



**Universidad
Rey Juan Carlos**

**GRADO EN INGENIERÍA DE ROBÓTICA
SOFTWARE**

Escuela de Ingeniería de Fuenlabrada

Curso académico 2023-2024

Trabajo Fin de Grado

Seguimiento Autónomo de Carril en Drones usando
Inteligencia Artificial y Aprendizaje por Refuerzo

Autor: Bárbara Villalba Herreros

Tutor: Dr. Roberto Calvo Palomino



Este trabajo se distribuye bajo los términos de la licencia internacional CC BY-NC-SA International License (Creative Commons AttributionNonCommercial-ShareAlike 4.0). Usted es libre de *(a) compartir*: copiar y redistribuir el material en cualquier medio o formato; y *(b) adaptar*: remezclar, transformar y crear a partir del material. El licenciador no puede revocar estas libertades mientras cumpla con los términos de la licencia:

- *Atribución.* Usted debe dar crédito de manera adecuada, brindar un enlace a la licencia, e indicar si se han realizado cambios. Puede hacerlo en cualquier forma razonable, pero no de forma tal que sugiera que usted o su uso tienen el apoyo de la licenciante.
- *No comercial.* Usted no puede hacer uso del material con propósitos comerciales.
- *Compartir igual.* Si remezcla, transforma o crea a partir del material, debe distribuir su contribución bajo la la misma licencia del original.

Agradecimientos

En primer lugar quería agradecer a todas las personas que han sido parte de este camino y trayectoria, agradezco a mis tres familias por apoyarme y no dejarme rendirme en ningún momento. Agradeciendo así a la madre de mi pareja por estar escuchandome tendida y aconsejando me.

Mención especial a mi pareja Renato Luigi por estar a pie de cañón en todo momento ayudando, apoyando, y escuchando, no me olvidaré de las charlas que teníamos en el coche mientras cenabamos. También quería agradecer a mi padre por sus charlas telefónicas de vuelta a casa, en donde me intentaba ayudar con sus ideas. Además de mi madre, mi hermana y abuelos por estar siempre a mi lado.

En segundo lugar, quiero mencionar a mi cuñado Angelo Vincenzo por ser compañero de carrera y poder haber compartido una variedad de recuerdos que nunca olvidaré.

Además de agradecer a mi tutor Roberto por la paciencia que ha tenido durante el desarrollo de este trabajo y brindarme ánimos en el camino.

Finalmente, dar las gracias a las personas que no pueden estar en estos momentos.

*A alguien especial,
que esta en el cielo, Vincenzo Barra.
Bárbara Villalba*

Resumen

Una de las principales áreas de estudio de la robótica es la navegación autónoma. En la actualidad, esta se lleva a cabo mediante dos tipos de algoritmos, los que se podrían llamar clásicos que se basan en la planificación de rutas y trayectorias mediante algoritmos clásicos, o cada vez tomando más protagonismo aquellos basados en Inteligencia Artificial. Dentro de la robótica aérea se están creando una gran número de aplicaciones variadas gracias a la capacidad de los drones de volar, y tener una perspectiva del entorno y alcanzar zonas a los que un robot con ruedas no podría llegar.

En este TFG se explora un método de navegación basado en Inteligencia Artificial, en el que se procura que el dron se mueva dentro de un área determinada, en este caso un carril de una carretera. Este tipo de navegación, podría tener muchas aplicaciones, por ejemplo, la inspección de carreteras a gran altura para analizar el estado de las mismas.

Para conseguirlo se han usado técnicas de Deep Learning, como las Redes Neuronales de segmentación para la detección del carril y extracción de sus características, así como otras técnicas clásicas para mejorar la información extraída. Así como algoritmos de Reinforcement Learning, en este caso, Q-Learning, para que el sistema de control del dron fuese capaz de aprender las acciones a realizar en función de la información sensorial obtenida.

Todo ello se ha realizado empleando el simulador AirSim, en el que se simula un entorno realista, que el dron deberá completar la tarea anteriormente descrita. Además de emplear el middleware robótico ROS para las comunicaciones entre el simulador, y el software implementado en esta solución.

Acrónimos

UAV *Unmanned Air Vehicle*

UAS *Unmanned Air System*

IA *Inteligencia Artificial*

Airsim *Aerial Informatics and Robotics Simulation*

PX4 *Pixhawk Autopilot*

TFG *Trabajo de fin de Grado*

ROS *Robotic Operative System*

Mavros *MAVlink to ROS Interface*

GPU *Unidad de procesamiento gráfico*

CPU *Unidad central de procesamiento*

GPS *Global Positioning System*

Lidar *Light Detection and Ranging*

YOLOP *You Only Look Once for Panoptic Driving Perception*

ONNX *Open Neural Network Exchange*

DBSCAN *Density-Based Spatial Clustering of Applications with Noise*

PID *Proporcional Integral Derivativo*

MCU *Movimiento Circular Uniforme*

RL *Reinforcement Learning*

Índice general

1. Introducción	1
1.1. La robótica	1
1.1.1. Enfoques de control en el mundo de la robótica	2
1.2. Robótica aérea	6
1.3. La inteligencia artificial en la navegación autónoma de drones	13
1.4. Seguimiento autónomo de carriles basado en inteligencia artificial y aprendizaje por refuerzo para drones	15
2. Objetivos	16
2.1. Descripción del problema	16
2.2. Requisitos	17
2.3. Metodología	17
2.4. Plan de trabajo	19
3. Plataforma de desarrollo	21
3.1. Lenguaje de programación	21
3.1.1. Python	21
3.2. YOLOP	24
3.3. ROS	27
3.3.1. Mavros	28
3.4. Airsim	29
3.4.1. Airsim ROS Wrapper	33
3.4.2. Client Airsim	34
3.5. PX4 AutoPilot	34
3.5.1. Software in The Loop(SITL)	35
3.5.2. Modos de vuelo	35
3.6. QGroundControl	37
4. Diseño e Implementación	38
4.1. Arquitectura	38
4.2. Configuración del sistema cliente-servidor	41
4.2.1. Preparación del entorno de simulación	42
4.3. Comportamiento de seguimiento de carril con drones	45

4.3.1. Percepción	45
4.3.2. Análisis del algoritmo de percepción	63
4.4. Comportamientos de control	64
4.4.1. Sigue-carril basado en PID	64
4.4.2. Sigue-carril basado en aprendizaje por refuerzo	66
4.4.3. Comparativa PID vs Q-learning	77
5. Conclusiones	82
5.1. Objetivos cumplidos	82
5.2. Requisitos satisfechos	83
5.3. Balance global y competencias adquiridas	83
5.4. Líneas futuras	84
6. Anexo	1
A. Bibliografía	4
Bibliografía	4

Índice de figuras

1.1.	Definición de robot	2
1.2.	Perseverance	3
1.3.	Nereus	5
1.4.	El dron de rescate Rega	6
1.5.	Historia de los drones	7
1.6.	El dron Ingenuity	8
1.7.	Drones en inspección eléctrica en Galicia	9
1.8.	El primer prototipo de dron de Prime Air	10
1.9.	El dron MK27-2	10
1.10.	El dron MK30	11
1.11.	Resultados de detección y seguimiento de carreteras en Efficient Road Detection and Tracking for Unmanned Aerial [22]	12
1.12.	Clasificación de Inteligencia Artificial [11]	13
1.13.	Resultados de la detección y clasificación de malas hierbas en un cultivo [10]	14
1.14.	Esquema de Reinforcement Learning [7]	14
2.1.	Ilustración del tablero Kanban durante el desarrollo del TFG	18
2.2.	Seguimiento de trabajo en GitHub	19
3.1.	Arquitectura de YOLOP	25
3.2.	Resultado de la salida de la red neuronal YOLOP	26
3.3.	Definición de ROS	27
3.4.	Arquitectura de ROS	28
3.5.	Infraestructura de Mavros	29
3.6.	Ejemplos de escenarios en Airsim	32
3.7.	Diagrama del comportamiento del modo de vuelo Position	36
3.8.	QGroundControl	37
4.1.	Arquitectura general del desarrollo en este TFG	39
4.2.	Diagrama de comunicaciones	42
4.3.	Visualización del entorno original ilustrando los objetos que dificultan al sistema perceptivo	43
4.4.	Configuración del dron mediante el fichero settings.json	44

4.5. Resultado de la detección de líneas utilizando visión clásica	46
4.6. Resultados de los diferentes modelos que ofrece la red YOLOP	51
4.7. Resultados de la inferencia del modelo yolop-320-320.onnx	52
4.8. Fases de reconstrucción de las líneas del carril	52
4.9. Proceso del resultado de los diferentes modelos de la red neuronal YOLOP	53
4.10. Resultado de la clasificación de las líneas por DBSCAN	56
4.11. Proceso de filtrado de las líneas clasificadas por DBSCAN	58
4.12. Resultado de la regresión cuadrática	60
4.13. Comparativa entre el resultado de detección Yolop-320-320.onnx y la construcción del carril desarrollada	62
4.14. Resultado del centro de masas	63
4.15. Sigue-carril basado en PID	65
4.16. Entorno en la fase de entrenamiento del sigue carril basado en Q-Learning .	67
4.17. Estados definidos para el sigue-carril basado en aprendizaje por refuerzo .	68
4.18. Acciones disponibles para el sigue-carril basado en aprendizaje por refuerzo	68
4.19. Gráfica del valor de epsilon respecto al episodio	71
4.20. Gráfica de las iteraciones respecto al episodio	73
4.21. Gráfica de la recompensa acumulativa en función de los episodios . . .	73
4.22. Distribución de acciones en inferencia	75
4.23. Resultado del comportamiento de aprendizaje por refuerzo ilustrando los estados y acciones en la fase de inferencia	75
4.24. Distribución de acciones en inferencia	77
4.25. Escenario de control tradicional y aprendizaje por refuerzo	77
4.26. Comparativa de trayectorias entre el control clásico y el control de aprendizaje por refuerzo	78
4.27. Resultado del comportamiento de aprendizaje por refuerzo	79
4.28. Escenario II de control tradicional y aprendizaje por refuerzo	80
4.29. Trayecto del comportamiento PID y Q-learning	80
4.30. Comparativa entre el PID y Q-learning	81

Listado de códigos

3.1. Ejemplo de código en Python de una función para calcular el factorial de un número	22
3.2. Ejemplo de código en Python de operaciones básicas utilizando la librería OpenCv	22
3.3. Ejemplo de código en Python de operaciones básicas utilizando la librería Numpy	23
3.4. Cargar modelo YOLOP con pesos preentrenados End-to-end.pth	26
4.1. comando	44
4.2. comando	45
4.3. Cargar modelo YOLOP con pesos preentrenados End-to-end.pth	47
4.4. Cargar modelo YOLOP escogiendo como dispositivo la GPU	48
4.5. Cargar modelo	48
4.6. Inferencia del modelo yolop-320-320.onnx	49
4.7. Inferencia del modelo	49
4.8. Resultado de la inferencia del modelo YOLOP	50
4.9. Algoritmo de clustering utilizando DBSCAN	55
4.10. Clasificación de clústeres según las dimensiones de la imagen	57
4.11. Función maximizada para escoger el grupo de cluster más cercano y denso respecto al punto P	57
4.12. Calculo de los coeficientes	59
4.13. Cálculo de la regresión cuadrática	59
4.14. Método de interpolación	61
4.15. Valores de las variables del PD del control de altura y del PID del controlador de velocidad angular	65
4.16. Función de recompensa	69

Listado de ecuaciones

4.1. Calculo del centro de masas	62
4.2. Función de recompensa	69
4.3. Ecuación de Bellman	71
4.4. Ecuación del movimiento circular uniforme	76

Índice de cuadros

4.1. Profiling	63
--------------------------	----

Capítulo 1

Introducción

La evolución tecnológica ha provocado una transformación radical en nuestra forma de vivir, trabajar y relacionarnos, desempeñando la tecnología un papel fundamental en el avance de la sociedad e impulsando una serie de innovaciones que se extienden desde la invención de la rueda hasta la era digital contemporánea. Por ejemplo, los ordenadores empezaron siendo grandes máquinas que ocupaban habitaciones enteras. Hoy en día, los ordenadores son dispositivos ligeros y eficientes que pueden realizar múltiples cálculos por segundos que se utilizan en diferentes ramas de las ingenierías como la informática, telecomunicaciones y, por supuesto, la robótica.

La robótica, en particular, se destaca como una de las ramas de la tecnología que más impacto significativo ha tenido en todo el mundo. Estos avances han facilitado numerosas tareas mejorando la eficiencia y la capacidad de enfrentar desafíos complejos dando pie a nuevas posibilidades en nuestro entorno. Un ejemplo de ello puede ser la robótica aérea con el uso de los drones, demostrando ser desafiantes y valiosos en la inspección de áreas de difícil acceso, el mapeo de terrenos, la realización de entregas de paquetería, la navegación autónoma o la captura de imágenes desde alturas elevadas. Su versatilidad y su capacidad para poder operar en entornos peligrosos o inaccesibles para los seres humanos los convierten en herramientas fundamentales en campos como la agricultura, la seguridad, la investigación medioambiental y la logística.

1.1. La robótica

Entre las diversas ramas de la tecnología, la robótica se destaca como una de las más prometedoras. Apareciendo como disciplina durante la década de los años 60, la robótica ha tenido un cambio asombroso pasando de ser simples máquinas programables a sistemas inteligentes capaces de aprender y adaptarse a su entorno teniendo avances en diversas disciplinas de la ingeniería, como la informática, la inteligencia artificial, la ingeniería de control, la mecánica, la electricidad y la electrónica. Los robots de hoy en día no solo tienen la capacidad de realizar tareas programadas y repetitivas, sino que también tienen la capacidad de interactuar con su entorno, tomar decisiones basadas en la información

sensorial y aprender de sus experiencias. Este avance en la robótica nos ha permitido tener una definición más precisa de lo que es la robótica moderna, definiendo la robótica como ciencia interdisciplinaria encargada de la creación, funcionamiento, estructuración, fabricación y uso de los robots. Esta definición incluye no solo los componentes mecánicos y eléctricos, sino que también los algoritmos de control, los sensores que les permiten recopilar datos de su entorno y los sistemas que procesan esta información y toman decisiones.

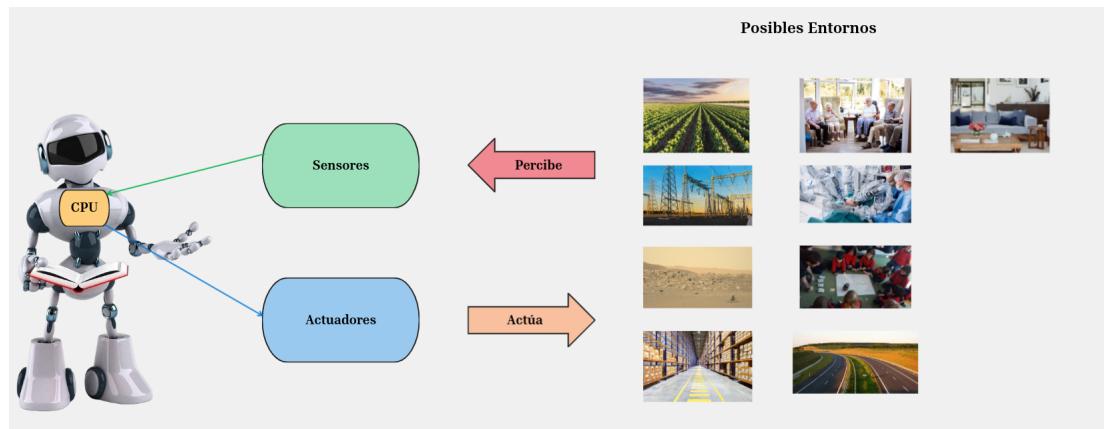


Figura 1.1: Definición de robot.

La capacidad de los robots para aprender y adaptarse a su entorno abre nuevas oportunidades en campos como la medicina, la exploración lunar, la asistencia personal, la automatización industrial, etc. Además de abrir nuevas aplicaciones y tareas como puede ser la navegación autónoma, la detección de objetos o la manipulación de objetos con sensores táctiles y de fuerza, dichas tareas que pueden realizar, pueden ser peligrosas, delicadas, sucias o monótonas (conocidas como las 4D's: dull,dirty, dangerous and dear)¹

1.1.1. Enfoques de control en el mundo de la robótica

A lo largo de la evolución de la robótica, han surgido tres enfoques fundamentales para el diseño y la operación de robots. Cada uno de estos enfoques presentan diferentes formas de interactuar y operar robots, con sus riesgos, características y aplicaciones únicas.

Teleoperación

La teleoperación surge de la necesidad de manipular objetos o realizar tareas en entornos complejos, peligrosos y distantes para el ser humano. Desde la historia, el

¹<https://www.forbes.com/sites/bernardmarr/2017/10/16/the-4-ds-of-robotization-dulldirty-dangerous-and-dear/?sh=40bb6cec3e0d>

ser humano ha utilizado una variedad de herramientas para ampliar su capacidad de manipulación como varas utilizadas para hacer caer la fruta madura de un árbol. Con el tiempo, se desarrollaron dispositivos más complejos, como pinzas que permitían manipular piezas o alcanzar objetos de difícil acceso, facilitando el trabajo para el operario. En la era moderna, la teleoperación ha evolucionado hasta el punto de incluir sistemas robóticos robustos controlados a distancia, permitiendo al operario poder realizar tareas en entornos peligrosos e inaccesibles para el ser humano como puede ser la exploración espacial, la medicina o la inspección nuclear. La intervención del operador humano en los sistemas de teleoperación de robots es imprescindible, debe ser capaz de poder interpretar los datos sensoriales que proporciona el robot, así como de tomar decisiones robustas y precisas dependiendo de la situación. Esto conlleva tener una capacidad de realizar múltiple tareas simultáneamente adaptándose a situaciones imprevistas.

Hoy en día, la teleoperación de robots tiene variedad de aplicaciones. Una de ellas es la exploración espacial, donde se utiliza la teleoperación como técnica de manipulación remota como el Perseverance. Como se muestra en la figura 1.2, el Perseverance² es un robot móvil compuesto por 6 ruedas diseñado por la NASA para la exploración marciana. Su objetivo principal es estudiar la habitabilidad del planeta y preparar el terreno para futuras misiones tripuladas. A futuro, se espera que las muestras recolectadas de este rover sean traídas a la Tierra para un análisis más detallado.

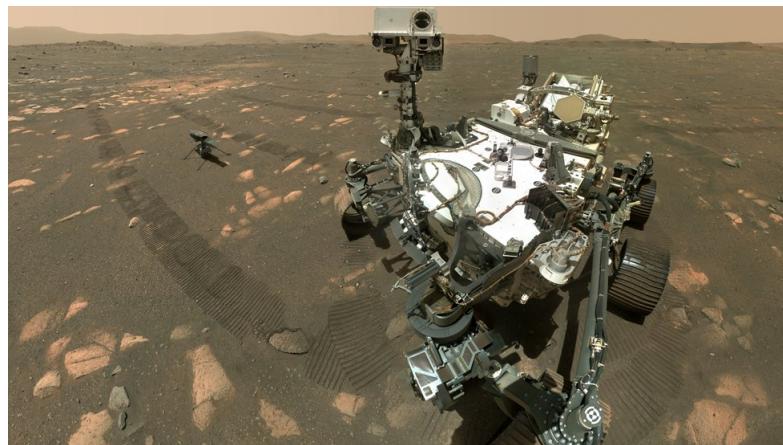


Figura 1.2: Perseverance

A pesar de ser un buen enfoque en cuanto a controlar un robot, presenta sus propias limitaciones, como la dependencia de una conexión continua y confiable entre el operario y el robot. Si la conexión se interrumpe, el control del robot podría perderse desembocando situaciones de grave peligro, como la perdida de conexión entre un rover entre la Tierra y Marte. Otro tipo de limitación puede ser la carga cognitiva que puede tener el operario

²<https://science.nasa.gov/mission/mars-2020-perseverance/>

al controlar el robot, ya que el operario debe permanecer concentrado monitorizando y controlando el robot de manera constante. Lo último puede conducir a errores humanos, especialmente durante operaciones de larga duración, lo que hace interesante tener otro tipo de enfoque de control.

Robótica Semiautónoma

Los robots pueden realizar tareas de forma independiente siguiendo instrucciones preprogramadas o tomando decisiones en tiempo real, este enfoque se le conoce como autonomía o semi-autonomía, siendo la diferencia que en el enfoque semi-autónomo todavía existe parte de teleoperación en el robot. Este enfoque permite que los robots puedan ser autónomos para poder percibir su entorno y en la toma de decisiones, pero con el handicap de que un operario humano pueda controlarlo para poder ajustar parámetros, cambiar objetivos o intervenir en caso de emergencia.

Aunque los robots semiautónomos puedan tomar decisiones en tiempo real, a menudo siguen instrucciones preprogramadas o reciben órdenes de un operario humano, esta toma de decisiones puede incluir elegir la ruta más eficiente para navegar por un entorno peligroso como puede ser el robot submarino llamado Nereus como se ilustra en la figura 1.3. El Nereus³ es un vehículo submarino semi-autónomo que entró en servicio en 2009, fue diseñado para explorar el Abismo Challenger en la Fosa de las Marianas, el punto más profundo conocido en los océanos. Este robot podía ser controlador remotamente desde un barco en la superficie, además de tener la capacidad de operar de manera autónoma, adaptándose a las condiciones del entorno sin intervención humana directa.

Lamentablemente, en 2014 durante una misión, el robot Nereus sufrió un colapso estructural y se perdió en el fondo del océano. A pesar de esta perdida, los datos recopilados siguen siendo una fuente de conocimiento sobre las profundidades marinas. Este incidente demuestra que los robots semiautónomos pueden operar en entornos peligrosos sin poner en riesgo vidas humanas, aunque requieran control por un operario⁴.

³https://www.bbc.com/mundo/ciencia_tecnologia/2009/06/090603_1541_nereus_robot_mar_mr

⁴<https://www.elperiodico.com/es/ciencia/20140512/famoso-sumergible-nereus-pierde-fondo-mar-3271389>

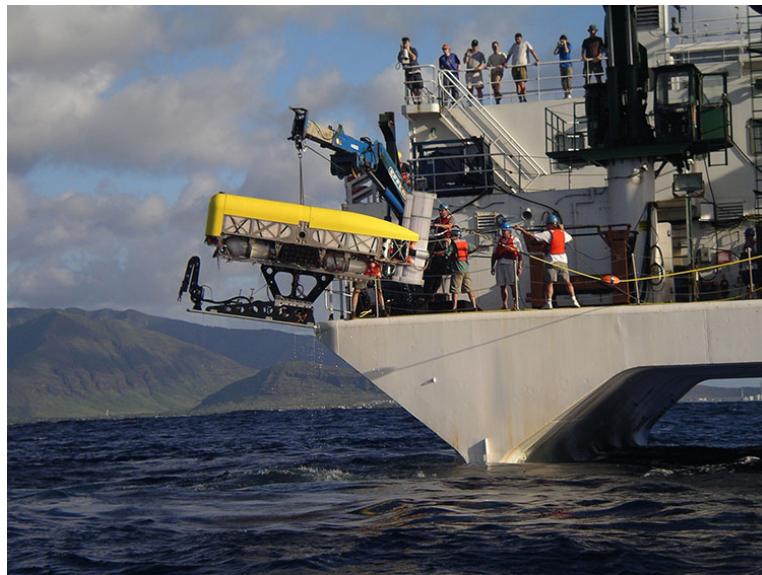


Figura 1.3: Nereus

En cuanto a las debilidades de la robótica semiautónoma, estos robots aún requieren intervención humana para tareas complejas o situaciones imprevistas, aunque la dependencia del operario sea menor todavía sigue siendo significativa. Se debe garantizar la seguridad y la fiabilidad de estos sistemas semi-autónomos, cualquier fallo en la autonomía del robot o en la intervención humana puede tener consecuencias peligrosas y asimismo de que los algoritmos perceptivos y de control de los robots semiautónomos deben ser eficientes y robustos ante situaciones cambiantes.

Robótica Autónoma

La robótica autónoma consiste en desarrollar robots que sean capaces de operar y realizar tareas de forma independiente sin la intervención de un ser humano. En contraste con los robots teleoperados, este tipo de robots necesitan un comportamiento más robusto y preciso para realizar tareas independientes basándose en la percepción del entorno y en la toma de decisiones autónomas. El concepto de autonomía en los sistemas robóticos se está convirtiendo en un área de investigación activa y en rápido desarrollo. Los avances en inteligencia artificial (IA), visión artificial, aprendizaje automático han facilitado la creación de robots autónomos capaces de llevar a cabo amplias variedades de tareas en entornos no estructurados y cambiantes. Uno de los grandes desafíos que enfrenta la robótica autónoma es cómo el robot puede realizar la percepción del entorno, identificando y comprendiendo objetos y situaciones de manera precisa y en tiempo real.

Por ejemplo, como se muestra en la figura 1.4, el robot autónomo parecido a un helicóptero diseñado por investigadores suizos es capaz de realizar tareas de rescate y

búsqueda en los Alpes suizos⁵. Este dron autónomo puede llegar a escanear amplias zonas de montaña y reconocer personas en tierra de manera autónoma mediante cámaras y algoritmos de aprendizaje automático desarrollados por la universidad ETH Zúrich. Facilitando las tareas de rescate al equipo de rescate Rega siguiendo rutas predefinidas sin intervención humana directa localizando a personas atrapadas o en peligro en áreas remotas o accidentadas.



Figura 1.4: El dron de rescate Rega

1.2. Robótica aérea

La robótica aérea se refiere al desarrollo y uso de robots capaces de volar en el aire. Estos robots pueden operar de forma autónoma o ser controlados a distancia.

Dentro del campo de la robótica aérea se encuentran los drones. Se define dron, como vehículo aéreo no tripulado (UAV), es un tipo de aeronave que puede operar sin la necesidad de un piloto humano a bordo. Estos dispositivos pueden ser controlados remotamente por un operador humano o navegar autonómicamente incorporando software en su sistema. En la figura 1.5 se ilustra la historia de los drones, desde la Primera Guerra Mundial hasta la actualidad.

El origen de los drones se remonta a la Primera Guerra Mundial con el biplano Kettering Bug. Este era un torpedo no tripulado de 240 kg (con una envergadura de 4,5 m, una longitud de 3,8 m y una altura de 2,3 m)⁶ que era capaz de volar de forma autónoma hasta un punto específico. Avanzando en la historia, en 1935 se desarrolló el DH.82 Queen Bee⁷, un blanco aéreo sin piloto controlado por radio, inspiró a la creación

⁵<https://www.swissinfo.ch/spa/ciencia/drones-suizos-al-rescate/46203902>

⁶<https://www.nationalmuseum.af.mil/Visit/Museum-Exhibits/Fact-Sheets/Display/Article/198095/kettering-aerial-torpedo-bug/>

⁷<https://dronewars.net/2014/10/06/rise-of-the-reapers-a-brief-history-of-drones/>

del término ”dron”.

Durante la Segunda Guerra Mundial, el V-1 ”Flying Bomb”⁸ fue el primer misil de crucero operativo del mundo, en donde su sistema de guía preestablecido incluía una brújula magnética que monitorizaba un auto-piloto con giroscopios. También en este periodo, se destaca el *Project Aphrodite* [9], fue un programa que tenía como objetivo convertir bombarderos en bombas voladoras no tripuladas que eran controladas por radio. Más adelante estos bombarderos no tripulados se utilizaron para volar a través de nubes de hongo después de las pruebas nucleares.



(a) Kettering Bug



(b) Queen Bee



(c) V-1



(d) Project Aphrodite



(e) Teledyne Ryan Firebee/Firefly



(f) Lockheed D-21



(g) El Condor



(h) Small UAV



(i) Dron

Figura 1.5: Historia de los drones

Destacando más UAVs, tenemos la familia Teledyne Ryan Firebee/Firefly⁹, estos sistemas generalmente se lanzaban desde el aire y se recuperaban mediante una combinación de paracaídas y helicópteros. El Lockheed D-21 fue uno de los sistemas más impresionantes durante la Guerra Fría. Este UAV fue propulsado por estatorreactor

⁸<https://migflug.com/jetflights/the-v1-flying-bomb/>

⁹<https://www.designation-systems.net/dusrm/m-34.html>

con velocidades mayores que Mach 3¹⁰. En la Edad Moderna, destacamos El Condor [4], fue el primer UAS en utilizar navegación GPS y tecnología de aterrizaje automático y el Predator¹¹. En la época dorada, gracias a los avances anteriores se pudo desarrollar sistemas militares esenciales que han demostrado su valor y el desarrollo de vehículos aéreos no tripulados pequeños (small UAV). Este último ha despertado un gran interés significativo resaltando como puntos de entrega al mercado civil ya que con sus cargas útiles reducidas pueden ser portátiles y tener un coste menor.

Cada vez es más común que los drones sean más sofisticados y accesibles. Por ejemplo, el dron Ingenuity de la NASA se ha convertido en el primer vehículo aéreo autónomo en poder volar sobre la superficie de otro planeta. Fue transportado a Marte mediante el rover Perseverance de la NASA, una vez fue posicionado el dron se elevó cerca de 3 metros realizando diferentes giros y desplazamientos tomando fotos a la superficie, teniendo la capacidad de escoger de forma autónoma los sitios de aterrizaje en el terreno marciano¹². Este dron operaba de manera autónoma, controlado por sistemas de guía, navegación y control a bordo ejecutando los diferentes algoritmos desarrollados por la NASA.

Uno de los grandes retos de este proyecto es demostrar la viabilidad del vuelo en la atmósfera de Marte, ya que su atmósfera esta compuesta por el 1% de la densidad terrestre dificultando el vuelo del dron. Sin embargo, gracias a su diseño ligero y a sus hélices especialmente diseñadas para crear suficiente sustentación en la atmósfera del planeta, el Ingenuity fue capaz de superar este desafío¹³.

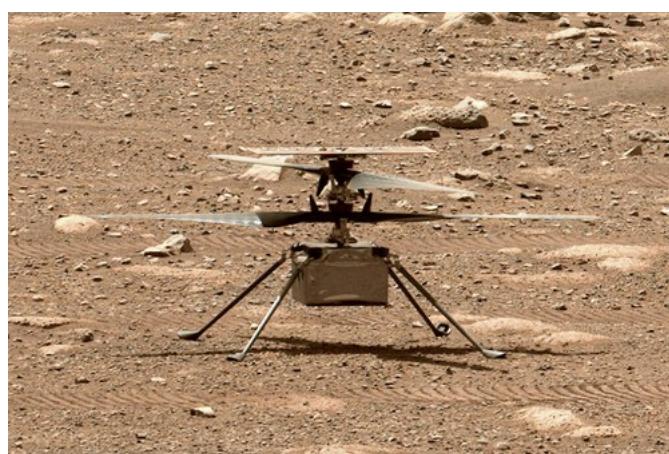


Figura 1.6: El dron Ingenuity

¹⁰<https://www.marchfield.org/aircraft/unmanned/d-21-drone-lockheed/>

¹¹<https://www.airforce-technology.com/projects/predator-uav/?cf-view>

¹²<https://ciencia.nasa.gov/sistema-solar/finaliza-la-mision-del-helicoptero-ingenuity-en-marte/>

¹³<https://www.bbc.com/mundo/noticias-56738201>

Además, en su última fase, el Ingenuity realizó pruebas de vuelo experimentales para ampliar el conocimiento sobre cuáles eran sus límites aerodinámicos¹⁴.

Otro ejemplo de uso de drones podemos tener el mantenimiento y control de redes eléctricas y otras infraestructuras. Algunas construcciones constan de grandes alturas y tamaños, lo que puede dificultar el trabajo y su correcto mantenimiento. No obstante, estas tareas con los drones se agilizan y se vuelven más eficientes y robustas, porque permiten poder inspeccionar dichas infraestructuras desde cerca sin poner en peligro a la seguridad de los operarios. Hay drones que se encargan en la monitorización de infraestructuras eléctricas.

Unión Fenosa, la distribuidora eléctrica en España de Naturgy, en 2018 incorporó drones a sus instalaciones eléctricas para realizar labores de supervisión. Estos drones aportan soluciones optimizadas y eficientes en costes. Si tenemos en cuenta la longitud que puede tener las redes eléctricas, el uso de estos vehículos autónomos facilita las tareas de supervisión equipados de cámaras de última generación permitiendo al operario observar en tiempo real el estado de las infraestructuras. Además de que los drones podrían acceder a zonas de difícil acceso para comprobar daños y poder repararlos¹⁵.

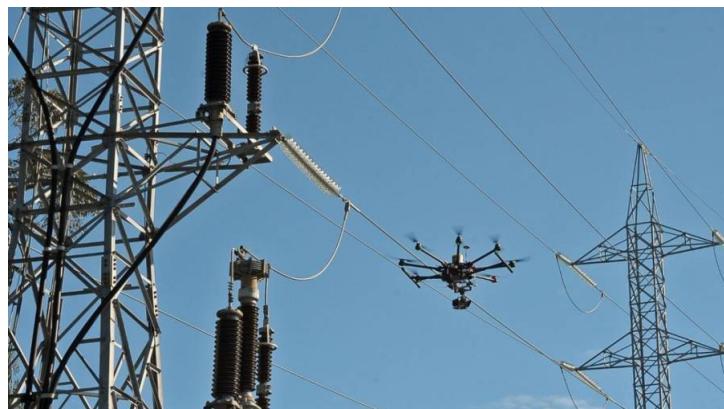


Figura 1.7: Drones en inspección eléctrica en Galicia

Es importante mencionar que estos drones son teleoperados, lo que significa que requieren la intervención y el control directo de un operador humano para volar y realizar sus tareas de inspección y mantenimiento.

Asimismo, Amazon ha estado trabajando en el desarrollo de drones autónomos para la entrega de paquetes durante varios años denominado así Prime Air[12], que consiste en un sistema de entrega de paquetes utilizando estos vehículos. Durante este programa, han realizado diferentes pruebas de reparto de paquetes a clientes en 60 minutos o menos.

¹⁴<https://science.nasa.gov/mission/mars-2020-perseverance/ingenuity-mars-helicopter/>

¹⁵<https://www.ufd.es/blog/primer-vuelo-de-un-dron-mas-allá-de-la-línea-visual/>



Figura 1.8: El primer prototipo de dron de Prime Air

A lo largo de los años, Amazon ha seguido investigando y diseñando nuevos modelos de drones como el dron autónomo MK27-2¹⁶. Fue el primer dron que utilizó Amazon para las primeras entregas dentro del programa Prime Air durante el año 2023, se basaba en un dron eléctrico capaz de entregar paquetes a los clientes en menos de una hora y capaz de realizar vuelos evitando obstáculos como puede ser las chimeneas o las torres de telefonía aunque no puede realizar entregas durante tormentas, vientos fuertes, temperaturas extremas o cualquier situación climatológica desfavorable.

Este servicio solamente está disponible para domicilios que tengan patios traseros que dispongan de espacio suficiente para que el dron pueda realizar el aterrizaje y la entrega del pedido.



Figura 1.9: El dron MK27-2

Sin embargo, gracias al dron autónomo MK30 creado y diseñado por Amazon. Este pequeño dron será capaz de volar en diferentes condiciones climatológicas y constará de un sistema capaz de identificar y evitar obstáculos en el área de entrega. Una novedad de este dron en comparación con los anteriores modelos es que será capaz de aterrizar en espacios más reducidos lo que conlleva a que este tipo de servicio pueda llegar a más vecindarios.

¹⁶<https://www.europapress.es/portaltic/gadgets/noticia-amazon-prime-air-comienza-entregar-pedidos-drones-estados-unidos-20221229115034.html>

Se tiene previsto que se llegue a probar en el año 2024 empezando por ciudades como Texas y California en Estados Unidos¹⁷.



Figura 1.10: El dron MK30

La navegación autónoma de drones sigue siendo un campo de investigación que busca permitir que los drones puedan volar de manera autónoma y segura. Dentro de este ámbito, la detención y el seguimiento de carreteras se destacan como áreas prometedoras, un ejemplo de investigación sobre este campo, podemos tener este artículo *Efficient Road Detection and Tracking for Unmanned Aerial* [22] que tiene como objetivo desarrollar un algoritmo de detección y seguimiento de carreteras específicas en vídeos capturados por vehículos aéreos no tripulados (UAV). Para la detección de carreteras utilizan un algoritmo denominado Graph-Cut¹⁸, que consiste en identificar y segmentar la imagen capturada por el dron para establecer la zona de interés, pero para obtener una segmentación más precisa y robusta de la carretera se combina con un modelo estadístico denominado GMM¹⁹ (Gaussian Mixture Model) para modelar las características de la imagen y representar regiones o clases en la imagen (por ejemplo, carretera, fondo, vehículos).

Una vez se realiza la identificación de la carretera, se utiliza un algoritmo basado en homografía (técnica geométrica), para ajustar la posición y la orientación del dron en relación con la carretera. Este tipo de algoritmos de seguimiento de carreteras permite al dron seguir automáticamente las áreas que se definieron de la carretera.

Este enfoque puede tener aplicaciones como la monitorización del tráfico y seguridad vial, seguimiento de vehículos terrestres o construcción de redes de carreteras para simulación. En un futuro cercano, puede que los drones sean más eficientes para las aplicaciones civiles y científicas incluyendo protección contra incendios forestales, misiones agrícolas, ayuda en catástrofes y más. Las demostraciones actuales del uso de los drones han revelado el potencial que pueden tener pero aun así el acceso al espacio aéreo sigue

¹⁷<https://www.forbesargentina.com/innovacion/asi-nuevo-asombroso-dron-amazon-mk30-n42612>

¹⁸<https://www.sciencedirect.com/topics/engineering/graph-cut-technique>

¹⁹<https://builtin.com/articles/gaussian-mixture-model>

siendo un factor limitante. Con el paso del tiempo, se irá desarrollando nuevas tecnologías prácticas para poder permitir una integración segura en el espacio aéreo [14].

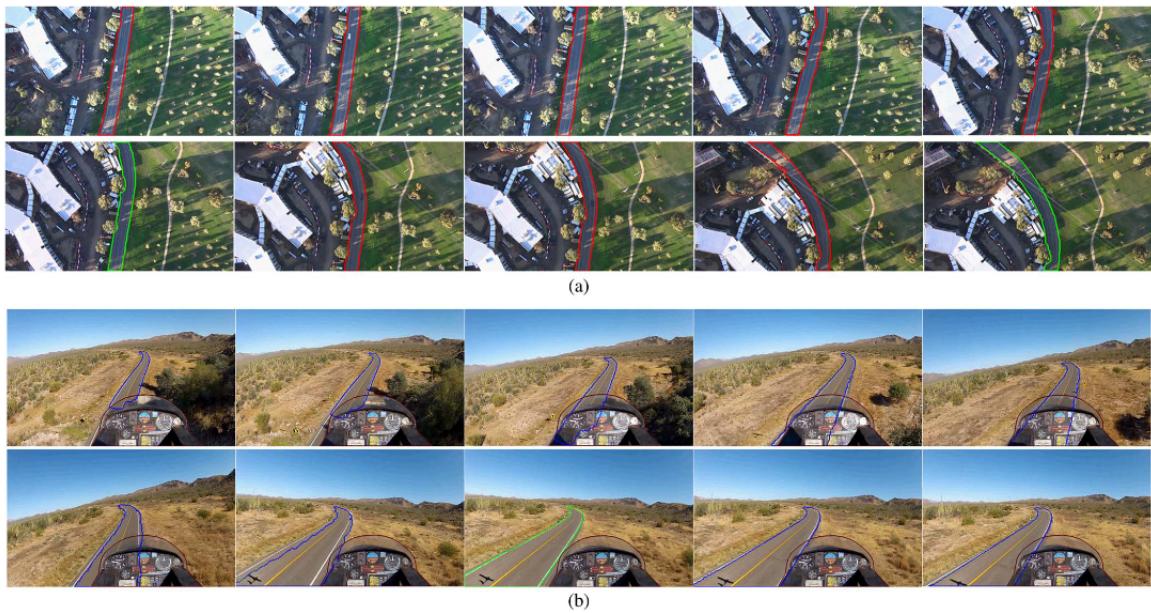


Figura 1.11: Resultados de detección y seguimiento de carreteras en Efficient Road Detection and Tracking for Unmanned Aerial [22]

En el artículo *Automatic Damage Detection and Diagnosis for Hydraulic Structures Using Drones and Artificial Intelligence Techniques* [23], se explora el uso de drones para monitorizar y diagnosticar daños estructurales en presas hidráulicas. El objetivo principal de este estudio es detectar y evaluar el estado de grietas en estas estructuras. Para lograrlo, se emplean algoritmos de visión por computadora e inteligencia artificial. En particular, se utiliza una red neuronal llamada Xception [17] junto con algoritmos de segmentación semántica de imágenes para detectar áreas afectadas. Los resultados experimentales, presentados en el artículo, demuestran la eficacia de esta detección de daños en las presas hidráulicas mediante el uso de técnicas de visión e inteligencia artificial.

En resumen, los drones son una tecnología emergente con un potencial significativo para transformar una variedad de industrias. Sin embargo, también plantean desafíos únicos que deben ser abordados a medida que se integran más plenamente en nuestra sociedad. Con el desarrollo continuo de la tecnología de los drones y la evolución de las regulaciones, es probable que veamos un aumento en la variedad de las aplicaciones de los drones en el futuro.

1.3. La inteligencia artificial en la navegación autónoma de drones

La incorporación de inteligencia artificial en el mundo de la robótica y en especial en los drones desempeña un papel crucial en la navegación autónoma, permitiéndoles tomar decisiones en tiempo real y adaptarse a entornos cambiantes de manera eficiente. Esto permite a los drones aprender de sus experiencias, comprender e interactuar con su entorno de forma más óptima.

Los drones equipados con IA de percepción o de control pueden realizar vuelos de precisión, mantener la estabilidad incluso en condiciones adversas como fuertes vientos, y evitar obstáculos de forma dinámica. Esto es posible gracias a la combinación de datos sensoriales junto con los algoritmos de IA, lo que permite al dron interpretar su entorno y tomar decisiones en tiempo real. Uno de los enfoques más destacados en la navegación autónoma de drones es el aprendizaje automático. Este enfoque permite a los drones mejorar su objetivo a través de la experiencia y los datos recopilados durante el vuelo.

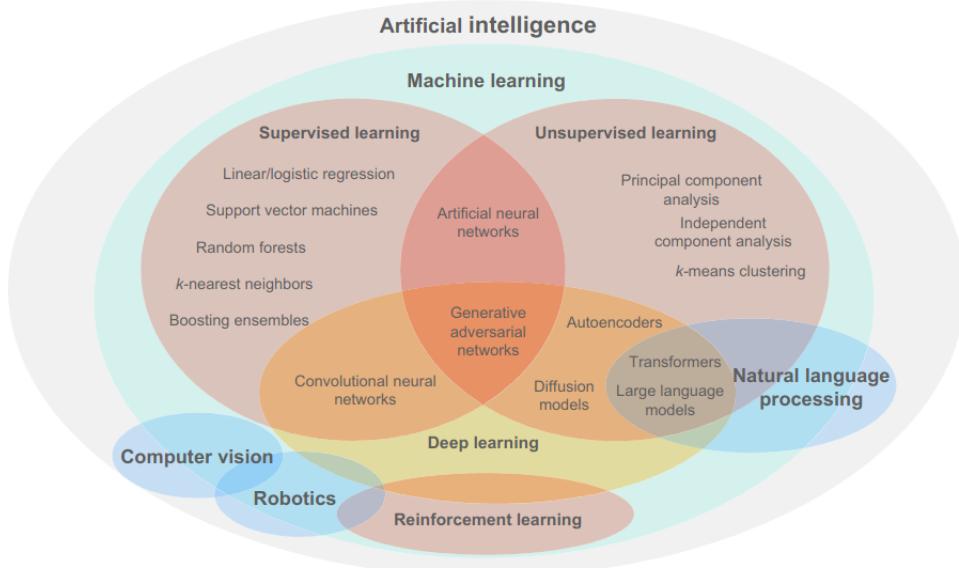


Figura 1.12: Clasificación de Inteligencia Artificial [11]

Por ejemplo, las CNN son capaces de analizar imágenes capturadas por las cámaras a bordo del dron para identificar obstáculos, peatones o vehículos. Un tipo de aplicación de uso de redes neuronales es la detección y clasificación de malas hierbas como se muestra en el artículo *Weed detection and classification using UAVs and deep neural networks: mapping for localized treatment* [10]. Mediante el sensor de la cámara, el dron es capaz de capturar imágenes en tiempo real para más adelante usar la red neuronal CNN YOLOv8 [19] para detectar y clasificar las diferentes hierbas que puede haber en un campo de

cultivo. Este tipo de aplicación es bastante útil para la inspección agrícola ya que los drones pueden crear mapas detallados que permiten a los agricultores aplicar herbicidas de manera más eficiente y precisa, también este tipo de aplicación puede ser útil para tener una monitorización general sobre la salud del cultivo. El resultado se puede visualizar en la figura 1.13.

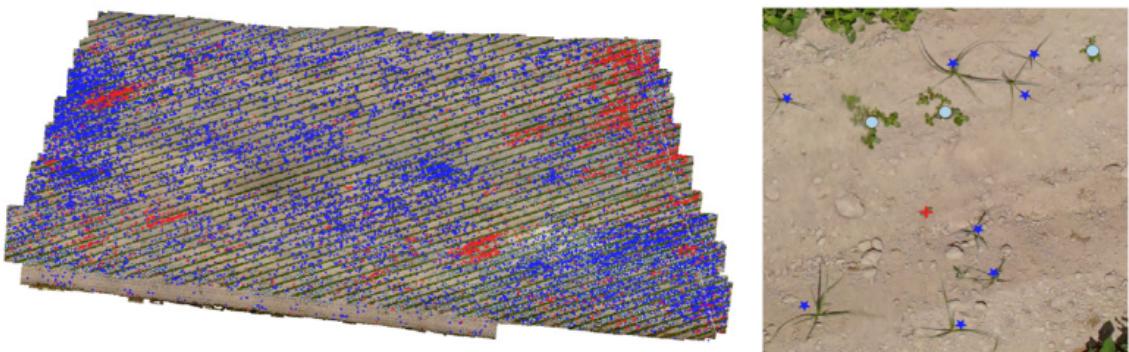


Figura 1.13: Resultados de la detección y clasificación de malas hierbas en un cultivo [10]

Por otro lado, reinforcement learning (RL) [16] es una técnica dentro del aprendizaje automático que es interesante utilizar en la navegación autónoma de drones. Este enfoque permite a los drones aprender a planificar rutas de forma autónoma, mejorando su desempeño a mediante un esquema de penalizaciones y recompensas permitiendo así al dron poder tomar decisiones decisivas en situaciones puntuales. En el artículo *Vision based drone obstacle avoidance by deep reinforcement learning* [20] precisamente se utiliza un algoritmo de RL para la evitación de obstáculos en un espacio continuo y se llega a conseguir que con estos tipos de algoritmos que un dron pueda llegar a aprender comportamientos y tomar decisiones por él mismo.

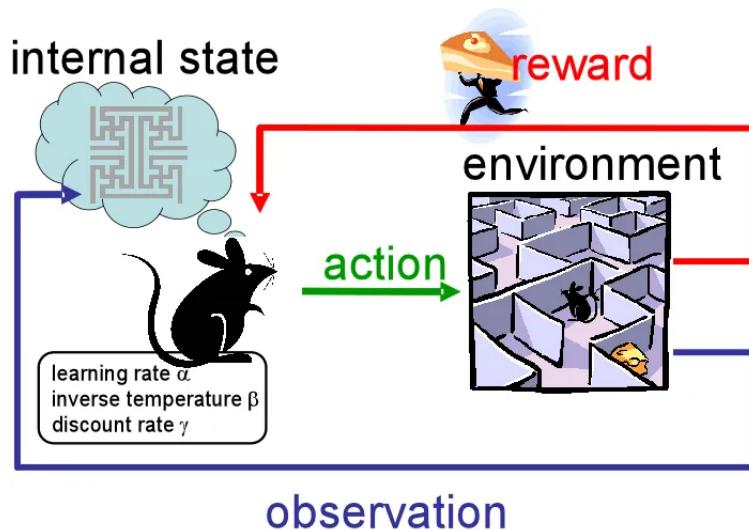


Figura 1.14: Esquema de Reinforcement Learning [7]

En conclusión, la inteligencia artificial es fundamental en la navegación autónoma de drones al permitirles percibir su entorno, podemos tomar decisiones y planificar acciones de manera anticipada y autónoma. A medida que la tecnología vaya avanzando, se espera que los drones tengan más sistemas de inteligencia artificial abordo para cubrir una amplia gama de tareas de manera autónoma, lo que abriría nuevas fronteras en campos como el rescate, la vigilancia, la logística y la exploración, y que promete seguir transformando la forma en que interactuamos con el espacio aéreo en un futuro.

1.4. Seguimiento autónomo de carriles basado en inteligencia artificial y aprendizaje por refuerzo para drones

En este TFG se desarrolla un algoritmo basado en navegación autónoma para drones, centrado en el seguimiento de carriles en entornos de carreteras. Para abordar esta problemática, se propone el uso de técnicas de Deep Learning, incluyendo Redes Neuronales, combinadas con métodos de aprendizaje autómático para tareas como la segmentación y reconocimiento de carriles. Además, de combinar algoritmos de Reinforcement Learning, en especial Q-Learning, lo que permite al dron adaptarse al entorno mediante datos sensoriales proporcionados por el sistema perceptivo.

Con este trabajo de investigación se busca impulsar el uso de la tecnología de drones en la planificación y eficiencia en el ámbito de las carreteras con la integración de IA y RL. Estos métodos tienen aplicaciones potenciales significativas, como la monitorización de carreteras para mejorar la seguridad vial mediante la identificación en tiempo real de vías y vehículos. Incluso pueden ser utilizados para la entrega autónoma de paquetes, siguiendo rutas de carreteras de manera eficiente. Además, los drones equipados con estos algoritmos pueden emplearse para la vigilancia de accidentes y situaciones de emergencia en áreas de carreteras, proporcionando una respuesta rápida y precisa.

Capítulo 2

Objetivos

En esta sección se describirá el problema a resolver junto con los objetivos y requisitos pautados en el desarrollo del TFG

2.1. Descripción del problema

El objetivo principal de este TFG, es desarrollar un comportamiento de navegación autónoma basado en seguimiento de carriles utilizando aprendizaje por refuerzo e inteligencia artificial, en el que el dron sea capaz de navegar de una manera robusta por escenarios urbanos. El enfoque de este trabajo de investigación se centra en la creación de una solución eficiente y completa ante la problemática que puede llegar a tener la navegación autónoma, se muestra un comportamiento capaz de realizar el seguimiento de un carril para demostrar la complejidad de mantener una trayectoria estable y precisa utilizando un dron.

A continuación, se definen los siguientes subobjetivos:

1. Estudio del arte de los drones con el objetivo de definir las posibilidades que puede ofrecer y definir la navegación autónoma dentro del entorno en el que se desarrollará el TFG.
2. Análisis y uso de un sistema perceptivo basado en redes neuronales en la navegación autónoma de drones.
3. Desarrollo de un sistema de control para drones utilizando técnicas de aprendizaje por refuerzo para lograr una navegación autónoma y eficaz.
4. Análisis y desarrollo de una aplicación de navegación autónoma de drones basándonos en el seguimiento de un carril.
5. Análisis y comparativas de los diferentes comportamientos desarrollados en este trabajo con el fin de lograr resultados interesantes acerca de la utilización de redes neuronales y aprendizaje por refuerzo en la navegación autónoma de drones.

2.2. Requisitos

Los requisitos que han de cumplirse en este trabajo son:

1. Uso del vehículo UAV en el entorno de simulación fotorrealista Airsim junto a UnRealEngine.
2. Utilización del middleware robótico ROS para así garantizar la interoperabilidad del trabajo en otro tipo de escenarios permitiendo la reutilización del proyecto en diversas aplicaciones.
3. Comportamiento robusto y en tiempo real para garantizar la navegación del dron dentro del escenario urbano.
4. Los sistemas desarrollados deben ser reactivos para poder reaccionar a su entorno de manera concisa y eficiente durante la navegación en tiempo real.
5. Uso del algoritmo de Q-learning para la navegación autónoma del dron.

2.3. Metodología

Este trabajo, comenzó oficialmente en Septiembre del 2023 aunque en Diciembre del 2022 se plantearon varias ideas a desarrollar, y se finalizó en Junio del 2024.

La metodología que se llevo a cabo fue:

1. Reuniones semanales mediante Teams¹ con una duración de media o una hora, con el fin de tener un control semanal y pactar los objetivos semanales a seguir. Gracias a estas reuniones, se tenia una organización global del proyecto.
2. Contacto vía email de la universidad con el fin de solventar problemas urgentes.
3. Utilización de la metodología Kanban²: Este tipo de metodología consiste en crear un flujo de trabajo en equipos mediante la gestión visual. Se compone de varias fases:

- **Fase 1: Inicio del proyecto.** Consiste en marcar los objetivos que se deben

¹<https://www.microsoft.com/es-es/microsoft-teams/group-chat-software>

²<https://canalinnova.com/que-es-y-para-que-sirve-el-kanban-definicion-fases-y-pasos/>

seguir dentro del proyecto junto la asignación e identificación de tareas. En esta fase, definimos el tipo de aplicación que se iba a seguir que es la navegación autónoma de drones considerando un entorno de simulación de carreteras, la infraestructura, los tipos de algoritmos a seguir y las analíticas de los comportamientos desarrollados durante el TFG.

- **Fase 2: Diseño del tablero Kanban.** Se crea un tablero en donde se visualizarán las tareas y su estado. Cada tarea se definen en las columnas del tablero: las tareas que se deben realizar junto con indicadores visuales como se muestra en la figura
- **Fase 3: Implementación del tablero Kanban.** Se asignan las tareas a realizar y se establecen límites de trabajo en progreso como el tiempo de desarrollo para completar dicha tarea.
- **Fase 4: Seguimiento y mejora continua.** Se realiza un flujo continuo y constante del trabajo identificando nuevas soluciones para mejorar la eficacia y la productividad hasta de poder cambiar estrategias existentes por nuevas alternativas previamente definidas. En este trabajo hemos realizado varias tareas y seguimientos para encontrar la mejor solución ante el problema a resolver.

A lo largo del desarrollo de este TFG, se marcaron estas 4 fases permitiendo cambios y mejoras continuas hasta llegar al resultado final.

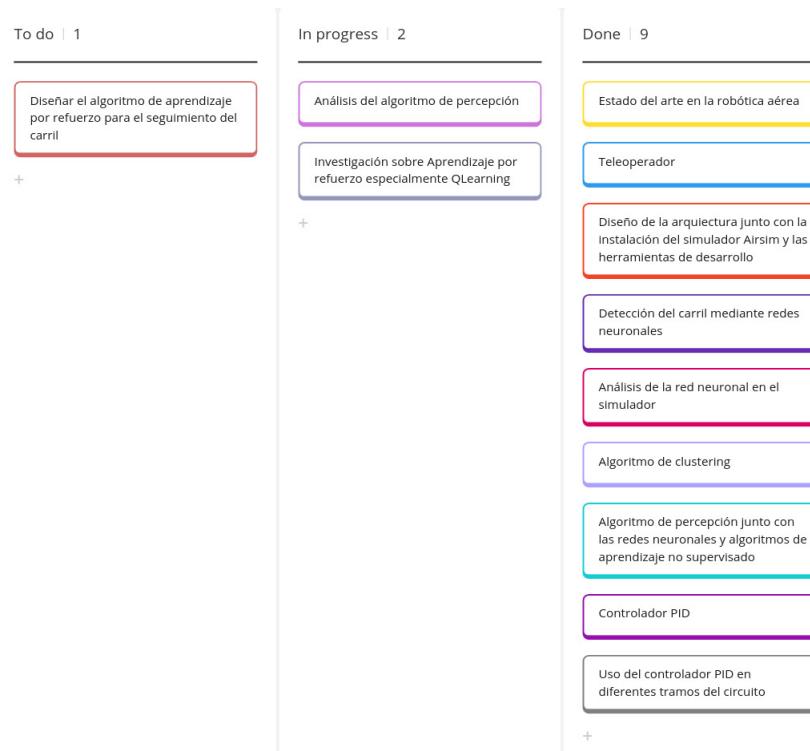


Figura 2.1: Ilustración del tablero Kanban durante el desarrollo del TFG

4. Tener un control de versiones mediante la plataforma GitHub³, con el objetivo de tener un almacenamiento de código y respaldos de ello.
5. El uso de un blog⁴, en el cual se describió brevemente los pasos que se siguieron para el desarrollo del TFG.

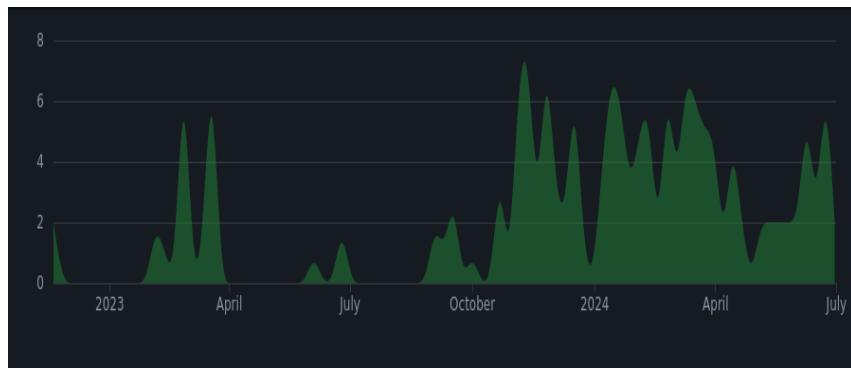


Figura 2.2: Seguimiento de trabajo en GitHub

2.4. Plan de trabajo

Finalmente, los pasos a seguir de este trabajo han sido:

1. Comienzo del trabajo.
 - Búsqueda del problema a desarrollar y análisis del estado del arte del uso de los drones en aplicaciones robóticas.
 - Instalación de las diferentes librerías y aplicaciones de software.
 - Preparación de configuración de toda la infraestructura, teniendo un análisis y estudio de comunicaciones para poder comenzar con el desarrollo.
2. Desarrollo: Una vez se tuvo listo toda la infraestructura tanto de comunicaciones como de librerías de software, se dio a pie el comienzo del desarrollo del código
 - En primer lugar, se desarrolló un teleoperador sencillo del drone para ver el funcionamiento del vehículo y dicho comportamiento.
 - Una vez finalizada la tarea del teleoperador, se comenzó con los algoritmos de percepción de detección de carril mediante redes neuronales.
 - Análisis y comparación de los resultados de los diferentes modelos que ofrece

³<https://github.com/RoboticsLabURJC/2022-tfg-barbara-villalba>

⁴<https://roboticslaburjc.github.io/2022-tfg-barbara-villalba/>

la red neuronal escogida con el propósito de tener la mejor solución.

- El siguiente paso fue estudiar la posibilidad de tener un algoritmo de aprendizaje no supervisado llamado clustering para clasificar las diferentes líneas que aparezcan en el escenario de la carretera.
 - Desarrollo del algoritmo de percepción junto con los dos puntos anteriormente mencionados.
 - Con el fin del algoritmo de percepción, se comenzó el desarrollo de un controlador sencillo PID para ver el funcionamiento de la percepción y de la navegación en el vehículo.
 - A continuación, se desarrolló el algoritmo de aprendizaje por refuerzo para el seguimiento del carril.
3. Evaluación: Se realizó la comparativa de los resultados obtenidos en el aprendizaje por refuerzo.
4. Redacción de la memoria del trabajo para la documentación de todo el proceso de investigación realizado.

Capítulo 3

Plataforma de desarrollo

En este capítulo se detalla sobre qué tecnologías se ha utilizado durante el desarrollo de este trabajo junto con el lenguaje de programación y las librerías utilizadas.

3.1. Lenguaje de programación

3.1.1. Python

Para el desarrollo de este TFG, se ha utilizado como lenguaje de programación Python. Python¹ es un lenguaje de programación interpretado, de tipado dinámico y orientado a objetos, utilizado para el desarrollo de software, aplicaciones web, data science y machine learning (ML). Fue creado por Guido van Rossum² en 1989, el nombre de "Python" se inspiró en el programa de televisión británico "Monty Python's Flying Circus".

Este lenguaje con los tiempos se ha convertido en uno de los lenguajes más populares del mundo, esto se puede deber a su sintaxis sencilla, clara y legible, aparte de estos beneficios también presenta una amplia gama de bibliotecas y marcos de trabajo. Además, se trata de un lenguaje de programación 'open source' (código abierto) y está disponible bajo una licencia de código abierto aprobada por la Iniciativa de Código Abierto (OSI), esto significa que Python es libre de usar, distribuir y modificar, incluso para uso comercial.

En el caso de este TFG, se utiliza Python3 para todo el desarrollo del código junto con el middleware robótico ROS (véase la sección 3.3) y para el desarrollo de los diferentes algoritmos.

Para el desarrollo del sistema de percepción, se ha utilizado varias librerías mediante el lenguaje de programación Python para poder percibir el entorno en el que se va a estar trabajando.

¹<https://www.python.org/>

²<https://gvanrossum.github.io/>

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)
print(factorial(5)) # Output: 120
```

Código 3.1: Ejemplo de código en Python de una función para calcular el factorial de un número

OpenCV

OpenCV (Open Source Computer Vision Library) es una biblioteca de open source dedicada para el tratamiento de imágenes y aprendizaje automático. Fue desarrollada por Intel y lanzada en el año 2000 y esta disponible para todos los públicos tanto para uso comercial como para uso personal³. Ofrece diferentes tareas como procesamiento de imágenes, detención de objetos, extracción de características, reconocimiento facial, estimación de movimiento entre otras. Además de ser compatible para múltiples sistemas operativos como Windows, Linux, MacOS, Android e iOS, lo que hace que sea una librería muy versátil para el desarrollo de aplicaciones en diferentes dispositivos y entornos.

En este caso, se utiliza esta biblioteca para poder obtener la imagen mediante la cámara que va abordo del dron.

```
import cv2

imagen = cv2.imread('ruta/a/tu/imagen.jpg')
cv2.imshow('Imagen', imagen)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

Código 3.2: Ejemplo de código en Python de operaciones básicas utilizando la libreria OpenCv

Numpy

Numpy⁴ (Numerical Python) es una librería dedicada para el cálculo científico en Python como arrays multidimensionales y matrices, junto con una amplia colección de

³<https://opencv.org/>

⁴<https://numpy.org/>

funciones matemáticas. Esta biblioteca es bastante utilizada en la comunidad debido a su eficiencia y a su de uso.

A continuación, se muestra un simple ejemplo de como podemos crear un array en NumPy e realizar operaciones básicas como la suma y calcular la matriz transpuesta

```
import numpy as np

array1d = np.array([1, 2, 3, 4, 5])

array2d = np.array([[1, 2, 3], [4, 5, 6]])

suma = np.sum(array1d)
transpuesta = np.transpose(array2d)
```

Código 3.3: Ejemplo de código en Python de operaciones básicas utilizando la librería Numpy

Se utiliza esta librería para el desarrollo del sistema de percepción que más adelante se explicará con más detalle.

Pytorch y ONNX Runtime

Pytorch⁵ es una biblioteca para el aprendizaje automático desarrollada en Python que permite a los desarrolladores crear y entrenar modelos de aprendizaje profundo. Proporciona una integración con otras bibliotecas de Python, como puede ser Numpy, lo que facilita la manipulación y el análisis de datos, además de tener soporte de GPU y CPU. Esta disponible para varios sistemas operativos como Windows, Linux, MacOs y Cloud Platforms (Amazon Web Services, Google Cloud Platform, Microsoft Azure).

ONNX⁶(Open Neural Network Exchange) es un formato abierto para representar modelos de aprendizaje profundo. Este tipo de formato permite la interoperabilidad entre diferentes herramientas de aprendizaje automático como Pytorch, Keras, Scikit-Learn y más. Por ejemplo, si un modelo ha sido creado desde Pytorch, ONNX permite convertirlo a un formato compatible sin necesidad de reescribir el código fuente desde cero. Dentro de Onnx existe un motor de inferencia de alto rendimiento denominado ONNX Runtime utilizado para ejecutar los modelos con formato ONNX, es compatible para diferentes sistemas operativos como Linux, Windows y Mac, además de permitir su uso en diversos entornos de programación como puede ser C, Python, C++, Java y más. Ofrece múltiples

⁵<https://pytorch.org/>

⁶<https://onnxruntime.ai/>

bibliotecas de aceleración hardware a través de Execution Providers (EP) para ejecutar de manera óptima los modelos de Onnx, cada proveedor de ejecución esta diseñado para aprovechar las capacidades de cómputo de una plataforma en particular, como CPU, GPU, FGPA o NPUs (Unidades de procesamiento neuronal). Algunos de los proveedores de ejecución compatibles con ONNX Runtime incluyen Nvidia CUDA, Nvidia TensorRT, Intel OpenVINO y más⁷.

En resumen, ONNX Runtime esta diseñado para la inferencia eficiente y ofrece una solución portátil y optimizada para ejecutar modelos ONNX en una variedad de entornos, en cambio Pytorch es muy utilizado en la comunidad para el desarrollo y entrenamiento de modelos.

En este TFG, se hace uso de ámbitos librerías/runtimes para las ejecuciones de los modelos y entrenamiento con RL.

Scikit-learn

Scikit-learn⁸ es una librería de aprendizaje automático open source que proporciona herramientas simples y eficientes para el análisis y modelado de datos. Esta librería soporta diferentes sistemas operativos como Windows, Linux y MacOs, esta construida específicamente para Python ya que se basa en bibliotecas científicas como Numpy, SciPy y Matplotlib. Se puede realizar múltiples algoritmos de aprendizaje automático como clasificaciones, regresiones, agrupamientos, reducciones de dimensionalidad y muchos más. Además de tener una gran comunidad debido a su facilidad de uso, documentación completa y su enfoque en la eficiencia y la simplicidad. En este trabajo se utiliza la librería para el uso de algoritmos de aprendizaje no supervisado específicamente clustering.

3.2. YOLOP

YOLOP [8] (You Only Look Once for Panoptic Driving Perception) es una red de percepción de conducción panóptica que ofrece múltiples tareas que puede manejar simultáneamente tres tareas cruciales en la conducción autónoma:

1. **Detección de objetos de tráfico:** Identifica los objetos presentes en la carretera, como otros vehículos, peatones, señales de tráfico, etc.

⁷<https://onnxruntime.ai/docs/execution-providers/>

⁸<https://scikit-learn.org/stable/>

2. **Segmentación del área transitable:** Determina las áreas de la carretera por las que un vehículo puede conducir de manera segura.
3. **Detección de carriles:** Identifica los carriles de la carretera.

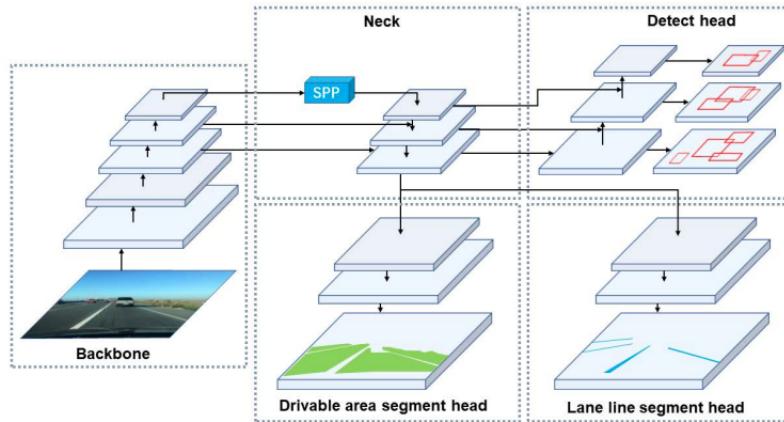


Figura 3.1: Arquitectura de YOLOP

YOLOP está compuesto por un codificador para la extracción de características y tres decodificadores para manejar las tareas específicas. Este modelo ha demostrado un rendimiento bueno en el desafiante conjunto de datos BDD100K[21], logrando el estado del arte en las tres tareas en términos de precisión y velocidad.

En este TFG, se utiliza los decodificadores de segmentación del área transitable y detección de carriles. El decodificador de segmentación del área transitable utiliza los mapas de características extraídos por el codificador para realizar una predicción semántica a nivel de píxeles. Esto significa que para cada píxel de la imagen, el decodificador de segmentación del área transitable predice si el píxel pertenece a un área transitable o no.

Ambos decodificadores, deben de recibir una entrada ($W/8, H/8, 256$):

1. **$W/8$ y $H/8$:** Representa la anchura y la altura de la imagen de entrada, respectivamente, divididas por 8.
2. **256:** es el número de canales en el tensor de entrada. En el contexto de las redes neuronales convolucionales, un canal puede ser una característica aprendida (como bordes, texturas, colores, etc.) o una capa de color en una imagen (como rojo, verde, azul en imágenes RGB)

Devuelven una salida de tipo $(W, H, 2)$, siendo W la anchura y H la altura de la imagen, solo hay dos canales en cada mapa de características, ya que cada píxel representa

si pertenece a una clase de objeto o al fondo.

Modelo de YOLOP

YOLOP utiliza una arquitectura de red neuronal CNN. Es un tipo de red neuronal artificial diseñada para procesar datos con una estructura de cuadrícula, como una imagen. Esta red presenta los diferentes modelos:

1. **End-to-end.pth**: Este archivo se ha construido a partir de la biblioteca Pytorch.
2. **Yolop-320-320.onnx**, **Yolop-640-640.onnx** y **Yolop-1020-1020.onnx**: Estos tres archivos se han construido usando ONNX Runtime.

Dependiendo de que modelo se quiera utilizar, se obtiene resultados diferentes en el ámbito de cómputo y rapidez a la hora de realizar la inferencia del modelo. En el código 3.4 se puede encontrar un ejemplo del uso de la red en la página Pytorch dedicada a la red neuronal YOLOP⁹ y en la figura 3.2 se muestra un ejemplo del uso de YOLOP utilizado en la página DebuggerCafe¹⁰

```
import torch

# load model
model = torch.hub.load('hustvl/yolop', 'yolop', pretrained=True)

#inference
img = torch.randn(1,3,640,640)
det_out, da_seg_out, ll_seg_out = model(img)
```

Código 3.4: Ejemplo básico de cómo poder utilizar YOLOP

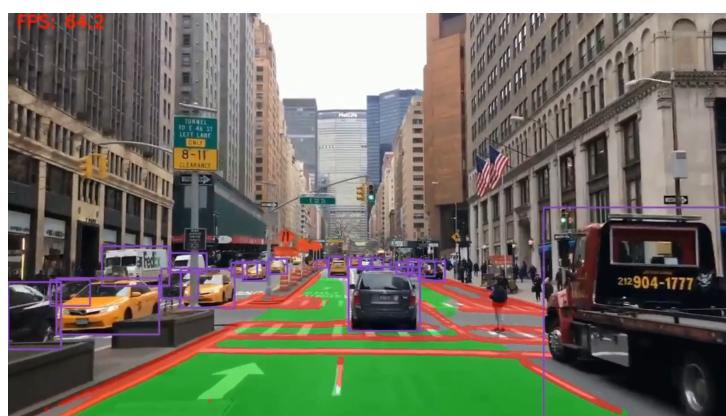


Figura 3.2: Resultado de la salida de la red neuronal YOLOP

⁹https://pytorch.org/hub/hustvl_yolop/

¹⁰<https://debuggercafe.com/yolop-for-object-detection-and-segmentation/>

Por lo que, se utiliza esta red neuronal para poder detectar los carriles que puede tener las áreas transitables en Airsim.

3.3. ROS

ROS (Robot Operating System)¹¹ es un conjunto de librerías de código abierto utilizadas principalmente para aplicaciones robóticas. Se puede definir este middleware como se muestra en la figura 3.3:



Figura 3.3: Definición de ROS

1. **Plumbing:** ROS proporciona una infraestructura de mensajería publicador-subscriptor diseñada para facilitar la construcción sencilla y rápida de sistemas informáticos distribuidos.
2. **Tools:** ROS proporciona introspección, lanzamiento, depuración, visualización, trazado, registro, reproducción y detener sistemas informáticos distribuidos.
3. **Capabilities:** ROS proporciona una amplia colección de bibliotecas que implementan funciones útiles para los robots, por ejemplo, movilidad, manipulación y percepción.
4. **Community:** ROS cuenta con el apoyo y la mejora de una gran comunidad, con un fuerte enfoque en la integración y la documentación, gracias a ello, es una ventaja poder aprender a cerca de los miles de paquetes que ofrece ROS que están disponibles de desarrolladores de todo el mundo.

Este middleware sigue un modelo parcialmente centralizado de publicación y suscripción, el cual el publicador genera mensajes y eventos asociados a un topic y el subscriptor es quien se subscribe al topic correspondiente y recibe la información que ha generado el publicador.

Este tipo sistema es bastante útil ya que permite a los desarrolladores cambiar, añadir o eliminar nodos (programas en ejecución) sin afectar al resto del sistema,

¹¹<https://www.ros.org/>

facilitando de forma asíncrona el desarrollo iterativo, permitiendo construir sistemas robóticos complejos, escalables y robustos mejorando la eficiencia y permitiendo el desarrollo y mantenimiento de aplicaciones robóticas.

Se utiliza ROS junto con la distribución Noetic¹² en el desarrollo del TFG para realizar la conexión con el simulador Airsim y el desarrollo del sistema de percepción del seguimiento del carril a través del controlador PID y aprendizaje por refuerzo.

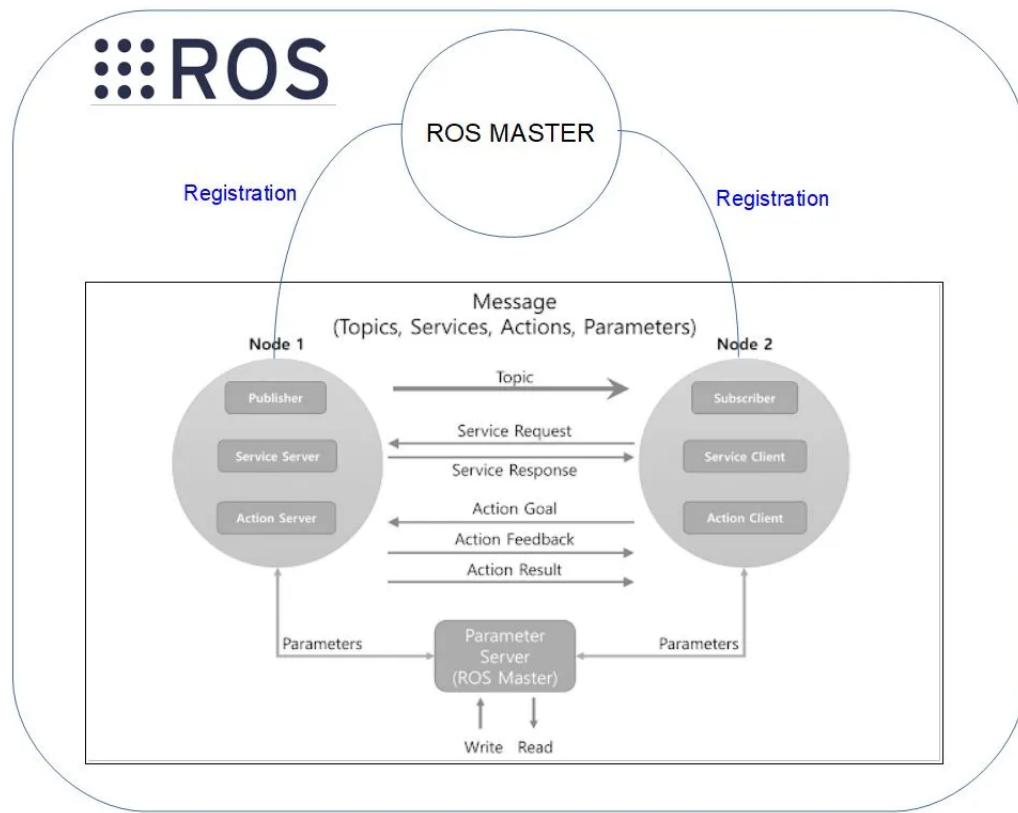


Figura 3.4: Arquitectura de ROS

3.3.1. Mavros

Mavros¹³ es un paquete formado por **ROS** y el protocolo de comunicaciones ligero **MAVLink** (Micro Air Vehicle Link) diseñado por Lorenz Meier¹⁴ bajo el LGPL licencia. Este protocolo es utilizado para enviar información de estado, para controlar el vehículo y recibir datos de telemetría. Fácil de implementar en sistemas con recursos limitados, lo que lo hace ideal para su uso en drones y otros vehículos aéreos no tripulados.

Ademas, **Mavros** traduce los mensajes **ROS** a mensajes **MAVLink** y viceversa

¹²<https://wiki.ros.org/noetic>

¹³<http://wiki.ros.org/mavros>

¹⁴<https://www.technologyreview.es/listas/35-innovadores-con-menos-de-35/2017/inventores/lorenz-meier>

por lo que permite que los datos y comandos fluyan entre **ROS** y el drone, permitiendo un control más sofisticado y una mayor funcionalidad.

Por lo que, en este TFG se estudia y se analiza si **Mavros** se podría utilizar para el control del dron junto con **PX4 AutoPilot** (3.5) a través de Airsim.

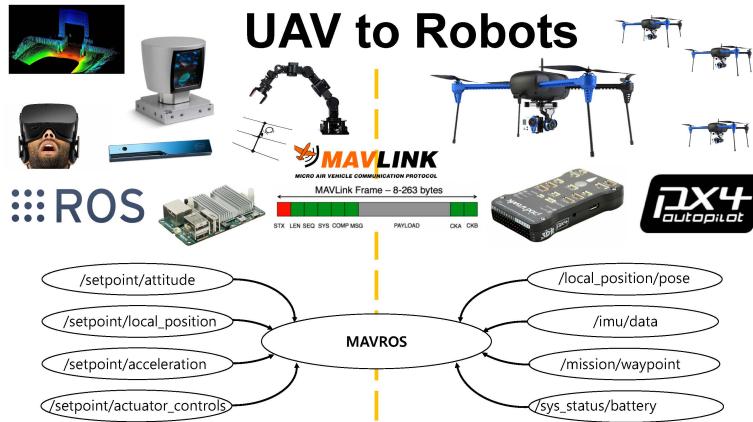


Figura 3.5: Infraestructura de Mavros

3.4. Airsim

El entorno de simulación en el que vamos a estar trabajando será **Airsim** junto con **UnRealEngine** de Epic Games¹⁵. **Airsim**¹⁶ es un simulador de código abierto que se utiliza en aplicaciones robóticas y aprendizaje automático. Se construye sobre entornos 3D creados con **UnRealEngine**, estos entornos son utilizados para simular el mundo real y probar cómo los vehículos autónomos se comportarían en diferentes situaciones. **Airsim** es compatible en varias plataformas como Linux, Windows, macOS y también para Docker y WSL. En nuestro caso, se utilizará en Linux junto con **ROS**.

Por otro lado **UnRealEngine** es un motor de videojuegos que se utiliza para la creación y simulación de entornos 3D realistas para videojuegos, películas animadas, experiencias interactivas y de realidad virtual. Es una propuesta innovadora utilizar este tipo de herramientas ya que puedes simular comportamientos físicos que se puedan producir en un entorno real.

Como hemos comentado anteriormente, **Airsim** es una buena opción de uso si queremos tener comportamientos similares a un entorno real. Ofrece una variedad de escenarios, tipos de vehículos, sensores y configuraciones del entorno según las necesidades u objetivos marcados de cada persona. Para ello se debe todo configurar en un fichero

¹⁵<https://www.unrealengine.com/es-ES>

¹⁶<https://microsoft.github.io/AirSim/>

de configuración con extensión json denominado settings.json ,lo cual para configurar el vehículo con nuestras necesidades necesitaremos definir diferentes variables.

Un archivo settings.json es un archivo de configuración específica de Airsim que define cómo se ejecutará la simulación en términos de propiedades del vehículo, configuración de sensores, condiciones climatológicas y más.

Un archivo settings.json consta de varias secciones:

1. **SimMode:** Este parámetro define el modo de simulación, se refiere si el modo de simulación es para coches, multirotores o visión de computador.
2. **ClockType:** Determina qué tipo de reloj se utiliza para medir el tiempo en la simulación.
3. **Vehicles:**Configuración de las propiedades de cada vehículo individualmente. Puedes especificar el tipo de vehículo, la posición inicial, la dinámica del vehículo, entre otros.
 - **VehicleType:** En este caso ese parámetro es el tipo de vehículo que utilizaremos en la simulación.
 - **UseSerial:** Es para saber si vamos a usar un puerto serial en físico si utilizamos un vehículo en un entorno real.
 - **ControlIp:** Esta opción es para especificar si el comportamiento se realizará simulado.
 - **ControlPortLocal:** Se especificará el puerto Local.
 - **ControlPortRemote:** Se especificará el puerto Remoto.
 - **LocalHostIp:** La dirección IP del ordenador en donde llevaremos la simulación.
 - **Sensors:** Permite personalizar la configuración de los sensores simulados, como Lidar, IMU (Unidad de Medición Inercial),GPS y sensor de distancia.
 - **Cameras:** Puedes configurar las cámaras utilizadas en la simulación, especificando sus propiedades como resolución, tipo de lente, posición y orientación relativas al vehículo.

Se usa una cámara que proporciona una imagen RGB de dimensiones 620x620 píxeles y con el flag de PublishToRos a 1 para poder acceder a ella mediante el Airsim ROS Wrapper.

Para más detalles sobre el archivo de settings.json está la página oficial de Airsim¹⁷

Escenarios

Los escenarios que ofrece Airsim depende en que sistema operativo nos encontremos, en nuestro caso al utilizar el escenario en Windows tenemos más variedad que en comparación con Linux.

Tiene escenarios desde carreteras y ciudades con coches simulados hasta entornos industriales como almacenes y entornos de montaña con carreteras como se muestra en la figura 3.6. En este caso se ha utilizado el escenario Coastline, consiste en un entorno fotorrealista de un recorrido amplio de 2 carriles con ambiente tropical. Con este entorno se obtiene la información del entorno para realizar el comportamiento sigue carril con el dron.

Todos estos escenarios se pueden encontrar en las releases de Airsim¹⁸ tanto para Windows como para Linux. En este caso, se ha utilizado el escenario Coastline, ya que se quiere desarrollar un comportamiento de seguimiento de carril y este escenario ofrece un amplio recorrido de 2 carriles para poder llevarlo a cabo.

Sensores

Ofrece sensores como cámaras, barómetros, Imus, GPS, Magnetómetros, sensores de distancia y Lidar. En este caso, se utiliza como sensores una cámara para poder realizar la detención del carril que queremos seguir, el sensor Lidar para saber a que altura se encuentra el dron respecto al suelo y el sensor GPS para poder obtener la localización del dron.

¹⁷<https://microsoft.github.io/AirSim/settings/>

¹⁸<https://github.com/Microsoft/AirSim/releases>

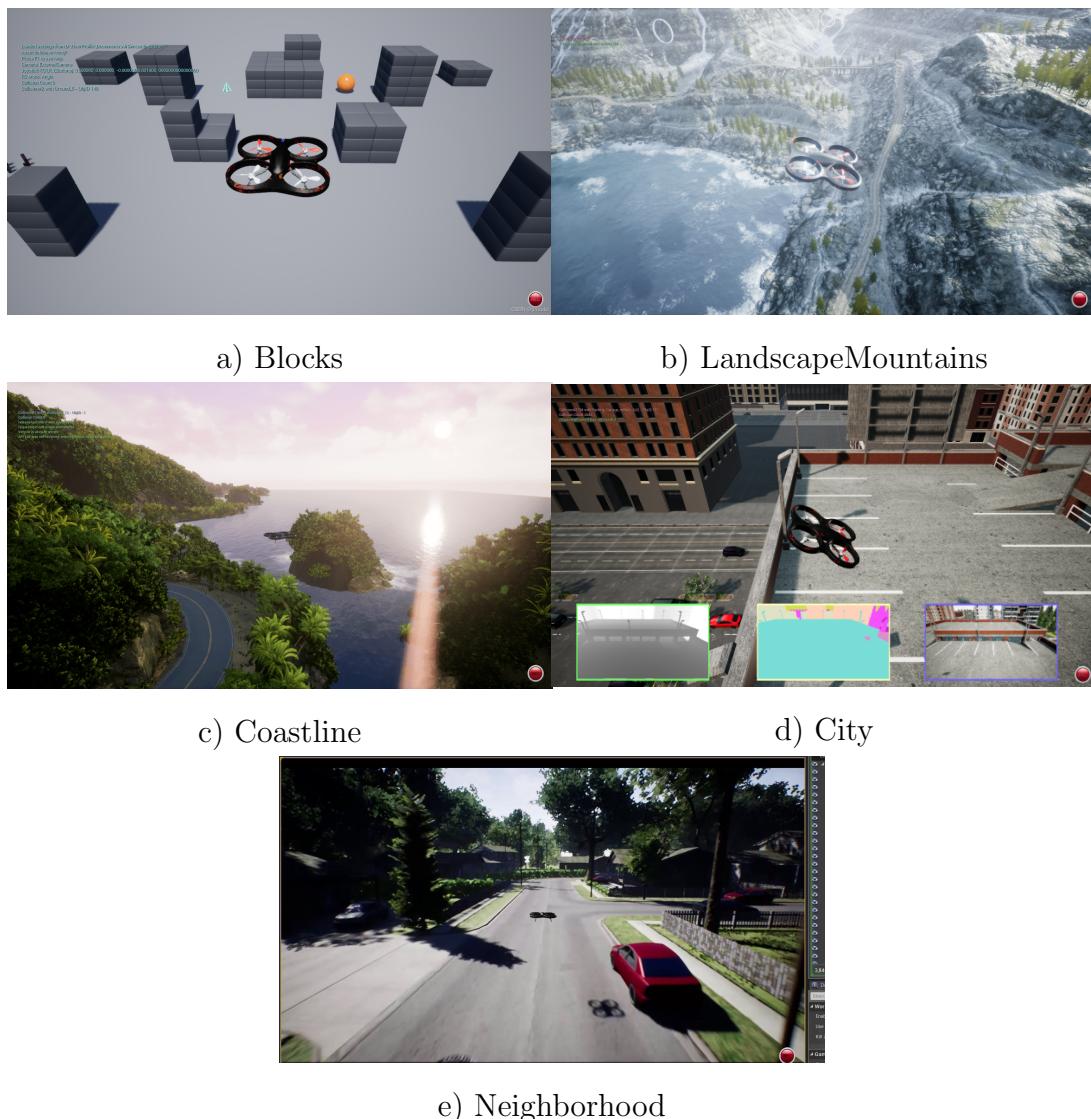


Figura 3.6: Ejemplos de escenarios en Airsim

Tipos de vehículos

Airsim ofrece dos tipos principales de vehículos para la simulación: coches y drones. Dentro de estos tipos se encuentran los subtipos de coches y drones que se puede utilizar.

1. Coche

- **PhysXCar**: Representa un vehículo en tierra con física realista basado en el motor de física PhysX.
 - **ArduRover**: Se utiliza para vehículos terrestres que sigan el estándar ArduRover. ArduRover¹⁹ se trata de un piloto automático de código abierto utilizado específicamente para vehículos terrestres.

¹⁹<https://ardupilot.org/rover/>

2. Dron

- **SimpleFlight:** Representa un dron con un modelo de vuelo simplificado. Este tipo de opción puede ser útil si queremos simular comportamientos de movimiento básico para los drones.
- **PX4Multirotor:** Representa un dron mediante PX4 ArduPilot.
- **ArduCopter:** Representa un dron pero siguiendo el estándar ArduCopter. ArduCopter²⁰ se trata de un piloto automático de código abierto utilizado para los drones

Como se ha enumerado anteriormente, este entorno de simulación tiene un catálogo de vehículos, sensores y cambios climatológicos dentro del entorno. En este TFG, se utiliza un dron de tipo "SimpleFlight" como vehículo.

3.4.1. Airsim ROS Wrapper

Se utiliza el paquete **Airsim ROS Wrapper**²¹ para poder acceder a ciertos sensores que serán necesarios como es la cámara, el Lidar y el GPS, pero antes de comentar lo que ofrece se hablará sobre qué es un ROS Wrapper.

ROS Wrapper es un componente que facilita la integración entre dos sistemas o entornos diferentes. Si se lleva al contexto de **ROS**, un wrapper es un nodo o paquete que permite que los componentes de **ROS** se comuniquen con otros sistemas o bibliotecas que no fueron originalmente diseñadas para trabajar con **ROS**. Este paquete puede proporcionar publicación de datos desde el sistema externo a **ROS** a través de topics, suscripción a topics de **ROS** para recibir comando o datos, adaptación de interfaces de llamada (por ejemplo, entre C++ y Python).

Por lo tanto, **Airsim ROS Wrapper** es un paquete de **ROS** que comunicará **ROS** y **Airsim**. Este paquete contiene dos nodos principales que han sido realizados mediante la comunidad de Airsim²²:

1. **AirSim ROS Wrapper Node:** Este nodo proporciona una interfaz **ROS** para acceder a los datos del vehículo simulado, por ejemplo, sus sensores, proporcionar velocidades, acceder a su sistema de referencia, etc.

²⁰<https://ardupilot.org/copter/>

²¹https://microsoft.github.io/AirSim/airsim_ros_pkgs/

²²<https://github.com/microsoft/AirSim>

2. **Simple PID Position Controller Node:** Este nodo es un controlador de posición simple basado en un controlador PID (proporcional-derivativo-integral). Ayuda controlar la posición del vehículo simulado en el entorno **Airsim**.

3.4.2. Client Airsim

Airsim ofrece una API implementada para Python denominada Client Airsim²³, en donde podemos conectarnos con el simulador y tener el control de los sensores y actuadores del vehículo. Esta interfaz es bastante útil ya que podemos tener el control del vehículo simulado sin necesidad de tener que utilizar los topics de ROS, es decir, existen métodos los cuales podemos comandar velocidades al vehículo, tener acceso a los sensores como las cámaras o cambiar configuraciones de la simulación como por ejemplo el clima, el viento, el tiempo del día o la densidad de tráfico en escenarios donde aparecen coches simulados.

Esta API se utiliza para el control del dron para realizar su navegación con los controladores PID y en el desarrollo de aprendizaje por refuerzo.

3.5. PX4 AutoPilot

PX4²⁴ es una plataforma de software de código abierto para desarrolladores de drones que les permite crear y controlar diversos tipos de drones, desde aplicaciones de consumo hasta aplicaciones industriales. Una de las principales características que ofrece esta plataforma es el soporte de múltiples tipos de vehículos, como aviones de ala fija, multirrotores, helicópteros, rovers y vehículos submarinos, también proporciona diferentes modos de vuelo, navegación por puntos de referencias predefinidos, estabilización del vehículo.

En este trabajo se realiza un análisis respecto a la navegación del dron mediante **PX4** con el modo de simulación Software in The Loop(SITL) para tener el control del dron junto con **Mavros** y **Airsim**.

²³<https://microsoft.github.io/AirSim/apis/>

²⁴<https://docs.px4.io/main/en/>

3.5.1. Software in The Loop(SITL)

Este modo de simulación permite a los desarrolladores probar y depurar códigos de control de drones sin necesidad de hardware físico, en lugar de ejecutar el código en un vehículo real, este modo simula el comportamiento del vehículo en una computadora. Es especialmente útil durante el desarrollo y la validación de control, navegación y planificación de misiones. Se puede tener diversos entornos de simulación con este modo como Gazebo, Airsim y jMAVSim. Dichos entornos de simulación permiten realizar simulaciones muy realistas y avanzadas de cualquier tipo de vehículo simulando una gran variedad de parámetros.

Para poder utilizar PX4 SITL, se debe configurar el entorno de desarrollo adecuado y seguir las instrucciones proporcionadas por la comunidad²⁵.

3.5.2. Modos de vuelo

PX4 ofrece varios modos de vuelo por ejemplo como Takeoff, Land, Hold, Position, Offboard, etc. Un modo de vuelo define como el usuario puede controlar el vehículo a través de comandos y ver que respuesta tiene.

1. **TAKEOFF:** Este modo de vuelo permite despegar el dron con una altitud y una velocidad de ascendente escogida por el usuario con los parametros de PX4 MIS_TAKEOFF_ALT y MPC_TKO_SPEED, dichos parámetros tienen valores por defecto definidos por la plataforma (2.5 m y 1.5 m/s respectivamente). Antes de realizar el despegue el dron será armado para poder realizarlo.

Una vez se realice el despegue del dron se pasa al modo HOLD

2. **LAND:** Permite aterrizar el dron en donde se encuentre en ese instante, una vez que el dron sea aterrizado, se desarmará por defecto. En este modo se puede cambiar por ejemplo la tasa de descenso durante se realiza el aterrizaje del dron con el parametro MPC_LAND_SPEED, tiempo en segundos para que se realice el desamardo del dron si se establece dicho tiempo con un valor de -1 el dron no se desarmará cuando aterrice.

Cuando de realice este modo de vuelo por defecto se cambiara al modo de POSITION

3. **HOLD:** A partir de este modo de vuelo podemos parar el dron manteniendolo en el

²⁵<https://docs.px4.io/v1.14/en/simulation/>

aire con su actual posición GPS y altitud. Este modo puede ser bastante útil para cuando se quiere pausar una misión o reiniciar el comportamiento que se quiera realizar.

4. **POSITION:** Es un modo de vuelo manual el cual puedes controlar el DRON mediante un joystick, dicho vuelo controla la posición del dron cuando comandemos velocidades mediante el joystick.

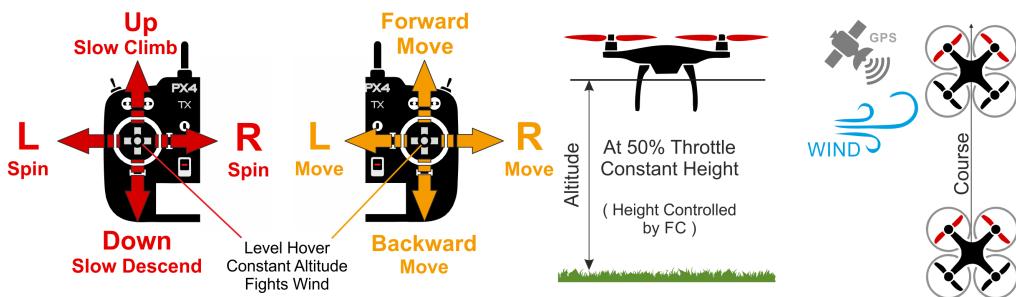


Figura 3.7: Diagrama del comportamiento del modo de vuelo Position

5. **OFFBOARD:** Con este modo de vuelo se puede controlar el movimiento y la altitud del vehículo a partir de comandos de posición, velocidad, aceleración, altitud, velocidades de altitud o puntos de ajuste de empuje/torque.

Dichos comandos deben ser una secuencia de mensajes de setpoint MavLink o a través de topics mediante ROS con Mavros.

En este modo, PX4 debe recibir una secuencia de mensajes continua. Si en algún momento dejamos de publicar mensajes, el control externo de PX4 dejará de estar en el modo OFFBOARD después de pasar un tiempo de espera establecido por el parámetro COM_OF_LOOS_T (por defecto está establecido a 1 s) e intentará aterrizar o realizar alguna acción de seguridad (dichas acciones de seguridad vienen definidas en la sección de Failsafes en PX4 Autopilot²⁶). La acción dependerá si el control RC está disponible, si este control está disponible pasará a otro tipo de modo de vuelo definido en el parámetro COM_OBL_RC_ACT.

Para comandar las velocidades al vehículo mediante Mavros, se tendrá que utilizar el topic denominado `/mavros/setpoint_velocity/cmd_vel_unstamped` dicho topic utiliza un marco de coordenadas por defecto definido en el archivo de configuración de px4.config.yaml LOCAL_NED. Si se quiere que el marco de coordenadas se mueva con el cuerpo del dron, se tiene que utilizar el marco de coordenadas BODY_NED

²⁶<https://docs.px4.io/v1.14/en/config/safety.html>

Los marcos de coordenadas que ofrece Mavros se puede ver a través del servicio SetMavFrame.srv²⁷

3.6. QGroundControl

QGroundControl²⁸ es una plataforma de software que proporciona un control completo de vuelo y configuraciones de vehículos para drones por **PX4 ArduPilot**. Además, ofrece un control total durante el vuelo y permite la planificación de vuelos autónomos mediante la definición de puntos de referencia, se muestra la posición del vehículo junto con su trayectoria, los puntos de referencia y los instrumentos del vehículo. Es una opción cómoda para poder visualizar tu vehículo y querer cambiar parámetros del vehículo mediante esta aplicación y poder teleoperar el vehículo a través de un mando joystick. Funciona en diferentes plataformas como Windows, macOS, Linux,iOS y dispositivos Android, en nuestro caso lo utilizaremos en Linux.

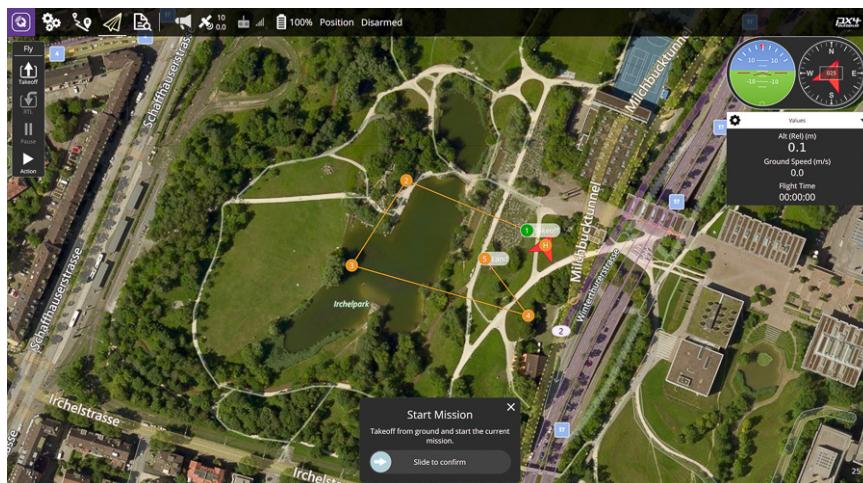


Figura 3.8: QGroundControl

²⁷https://github.com/mavlink/mavros/blob/master/mavros_msgs/srv/SetMavFrame.srv

²⁸<http://qgroundcontrol.com>

Capítulo 4

Diseño e Implementación

El principal objetivo de este TFG es desarrollar un sistema autónomo de navegación sigue-carril para un dron basado en aprendizaje por refuerzo, siendo este capaz de desenvolverse por carreteras urbanas con un comportamiento reactivo. En este capítulo se desarrolla como se ha alcanzando este objetivo.

En primer lugar se diseña la infraestructura de comunicaciones entre el entorno de simulación y las diversas plataformas de desarrollo. Esta infraestructura de comunicaciones, permite la transferencia de datos en tiempo real y la integración de diferentes módulos del sistema, garantizando una comunicación fluida y eficiente.

A continuación, se desarrolla el sistema perceptivo mediante el uso de inteligencia artificial junto con la implementación de comportamientos autónomos tradicionales empleados en diferentes robots, como el seguimiento de carriles. Entre los métodos utilizados se incluye el control clásico basado en controladores y métodos de control avanzados basados en aprendizaje por refuerzo, con el objetivo de que el dron aprenda y se adapte a distintos escenarios urbanos. Una vez desarrollados estos comportamientos, se procede a realizar una evaluación de los comportamientos obtenidos. Se recopilan diversas métricas para determinar la efectividad de cada enfoque y adaptabilidad del dron.

Finalmente, se analizan los resultados y se realiza una comparativa, destacando las ventajas y limitaciones de cada enfoque, proporcionando así una visión global del desempeño del sistema autónomo de navegación del dron en carreteras.

4.1. Arquitectura

La arquitectura propuesta para este trabajo se muestra en la figura 4.1. Consta de dos componentes principales comunicados entre sí por medio de un router y conexión Ethernet: el entorno de simulación y las plataformas de desarrollo. Estos componentes están separados en dos ordenadores distintos, siendo PC1 el entorno de simulación y PC2

las plataformas de desarrollo junto con los algoritmos de percepción y de sigue-carril. El objetivo de esta separación en dos ordenadores diferentes es para poder dividir la carga computacional, ya que una sola máquina es incapaz de ejecutar el simulador y los algoritmos desarrollados en este TFG dada la alta demanda computacional del simulador Airsim.

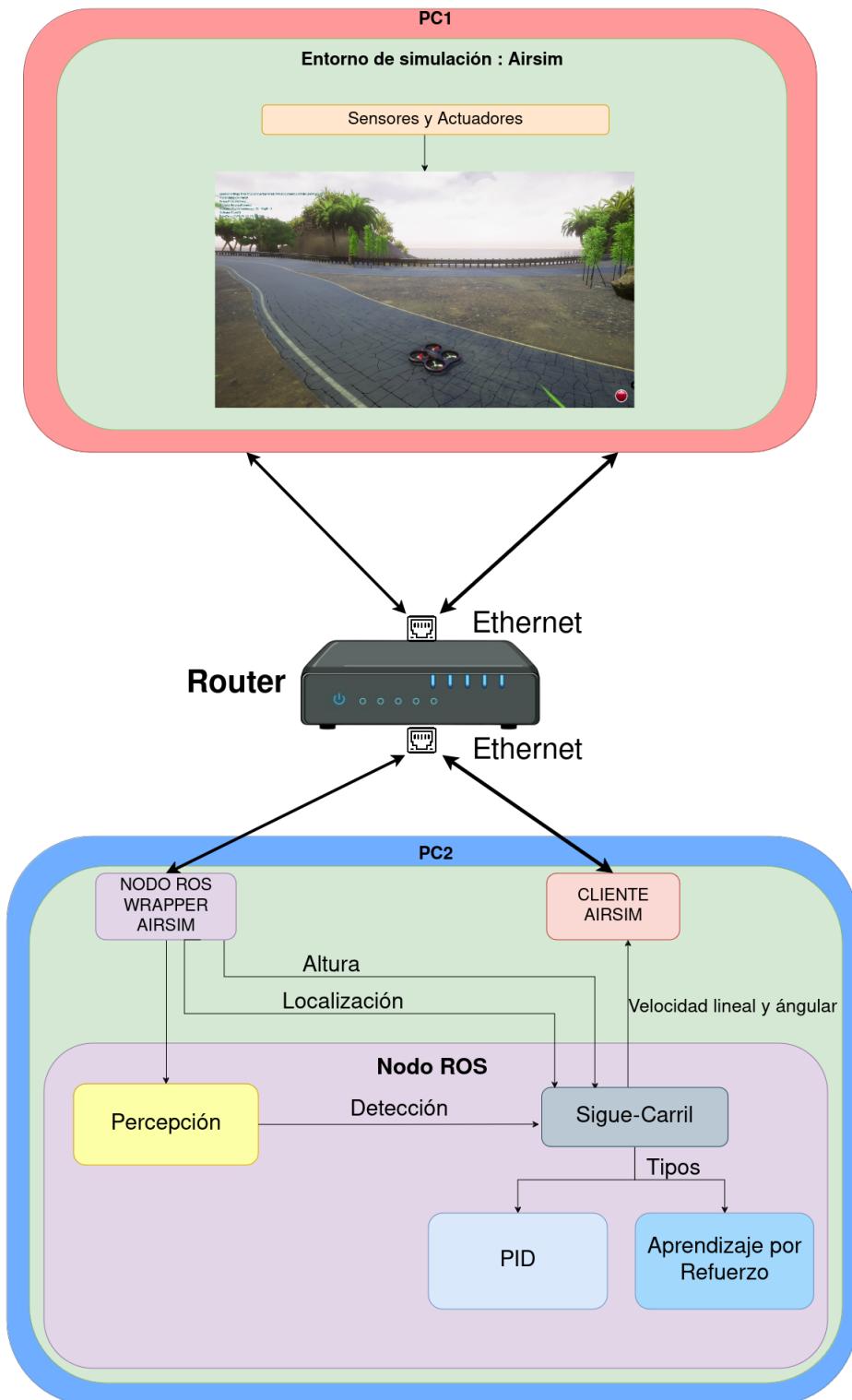


Figura 4.1: Arquitectura general del desarrollo en este TFG

Por un lado, el primer componente está compuesto por el entorno de simulación Airsim, ejecutado en el PC1. Incluye la configuración de sensores y actuadores necesarios para la navegación del dron. La comunicación entre este componente y el segundo componente (PC2), se realiza a través de dos interfaces principales:

1. **Nodo ROS Wrapper Airsim** : Este componente actúa como una interfaz que facilita la integración de Airsim con el middleware ROS. A través del ROS Wrapper Airsim, se recogen tres tipos de salidas:

- a) **Imágenes RGB**: Capturadas por las cámaras a bordo del dron, estas imágenes son fundamentales para el sistema perceptivo, permitiendo la detección y el seguimiento del carril.
- b) **Altura**: Obtenida a través del sensor Lidar, esta medida es esencial para conocer la altura a la que vuela el dron durante su navegación.
- c) **Localización**: Proporcionada por el GPS, esta información permite conocer la posición del dron en el entorno simulado.

2. **Cliente Airsim**: Este componente permite el control del dron, gestionando las velocidades lineales y angulares necesarias para su navegación.

Dentro de PC2, se encuentra el componente que encapsula el sistema perceptivo y el seguimiento de carril, actuando como el núcleo de procesamiento. Este componente recibe y procesa las entradas de los sensores (imágenes RGB, altura y localización), y genera velocidades lineales y angulares.

Toda la funcionalidad se implementa dentro de dos submódulos:

1. **Percepción**: Utilizando las imágenes RGB proporcionadas por ROS Wrapper Airsim, este sistema procesa la información visual para detectar el carril en el que debe navegar el dron. La detección del carril es fundamental para la ejecución del comportamiento sigue-carril, que guía al dron a lo largo del carril deseado.

2. **Sigue-Carril**: Este comportamiento se encarga de seguir el carril en función de la información obtenida de la percepción. Dentro de este componente, se utiliza dos enfoques diferentes:

- a) **PID**: Un enfoque clásico de control que ajusta las velocidad angulares en función de las desviaciones del carril detectado. En este controlador, la velocidad lineal del dron es constante.

b) **Aprendizaje por Refuerzo:** Un enfoque más avanzado que utiliza técnicas de aprendizaje automático para optimizar el comportamiento del dron. Este método permite al dron aprender de su entorno en diferentes condiciones, dotándole de la capacidad de generalizar el comportamiento aprendido, provocando que sea capaz de funcionar correctamente en entornos nunca vistos anteriormente.

Ambos enfoques de control generan comandos de velocidades lineales y angulares que son enviados al Client Airsim. Este, a su vez, traduce estos comandos en acciones físicas en el entorno simulado, permitiendo un movimiento controlado y preciso del dron.

4.2. Configuración del sistema cliente-servidor

En el desarrollo de este TFG, se ha decidido adoptar un enfoque cliente-servidor. El entorno de simulación, compuesto por Airsim y UnrealEngine, se ejecuta en un ordenador de sobremesa con un sistema operativo Windows 10, una GPU Nvidia RTX 2070 Super, Intel Core i9-9900KF y 32 GiB de memoria. Por otra parte, las plataformas de desarrollo y control, que incluyen ROS, AirSim ROS Wrapper Node y Client Airsim se ejecutan en un ordenador portátil con un sistema operativo Ubuntu 20.04, una GPU Nvidia RTX 2070, un procesador Intel Core i7-10875H y 32 GiB de memoria. Esta propuesta se tomó con la premisa de no encapsular en un único ordenador el entorno de simulación y las plataformas de desarrollo. Inicialmente, todo el sistema seguía una configuración centralizada en un solo equipo con Ubuntu 20.04, generando cuellos de botella y limitando al rendimiento del sistema. Para solventar estos problemas, se configuró una infraestructura entre dos ordenadores distintos, dividiendo así la carga de trabajo entre los dos sistemas. Ambos ordenadores se comunican a través de un router teniendo cada uno una dirección IP distinta.

Con la configuración en un único ordenador, el simulador generaba la imagen de la cámara a una frecuencia de 6 fotogramas por segundo, la cual era muy baja para el comportamiento que se deseaba implementar. Sin embargo, en la configuración separada, el simulador es capaz de generar imágenes a una frecuencia de 30 fotogramas por segundo, valor máximo que el simulador es capaz de ofrecer debido a la configuración que tiene la cámara simulada.

Al principio del desarrollo, en esta distribución de equipos, se utilizó una

comunicación con PX4 y Mavros, siguiendo la documentación oficial¹ junto con el entorno de simulación Airsim. Sin embargo, se decidió cambiar por una nueva configuración que emplea ROS Wrapper Airsim Node y Client Airsim, ya que la comunicación original añadía una capa de procesamiento adicional, produciendo bajos rendimientos que afectaba a la sincronización con el simulador. También este cambio, se realizó debido a que el funcionamiento del simulador se degradaba rápidamente, provocando que el simulador sufriese de cierres inesperados, además de introducir latencias innecesarias. Si no hubiese sido por este cambio, no se hubiese podido desarrollar el entrenamiento por refuerzo que se explicará posteriormente, ya que todos los comportamientos que hubiese podido aprender, se perderían, ya que manejar esta situación en el programa de entrenamiento no era posible.

En resumen, el sistema de comunicaciones se compone de dos ordenadores interconectados en la misma red. El diagrama de comunicaciones ilustrado en la figura 4.2, muestra esta implementación cliente-servidor en diferentes ordenadores. Cada ordenador asume roles distintos: el primer ordenador ejecuta el entorno de simulación en cambio el segundo ordenador, gestiona las plataformas de desarrollo en donde se implementan los algoritmos de percepción y seguimiento de carril.

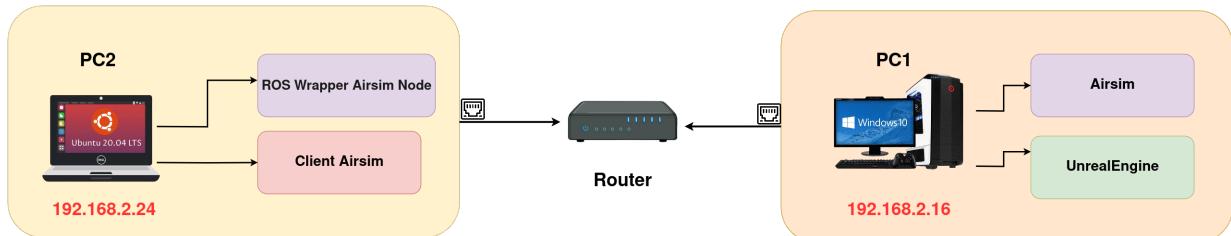


Figura 4.2: Diagrama de comunicaciones

4.2.1. Preparación del entorno de simulación

Como se menciona en la sección 3.4, se utiliza como simulador Airsim junto con el motor UnrealEngine. Para construir el entorno de simulación, primero se necesita instalar UnrealEngine. Para ello, se sigue las instrucciones marcadas por la página oficial de Epic Games², utilizando la versión 4.27.2.

Una vez que UnrealEngine esté instalado, se procede a configurar el entorno de simulación mediante el archivo settings.json. Por defecto, al ejecutar Airsim por primera vez, el simulador crea este archivo automáticamente dentro de una carpeta denominada

¹<https://docs.px4.io/main/en/simulation/>

²<https://www.unrealengine.com/en-US/download>

Airsim en la carpeta Documentos en Windows.

Configuración del dron y del entorno

En primer lugar se utiliza el mapa de simulación Coastline de entre los diferentes mapas que te ofrece Airsim para Windows. Al descargar esta carpeta, se obtiene un fichero ejecutable para abrir el entorno con Airsim, junto con carpetas de modelos de simulación como las carreteras, montañas y vegetación. Dentro de estas carpetas, se eliminan dichos componentes de simulación que pueden dificultar al sistema perceptivo, como plantas y señales de tráfico que se ubican en las carreteras, facilitando así la percepción del entorno y haciéndolo más manejable. En la figura 4.3 se muestra marcado en color rojo, los componentes eliminados del entorno para facilitar al sistema perceptivo además de que el dron de volar a la misma altura que las señales de tráfico, cuando este se salía de la carretera durante el entrenamiento que se explicará posteriormente, en algunas ocasiones se chocaba con estas provocando de que no se pudiese restablecer el simulador.

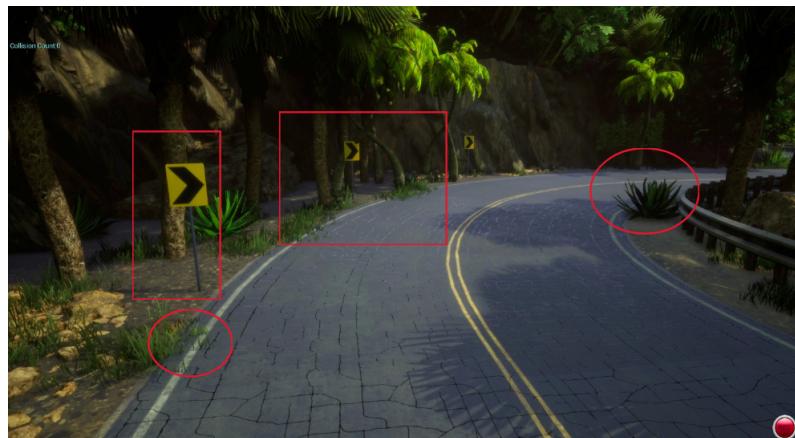
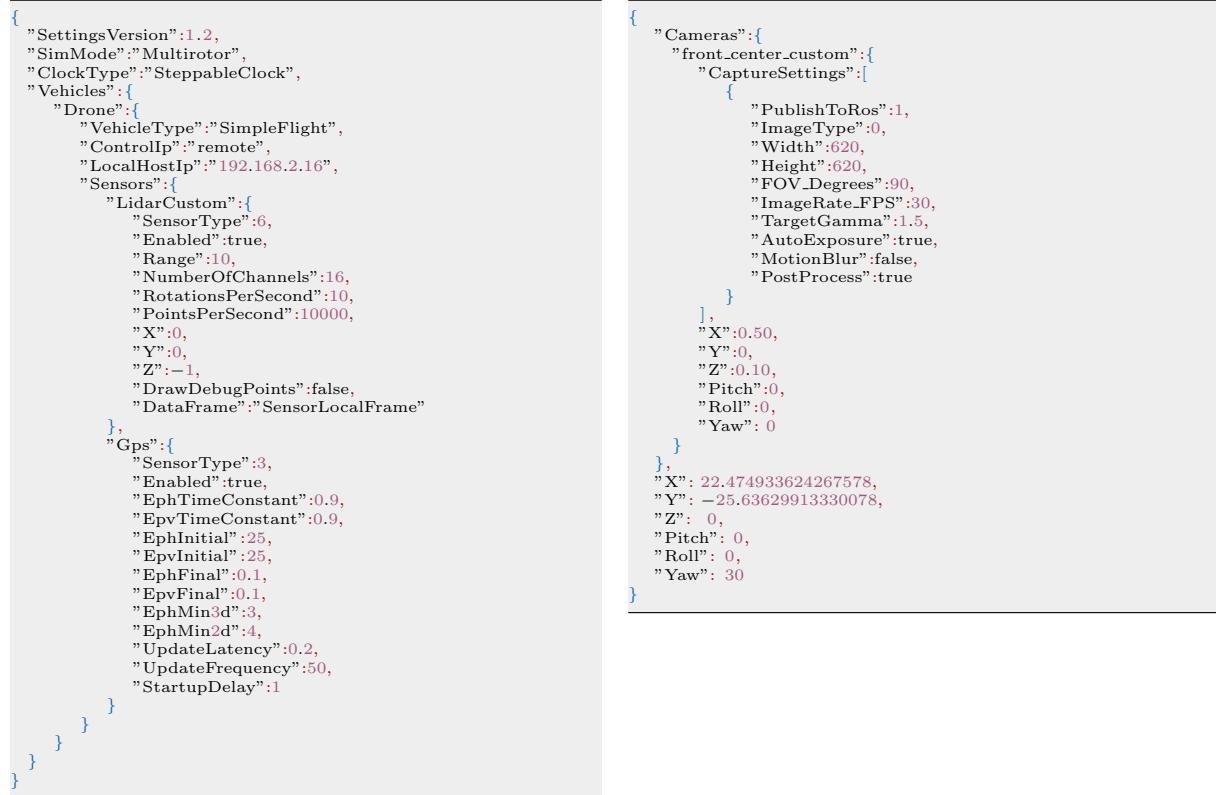


Figura 4.3: Visualización del entorno original ilustrando los objetos que dificultan al sistema perceptivo

Cuando el fichero settings.json este creado, ya se puede equipar al dron con características como qué sensores va a utilizar, el tipo de vehículo, el tipo de simulación, la comunicación y más. Como se muestra en la figura 4.4, se definen como sensores el Lidar para saber a que altura esta volando el dron respecto al suelo, el GPS para conocer la localización que tiene el dron en el entorno y la cámara con imágenes RGB. Cada sensor debe llevar su propia configuración, siguiendo los parámetros que dicta la guía oficial de Airsim³. La cámara tiene una peculiaridad de que se configura siendo otro sensor a parte de la lista de sensores de Airsim, consta de varios parámetros que se tienen que configurar como el tamaño de la imagen, el uso de ROS, los FPS de la imagen transmitida, la posición,

³<https://microsoft.github.io/AirSim/sensors/>

etc. Respecto de los actuadores, no se debe realizar ninguna configuración especial para ellos.



```
{
  "SettingsVersion":1.2,
  "SimMode":"Multirotor",
  "ClockType":"SteppableClock",
  "Vehicles":{
    "Drone": {
      "VehicleType":"SimpleFlight",
      "ControlIp":"remote",
      "LocalHostIp":"192.168.2.16",
      "Sensors": {
        "LidarCustom": {
          "SensorType":6,
          "Enabled":true,
          "Range":10,
          "NumberOfChannels":16,
          "RotationsPerSecond":10,
          "PointsPerSecond":10000,
          "X":0,
          "Y":0,
          "Z":-1,
          "DrawDebugPoints":false,
          "DataFrame":"SensorLocalFrame"
        },
        "Gps": {
          "SensorType":3,
          "Enabled":true,
          "EphTimeConstant":0.9,
          "EpvTimeConstant":0.9,
          "EphInitial":25,
          "EpvInitial":25,
          "EphFinal":0.1,
          "EpvFinal":0.1,
          "EphMin3d":3,
          "EphMin2d":4,
          "UpdateLatency":0.2,
          "UpdateFrequency":50,
          "StartupDelay":1
        }
      }
    }
  },
  "Cameras": {
    "front_center_custom": {
      "CaptureSettings": [
        {
          "PublishToRos":1,
          "ImageType":0,
          "Width":620,
          "Height":620,
          "FOV_Degrees":90,
          "ImageRate_FPS":30,
          "TargetGamma":1.5,
          "AutoExposure":true,
          "MotionBlur":false,
          "PostProcess":true
        }
      ],
      "X":0.5,
      "Y":0,
      "Z":0.1,
      "Pitch":0,
      "Roll":0,
      "Yaw": 0
    }
  },
  "X": 22.474933624267578,
  "Y": -25.63629913330078,
  "Z": 0,
  "Pitch": 0,
  "Roll": 0,
  "Yaw": 30
}
```

Figura 4.4: Configuración del dron mediante el fichero settings.json

Si no se define la posición inicial del dron como aparece en la última parte del fichero settings.json, por defecto el propio simulador te establece el dron en el punto que decida Airsim.

Para poder utilizar Airsim ROS Wrapper y Client Airsim junto con el simulador, se debe especificar la dirección IP en donde se encuentra el simulador Airsim. En el código 4.1 se muestra la ejecución del launcher `airsim_node.launch` proporcionado por Airsim ROS Wrapper y en el código 4.2 se muestra el uso del método `airsim.MultirrotorClient` para establecer la comunicación con Airsim.

```
roslaunch airsim_ros_pkgs airsim_node.launch output:=screen
host:=192.168.2.16
```

Código 4.1: Lanzamiento del nodo AirSim ROS Wrapper Node especificando la dirección IP del simulador

```
import airsim
client_airsim = airsim.MultirotorClient(ip="192.168.2.16")
```

Código 4.2: Conexión al simulador Airsim utilizando Client Airsim especificando la dirección IP

4.3. Comportamiento de seguimiento de carril con drones

Para desarrollar el comportamiento de seguimiento de carriles, es crucial diseñar un sistema perceptivo que detecte el carril a seguir. Una vez construido este sistema, se procede al desarrollo de un algoritmo de seguimiento de carril. Dicho algoritmo se enfoca en dos estrategias distintas para comandar las velocidades lineales y angulares del dron, como se comenta en la sección 4.4.

4.3.1. Percepción

En las fases iniciales del desarrollo del sistema perceptivo, se optó por emplear técnicas de visión tradicional utilizando métodos que ofrece la biblioteca OpenCV, tales como GaussianBlur, Canny e HoughLines. Antes de poder utilizarlos, se necesita obtener la imagen proporcionada por la cámara a bordo del dron. Esto se logra suscribiéndose al topic `/airsim_node/Drone/front_center_custom/Scene`, facilitada por el nodo AirSim ROS Wrapper Node. Posteriormente, mediante el uso de CvBridge⁴, se transforma la imagen al formato de OpenCV para su posterior procesamiento con esta biblioteca.

Para los métodos GaussianBlur, Canny e HoughLines se necesita trabajar con imágenes en escala de grises, por lo tanto, primero se realiza una transformación de la imagen de color a escala de grises. A continuación se utiliza el método GaussianBlur⁵, que aplica un desenfoque gaussiano a la imagen. Este proceso suaviza la imagen, reduciendo el ruido y mejorando la detección de bordes y líneas. Seguidamente, se aplica el método Canny⁶ para la detección de bordes en la imagen. Finalmente, se aplica el método HoughLines⁷ para detectar líneas en la imagen mediante la transformación de Hough, permitiendo identificar líneas rectas dentro de una imagen.

⁴http://wiki.ros.org/cv_bridge

⁵https://docs.opencv.org/4.x/d4/d13/tutorial_py_filtering.html

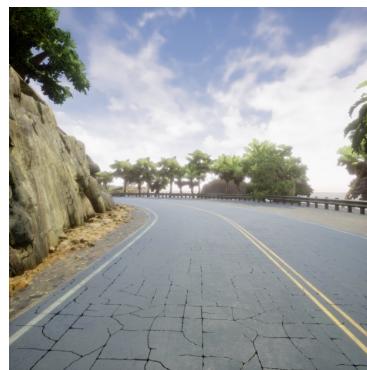
⁶https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html

⁷https://docs.opencv.org/3.4/d9/db0/tutorial_hough_lines.html

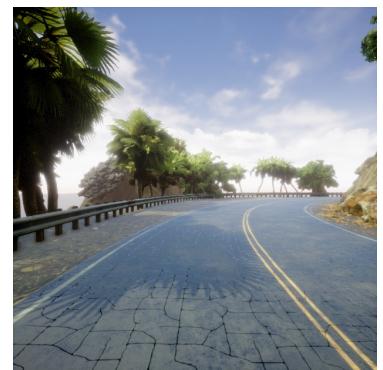
El resultado final de este enfoque tradicional de estos algoritmos de visión clásica se puede observar en la figura 4.5. En la parte superior de la figura se presentan diferentes segmentos de carril: Una zona con curvas, una zona recta y una zona semirrecta. Estas imágenes representan la entrada original capturada por la cámara del dron. En la parte inferior de la figura, se muestran los resultados de la detección de líneas sobre las imágenes originales mediante el uso de los algoritmos mencionados. En la zona recta se puede ver como las líneas detectadas son claramente visibles y paralelas, lo que indica una correcta detección en este tipo de trayectos rectos. Mientras que en las zonas curvas y las zonas semirrectas, presentan dificultades en la detección de las líneas, debido a que HoughLines esta optimizada para la detección de líneas rectas y no para la detección de líneas con curvas.



a: Zona con curvas



b: Zona recta



c: Zona semirrecta



d: Detección en la zona con curvas



e: Detección en la zona recta



f: Detección en la zona semirrecta

Figura 4.5: Resultado de la detección de líneas utilizando visión clásica

Debido al mal funcionamiento de estos algoritmos de visión clásica en la detección de líneas en los carriles, hemos optado por utilizar técnicas de Inteligencia Artificial, en especial Deep Learning, utilizando la red neuronal YOLOP. Como se comenta en la sección 3.2, YOLOP es una red neuronal que realiza una detección de objetos de tráfico, una segmentación del área transitable y una detección de carriles. Es importante destacar

que se realiza solamente la inferencia de los distintos modelos ya entrenados con datos reales que ofrece YOLOP, sin realizar un nuevo entrenamiento. Este enfoque nos permite aprovechar la detección de las líneas proporcionadas por la red para mejorar la percepción del entorno del dron, proporcionando una detección más robusta y precisa a una gran variedad de configuraciones de las carreteras urbanas dentro del entorno de Airsim.

Inferencia de YOLOP

Para realizar la detección de las líneas en la carretera, se utiliza la red neuronal YOLOP⁸. En primer lugar, se realiza la inferencia del modelo `End-to-end.pth` y más adelante la inferencia de los modelos `Yolop-320-320.onnx`, `Yolop-640-640.onnx` e `Yolop-1280-1280.onnx` mediante el uso de ONNX Runtime, todos ellos entrenados con datos reales. Para más adelante realizar un análisis y comparativa de cada modelo de YOLOP y seleccionar el modelo que tenga mejores resultados en cuanto a eficiencia y detecciones en las líneas en la carretera.

El archivo `End-to-end.pth` es un modelo preentrenado con datos reales construido en formato Pytorch que contiene los pesos y sesgos de la red neuronal después del entrenamiento. En el contexto de redes neuronales, `end-to-end` significa que la red se entrena para realizar todas las etapas de un proceso completo. En el caso de YOLOP, esto significa que se ha entrenado de manera integral para realizar simultáneamente todas sus tareas (detección de objetos, segmentación de áreas transitables y detección de líneas). Para poder cargar este modelo, se realiza su carga mediante el repositorio de Github⁹ especificando el modelo de la red neuronal y la opción '`pretrained`' definida con valor `True`, especificando que los pesos del modelo de la red se cargan del archivo `End-to-end.pth`.

```
import torch
model = torch.hub.load('hustvl/yolop', 'yolop', pretrained=True)
```

Código 4.3: Cargar modelo YOLOP con pesos preentrenados `End-to-end.pth`

Una vez se tenga el modelo cargado del repositorio como se ilustra en el código 4.3, se puede escoger si se quiere realizar la inferencia mediante el uso de la CPU o el uso de la GPU. Para una mejor comportamiento de la red neuronal, se utiliza la GPU para realizar la inferencia asignándole el dispositivo GPU como se muestra en el código 4.4.

Esta red neuronal ofrece modelos con formato ONNX: `Yolop-320-320.onnx`,

⁸<https://github.com/hustvl/YOLOP>

⁹<https://github.com/hustvl/YOLOP>

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = model.to(device)
```

Código 4.4: Cargar modelo YOLOP escogiendo como dispositivo la GPU

`Yolop-640-640.onnx` e `Yolop-1280-1280.onnx`. Estos modelos son construidos a partir del archivo `End-to-end.pth` y se convierten en modelos con formato Onnx. Cada modelo proporcionado con formato onnx tiene una resolución distinta de entrada en cuanto a las dimensiones de las imágenes. El archivo `Yolop-320-320.onnx` necesita una entrada con unas dimensiones de 320x320 píxeles en las imágenes, el archivo `Yolop-640-640.onnx` necesita una entrada con unas dimensiones de 640x640 píxeles en las imágenes y por último el archivo `Yolop-1280-1280.onnx` se necesita unas dimensiones de imágenes de 1280x1280 píxeles. En cuanto al uso de estos modelos, se debe realizar la configuración de los drivers de CUDA con la versión disponible para ONNX Runtime, dichas versiones tienen que ser compatible entre sí. Para llevar a cabo este proceso, se emplea la tabla requisitos proporcionada en la página oficial de ONNX Runtime¹⁰.

Cuando se trabaja con ONNX Runtime, se puede especificar qué proveedores de ejecución utilizar para ejecutar la inferencia del modelo de ONNX elegido. Cada proveedor contiene un conjunto de núcleos optimizados para un objetivo específico (por ejemplo, CPU, GPU, IoT), cada proveedor se especifica en una lista en orden de prioridad. Para realizar inferencia de estos modelos, se escoge CUDA para ejecutar mediante la GPU. En el código 4.5 se muestra la carga del modelo `Yolop-320-320.onnx` junto con la configuración del provider `CUDAExecutionProvider`.

```
import onnxruntime as ort

ROUTE_MODEL = "/home/bb6/YOLOP/weights/yolop-320-320.onnx"
ort_session =
    ort.InferenceSession(ROUTE_MODEL, providers=['CUDAExecutionProvider'])
```

Código 4.5: Cargar modelo por ejemplo YOLOP-320-320.onnx

Para realizar la redimensión de las imágenes dependiendo de cada modelo, se utiliza un método implementado en la propia página de YOLOP en Github que se puede encontrar en la siguiente página¹¹, el cuál redimensiona la imagen de entrada según cada modelo. En el código se realiza la inferencia del modelo `Yolop-320-320.onnx`.

La inferencia en el resto de modelos de ONNX siguen un procedimiento similar, sin

¹⁰<https://onnxruntime.ai/docs/execution-providers/CUDA-ExecutionProvider.html>

¹¹https://github.com/hustvl/YOLOP/blob/main/test_onnx.py

```

    _, da_seg_out, ll_seg_out = self.ort_session.run(
        ['det_out', 'drive_area_seg', 'lane_line_seg'],
        input_feed={"images": img}
)

```

Código 4.6: Inferencia del modelo yolop-320-320.onnx

embargo, es importante considerar que se deben ajustar las dimensiones de las imágenes de entrada y la ruta de almacenamiento del modelo correspondiente.

Por último, para poder utilizar imágenes en la inferencia de YOLOP tanto del modelo de Pytorch como de los modelos de ONNX, se debe convertir la imagen en un tensor. Para poder realizarlo, se utiliza la función `transforms.ToTensor`¹² para realizar la transformación a tensor. Una vez se obtenga el tensor, se puede realizar la inferencia del modelo `End-to-end.pth` como se ilustra en el código 4.7.

```

from torchvision import transforms

transform = transforms.ToTensor()

imagen_tensor = transform(cv_image).to(device).unsqueeze(0)
_, da_seg_out, ll_seg_out = self.model(imagen_tensor)

```

Código 4.7: Inferencia del modelo en Pytorch

Como resultado de la inferencia se obtiene un tensor de salida que corresponde a la probabilidad de detección de la segmentación y la detección de las líneas de la calzada. Dicho tensor se debe de convertir en una imagen para poder visualizar dicho resultado como una imagen. Por ello, se convierte el tensor en un array numpy y se realiza una transformación para cambiar las dimensiones de (H,W,C) a (C,H,W) esto se debe a que OpenCV representa las imágenes en formato numpy array y se transpone las dimensiones¹³.

Después de este proceso, se obtiene un array de numpy, se normaliza con valores de 0-1. Con la normalización se ayuda a igualar la escala de los píxeles de la imagen. Finalmente se muestra el resultado de la inferencia de la red en una imagen como se ilustra en el código 4.8. Al normalizar los valores de 0-1, los píxeles con un valor 1 se muestran en la imagen final, asignándoles color verde el resultado de la segmentación de la calzada y de color rojo la detección de las líneas de la calzada.

¹²<https://pytorch.org/vision/main/generated/torchvision.transforms.ToTensor.html>

¹³<https://lindevs.com/convert-pytorch-tensor-to-opencv-image-using-python>

```

import cv2
import numpy as np
for image in (da_seg_out,ll_seg_out):
    image_np = image.detach().cpu().numpy()
    image_array = np.transpose(image_np, (2, 3, 1, 0))

    image_norm = cv2.normalize(image_array[:, :, 1, :], None, 0, 1,
                               cv2.NORM_MINMAX, cv2.CV_8U)

    images.append(image_norm)

cv_image[images[0] == 1] = [0, 255, 0]
cv_image[images[1] == 1] = [0, 0, 255]

cv2.imshow('Image', cv_image)
cv2.waitKey(1)

```

Código 4.8: Inferencia de YOLOP mediante los pesos End-to-end.pth

Resultados de YOLOP

En esta sección se contrastan los resultados de la inferencia de YOLOP con los diferentes modelos preentrenados que ofrece esta red. Para realizar esta comparación, se usa el ordenador PC2 junto con la GPU Nvidia 2070 RTX para realizar la inferencia de cada modelo. Se recopilan datos del tiempo medio de inferencia en milisegundos de cada modelo de YOLOP durante aproximadamente un minuto, mientras se teleopera con un joystick el dron por las carreteras. El resultado de la inferencia de estos modelos se puede ver en este [vídeo](#), en donde se muestra los frames por segundo que realiza cada uno la inferencia.

El objetivo de este análisis es determinar qué modelo ofrece mejores resultados en términos de eficiencia y rendimiento. En la figura 4.6 se muestra este análisis mostrando la media en realizar la inferencia de YOLOP utilizando los modelos `End-to-end.pth`, `Yolop-320-320.onnx`, `Yolop-640-640.onnx` e `Yolop-1280-1280.onnx` en milisegundos.

El modelo que tiene un tiempo de inferencia menor al resto se trata de `Yolop-320-320.onnx`, tiene un tiempo de inferencia alrededor de 10 milisegundos, esto significa que el modelo `Yolop-320-320.onnx` tiene aproximadamente un *framerate* de 100 FPS. Si se compara este resultado con los restantes modelos, es el ganador en cuanto en tiempo de inferencia, rate y mejores resultados. ONNX está diseñado para ser más eficiente en términos de memoria y velocidad de inferencia respecto a Pytorch, lo que puede mejorar la velocidad y la precisión del modelo, una menor resolución en cuanto al

tamaño de las imágenes reduciendo la cantidad de información al procesarlas. Aun así, depende de varios factores y según las necesidades, en este caso se busca un equilibrio entre velocidad de inferencia y mejores resultados en cuanto a la detección de las líneas de la calzada.

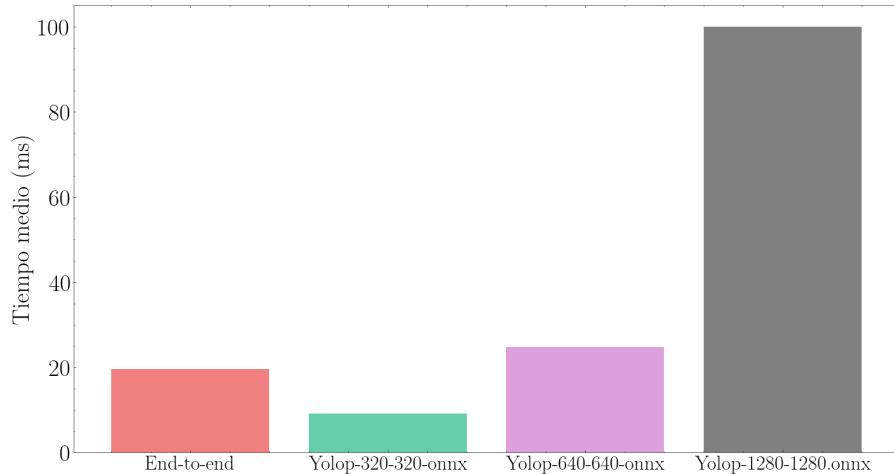


Figura 4.6: Resultados de los diferentes modelos que ofrece la red YOLOP

Por ello, se escoge el modelo `Yolop320-320.onnx` ya que obtiene mejores resultados para poder realizar la detección de las líneas del carril. En cuanto a los resultados de la inferencia, en la figura 4.9 se puede observar los diferentes resultados que tiene la red neuronal YOLOP dependiendo del modelo empleado. Para ello, se utilizan dos zonas totalmente distintas para realizar la inferencia de los modelos. Aunque se escoja el modelo `Yolop-320-320.onnx`, la detección de las líneas no es perfecta como se muestra en la figura 4.7.

Como se puede ver en la figura 4.7, el margen izquierdo del carril no se detecta correctamente, solo se pueden apreciar unos pocos puntos, además la detección de la zona que pertenece al carril presenta parches que la red no detecta como zona transitable, que se pueden observar en la parte central de la imagen como en la parte inferior. Debido a esto, se realiza un post-procesado que se explicará a continuación, para que estos errores provenientes de la red neuronal, no afecten al posterior control del dron.

Estos fallos en la detección se deben principalmente a que los modelos preentrenados que se proporcionan junto al YOLOP, han sido entrenados únicamente con datos reales como se ha mencionado anteriormente.



Figura 4.7: Resultados de la inferencia del modelo yolop-320-320.onnx

Para lograr una reconstrucción de las líneas dentro del carril que se desea seguir y mejorar la información obtenida de la red neuronal, se deben de seguir una serie de pasos que se pueden observar en la figura 4.8. En primer lugar, se seleccionan las líneas que se necesitan mediante DBSCAN, es un algoritmo de clustering basado en aprendizaje no supervisado. Con este algoritmo, se realiza una segmentación de las diferentes líneas detectadas por parte del modelo Yolop-320-320.onnx para en segundo lugar aplicar un filtro y seleccionar las líneas para construir el carril. Las líneas de la red neuronal no llegan a ser exactas, por lo que, aunque se realice el filtrado de estas líneas, estas deben ser reconstruidas para lo cual, en último lugar se emplearán regresiones cuadráticas.

Con las líneas que delimitan el carril, se puede obtener el área transitable del mismo para posteriormente obtener su centroide y realizar el control del dron como se explicará en la sección 4.4.



Figura 4.8: Fases de reconstrucción de las líneas del carril

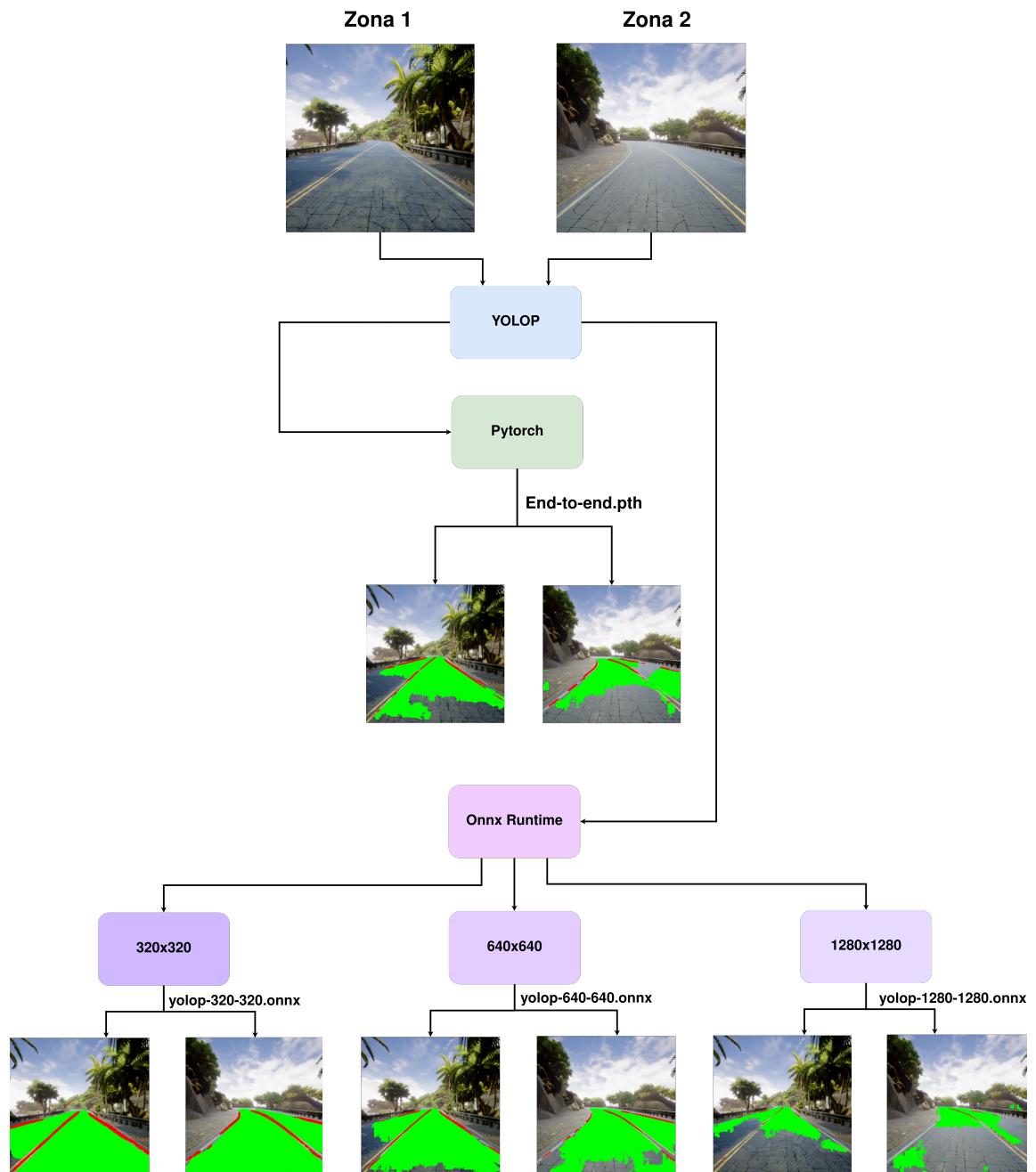


Figura 4.9: Proceso del resultado de los diferentes modelos de la red neuronal YOLOP

DBSCAN

Para clasificar las líneas detectadas por parte de la red neuronal, se utiliza un algoritmo de clustering basado en aprendizaje no supervisado, en particular DBSCAN(Density-Based Spatial Clustering of Applications with Noise)[1].

Este algoritmo clasifica un conjunto de muestras en grupos (clústeres) basándose en la densidad de puntos en el espacio de características, que es una forma de representar datos utilizando diferentes medidas o atributos. A diferencia de otros métodos de clustering, como K-Means¹⁴, DBSCAN no requiere especificar el número de clústeres de antemano, siendo capaz de identificar patrones de forma arbitraria y especialmente útil para detectar estructuras complejas en los datos.

DBSCAN requiere la configuración de varios parámetros importantes:

1. **Eps:** Consiste en la distancia máxima que puede existir entre dos muestras para que una se considere vecina de la otra. Dicha distancia no se trata de un límite máximo entre las distancias que puede haber dentro de un cluster. Si se elige un valor de eps muy alto, puede provocar que los clústeres distintos se combinen en el mismo grupo. En cambio, un valor de eps muy pequeño puede resultar una mayor generación de clústeres pequeños.
2. **Min_samples:** Es el número mínimo de muestras dentro de un vecindario para que un punto se considere como un punto central incluyendo al propio punto. Si min_samples se establece en un valor alto, DBSCAN encontrará clústeres más densos y si el valor de min_samples es bajo, los clústeres serán más dispersos.
3. **Metric:** La métrica utilizada para calcular la distancia entre los conjuntos de clústeres. Por defecto se utiliza la distancia euclídea.

Antes de ejecutar el algoritmo, es necesario convertir la imagen resultante de la red neuronal en un array bidimensional compuesto por coordenadas de puntos x e y, por ello se utiliza la función `column_stack`¹⁵ de numpy, transformando un array unidimensional en uno bidimensional. Cuando se realice la conversión se da pie a escoger los valores del algoritmo DBSCAN. Se ha seleccionado un valor de `eps` de 10, esto significa que, para que un punto sea considerado parte de un mismo clúster, debe estar a una distancia máxima de 10 píxeles para que sean vecinos y pertenezcan al mismo grupo de clúster. El parámetro

¹⁴<https://www.iebschool.com/blog/algoritmo-k-means-que-es-y-como-funciona-big-data/>

¹⁵https://numpy.org/doc/stable/reference/generated/numpy.column_stack.html

`min_samples` se establece en 5, al menos 5 muestras deben estar presentes en la vecindad de un punto para que se considere parte de un mismo clúster.

```

def clustering(self, cv_image):

    points_lane = np.column_stack(np.where(cv_image > 0))
    dbscan = DBSCAN(eps=10, min_samples=5, metric="euclidean")

    if points_lane.size > 0:
        dbscan.fit(points_lane)
        labels = dbscan.labels_

        clusters = set(labels)
        if -1 in clusters:
            clusters.remove(-1)

        for cluster in clusters:
            points_cluster = points_lane[labels==cluster,:]
            centroid = points_cluster.mean(axis=0).astype(int)
            color = self.colors[cluster % len(self.colors)]
            cv_image[points_cluster[:,0], points_cluster[:,1]] = color

    return cv_image

```

Código 4.9: Algoritmo de clustering utilizando DBSCAN

Al aplicar DBSCAN, el algoritmo devuelve una lista de etiquetas que van desde 0 a n, donde 0 representa el primer clúster detectado y n el último. Las muestras etiquetadas como ruido o no clasificadas reciben la etiqueta -1 y se eliminan de la lista de clústeres. Para visualizar el resultado, se itera sobre la lista de etiquetas y se muestran los puntos correspondientes de cada etiqueta en la imagen de salida. El proceso completo, incluyendo los valores de los parámetros de DBSCAN así como el uso de la clasificación de las líneas, se puede ver en el código 4.9.

La figura 4.10 se muestran los diferentes resultados que tiene este algoritmo en localizaciones distintas. En la primera fila, se muestra las imágenes originales. En la segunda fila se muestra el resultado del modelo Yolop-320-320.onnx y por último en la tercera fila se muestra el resultado de la clasificación de las líneas detectadas por la red utilizando el algoritmo DBSCAN. Cada clúster detectado es representado por un color distinto para poder visualizar el total de grupos que ha sido capaz clasificar DBSCAN. Dependiendo de la ubicación del dron, DBSCAN clasifica las líneas en aproximadamente entre 3 y 4 grupos de clústeres. En función de la clasificación de DBSCAN, se realiza un filtrado de clústeres para seleccionar específicamente aquellos grupos de puntos que representan las líneas del carril deseado.

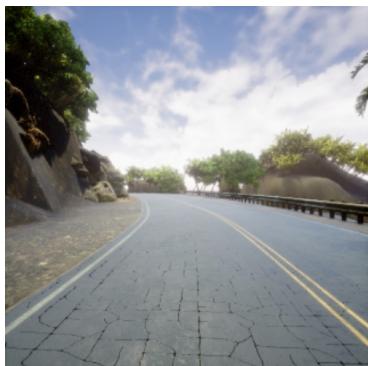
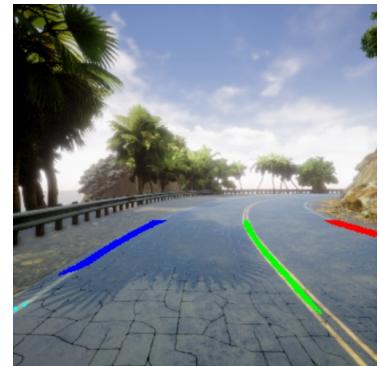
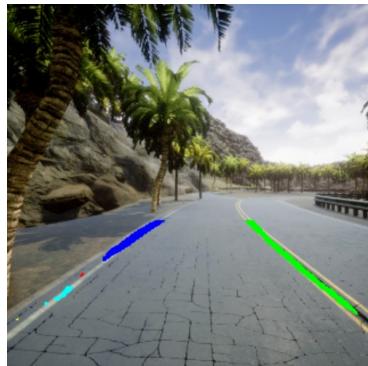
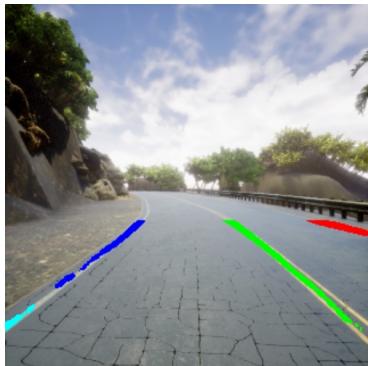
Imágenes Originales**Detección de Yolop-320-320.onnx****Resultado de DBSCAN**

Figura 4.10: Resultado de la clasificación de las líneas por DBSCAN

Filtrado de clústeres

Después de ejecutar el algoritmo DBSCAN, es crucial determinar qué grupos de clústeres pertenecen a las líneas que definen el carril que se desea seguir. Para lograrlo,

se realiza un filtrado de los centroides de cada grupo de clústeres detectado y se filtran según su posición respecto al centro de la imagen. La imagen tiene unas dimensiones de 320x320 píxeles, con su centro en (160,160) píxeles . Se consideran exclusivamente los valores horizontales de los centroides para realizar el filtrado con respecto al centro de la imagen. Basándose en este filtrado, los centroides se dividen en dos grupos distintos: izquierda y derecha de la imagen, como se detalla en el código 4.10.

```
WIDTH = cv_image.shape[1]
if centroid[1] < WIDTH/2: # left lane
    left_clusters.append((points_cluster,centroid))

elif centroid[1] >= WIDTH/2: # right lane
    right_clusters.append((points_cluster, centroid))
```

Código 4.10: Clasificación de clústeres respecto a las dimensiones de la imagen

Durante el proceso de filtrado de los grupos de clústeres, se seleccionan subgrupos específicos de cada grupo (izquierda y derecha). Los subgrupos se eligen utilizando una función maximizada con respecto a un punto central predefinido P, cuyo valor es (160,220), ajustado según las dimensiones de la imagen de 320x320 píxeles. El resto de clústeres son descartados. Esta función maximizada se detalla en el código 4.11.

Al manipular puntos en numpy, es importante recordar que las coordenadas están invertidas (y,x en lugar de x,y). Además de seleccionar subgrupos en relación con el punto central P, se considera también la densidad de puntos en los grupos de clústeres detectados tanto en la izquierda como en la derecha. Este enfoque garantiza que la selección no solo dependa de la proximidad al punto central, sino también de la cantidad de puntos presentes.

```
def score_cluster(self,cluster, center):
    points_cluster, centroid = cluster

    proximity = np.linalg.norm(centroid - center)
    density = len(points_cluster)
    return density / proximity
```

Código 4.11: Función maximizada para escoger el grupo de clúster más cercano y denso respecto al punto P

En la figura 4.11 se muestra el proceso desde la clasificación de líneas por DBSCAN hasta la selección de los clústeres. DBSCAN es capaz de detectar hasta 4 clústeres, a partir de los cuales se realiza un filtrado basado en las coordenadas x de cada centroide de cada clúster. Posteriormente, se aplica una función maximizada para obtener los 2 clústeres

requeridos. Estos clústeres están representados en color rojo para el grupo izquierdo y en verde para el grupo derecho.

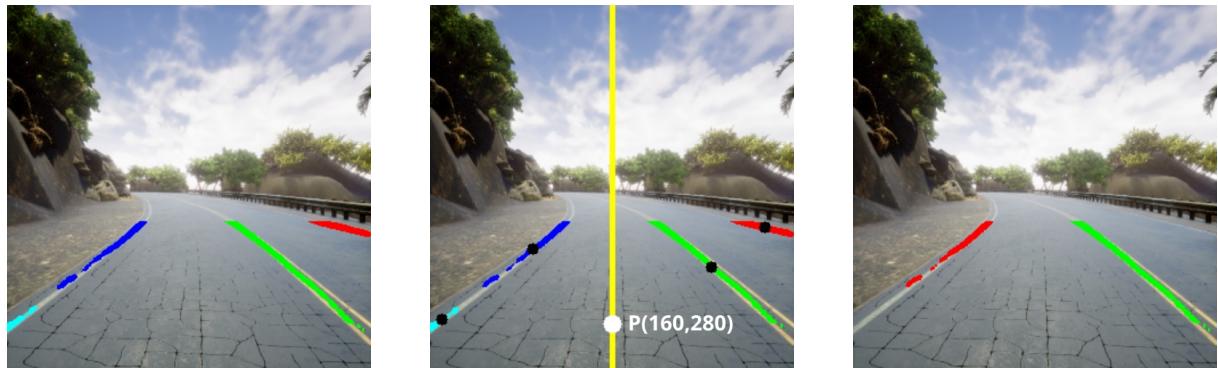


Figura 4.11: Proceso de filtrado de las líneas clasificadas por DBSCAN

Regresión cuadrática

Al tener los clústeres seleccionados, es necesario construir dos líneas que representen la naturaleza del carril basándose en estos puntos. Para ello, se emplea una técnica de aprendizaje automático conocida como regresión, que permite aproximar a un número N de puntos a una recta, curva u otra forma matemática que se pueda definir con una función. Debido a que las líneas de los carriles rara vez son completamente rectas, suelen ser curvilíneas, se opta por utilizar una regresión de tipo cuadrática.

La regresión cuadrática permite modelar los puntos de cada clúster como curvas, lo cual es más adecuado para capturar las variaciones y curvaturas típicas de los carriles. Esta técnica no solo ayuda a interpretar mejor la forma del carril, sino que también facilita la navegación del dron al seguir la trayectoria por el carril.

Para la construcción de las regresiones cuadráticas se utiliza las funciones de la librería numpy denominada `Polyfit`¹⁶ y `Polyval`¹⁷. `Polyfit` es una función que calcula los coeficientes del polinomio que mejor se ajusta a los datos utilizando el método de los mínimos cuadrados para la ecuación cuadrática, obteniendo tres coeficientes, denominados a , b y c . Se define los valores de las coordenadas x como los puntos que se quiere realizar dicho cálculo, se realiza un rango entre un mínimo y un máximo ajustándolo a las dimensiones de la imagen, además de la ayuda de dos puntos auxiliares en los extremos inferiores en el cálculo de la regresión, esto se realiza para que la regresión cuadrática tienda a las esquinas superiores de la imagen como se muestra en el código 4.12.

¹⁶<https://numpy.org/doc/stable/reference/generated/numpy.polyfit.html>

¹⁷<https://numpy.org/doc/stable/reference/generated/numpy.polyval.html>

```

FACTOR_PIXEL = 20
MIN_VALUE_X = (cv_image.shape[1] // 2) + FACTOR_PIXEL
MAX_VALUE_X = cv_image.shape[1]

valuesX = np.arange(MIN_VALUE_X,MAX_VALUE_X)
point = np.array([cv_image.shape[1]/1.104,cv_image.shape[1]])
coefficients = np.polyfit(points_cluster[:,0],points_cluster[:,1],2)

```

Código 4.12: Cálculo de los coeficientes de la regresión cuadrática

Se realiza un promedio de los últimos diez valores utilizando los coeficientes mencionados, como se ilustra en el código 4.13. Este promedio se recalcula cada cinco iteraciones. Este paso es crucial para reducir las oscilaciones causadas por las detecciones de la red neuronal, las cuales son fundamentales para el comportamiento sigue-carril. Una vez calculados los coeficientes, se calcula la regresión cuadrática mediante la función `Polyval`, dicha función calcula la función cuadrática con los coeficientes calculados anteriormente.

```

self.list_coeff_a.append(coefficients[0])
self.list_coeff_b.append(coefficients[1])
self.list_coeff_c.append(coefficients[2])

a = np.mean(self.list_coeff_a[-10:])
b = np.mean(self.list_coeff_b[-10:])
c = np.mean(self.list_coeff_c[-10:])

mean_coeff = np.array([a,b,c])

self.counter += 1

if(self.counter > 5):
    self.list_coeff_a.clear()
    self.list_coeff_b.clear()
    self.list_coeff_c.clear()

values_fy = np.polyval(mean_coeff,valuesX).astype(int)
fitLine_filtered = [(x, y) for x, y in zip(valuesX, values_fy) if 0 <= y
    <= (cvimage.shape[1] - 1)]
line = np.array(fitLine_filtered)

```

Código 4.13: Cálculo de la regresión cuadrática

Al obtener los valores de la función cuadrática, se seleccionan los puntos de dicha función que se encuentran en el eje y, entre 0 y el máximo de la imagen menos uno (es

decir, entre 0 y 319). Este proceso se debe realizar debido a que el resultado de la función `Polyval` puede dar como resultado números negativos. En una imagen no se puede indexar con números negativos ya que se corresponderían a píxeles que se encuentran fuera de la propia imagen.

Este proceso se realiza dos veces, una regresión cuadrática para el grupo de clústeres detectados escogidos de la derecha y otra regresión cuadrática para el grupo de clústeres detectados escogidos de la izquierda. Finalmente, con dichas regresiones cuadráticas se procede a realizar una dilatación de los puntos para tener un resultado más llamativo y visual al poder ver las regresiones cuadráticas. El resultado se puede visualizar en la figura 4.12 teniendo presente los pasos que se ha tenido que seguir para realizar las regresiones, dichas líneas se representan de color amarillo.

Detección de Yolop-320-320.onnx



Resultado de las regresiones cuadráticas

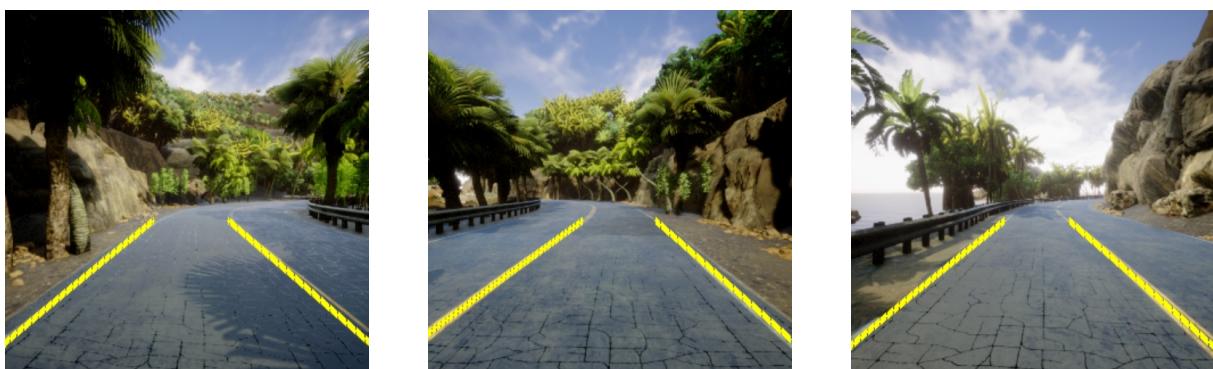


Figura 4.12: Resultado de la regresión cuadrática

Interpolación y cálculo del centro de masas del carril

Una vez obtenidas las líneas reconstruidas, se necesita definir el área de carril dentro de estas líneas. Como se menciona en la sección 4.3.1, los resultados del modelo `Yolop-320-320.onnx` respecto al área del carril no son totalmente exactos produciéndose una mala detección por parte de la red. Por lo tanto, se requiere una técnica para construir el área del carril a partir de las líneas obtenidas mediante regresiones cuadráticas. Por ello, se utiliza una interpolación para recorrer y calcular los puntos que se encuentran dentro de los límites de las regresiones.

Se realizan dos interpolaciones, una para los puntos de la regresión de la derecha y otra para los puntos de la regresión izquierda.

```
def interpolate_lines(self, cvimage, points_line_left, points_line_right):

    gray_image = cv2.cvtColor(cvimage, cv2.COLOR_BGR2GRAY)
    np_gray = np.array(gray_image)
    x, y = np.nonzero(np_gray)
    img_points = np.column_stack((x, y))

    f1 = interp1d(points_line_left[:, 0], points_line_left[:, 1], kind='slinear', fill_value="extrapolate")
    f2 = interp1d(points_line_right[:, 0], points_line_right[:, 1], kind='slinear', fill_value="extrapolate")
    y_values_f1 = f1(img_points[:, 0])
    y_values_f2 = f2(img_points[:, 0])
    indices = np.where((y_values_f1 < img_points[:, 1]) & (img_points[:, 1] <= y_values_f2))

    points_between_lines = img_points[indices]
    filtered_points_between_lines =
        points_between_lines[points_between_lines[:, 0] > 180]
    return filtered_points_between_lines
```

Código 4.14: Método del cálculo de las funciones de interpolación

Estas funciones interpolan los valores de los puntos en y en función de los valores de los puntos en x , más adelante se evalúa los valores de los puntos que se encuentran entre ambas regresiones cuadráticas como se muestra en el código 4.14. Cuando se proceda a dicha evaluación, se desarrolla un filtro para seleccionar un fragmento de carril. El carril se representa en color azul en la imagen resultante.

En la figura 4.13, se contrasta los resultados obtenidos por parte del modelo `Yolop-320-320.onnx` y la construcción del carril calculada a partir de las regresiones.

Como se puede observar, la construcción del carril desarrollada es bastante robusta en comparación con la salida que te ofrece la red, siendo así una forma de poder construir un carril. Finalmente, para poder realizar el comportamiento sigue-carril, el dron necesita mantener una alineación respecto al carril calculado, por ello se calcula el centro de masas que tiene el carril.

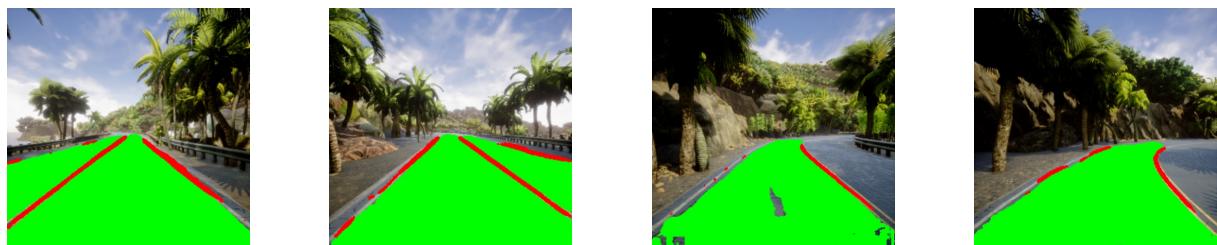
Para el cálculo del centroide del carril, se toma ecuación 4.1, que consiste en el cálculo de centro de masas de una superficie.

$$\vec{r}_{CM} = \frac{\sum_i m_i \vec{r}_i}{\sum_i m_i} = \frac{\sum_i m_i \vec{r}_i}{M} \quad (4.1)$$

Ecuación 4.1: Calculo del centro de masas

Siendo \vec{r}_{CM} las coordenadas del centroide compuesta por (x,y), \vec{r}_i los puntos del carril construido y m_i la masa que constituye cada punto, se supone que todos los puntos del carril tienen masa igual a 1. Dicha suposición, simplifica el cálculo, pero en aplicaciones del mundo real, las masas pueden variar.

Detección de Yolop-320-320.onnx



Construcción del carril desarrollada

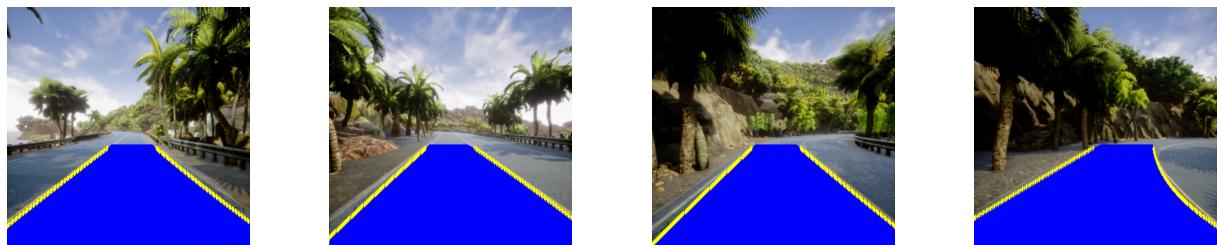


Figura 4.13: Comparativa entre el resultado de detección Yolop-320-320.onnx y la construcción del carril desarrollada

El cálculo de masa total ($\sum_i m_i \vec{r}_i$), consiste en el producto entre la masa individual de cada punto por el tamaño de los puntos que componen el carril. A continuación, se realiza el sumatorio de las coordenadas de los puntos ponderados por su masa y la división por el cálculo de masa total. El resultado de la ecuación es el conocido centroide compuesto

por (x,y).

El resultado final del cálculo del centroide se muestra en la figura 4.14 resaltando con una flecha de color rojo donde se encuentra el centro de masas del carril representado por un círculo de color blanco. Este centroide se utilizará en los controladores que se explicarán en la sección 4.4, para poder determinar que tan desviado está el dron respecto al centro del carril, que se representa con este centro de masas.

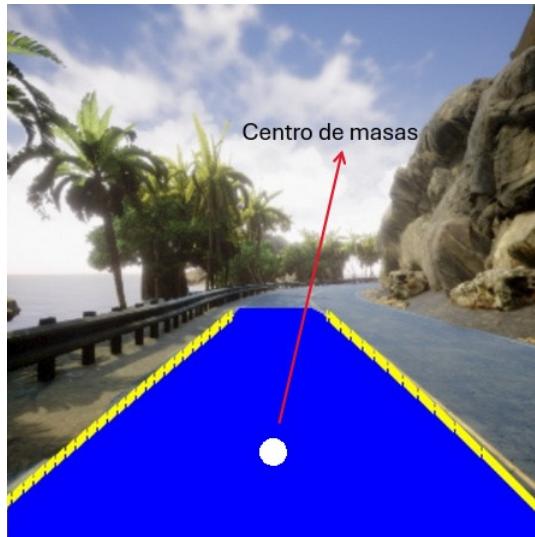


Figura 4.14: Resultado del centro de masas

4.3.2. Análisis del algoritmo de percepción

A continuación, se realiza un análisis detallado de cada submódulo del sistema perceptivo. Esta evaluación se presenta en la tabla 4.1, incluye tres columnas indicando el nombre de cada componente del sistema perceptivo, el dispositivo donde se ejecuta y el tiempo medio calculado en milisegundos.

Componentes Percepción	Dispositivo	Tiempo (ms)
YOLOP	GPU	9.523
Procesamiento de YOLOP	CPU	0.195
DBSCAN	CPU	0.000301
Filtrado de clústeres	CPU	6.867
Regresiones cuadráticas	CPU	3.593
Interpolación	CPU	8.532
Cálculo del centro de masas	CPU	0.0427
Tiempo Total		28.753

Cuadro 4.1: Profiling

Estos datos se recopilan utilizando un ordenador portátil (PC2) con una GPU Nvidia

RTX 2070, como se menciona en la sección de 4.1. La tabla muestra que los componentes perceptivos que requieren más tiempo de procesamiento son la inferencia del modelo `Yolop-320-320.onnx` y el cálculo de las interpolaciones. Aunque la inferencia se realice en la GPU, está constantemente procesando imágenes para generar la salida. Por otro lado, la interpolación requiere procesar todos los puntos que pertenecen a las dos regresiones, lo cual consume recursos de la CPU.

En comparación, el componente DBSCAN es el más eficiente en términos de tiempo de cómputo, lo que hace interesante trabajar con este algoritmo de clustering. Sumando los tiempos de cada componente, se obtiene un tiempo medio total de 28.753 milisegundos, lo que se traduce en un procesamiento aproximado de 30 FPS (frames por segundo).

Este análisis es crucial para conseguir un sistema perceptivo en tiempo real a la hora de implementar los comportamientos de control.

4.4. Comportamientos de control

En este sub-apartado, se describe el desarrollo de los comportamientos de control junto con el sistema perceptivo desarrollado para posteriormente realizar una comparativa entre ambos y contrastar los resultados obtenidos.

4.4.1. Sigue-carril basado en PID

A partir del sistema perceptivo, se desarrolla un comportamiento autónomo de sigue-carril utilizando un controlador PID. Este comportamiento tiene como objetivo demostrar la efectividad del sistema perceptivo y la capacidad del dron para mantenerse centrado y corregir desviaciones en relación con el centroide del carril en tiempo real. El controlador PID ajusta continuamente las velocidades angulares del dron para mantener un ángulo lo más alineado posible respecto con el centro del carril, reduciendo así el error de desviación.

El proceso de ajuste de las constantes del controlador PID es crucial y debe realizarse de manera iterativa, adaptándose a las condiciones específicas del entorno. Es importante destacar que los valores óptimos de las constantes para un escenario particular pueden no ser aplicables directamente a otros escenarios, lo que requiere ajustes personalizados. En cuanto a la velocidad lineal, se mantiene un valor constante.

Además del controlador PID utilizado para el seguimiento del carril, se implementa

otro controlador PD para mantener la altura del dron con respecto al suelo durante su navegación. Esto se debe a la naturaleza del simulador, donde las carreteras presentan variaciones en su elevación, lo que puede afectar la detección del carril del sistema perceptivo. Por lo cual, con este controlador PD, permite al dron ajustar su altitud, asegurando así una altura constante y estable. En el código 4.15 se expone los valores de cada constante de los controlador PID Y PD ajustadas.

En la figura 4.15, se muestran diferentes frames de la navegación autónoma del dron utilizando el controlador anteriormente descrito. En las imágenes se muestran los valores de la velocidad lineal, angular, el error con respecto al centro de masas y una línea representando la mitad de la imagen. En la primera imagen y en la segunda, el error se encuentra entre 4 y 7 píxeles, mostrando que el dron consigue una alineación centrada respecto al carril. Por último, en la tercera imagen se muestra una desviación significativa con un error de 16 píxeles, el controlador PID realiza ajustes en la velocidad angular para corregir la trayectoria.

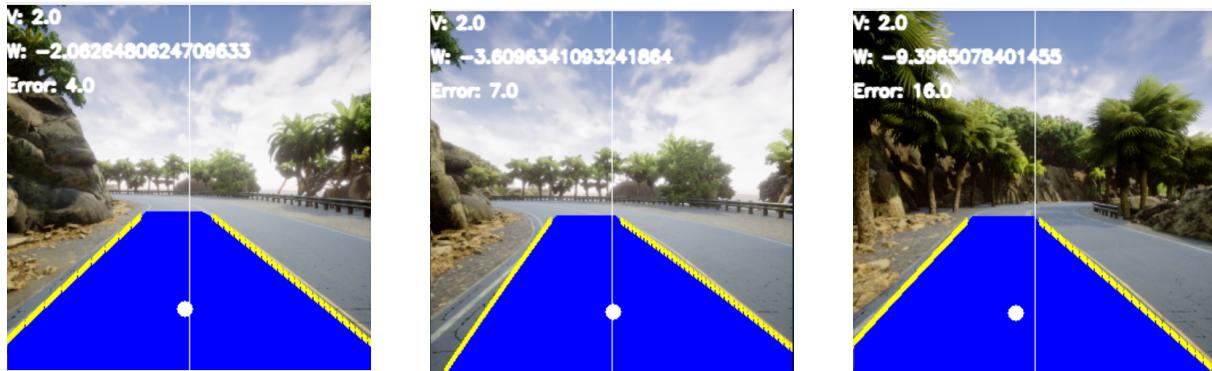


Figura 4.15: Sigue-carril basado en PID

```

kp_height = 0.1
kd_height = 0.4

kp_speed_controller = 0.09
kd_speed_controller = 0.1
ki_speed_controller = 0.008

```

Código 4.15: Valores de las variables del PD del control de altura y del PID del controlador de velocidad angular

Este controlador PID, aunque funcional, presenta problemas ya que es incapaz de adaptarse a todos los escenarios, ya que en algunos casos más complicados en los que las curvas son más cerradas el controlador es incapaz de trazar la curva provocando una salida de carril por parte del dron.

Debido a esto, se considera una implementación de técnicas de aprendizaje por refuerzo. Estas técnicas, permiten al dron aprender y adaptarse de manera autónoma a diversas condiciones del entorno, manteniéndose centrado en el carril de manera más eficiente y robusta en comparación con los métodos de control tradicionales. Este método a diferencia del anterior, permite al dron generalizar las acciones que debe de tomar ante distintas situaciones del entorno, lo que tiene como consecuencia que este pueda funcionar correctamente ante entornos desconocidos.

4.4.2. Sigue-carril basado en aprendizaje por refuerzo

Como se menciona en la sección 1.3, el aprendizaje por refuerzo consiste en enseñar a un agente a desempeñar un comportamiento deseado mediante recompensas y penalizaciones. Este comportamiento se aprende a base de interacciones con el entorno de trabajo y observaciones de como puede responder, de forma similar a los niños que exploran el mundo que les rodea y aprenden las acciones que les ayudan a alcanzar un objetivo.

En este TFG se emplea como algoritmo de aprendizaje por refuerzo, el algoritmo Q-learning. Este se basa en una tabla de correspondencias estado-acción que almacena ante distintos estados la acción que se debe de tomar para maximizar la recompensa. Para este algoritmo es necesario definir distintos componentes, que son: agente, entorno, estados, acciones, función de recompensa y la política.

Los componentes del algoritmo Q-learning, que se definen para la tarea del sigue-carril son los siguientes:

- **Agente:** En este caso, el dron. Es la entidad o modelo que se pretende entrenar para que aprenda a tomar decisiones (acciones) en función del estado en el que se encuentre y de las observaciones que vea en el entorno.
- **Entorno:** El dron interactuará con el entorno de simulación Airsim, específicamente en el mapa Coastline, durante su navegación, aprendiendo el comportamiento deseado.



Figura 4.16: Entorno en la fase de entrenamiento del sigue carril basado en Q-Learning

Dentro de este escenario, se ha considerado tres localizaciones distintas para el dron a la hora de realizar el entrenamiento, así como un punto final del recorrido, como se muestra en la figura 4.16. La idea de utilizar diferentes localizaciones de inicio es evitar que el modelo haga sobreajuste (overfitting)¹⁸, es decir, que se aprenda de memoria los movimientos que debe hacer, para completar la ruta. En su lugar, se quiere que el dron sea capaz de aprender y adaptarse a diversas situaciones, desarrollando así un comportamiento más generalizado y robusto. Esto se consigue debido a que en los distintos entrenamientos, para una misma iteración, el dron se encontrará en posiciones distintas de la carretera, ya que la posición de inicio se elige de forma aleatoria (equiprobable) para cada episodio.

- **Estados:** Se necesita definir las situaciones que se puede encontrar el dron navegando dentro por el carril. Por ello, se divide verticalmente la imagen del sistema perceptivo en múltiples franjas representando cada una franja un estado. En total son 14 franjas equidistantes de 10 píxeles, empezando desde la izquierda de la imagen hasta la derecha. La determinación del estado actual del dron se basa en las coordenadas en píxeles que se encuentra ubicado el centro de masas del carril calculado. Se puede observar el resultado de los estados en la figura 4.17. En total se definen 15 estados.

¹⁸<https://iartificial.blog/aprendizaje/estrategias-para-evitar-el-overfitting-en-modelos-de-ml/>

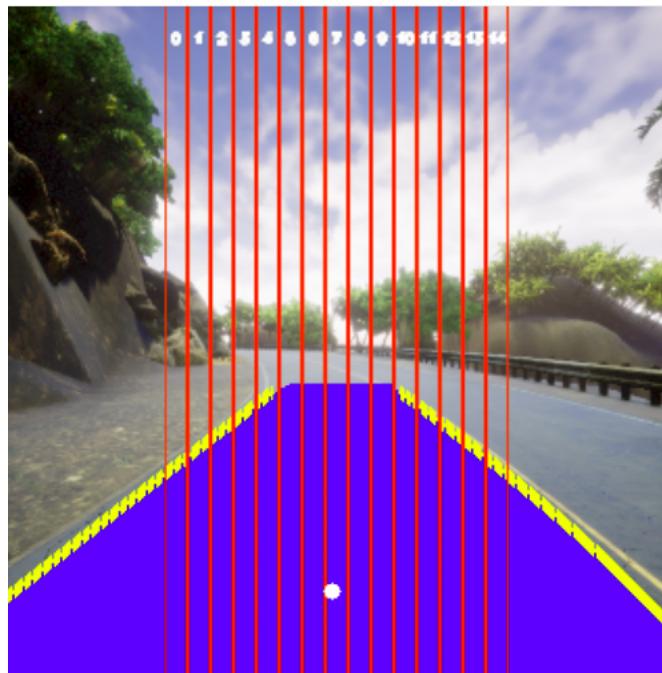


Figura 4.17: Estados definidos para el sigue-carril basado en aprendizaje por refuerzo

- Acciones:** En total, se han definido 21 acciones que el agente puede llevar a cabo. Se componen de pares de velocidades lineales y angulares, las velocidades lineales tienen un intervalo de 0.1 m/s hasta 2.0 m/s. Así siendo el intervalo de la velocidad angular de -25 hasta 25 grados/segundo, teniendo giros hacia la izquierda y hacia la derecha. Dichos pares de velocidades están formados a partir de la función que nos proporciona numpy llamada `linspace`¹⁹.

En la figura 4.18, se puede observar las acciones definidas en 3 partes. En color naranja se representa las acciones con giro izquierdo, en color rojo se presenta la acción central definida así como velocidad lineal sin velocidad angular y en color azul, las acciones con giro derecho. Cada pareja de acción tiene un identificador, desde el 0 siendo la primera acción hasta el 20 siendo la última acción.

ACCIONES IZQUIERDA	A0	A1	A2	A3	A4	A5	A6	A7	A8	A9
	V = 0.1 W = -25.0	V = 0.29 W = -22.5	V = 0.48 W = -20.0	V = 0.67 W = -17.5	V = 0.86 W = -15.0	V = 1.05 W = -12.5	V = 1.24 W = -10.0	V = 1.43 W = -7.5	V = 1.62 W = -5.0	V = 1.81 W = -2.5
ACCIÓN CENTRAL										
	A10 V = 2.0 W = 0.0									
ACCIONES DERECHA	A11	A12	A13	A14	A15	A16	A17	A18	A19	A20
	V = 1.81 W = 2.5	V = 1.62 W = 5.0	V = 1.43 W = 7.5	V = 1.24 W = 10.0	V = 1.05 W = 12.5	V = 0.86 W = 15.0	V = 0.67 W = 17.5	V = 0.48 W = 20.0	V = 0.29 W = 22.5	V = 0.1 W = 25.0

Figura 4.18: Acciones disponibles para el sigue-carril basado en aprendizaje por refuerzo

¹⁹<https://numpy.org/doc/stable/reference/generated/numpy.linspace.html>

- **Función de Recompensa o Penalizaciones:** Determina cómo se premia o penaliza el agente para que alcance el objetivo deseado. Las recompensas o penalizaciones se definen en una función específica, la cual varía según el objetivo que se quiere lograr. La función de recompensa se compone en dos partes: premiar al dron por permanecer centrado en el carril y premiarlo por mantener una orientación adecuada con respecto al carril.

Cada parte de la función tiene un peso de importancia diferente: un 85 % por permanecer centrado en el carril y un 15 % para la orientación respecto al carril. Ambas partes son normalizadas en un rango de 0 a 1. Además, se penaliza con un valor constante de -10 al dron cuando se sale del carril o pierde la percepción. En la ecuación 4.2 se muestra las dos componentes descritas, siendo $norm(error_{centro})$ y $norm(error_{angulo})$ las normalizaciones del error del centro de masas del carril y la orientación, respectivamente, multiplicadas por sus pesos correspondientes.

$$R(s, a) = (1 - norm(error_{centro})) \cdot w_1 + (1 - norm(error_{angulo})) \cdot w_2 \quad (4.2)$$

Ecuación 4.2: Función de recompensa

```
def reward_function(self,cx,angle):

    reward = 0
    target_heading = 0
    error_lane_center = (WIDTH/2 - cx)
    heading_difference = (target_heading - angle)

    if (self.is_exit_lane(cx)):
        reward = -10

    else:
        normalise_error_centre = (abs(error_lane_center) - MIN_ERROR) /
            (MAX_ERROR - MIN_ERROR)
        reward_centre = 1 - normalise_error_centre

        normalise_error_angle = (abs(heading_difference) - MIN_ANGLE) /
            (MAX_ANGLE - MIN_ANGLE)
        reward_angle = 1 - normalise_error_angle

        reward = (reward_centre * CENTRE_WEIGHT) + (reward_angle *
            ANGLE_WEIGHT)

    return reward
```

Código 4.16: Función de recompensa

La implementación de la función de recompensa y las penalizaciones se puede ver en el código 4.16.

- **Política:** Determina que acción realizar en cada estado que se encuentre el agente. Dicha política varía según el tipo de algoritmo que se quiera seguir dentro de aprendizaje por refuerzo. Puede ser determinista o estocástico.

En este TFG, se sigue una política epsilon-greedy[6], que consiste en un equilibrio entre la fase de exploración y explotación dentro del entrenamiento. Cuando el agente tenga que escoger la acción, tiene en cuenta dos enfoques:

- Exploración (con probabilidad ϵ): El agente escoge una acción al azar para explorar
- Explotación (con probabilidad $1 - \epsilon$): El agente escoge la acción con el valor de la tabla $Q(S,A)$ más alto, es decir, la mejor acción conocida para el estado actual.

El parámetro ϵ es el responsable de controlar la proporción de exploración frente a explotación, si ϵ es alto el agente explorará más, en cambio si ϵ es bajo el agente se centrará en la explotación. Por lo que, en la elección de acción se genera un número n aleatorio. Si n es menor que la probabilidad ϵ , la acción se escoge aleatoriamente, en cambio si n es mayor que la probabilidad de ϵ , la acción seleccionada es aquella que tenga el mayor valor en la tabla $Q(S,A)$ para el estado correspondiente.

Dentro de esta política, la probabilidad de ϵ es decayente, es decir, no tiene un valor constante en cada episodio en el entrenamiento. Para ello, se realiza una disminución de esta probabilidad para que el dron explote poco a poco lo aprendido provocando que el modelo converja.

Hay varias formas de realizar el descenso de ϵ , por ejemplo se puede realizar linealmente, logarítmicamente, exponencial o escalonado. En este caso, el descenso de ϵ se realiza de una forma lineal, teniendo pequeños decrementos en cada episodio. Durante el entrenamiento, se puede observar el resultado del descenso de ϵ en la figura 4.19. A medida que el valor de ϵ disminuye, el agente gradualmente explota lo aprendido, alcanzando una explotación total cuando el valor de ϵ es igual a 0.

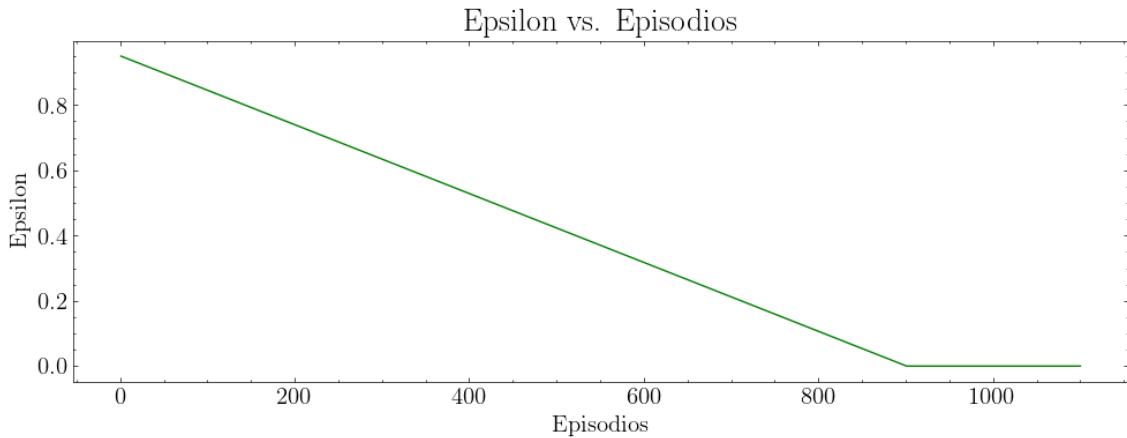


Figura 4.19: Gráfica del valor de epsilon respecto al episodio

El algoritmo Q-learning se basa en seguir una función acción-recompensa formada por un tabla estado-acción la cual se va rellenando siguiendo la ecuación de Bellman[5] mostrada en la ecuación 4.3. en donde,

$$Q(s, a) = Q(s, a) + \alpha \cdot [R(s, a) + \gamma \cdot \max Q(s', a') - Q(s, a)] \quad (4.3)$$

Ecuación 4.3: Ecuación de Bellman

- $Q(s, a)$: Valor Q para el estado s y la acción a . Se trata de una matriz formada por estado y acción que se rellena durante la fase de entrenamiento para utilizarla en la fase de inferencia.
- α : Tasa de aprendizaje entre un valor de 0 a 1. Consiste en el porcentaje que daremos al agente para el proceso de aprendizaje, si dicho valor es alto daremos más peso al valor aprendido. Dicho valor se define con un valor de 0.5
- $R(s, a)$: Recompensa por tomar la acción a en el estado s . La función de recompensa es crucial en el proceso de aprendizaje del agente, evalúa como es de buena la acción tomada por el agente en el estado que se encuentre. Proporciona información al agente sobre que acciones maximizan la recompensa total a lo largo del tiempo. Dicha función de recompensa es diseñada dependiendo de cual sea el objetivo de tu agente.
- γ : Factor de descuento entre un valor de 0 a 1. Modela la importancia de las recompensas futuras en relación con las recompensas inmediatas, refleja la importancia del agente por las recompensas a largo plazo. Un valor alto significará que le agente valorará mucho las recompensas futuras, mientras que un valor bajo indica que se enfocará más en las recompensas inmediatas. Dicho valor se define con

un valor de 0.7.

- $\max Q(s', a')$: Valor Q máximo en el próximo estado s' para todas las acciones posibles a' .

Fase de entrenamiento

Dentro de esta fase, existen dos conceptos importantes a destacar:

- **Episodios:** Se define episodio como una secuencia completa de interacciones que se produce entre el agente y el entorno. Cada episodio comienza con un estado inicial y consta de una serie de pasos o acciones tomadas por el agente.
- **Iteraciones:** Son los pasos que puede dar un agente en el entorno dentro de un episodio. Estos pasos pueden incluir observaciones del entorno, decisiones tomadas por el agente y las consecuentes recompensas o penalizaciones recibidas.

La fase de entrenamiento consiste que el agente explore todo lo máximo posible en el entorno respecto a los estados que tiene y las acciones que puede tomar, es decir, al comienzo del entrenamiento se inicializa la tabla $Q(S,A)$ a cero todos sus valores y mediante el algoritmo de Q-learning, se va llenando hasta converger el modelo.

Básicamente, en cada iteración del algoritmo, se escoge una acción según si el agente se encuentra en exploración o explotación, se calcula la recompensa obtenida y se actualiza la tabla de nuevo sobre el algoritmo.

El entrenamiento finaliza cuando el agente haya aprendido, es decir, el agente ha cumplido el objetivo. Esto sucede cuando las iteraciones del algoritmo y la recompensa acumulada del agente en esta fase se estabilicen y tengan valores constantes, es decir, no cambian significativamente con más iteraciones. Esto significa que el modelo ha convergido y a continuación se pasa a la fase de inferencia.

Durante la fase de entrenamiento, el algoritmo se ejecuta continuamente. Como se menciona en la sección 4.3.2, el sistema de percepción tiene 30 frames por segundo, proporcionando al agente información visual del entorno en tiempo real. Sin embargo, para permitir al agente tomar decisiones más efectivas y gestionar adecuadamente las acciones y estados del entorno, se reduce el tiempo global de procesamiento del algoritmo a 10 frames por segundo. Esto asegura que el agente tenga suficiente tiempo para analizar cada acción y estado. La ejecución de este algoritmo de entrenamiento es aproximadamente de 12

horas, asegurando que el algoritmo va ser capaz de completar el entrenamiento de manera correcta consiguiendo converger.

En la figura 4.20 se muestra la evolución de las iteraciones que tiene el dron con el entorno en función del episodio. Al comienzo del entrenamiento, el dron obtiene bajas iteraciones, esto se debe a que al comienzo del entrenamiento desconoce completamente el entorno y las acciones que debe realizar. Pero, no obstante, a medida que el entrenamiento va avanzando, las iteraciones obtienen valores altos hasta llegar al episodio 900 donde la fase de exploración finaliza y comienza la fase de explotación siendo epsilon 0 como se muestra en la figura 4.19. A partir de este punto, las iteraciones tienen una tendencia ascendente con pequeñas variaciones, estas variaciones se deben a que en la fase de entrenamiento el dron no comienza siempre en el mismo punto de inicio como se muestra en la figura 4.16, obteniendo en cada punto de partida hasta el final del recorrido una serie de iteraciones. Aunque se muestren estas pequeñas variaciones, el modelo ha sido capaz de converger debido a que el dron es capaz de completar el recorrido continuamente sin salirse del carril.

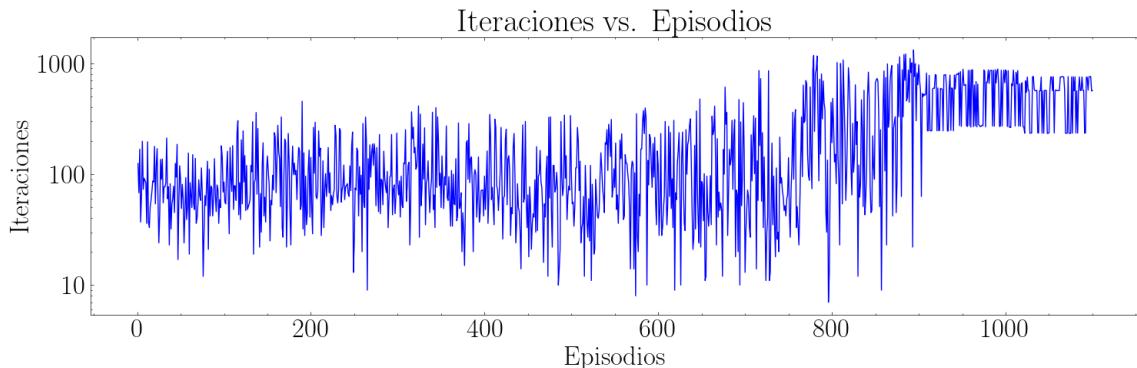


Figura 4.20: Gráfica de las iteraciones respecto al episodio

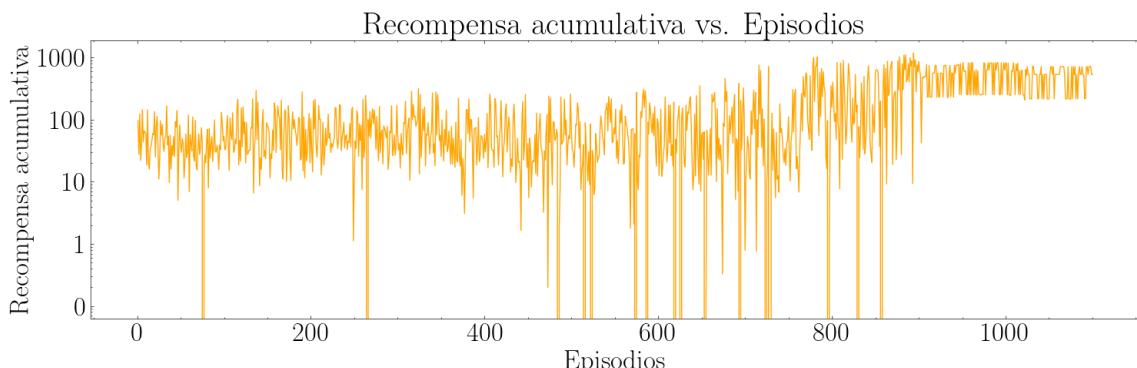


Figura 4.21: Gráfica de la recompensa acumulativa en función de los episodios

Respecto a la evolución de la función de recompensa, se puede observar en la figura 4.21 como la recompensa acumulativa va obteniendo valores ascendentes a medida que el dron aprende del entorno para no salirse del carril. En el momento en que se llega

al episodio 900, la recompensa acumulativa tiende a ser siempre ascendente teniendo pequeñas variaciones entre los episodios. Este comportamiento ocurre como las iteraciones, se producen pequeñas variaciones debido a los puntos de partida que tiene el dron durante cada comienzo de su entrenamiento. A partir del momento que no se produzca grandes variaciones respecto a las iteraciones y la recompensa acumulativa en función de los episodios, se puede decir que el modelo ha sido capaz de converger. Esto significa que el dron completa satisfactoriamente el recorrido en cualquier punto de partida definido. Una vez analizado los resultados del entrenamiento con las diferentes métricas, se da a pie a la fase de inferencia para verificar el resultado obtenido con el modelo convergido.

Fase de inferencia

La fase de inferencia consiste en indexar la tabla $Q(S,A)$ que se ha ido llenando en la fase de entrenamiento. El dron cuando se encuentre en un estado específico consultará la tabla $Q(S,A)$ para encontrar la mejor acción en ese estado (cuando mencionamos la mejor acción nos referimos al máximo valor en la tabla que tenga en ese estado). Cuando el dron tome dicha acción se moverá al siguiente estado siendo así un proceso iterativo hasta alcanzar la meta. En esta fase, los valores de la tabla permanecen constantes y se utiliza para la toma de decisiones basadas en el conocimiento aprendido en la fase de entrenamiento.

En la figura 4.22 se puede observar las diferentes acciones que ha utilizado el dron para completar el recorrido utilizando cuatro de las veintiuno acciones disponibles. Aunque se muestre solamente cuatro acciones utilizadas, durante la fase de entrenamiento, el dron ha aprendido muchas más acciones y estados como se muestran.

Se destaca que la acción más usada se trata de una de las acciones que presentan poco giro y alta velocidad lineal dentro de las acciones con más velocidad lineal, esto se debe a la importancia de la función de recompensa al considerar que queremos que el dron se mantenga constantemente en el centro del carril manteniendo un ángulo de orientación adecuado sin salirse del carril. En la figura 4.23, se muestra una serie de imágenes durante la inferencia del modelo en el recorrido donde el dron ha sido entrenado, manteniendo una trayectoria estable respecto al centro del carril.

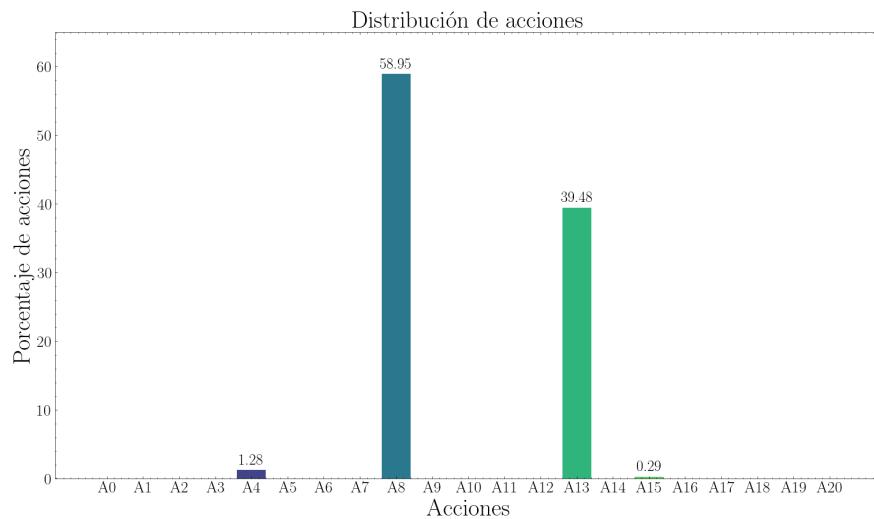


Figura 4.22: Distribución de acciones en inferencia

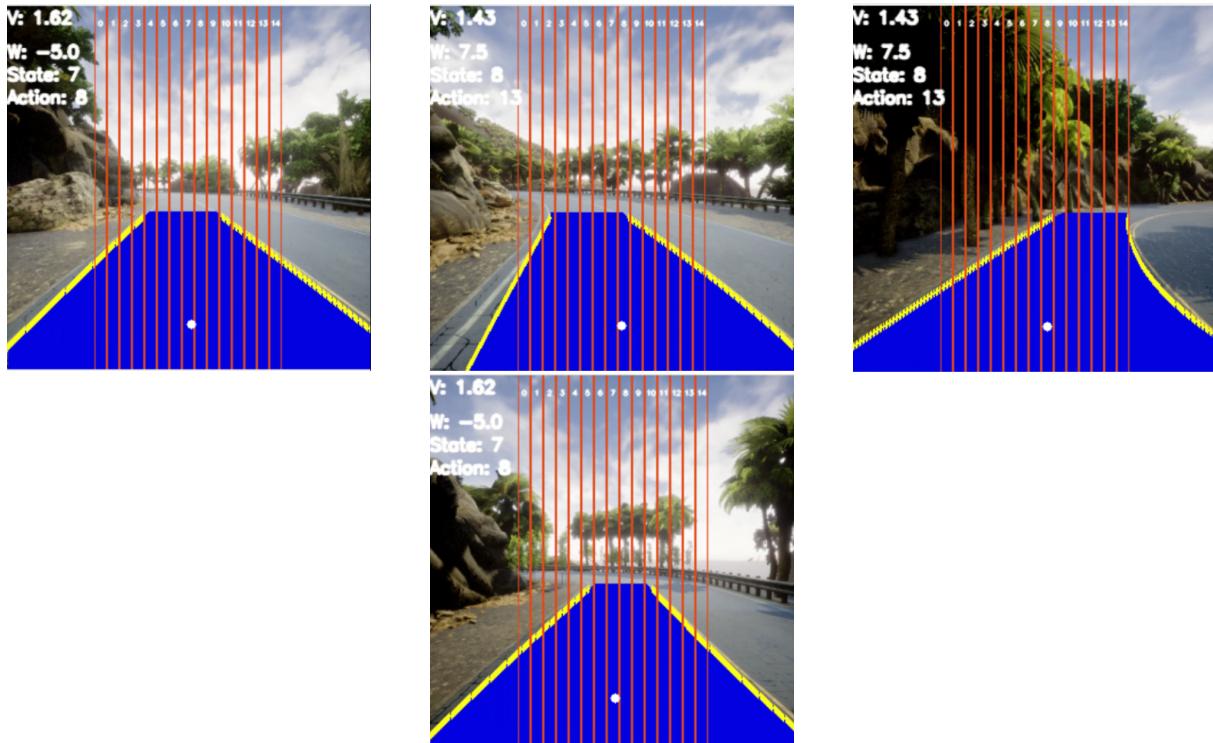


Figura 4.23: Resultado del comportamiento de aprendizaje por refuerzo ilustrando los estados y acciones en la fase de inferencia

Como resultado al utilizar el algoritmo de Q-learning, el modelo entrenado ofrece tener un resultado eficaz a la hora de navegar por el entorno, pudiendo observar el modelo entrenado final en diferentes localizaciones aleatorias en el entorno como se muestra en la figura 4.16.

Escalado de acciones

Como se menciona en la sección 4.4.2, durante la fase de entrenamiento, el modelo ejecutaba a una velocidad de 10 frames por segundo, lo que permitía al dron analizar y responder a cada acción y estado de manera efectiva.

Sin embargo, durante la fase de inferencia, el modelo es capaz de ejecutar a 25 frames por segundo. Por lo que, para aprovechar al máximo este modelo ya entrenado y esta velocidad de inferencia más alta, se decide buscar los valores de las acciones que el modelo puede tomar, es decir, ver a qué valores se pueden aumentar las velocidades que tienen asignadas las acciones manteniéndose el dron capaz de funcionar correctamente.

Estas acciones se refieren a las velocidades lineales y angulares que el dron puede alcanzar. En estos experimentos, se incrementan los valores de las velocidades lineales y angulares multiplicándolas por un factor específico. Las velocidades lineales se multiplican por 2.0, lo que significa un incremento del 100 % y las velocidades angulares se multiplican por 1.8, que representa un incremento del 80 %.

Al buscar los nuevos valores de las velocidades lineales y angulares, es importante tener en cuenta la relación entre ambas. El dron al tener un movimiento circular uniforme (MCU), la velocidad angular y la velocidad lineal están interrelacionadas. Si se aumenta la velocidad lineal, también se tiene que ajustar la velocidad angular para mantener el equilibrio del movimiento, es decir, que para describir una misma curva a una velocidad lineal más alta, se debe incrementar en un valor similar la velocidad angular. Dicha relación se expresa siguiendo la ecuación 4.4 del MCU, donde v es la velocidad lineal, r es el radio de la curva y w es la velocidad angular.

$$v = r * w \quad (4.4)$$

Ecuación 4.4: Ecuación del movimiento circular uniforme

Por lo tanto, al aumentar la velocidad lineal, también se debe aumentar la velocidad angular. Este es un aspecto crucial a tener en cuenta al modificar los valores de las velocidades lineales y angulares en un modelo de Q-learning.

A partir de ahora, a la hora de inferir el modelo, se utilizan estos nuevos valores de velocidades lineales y angulares. En la figura 4.22 se puede observar que el dron sigue escogiendo acciones con velocidad lineal alta y velocidad angular baja para mantenerse dentro del recorrido sin salirse, además de aumentar el porcentaje de las acciones escogidas respecto a la distribución de acciones.

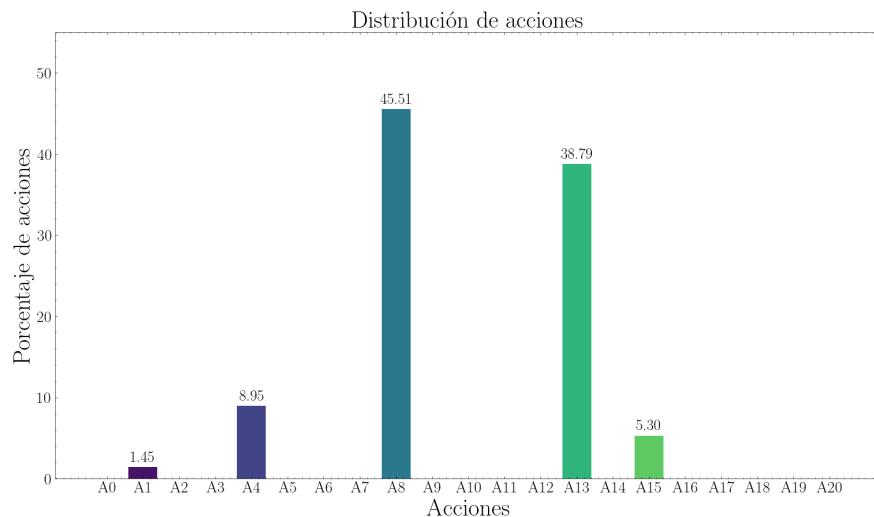


Figura 4.24: Distribución de acciones en inferencia

4.4.3. Comparativa PID vs Q-learning

Para comprobar la robustez y la eficacia que demuestra el modelo entrenado con aprendizaje por refuerzo respecto al seguimiento de carril mediante el PID, se realiza varias comparativas, en ambos comportamientos se utiliza el mismo sistema perceptivo que se menciona en la sección 4.3.1.

Se definen límites a las velocidades (lineales y angulares) del controlador PID para que ambos tengan las mismas condiciones, siendo que el controlador PID tenga los mismos rangos mínimos y máximos de velocidades como Q-learning. Para realizar la comparativa, se ha escogido un escenario en el que ambos comportamientos visitan por primera vez sin que ninguno de los dos comportamientos se salgan del carril. En la figura 4.25 se muestra el escenario marcando el inicio y el final del recorrido, además de la vista que tiene el dron al comienzo del recorrido.



Figura 4.25: Escenario de control tradicional y aprendizaje por refuerzo

Para realizar como ha sido la trayectoria de ambos comportamientos dentro del escenario, se ha ido recogiendo en cada momento la posición del dron (x,y) durante su navegación para más adelante contrastarlos. En la figura 4.26, se muestra una comparativa de trayectorias entre PID y Q-learning realizando un zoom en una de las partes de ambas trayectorias.

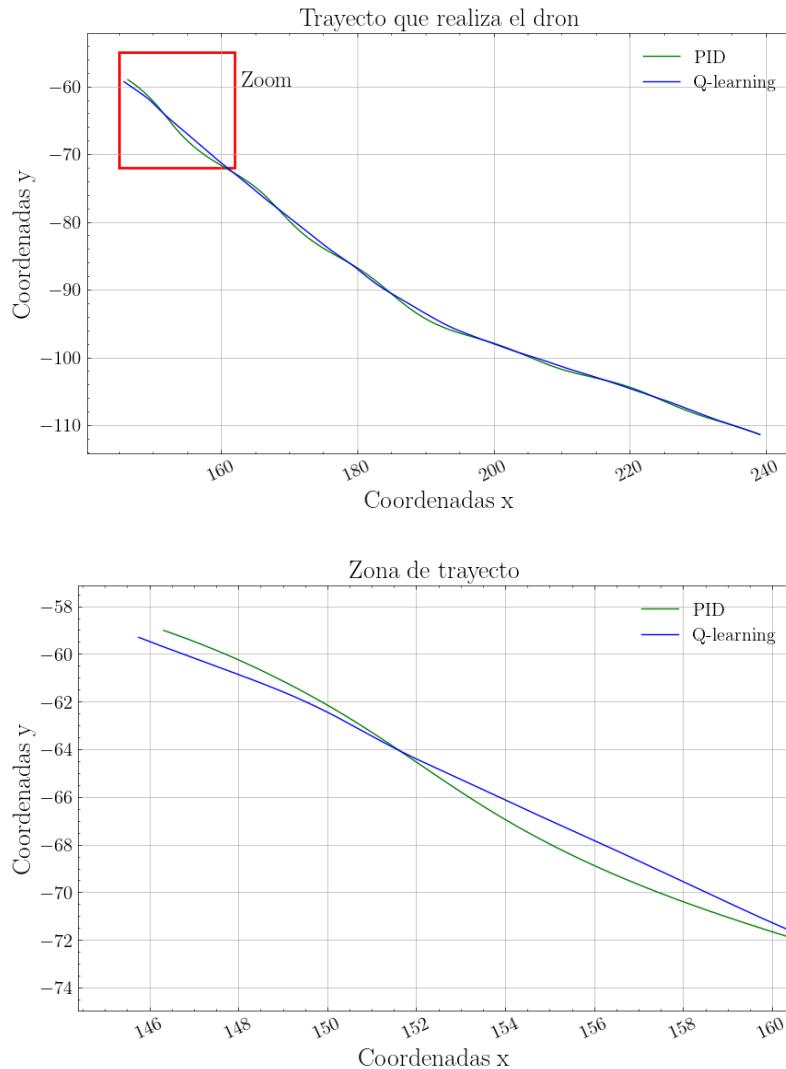


Figura 4.26: Comparativa de trayectorias entre el control clásico y el control de aprendizaje por refuerzo

Como se observa en esta comparativa, una de las principales diferencias entre ambos comportamientos es que el control clásico basado en el PID presenta oscilaciones durante el recorrido realizando eses consiguiendo una velocidad angular media alrededor de 0.9 grados/segundos hasta llegar al punto final.

En cambio, el comportamiento con aprendizaje por refuerzo destaca por su trayectoria constante durante todo el recorrido sin apenas realizar oscilaciones obteniendo una velocidad angular media de 0.1 grados/segundos siendo menor que el resultado del

controlador PID. Esto se debe a que el controlador PID constantemente tiene que estar realizando ajustes de velocidades angulares para mantenerse centrado y dentro del carril, sin embargo con comportamiento de aprendizaje por refuerzo es capaz de aprender y adaptarse con una velocidad angular baja realizando giros menos bruscos.

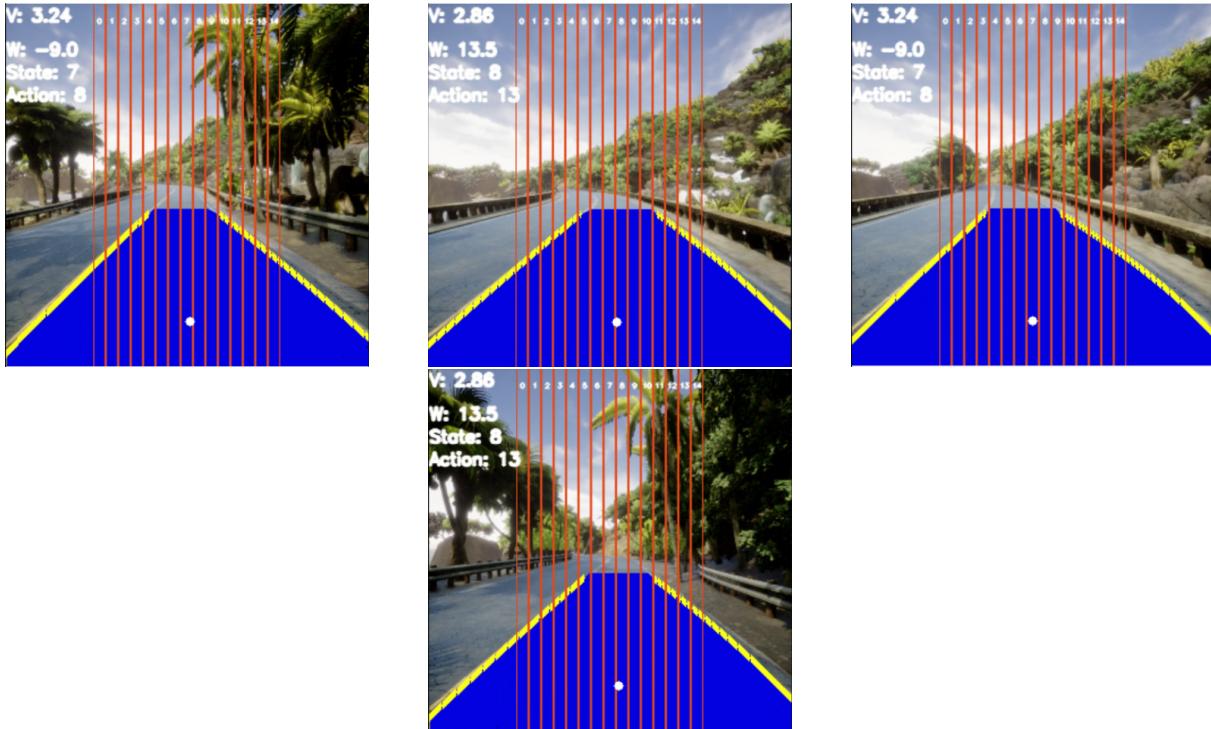


Figura 4.27: Resultado del comportamiento de aprendizaje por refuerzo

Demostrando así que el modelo entrenado es capaz de obtener comportamientos satisfactorios tomando acciones acordes a diferentes escenarios. En la figura 4.27, se muestra una secuencia de frames recogidas en este [vídeo](#) del comportamiento de aprendizaje por refuerzo durante el recorrido siendo capaz de mantenerse centrado al carril, su comportamiento se debe a que las acciones son discretas. Es importante recordar que el objetivo de usar el algoritmo de Q-learning es su capacidad de generalización a distintos escenarios.

En este otro experimento, se realiza un análisis entre el PID y Q-learning, en el cual ambos comportamientos tienen los mismos rangos de velocidades mínimas y máximas y demostrando que el controlador PID no es capaz de mantenerse dentro del carril desembocando que se salga del carril. Se utiliza un escenario en el que ambos comportamientos visitan por primera vez, este escenario se puede visualizar en la figura 4.28 junto con la vista del dron. Se trata de un escenario en donde presenta curvas hacia la derecha e izquierda.



Figura 4.28: Escenario II de control tradicional y aprendizaje por refuerzo

En este análisis, se evalúa trayectoria que han seguido ambos comportamientos como se muestra en la figura 4.29.

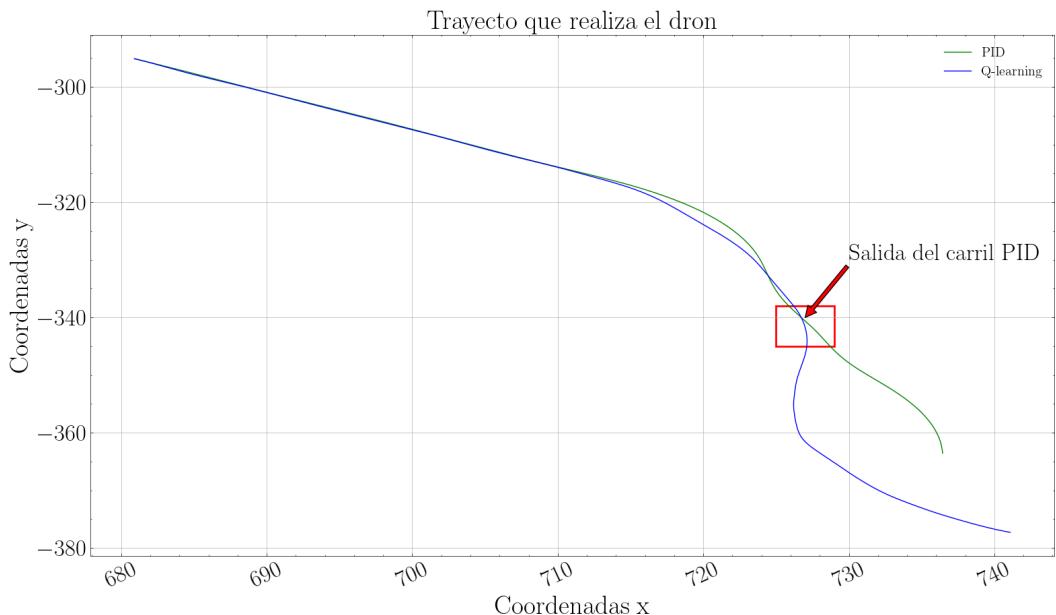


Figura 4.29: Trayecto del comportamiento PID y Q-learning

Al comienzo del recorrido, tanto el controlador PID como Q-learning son capaces de mantener una trayectoria suave y centrada respecto al centro del carril. Esto indica que ambos métodos pueden manejar bien las condiciones iniciales del entorno. En cambio, cuando el dron se enfrenta a la primera curva, comienza a surgir diferencias notables. El controlador PID empieza a oscilar, tratando de corregir su posición para mantenerse dentro del carril. Estas oscilaciones son típicas de un sistema PID tratando de adaptarse rápidamente a cambios en la trayectoria. En contraste, el algoritmo Q-learning mantiene una trayectoria acorde a la curva sin realizar ninguna oscilación significativa, mostrando una mayor estabilidad. Debido a las oscilaciones del PID, el dron llega a un punto donde no puede mantenerse dentro del carril, saliéndose de la trayectoria. Esto se destaca en la gráfica, donde se marca la **"Salida del carril PID"**. Por otro lado, el comportamiento de

Q-learning sigue la trayectoria de forma consistente y centrada, completando el recorrido sin salirse del carril.

Esta comparativa de las trayectorias revela que el algoritmo de Q-learning ofrece mejores resultados en cuanto al controlador PID. Mientras que el PID sufre oscilaciones y se sale de la trayectoria, Q-learning demuestra una capacidad para adaptarse al recorrido, completándolo de manera eficiente y manteniéndose dentro del carril.

En la figura 4.30 se muestran varios frames recogidos en este [vídeo](#) de ambos comportamientos durante la trayectoria mostrando como Q-learning mantiene una trayectoria robusta y centrada en comparación con el PID. En particular, en la tercera columna se observa cómo el controlador PID se desvía su trayectoria, mientras que Q-learning sigue su trayecto con éxito. Estos resultados subrayan la importancia de utilizar métodos de control adaptativos que tienen la capacidad de generalizar y ser flexibles como Q-learning para aplicaciones de navegación autónomas con drones en carreteras, donde los métodos tradicionales pueden no ser suficientes para garantizar un rendimiento fiable.

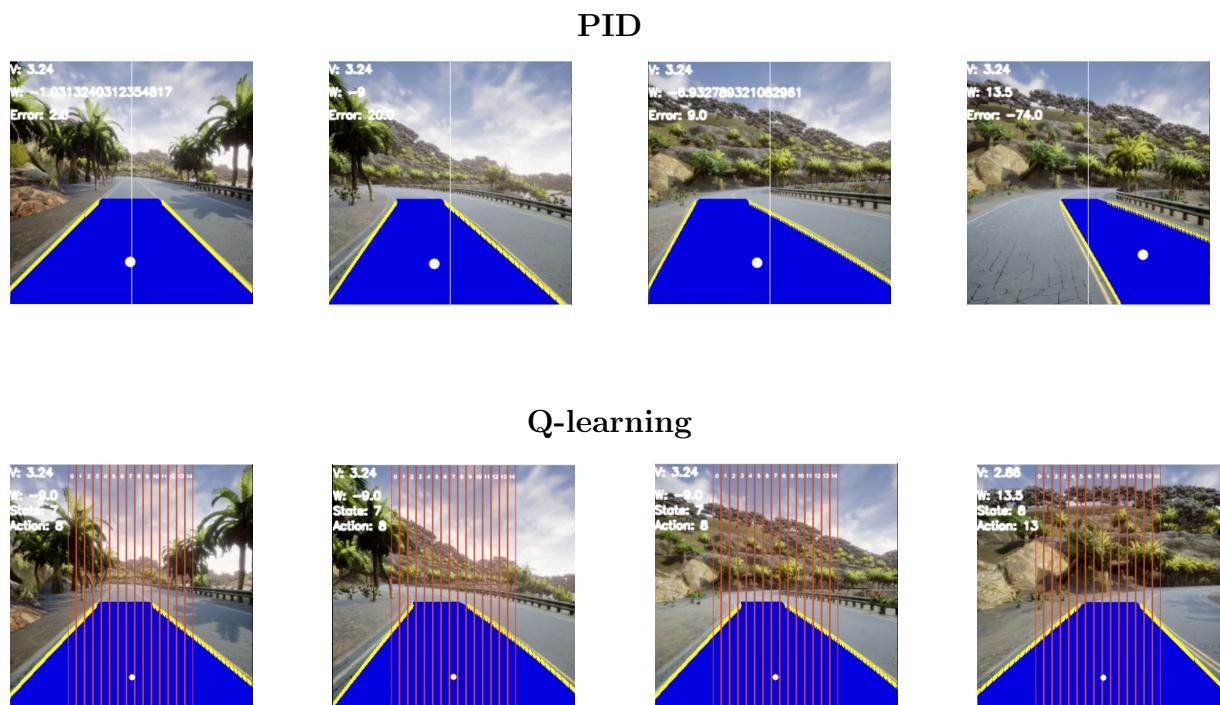


Figura 4.30: Comparativa entre el PID y Q-learning

Capítulo 5

Conclusiones

En este TFG se han cumplido los objetivos marcados durante el desarrollo de él mismo. Primero la creación de algoritmos de navegación autónoma para drones para resolver la problemática de seguimiento de carril en entornos de carreteras urbanas. En segundo lugar la utilización de algoritmos basados en inteligencia artificial y aprendizaje no supervisado con el fin de evaluar su efectividad. Tercero, analizar el desarrollo de aplicaciones de navegación autónoma drones utilizando el simulador fotorrealista Airsim junto con el middleware robóticos ROS empleando una arquitectura de comunicación distribuida.

En conclusión, se comenta los objetivos que se han cumplido a lo largo del desarrollo como los requisitos cumplidos que se expusieron en el capítulo 2, englobando líneas futuras dentro del trabajo.

5.1. Objetivos cumplidos

Los objetivos presentados en la sección 2.1, han sido cumplidos exitosamente

1. Se logró la instalación y configuración del simulador Airsim junto con ROS, estableciendo la comunicación efectiva entre dos equipos mediante protocolos de red.
2. Se implementó satisfactoriamente un sistema perceptivo capaz de realizar una detección del carril utilizando redes neuronales junto con algoritmos de aprendizaje automático.
3. Se implementó satisfactoriamente un sistema de control utilizando técnicas de aprendizaje por refuerzo, en especial Q-learning para lograr el seguimiento del carril del dron de manera robusta y eficaz.
4. Se completó de manera exitosa la aplicación para el seguimiento del carril para drones.

5. Se han completado análisis efectivos para cada uno de los comportamientos desarrollados contrastando sus métricas de forma exitosa tanto en el sistema perceptivo como de control.

5.2. Requisitos satisfechos

En la sección 2.2 se presentaron los requisitos que ha tenido este TFG, que se han ido resolviendo de la siguiente forma:

1. Durante todo el proceso del trabajo se utiliza Airsim junto con UnrealEngine como entorno de simulador.
2. Los comportamientos desarrollados durante el trabajo se ha utilizado la estructura de ROS junto con la comunicación del entorno de simulación Airsim.
3. La navegación autónoma basada en aprendizaje por refuerzo ha demostrado ser lo suficiente robusta y eficiente en los diferentes trayectos dentro del escenario Coastline.
4. Uso del algoritmo de Q-Learning para desarrollar el comportamiento sigue carril y de carreteras basados en aprendizaje por refuerzo.

5.3. Balance global y competencias adquiridas

Durante el desarrollo de este TFG, el desafío de implementar una aplicación de navegación autónoma de drones basada en aprendizaje por refuerzo y uso de redes neuronales resultó ser un proceso complejo y enriquecedor. Además de que la combinación de aprendizaje por refuerzo e inteligencia artificial es una estrategia viable y efectiva para la navegación autónoma de drones. Los resultados obtenidos son prometedores y sugieren que, con más investigación y desarrollo, estos sistemas pueden ofrecer soluciones robustas y eficientes para la navegación autónoma en una variedad de entornos sumando la arquitectura de conexión que se llegó a implementar para ello.

Al comienzo de este trabajo apenas tenía suficientes conocimientos básicos sobre inteligencia artificial enfocada en redes neuronales y aprendizaje por refuerzo. Destacando que ha sido mi primer trabajo de investigación en donde he aprendido múltiples conceptos, así como adquirir técnicas de análisis de errores que han ido apareciendo durante todo el

proceso. Destacando las siguientes competencias:

- Organización en cuanto a tiempos y tareas utilizando la metodología Kanban.
- Nuevos conocimientos en la utilización de simuladores robóticos.
- Nuevos conocimientos dentro del área de inteligencia artificial, así como redes neuronales y algoritmos de aprendizaje no supervisado.
- Nuevos conocimientos desde cero sobre comportamientos basados en aprendizaje por refuerzo.
- Capacidad de analizar diferentes resultados recogidos en los diferentes comportamientos implementados.
- Mejora en la capacidad de implementar una arquitectura distribuida entre ambos equipos.
- Nuevos conocimientos sobre la integración de ROS junto con aplicaciones externas.
- Capacidad de documentación a la hora de desarrollar los diferentes comportamientos.

5.4. Líneas futuras

Aunque hemos obtenido resultados exitosos y satisfactorios a lo largo del TFG, existen diferentes líneas futuras que se podrían desarrollar a partir de este trabajo.

- Utilizar la red neuronal usada en el trabajo y volver a entrenar dicha red en el entorno de trabajo teniendo en cuenta la visión de un dron.
- Búsqueda de algoritmos alternativos que produzcan mejores resultados a la hora de refinar resultados de redes neuronales de segmentación.
- Búsqueda sobre la infraestructura que existe entre el simulador PX4 Autopilot y Airsim, para tener mejores comportamientos en cuanto a rendimiento y latencia.
- Mejorar los sistemas de seguimiento de carril de aprendizaje por refuerzo sin las propias limitaciones que puede presentar el algoritmo de Q-learning, usando algoritmos de Deep Reinforcement Learning.

- Probar el comportamiento en diferentes escenarios distintos, con circuitos con curvas más cerradas, con situaciones de tiempo adversas.

Capítulo 6

Anexo

A continuación se muestra las diferentes referencias a las figuras que hemos visto a lo largo de este trabajo junto con el enlace de donde ha sido obtenida. Las imágenes que no incluidas en este capítulo han sido formadas en el desarrollo de este trabajo provienen del mismo:

Referencia de las imágenes	Enlaces de donde se ha obtenido
1.2	https://www.bbc.com/mundo/noticias-57317022
1.3	https://www.whoi.edu/oceanrobots/robots/nereus-phone.html
1.4	https://www.swissinfo.ch/spa/ciencia/drones-suizos-al-rescate/46203902

	https://www.smithsonianmag.com/arts-culture/unmanned-drones-have-been-around-since-world-war-i-16055939/ https://web.happystays.com/?m=file-winston-churchill-and-the-secretary-of-state-for-tt-YQ3jGVI4 https://en.wikipedia.org/wiki/V-1_flying_bomb https://www.google.com/url?sa=i&url=https%3A%2F%2Fthefrontlines.com%2Fstory%2Fww2-project-aphrodite%2F&psig=A0vVaw20cBlgMDH1HVU5qsiJ9_Fg&ust=1714151925046000&source=images&cd=vfe&opi=89978449&ved=OCBQQjhxqFwoTCNjGs9bv3YUDFQAAAAAdAAAAABAE https://en.wikipedia.org/wiki/Ryan_Firebee https://en.wikipedia.org/wiki/Lockheed_D-21 https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.researchgate.net%2Ffigure%2FBoeing-Condor-UAV-23_fig10_261209014&psig=A0vVaw1q3J6eh2YCyEUzy1QM9z_K&ust=1714152091161000&source=images&cd=vfe&opi=89978449&ved=OCBQQjhxqFwoTCJDAyqXw3YUDFQAAAAAdAAAAABAE https://www.timesofisrael.com/idf-launches-probe-after-two-more-mini-drones-crash/ https://www.google.com/url?sa=i&url=https%3A%2F%2Fnews.usni.org%2F2020%2F09%2F15%2Fmarines-placing-small-uavs-into-ground-combat-element-as-aviators-still-refining-large-uas-requirement&psig=A0vVaw2csv7vma6UxJokGv1G8j7h&ust=1714152048517000&source=images&cd=vfe&opi=89978449&ved=OCBQQjhxqFwoTCJDAyqXw3YUDFQAAAAAdAAAAABAE
1.5	https://www.xataka.com/espacio/helicoptero-ingenuity-ha-terrizado-lugares-marte-que-nasa-se-esta-quedando-letras-para-nombrarlos
1.6	https://www.elcorreogallego.es/hemeroteca/union-fenosa-distribucion-implanta-uso-drones-supervisar-sus-lineas-alta-tension-galicia-MQCG1024080
1.8	https://www.xataka.com/drones/asi-es-el-dron-repartidor-de-amazon-todavia-poco-mas-que-humo-que-promete-entregar-paquetes-en-media-hora
1.9	https://www.techtimes.com/articles/285562/20221228/amazon-begins-prime-air-drone-deliveries-california-texas.htm
1.10	https://emprendedores.es/marketing-y-ventas/ecommerce-marketing-y-ventas/drones-amazon-europa/

1.11	https://www.researchgate.net/publication/273392596_Efficient_Road_Detection_and_Tracking_for_Unmanned_Aerial_Vehicle
1.12	https://www.researchgate.net/publication/378676909_The_prospect_of_artificial_intelligence_to_personalize_assisted_reproductive_technology
1.13	https://revistas.rcaap.pt/rca/article/view/34973/24651
1.14	https://becominghuman.ai/the-very-basics-of-reinforcement-learning-154f28a79071
2.1	https://www.lifeder.com/modelo-espiral/
2.2	https://github.com/RoboticsLabURJC/2022-tfg-barbara-villalba/graphs/contributors
3.1	https://pytorch.org/hub/hustvl_yolop/
3.3	https://www.ros.org/imgs/ros-equation.png
3.4	https://medium.com/@robtech.impaciente/ros-robot-operating-system-fundamentos-e92478c26e02
3.5	https://404warehouse.net/2015/12/20/autopilot-offboard-control-using-mavros-package-on-ros/
3.6	https://img-blog.csdnimg.cn/272026cef41047cdb7e523fb9a28e173.png?x-oss-process=image/watermark,type_d3F5LXplbmhlaQ,shadow_50,text_Q1NETiBAamluYXV0bw==,size_20,color_FFFFFF,t_70,g_se,x_16 https://www.scrimmagesim.org/sphinx/html/_images/Asset_LandscapeMountains_1.png https://www.researchgate.net/figure/Appearance-of-the-maps-for-training-a-City-environment-b-Coastline-c_fig7_359436337 https://www.scrimmagesim.org/sphinx/html/_images/city_airsim_view.png https://github.com/Microsoft/AirSim/wiki/moveOnPath-demo
3.7	https://docs.px4.io/v1.14/en/flight_modes_mc/position.html
3.8	https://flathub.org/es/apps/org.mavlink.qgroundcontrol
4.2	https://fossbytes.com/dell-xps-13-developer-edition-linux-laptop-ubuntu-20-04/ https://www.flaticon.es/icono-gratis/router-de-wifi_3096440 https://www.pngwing.com/es/free-png-twcfq

Apéndice A

Bibliografía

- [1] (2017). Dbscan:density-based spatial clustering of applications with noise. *Scikit learn*.
- [2] (2023). Distributions of ros. *ROS.org*.
- [Akbar] Akbar, R. Onnx runtime: The only deployment framework you will ever need.
- [4] Alavarez, D. A. R. (2016). The condor uav system.
- [5] Ausin, M. S. (2020). Introducción al aprendizaje por refuerzo. parte 2: Q-learning. *Medium*.
- [6] Bealdung (2023). Política epsilon greedy. *Bealdung*.
- [7] Das, A. (2017). The very basics of reinforcement learning. *Becoming Human: Artificial Intelligence Magazine*. This article provides an introduction to the fundamental concepts of reinforcement learning, laying the groundwork for understanding more advanced topics.
- [8] Dong, W., Man-Wen, L., Wei-Tian, Z., Xing-Gang, W., Xiang, B., Wen-Qing, C., and Wen-Yu, L. (2012). Yolop: You only look once for panoptic driving perception. *Machine Intelligence Research*, 19:253–266.
- [9] Frisbee, J. L. (1997). Proyect aphrodite.
- [10] Gustavo Mesías-Ruiz, J. P., Ana de Castro, I. B.-S., and Dorado, J. (2024). Detección y clasificación de malas hierbas mediante drones y redes neuronales profundas: creación de mapas para tratamiento localizado. pages 1–5.
- [11] Hanassab, S., Abbara, A., Yeung, A., Voliotis, M., Tsaneva-Atanasova, K., Kelsey, T., Trew, G., Nelson, S., Heinis, T., and Dhillo, W. (2024). The prospect of artificial intelligence to personalize assisted reproductive technology. *npj Digital Medicine*, 7.
- [12] Jung, S. and Kim, H. (2017). Analysis of amazon prime air uav delivery service. *Journal of Knowledge Information Technology and Systems*, 12:253–266.

- [13] Khater, I., Nabi, I., and Hamarneh, G. (2020). A review of super-resolution single-molecule localization microscopy cluster analysis and quantification methods. *Patterns*, 1:100038.
- [14] Krejci Garzon, E. (2014). Drones el futuro de hoy. *ashtag*, pages 96–103.
- [15] Loja Romero, J. D. (2022). Exploración autónoma en interiores para el robot spot basado en la red yolo. No Publicado.
- [16] Qiang, W. and Zhongli, Z. (2011). Reinforcement learning model, algorithms and its application. In *2011 International Conference on Mechatronic Science, Electric Engineering and Computer (MEC)*, pages 1143–1146.
- [17] Tsang, S.-H. (2019). Deeplabv3+-atrous separable convolution(semantic segmentation). *Medium*.
- [18] Ultralytics (2021a). *GitHub*. Find the source code used in the YOLOv8 model as well as a quick-start guide to help you start using YOLOv8.
- [19] Ultralytics (2021b). Ultralytics/ultralytics: New - yolov8 in pytorch. *GitHub*. Find the source code used in the YOLOv8 model as well as a quick-start guide to help you start using YOLOv8.
- [20] Xue, Z. and Gonsalves, T. (2021). Vision based drone obstacle avoidance by deep reinforcement learning. 2:366–380.
- [21] Yu, F., Chen, H., Wang, X., Xian, W., Chen, Y., Liu, F., Madhavan, V., and Darrell, T. (2018). Bdd100k: A diverse driving dataset for heterogeneous multitask learning.
- [22] Zhou, H., Kong, H., Wei, L., Creighton, D., and Nahavandi, S. (2015). Efficient road detection and tracking for unmanned aerial vehicle. *Intelligent Transportation Systems, IEEE Transactions on*, 16:297–309.
- [23] Zhu, Y. and Tang, H. (2023). Automatic damage detection and diagnosis for hydraulic structures using drones and artificial intelligence techniques. *Remote Sensing*, 15(3).