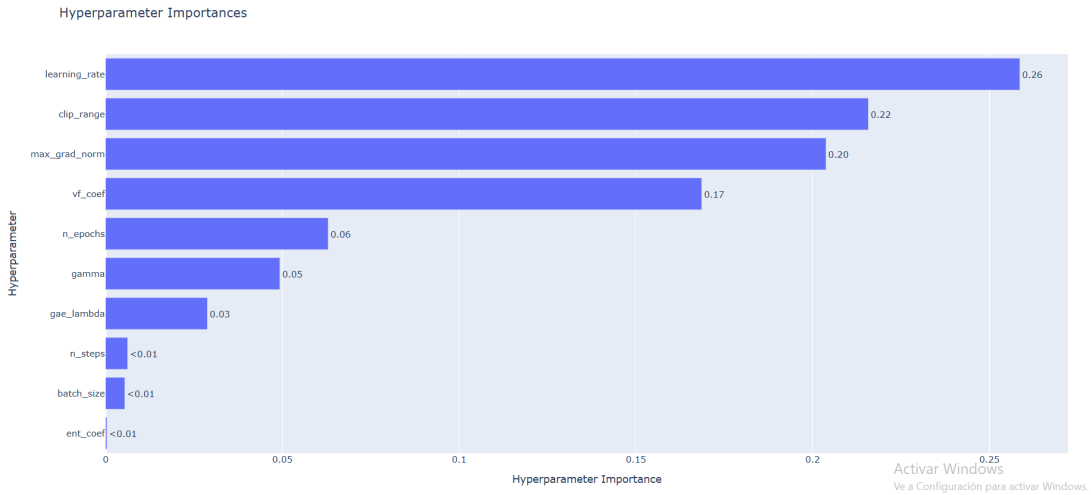

Figura 8: Importancia relativa de los hiperparámetros según el estudio de Optuna.

Una vez finalizado el estudio, se seleccionó la configuración correspondiente al mejor *trial*, que fue utilizada para entrenar el agente final desde cero en una ejecución prolongada.

7.2. Curvas de Aprendizaje del Entrenamiento Final

El agente entrenado con los mejores hiperparámetros encontrados por Optuna fue entrenado durante aproximadamente 5 millones de pasos. Durante este proceso se registraron métricas tanto de entrenamiento como de evaluación, lo que permite analizar la dinámica de aprendizaje del agente.

La Figura 9 muestra la evolución de la recompensa media por episodio durante los *rollouts* de entrenamiento. Se observa una fase inicial de exploración con recompensas bajas, seguida de un incremento progresivo del rendimiento y una posterior estabilización, lo que indica un proceso de aprendizaje consistente.

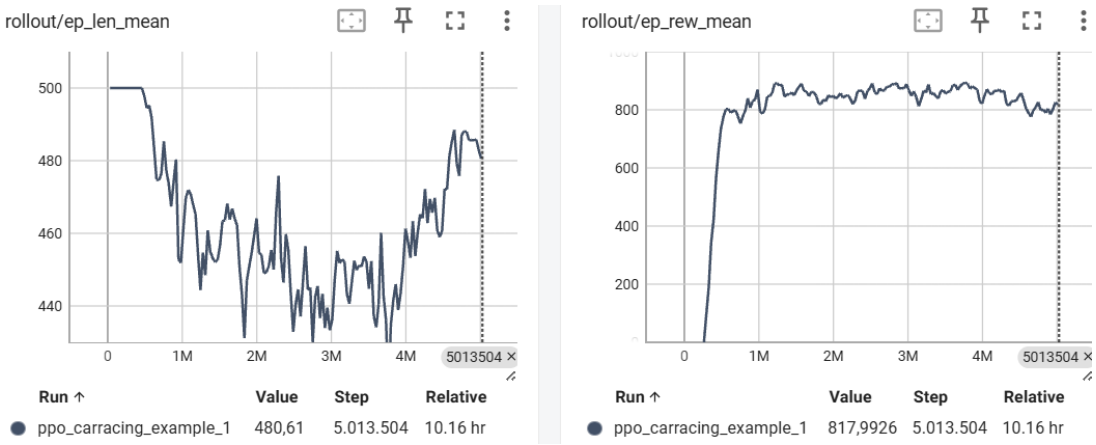

Figura 9: Evolución de la recompensa media durante el entrenamiento final con hiperparámetros optimizados.

De forma complementaria, la Figura 10 muestra la evolución de las métricas de evaluación obtenidas con la política congelada. La coherencia entre las tendencias observadas en entrenamiento y evaluación sugiere que el rendimiento alcanzado no se debe únicamente a sobreajuste durante el aprendizaje.

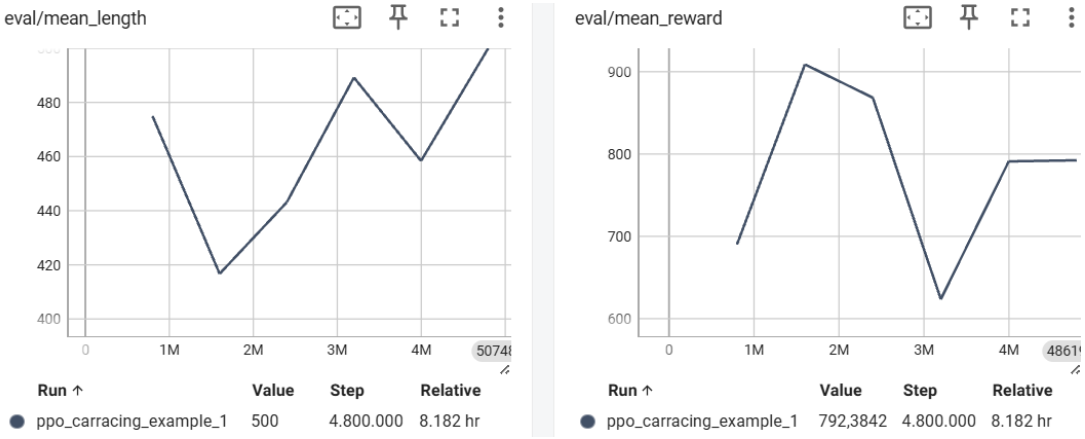

Figura 10: Evolución de las métricas de evaluación durante el entrenamiento final.

7.3. Análisis Cualitativo del Comportamiento

Además de las métricas cuantitativas, se realizó una inspección visual de episodios generados por el agente entrenado. De forma general, el agente es capaz de mantener el vehículo en pista durante episodios prolongados, anticipar curvas y realizar correcciones suaves ante pequeñas desviaciones.

No obstante, se observaron también comportamientos subóptimos en situaciones concretas, como curvas muy cerradas o cambios bruscos de geometría del circuito, donde el agente puede perder el control temporalmente. Estos comportamientos sugieren posibles líneas de mejora futuras mediante ajustes adicionales en la función de recompensa o en el preprocesamiento de observaciones.

7.4. Resumen de Resultados

En conjunto, los resultados obtenidos muestran que la optimización de hiperparámetros mediante Optuna permitió identificar configuraciones más estables y eficientes que las obtenidas mediante ajuste manual. El entrenamiento prolongado con dicha configuración conduce a un aprendizaje consistente y a un comportamiento razonable del agente en el entorno CarRacing.

Si bien no se ha realizado una evaluación exhaustiva de generalización ni estudios de ablación sistemáticos, los resultados presentados proporcionan una base sólida sobre la que extender el trabajo en futuras iteraciones.

8. Modificaciones del entorno y *reward shaping*

Durante los experimentos iniciales con el entorno `CarRacing-v3` y el algoritmo PPO, se observó un comportamiento recurrente en el agente consistente en pérdidas de control y giros excesivos al afrontar curvas cerradas, incluso tras un número elevado de pasos de entrenamiento. Este fenómeno es habitual en este entorno y se debe principalmente a limitaciones del diseño de la señal de recompensa original y a la naturaleza continua del espacio de acciones.

En particular, la recompensa proporcionada por el entorno base prioriza la cobertura de segmentos de pista, pero no penaliza explícitamente comportamientos dinámicamente inestables como el sobreviraje, el uso excesivo del acelerador durante giros pronunciados o la ausencia de frenado. Como consecuencia, el agente puede aprender políticas que maximizan la recompensa a corto plazo pero resultan ineficientes o inestables en curvas de radio reducido.

Para mitigar este problema, se introdujo una estrategia de *reward shaping* mediante un *wrapper* adicional del entorno. El objetivo de esta modificación es guiar el aprendizaje hacia políticas de conducción más suaves y estables sin alterar la función objetivo fundamental del entorno. Las contribuciones añadidas a la recompensa son las siguientes:

- **Penalización de la magnitud del giro**: se introduce un término proporcional al valor absoluto del ángulo de dirección, con el fin de favorecer trayectorias más suaves y evitar oscilaciones excesivas del volante.
- **Penalización del giro combinado con aceleración**: se añade una penalización adicional cuando el agente aplica simultáneamente un alto ángulo de dirección y aceleración. Este término resulta clave para reducir el sobreviraje y los trompos, incentivando al agente a reducir la velocidad antes de afrontar curvas cerradas.
- **Recompensa leve por el uso del freno**: dado que el agente tiende a ignorar la acción de frenado en ausencia de incentivos explícitos, se introduce una pequeña recompensa asociada a su utilización. Este término facilita la aparición de comportamientos de desaceleración controlada en situaciones de alta curvatura.

Es importante destacar que los coeficientes asociados a estos términos se mantienen deliberadamente pequeños en comparación con la recompensa original del entorno. De este modo, el *reward shaping* actúa como una guía inductiva durante el aprendizaje, pero no modifica la política óptima subyacente ni introduce información privilegiada sobre el estado interno del simulador.

La introducción de esta modificación permitió mejorar notablemente la estabilidad del agente, reduciendo de forma significativa los giros incontrolados en curvas pronunciadas y acelerando la convergencia hacia políticas de conducción más realistas y eficientes.

8.1. Evaluación

Debido al cambio de recompensas, el resultado será evaluado de forma más cualitativa, ya que las recompensas medias no son comparables.

En definitiva, se observa un comportamiento más suavizado del agente, evitando derrapes en las curvas pronunciadas, y un manejo más estable con giros menos bruscos. Son cambios sutiles, que pulen el comportamiento del agente.

9. Aproximación alternativa: Curriculum Learning

9.1. Motivación

El entorno `CarRacing-v3` plantea un problema especialmente complejo para los algoritmos de Aprendizaje por Refuerzo. El agente debe aprender una política de control continuo a partir de observaciones visuales de alta dimensionalidad, con recompensas escasas y una dinámica del entorno altamente no lineal. Entrenar directamente sobre el entorno completo desde el inicio suele provocar dificultades de exploración, aprendizaje inestable y políticas subóptimas, especialmente en fases tempranas del entrenamiento.

Para mitigar estos problemas, se ha adoptado una estrategia de *Curriculum Learning*, inspirada en el aprendizaje progresivo humano. La idea fundamental consiste en presentar al agente una secuencia de tareas cuya dificultad aumenta gradualmente, permitiéndole adquirir primero habilidades básicas antes de enfrentarse a escenarios más complejos.

En este proyecto, el curriculum no se ha implementado modificando la función de recompensa ni la dinámica del entorno, sino a través de una progresión en la **complejidad geométrica de los circuitos**. Esta aproximación resulta especialmente natural en CarRacing, donde la dificultad de la tarea está directamente relacionada con la curvatura y estructura del trazado.

9.2. Curriculum basado en semillas del entorno

El entorno CarRacing genera los circuitos de forma procedural a partir de una semilla (*seed*). Variando dicha semilla es posible obtener pistas con características geométricas muy diferentes: desde circuitos con largas rectas y curvas suaves hasta trazados altamente sinuosos con giros cerrados.

Aprovechando esta propiedad, se ha diseñado un curriculum learning basado en conjuntos de semillas ordenadas por complejidad. El entrenamiento se divide en tres fases:

- **Fase 1 (Easy):** circuitos de baja curvatura, dominados por rectas y giros suaves.
- **Fase 2 (Medium):** circuitos con curvas amplias y estructura ovalada.
- **Fase 3 (Hard):** circuitos completos generados proceduralmente, con giros cerrados y alta variabilidad.

9.3. Estimación de la complejidad del circuito

Para clasificar las semillas en función de la complejidad del circuito generado, se emplea una métrica basada en la variación angular media del trazado. Cada circuito en CarRacing está compuesto por una secuencia de segmentos, cada uno asociado a un ángulo de orientación. La complejidad se estima como la media del valor absoluto de la diferencia entre ángulos consecutivos.

El siguiente fragmento de código ilustra el cálculo de esta métrica:

```
1
2 def track_complexity(seed):
3     # Inicializamos el entorno
4     env = gym.make("CarRacing-v3")
5     env.reset(seed=seed)
6
7     # Accedemos a los datos internos de la pista
8     track = env.unwrapped.track
9     angles = [tile[1] for tile in track]
10
11    # Calculamos la curvatura media
12    curvature = np.mean(np.abs(np.diff(angles)))
13
14    env.close()
15    return curvature
```

Listing 9: Cálculo de complejidad de la pista en CarRacing

Evaluando esta métrica sobre un conjunto amplio de semillas y ordenándolas por curvatura creciente, se obtienen listas de semillas representativas de circuitos simples, intermedios y complejos. Estas listas se almacenan en un fichero `JSON` que define el curriculum del entrenamiento.

9.4. Modificaciones introducidas en el entorno

El entorno original de CarRacing se ha extendido de forma mínima y modular para soportar el curriculum learning. En particular, se ha añadido un *wrapper* encargado de seleccionar dinámicamente la semilla del circuito en cada reinicio del entorno, a partir de un conjunto predefinido:

```
1 class CurriculumSeedWrapper(gym.Wrapper):
2     def __init__(self, env, seeds):
3         super().__init__(env)
4         self.seeds = seeds
5
6     def reset(self, **kwargs):
7         seed = random.choice(self.seeds)
8         return self.env.reset(seed=seed)
```

Listing 10: Estructura del wrapper.

Este wrapper se integra en la función de creación del entorno sin alterar el resto del pipeline, que incluye *frame skipping*, preprocesado visual, *reward shaping* y apilado de frames. De este modo, el curriculum afecta exclusivamente a la distribución de los circuitos observados durante el entrenamiento, manteniendo inalterada la dinámica subyacente del entorno.

9.5. Planteamiento del entrenamiento

El entrenamiento se realiza de forma secuencial siguiendo las fases del curriculum. Se utiliza el algoritmo PPO (Proximal Policy Optimization) con una arquitectura convolucional para el procesamiento de las observaciones visuales, es decir, se ha respetado la configuración previa.

En cada fase:

1. Se crea un entorno vectorizado configurado con el conjunto de semillas correspondiente a la dificultad actual.
2. Se entrena el agente durante un número determinado de pasos, reutilizando los parámetros aprendidos en fases anteriores.
3. El entorno de evaluación se mantiene fijo y utiliza únicamente circuitos complejos, con el objetivo de medir la capacidad de generalización del agente.

Este enfoque permite que el agente aprenda primero el control longitudinal y la estabilidad del vehículo en circuitos simples, posteriormente la negociación de curvas suaves, y finalmente la conducción robusta en trazados complejos. Además, la separación clara entre entrenamiento y evaluación facilita el análisis del impacto real del curriculum learning sobre el rendimiento final.

9.6. Entrenamiento

La Figura 11 muestra la evolución de la recompensa media por episodio (`rollout/ep_rew_mean`) a lo largo del entrenamiento completo empleando *curriculum learning* basado en la complejidad del circuito. El eje horizontal representa el número total de *timesteps* acumulados, mientras que el eje vertical corresponde a la recompensa media obtenida por el agente.

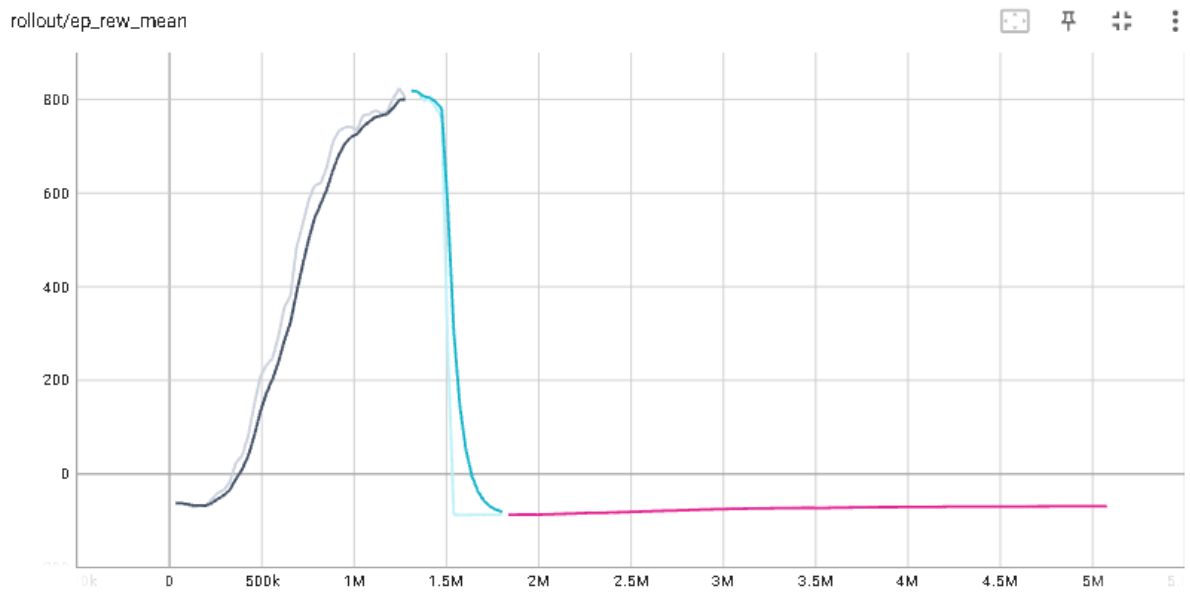

Figura 11: Evolución de la recompensa media por episodio durante el entrenamiento con curriculum learning. Cada cambio de color corresponde a una transición entre etapas del currículo.

Primera etapa: circuito sencillo Durante la primera fase del entrenamiento (hasta aproximadamente $1{,}3 \times 10^6$ timesteps), el agente se entrena en un conjunto de semillas que generan circuitos de baja complejidad (tramos largos y curvatura reducida). En esta etapa se observa un incremento rápido y estable de la recompensa media, alcanzando valores cercanos a 800. Este comportamiento indica que el agente aprende con eficacia las habilidades básicas de conducción: mantener el vehículo sobre la pista, controlar la aceleración y minimizar derrapes innecesarios.

El crecimiento casi monótono de la recompensa sugiere que el espacio de estados es suficientemente simple como para permitir una exploración eficiente y una señal de recompensa consistente.

Transición del currículo Alrededor de $1{,}4 \times 10^6$ timesteps se produce una caída abrupta de la recompensa media. Este descenso coincide con el cambio de etapa en el currículo, donde el agente pasa a entrenarse en circuitos de mayor complejidad (por ejemplo, circuitos ovalados o con curvas más cerradas).

Este fenómeno es esperable y constituye una señal positiva: la política aprendida en la etapa anterior no es directamente óptima para el nuevo dominio, pero proporciona una base inicial razonable. El agente debe reajustar su comportamiento para adaptarse a trayectorias más exigentes, lo que temporalmente degrada su rendimiento.

Segunda y tercera etapa: circuitos complejos Tras la transición, la recompensa media se estabiliza inicialmente en valores negativos, seguida de una recuperación lenta pero progresiva a lo largo del resto del entrenamiento (hasta los 5 millones de timesteps). Esta dinámica refleja la mayor dificultad del entorno final: los circuitos originales de

CarRacing presentan curvas pronunciadas, cambios frecuentes de dirección y una mayor penalización por salirse de la pista.

La pendiente suave de mejora indica que el agente sigue aprendiendo, aunque a un ritmo más lento que en la primera etapa. Esto es coherente con un espacio de estados más amplio y una función de recompensa más ruidosa.

Conclusión En conjunto, los resultados confirman la utilidad del enfoque de *curriculum learning*. El agente es capaz de adquirir primero habilidades fundamentales en entornos simplificados y reutilizarlas posteriormente al enfrentarse a tareas más complejas. Aunque cada transición del currículo introduce una caída temporal del rendimiento, el aprendizaje acumulado permite una adaptación más estable que si el entrenamiento se realizara directamente sobre circuitos complejos desde el inicio.

9.7. Evaluación

El modelo entrenado mediante *curriculum learning* fue evaluado sobre 20 episodios independientes en el entorno completo de *CarRacing*. Los resultados obtenidos muestran una recompensa media de $825{,}6 \pm 72{,}7$, junto con una longitud media de episodio de $493{,}1 \pm 16{,}0$ pasos. Estos valores indican que el agente es capaz de completar la mayor parte del circuito de forma consistente, manteniendo un comportamiento de conducción estable y eficiente. La desviación estándar moderada observada en la recompensa sugiere una buena robustez frente a la variabilidad inherente de los circuitos procedurales, mientras que la baja variabilidad en la duración de los episodios refleja un control fiable del vehículo y una tasa reducida de salidas de pista prematuras. En conjunto, estos resultados confirman que el enfoque propuesto permite aprender una política de conducción de alto rendimiento que generaliza adecuadamente a circuitos no vistos durante el entrenamiento.

9.8. Discusión

Si bien el entrenamiento mediante curriculum learning ha resuelto el entorno según los criterios preestablecidos, ha resultado una aproximación objetivamente mediocre, comparable a un entrenamiento ordinario.

Esto se debe a que la principal dificultad a la que se enfrenta el algoritmo reside en la interpretación de las imágenes que componen su espacio de observaciones, y es a esto a lo que dedica el grueso del entrenamiento. Mientras tanto, la dificultad inherente del circuito al que se enfrenta es meramente un añadido. Por ello, seleccionar semillas más difíciles, aquellas que en entrenamientos regulares serían meramente ruido para el agente, no tiene ningún interés añadido.

10. Conclusiones y Trabajo Futuro

En este proyecto se ha abordado el entrenamiento de agentes mediante aprendizaje por refuerzo profundo en el entorno CarRacing, utilizando el algoritmo PPO y un pipeline de preprocesamiento visual orientado a entornos continuos y de alta dimensionalidad. El objetivo principal fue desarrollar un agente capaz de aprender una política de conducción estable y analizar el impacto de distintas decisiones de diseño sobre el proceso de aprendizaje.

Conclusiones Principales

A lo largo del trabajo se han obtenido las siguientes conclusiones principales:

- Se ha implementado con éxito un **pipeline completo de entrenamiento y evaluación** basado en Gymnasium, Stable-Baselines3 y TensorBoard, permitiendo un seguimiento detallado del proceso de aprendizaje.
- La utilización de **Optuna para la optimización de hiperparámetros** ha demostrado ser una herramienta eficaz para sistematizar el ajuste del algoritmo PPO, reduciendo la dependencia de ajustes manuales y mejorando la estabilidad del entrenamiento.
- El entrenamiento final con los hiperparámetros seleccionados mostró una **dinámica de aprendizaje consistente**, con una rápida fase inicial de mejora seguida de una estabilización progresiva del comportamiento del agente.
- Desde el punto de vista cualitativo, el agente es capaz de **mantener el vehículo en pista durante episodios prolongados**, anticipar curvas y realizar correcciones suaves ante desviaciones moderadas.

No obstante, también se han identificado limitaciones claras. En situaciones de mayor dificultad —como curvas muy cerradas o cambios bruscos en la geometría del circuito— el control del vehículo puede degradarse temporalmente, lo que evidencia márgenes de mejora en la robustez del agente.

Análisis de Extensiones Exploradas

Además del pipeline principal, se exploraron distintas extensiones del enfoque base:

- El uso de **reward shaping** permitió estudiar el impacto de introducir preferencias de conducción más suaves y estables. Si bien este enfoque influyó en el comportamiento del agente, su efectividad depende en gran medida del diseño cuidadoso de la función de recompensa.
- Se implementó un enfoque de **curriculum learning** basado en la complejidad geométrica de las pistas, con el objetivo de facilitar un aprendizaje progresivo. Aunque este método resulta conceptualmente atractivo y mostró una dinámica razonable, no se observaron mejoras claras y consistentes frente al entrenamiento estándar.

Estos resultados ponen de manifiesto que no todas las extensiones habituales del aprendizaje por refuerzo conducen necesariamente a mejoras automáticas, y que su efectividad depende de un diseño experimental cuidadoso y del contexto concreto del problema.

Trabajo Futuro

A partir del trabajo realizado, se identifican varias líneas de investigación futuras:

- Realizar una **evaluación más exhaustiva de la capacidad de generalización** del agente, utilizando un mayor número de episodios y semillas de evaluación.
- Refinar el enfoque de **curriculum learning** mediante criterios de dificultad más informativos o estrategias adaptativas basadas en el rendimiento del agente.
- Explorar **algoritmos alternativos** al PPO, como SAC o TD3, así como arquitecturas de red que incorporen memoria temporal explícita.
- Analizar de forma sistemática el impacto de diferentes esquemas de **reward shaping**, con el fin de comprender mejor su influencia sobre el comportamiento emergente del agente.

En conclusión, este proyecto establece una base sólida para futuras extensiones en el aprendizaje por refuerzo aplicado a entornos visuales continuos y proporciona una visión práctica y crítica de los retos asociados al entrenamiento de agentes en este tipo de escenarios.