



GRADO EN INGENIERÍA DE ROBÓTICA SOFTWARE

Escuela de Ingeniería de Fuenlabrada

Curso académico 2023-2024

Trabajo Fin de Grado

Navegación Autónoma de drones basado en
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 licenciatante.
- *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 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 telefonicas 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

Dentro del mundo de la robótica, la navegación autónoma emerge una de las áreas más emocionantes especialmente la navegación autónoma de drones con sistemas de inteligencia artificial. Esto permite que los vehículos aéreos no tripulados no solo ejecuten tareas preprogramadas, sino que también sean capaces de aprender la adaptación de entornos dinámicos y cambiantes. Los drones se han convertido en un gran desafío en el mundo de la robótica aérea. Pueden ser programados para realizar tareas específicas, como mapear terrenos en aplicaciones cartográficas o entregar suministros médicos en zonas de difícil acceso. Su versatilidad y capacidad de operar de manera autónoma convierte a estos robots aéreos en herramientas valiosas en diversas aplicaciones.

Además de la navegación autónoma, la inteligencia artificial permite a los drones, por ejemplo, ser entrenados para reconocer patrones y objetos en su entorno permitiéndoles realizar tareas como la identificación de personas en situaciones de búsqueda y rescate o la detección de anomalías en infraestructuras con algoritmos de aprendizaje automático. Sin embargo, a pesar de estos avances, la navegación autónoma e inteligencia artificial en drones sigue siendo un área de investigación debido a la necesidad de algoritmos de aprendizaje más robustos que puedan cumplir con éxito en un futuro próximo.

Con este Trabajo de Fin de Grado se demuestra que la navegación autónoma de drones es capaz de tener un comportamiento autónomo en entornos realistas y complejos de carreteras tomando decisiones en tiempo real para alcanzar sus objetivos de manera eficiente y segura. Para poder lograr esto, se explorarán y se implementarán técnicas de algoritmos de aprendizaje automático y de inteligencia artificial con un enfoque en particular el aprendizaje por refuerzo utilizando entornos de simulación de Airsim.

Acrónimos

UAV *Unmanned Air Vehicle*

UAS *Unmanned Air System*

IA *Inteligencia Artificial*

Airsim *Aerial Informatics and Robotics Simulation*

TFG *Trabajo de fin de Grado*

ROS *Robotic Operative System*

Mavros *MAVlink to ROS Interface*

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	7
1.3. La inteligencia artificial en la navegación autónoma de drones	16
1.4. Navegación autónoma en Airsim basada en inteligencia artificial y aprendizaje por refuerzo	19
2. Objetivos	20
2.1. Descripción del problema	20
2.2. Requisitos	21
2.3. Metodología	21
2.4. Plan de trabajo	24
3. Plataforma de desarrollo	26
3.1. Lenguaje de programación	26
3.1.1. Python	26
3.2. YOLOP	30
3.3. ROS	34
3.3.1. Mavros	36
3.4. Airsim	37
3.4.1. Airsim ROS Wrapper	41
3.4.2. Client Airsim	42
3.5. PX4 AutoPilot	42
3.5.1. Software in The Loop(SITL)	42
3.5.2. Modos de vuelo	43
3.6. QGroundControl	45

4. Diseño	46
4.1. Arquitectura	47
4.1.1. Configuración de Integración Distribuida en equipos separados .	48
4.2. Preparación del entorno de simulación	51
4.2.1. Configuración del dron y del entorno	51
4.2.2. Instalación de las herramientas de desarrollo	54
4.3. Percepción	55
4.3.1. Inferencia de YOLOP	56
4.3.2. DBSCAN	61
4.3.3. Clasificación de clústeres	64
4.3.4. Regresión cuadrática	66
4.3.5. Interpolación y cálculo del centro de masas del carril	69
4.3.6. Análisis del algoritmo de percepción	71
4.4. Seguimiento del carril mediante control clásico	72
4.5. Seguimiento del carril mediante aprendizaje por refuerzo	73
4.5.1. Fase de entrenamiento	79
4.5.2. Fase de inferencia	81
4.5.3. Análisis y comparativa entre el seguimiento de carril clásico	83
5. Conclusiones	86
5.1. Conclusiones	86
5.2. Corrector ortográfico	87
6. Anexo	1
A. Bibliografía	5
Bibliografía	5

Índice de figuras

1.1.	Definición de robot	2
1.2.	Sojourner Rover	4
1.3.	Nereus	5
1.4.	El dron de rescate Rega	7
1.5.	Historia de los drones	9
1.6.	El dron Ingenuity	10
1.7.	Drones en inspección eléctrica en Galicia	11
1.8.	El primer prototipo de dron de Prime Air	12
1.9.	El dron MK27-2	12
1.10.	El dron MK30	13
1.11.	Resultados de detección y seguimiento de carreteras en Efficient Road Detection and Tracking for Unmanned Aerial [21]	14
1.12.	Demostración de la identificación de diferentes tipos de grietas [17] . . .	15
1.13.	Clasificación de Inteligencia Artificial [11]	16
1.14.	Resultados de la detección y clasificación de malas hierbas en un cultivo [10]	17
1.15.	Esquema de Reinforcement Learning [7]	18
2.1.	Ilustración del tablero Kanban durante el desarrollo del TFG	23
2.2.	Seguimiento de trabajo en GitHub	24
3.1.	Arquitectura de YOLOP	30
3.2.	Resultados de la salida de la red neuronal YOLOP[8]	33
3.3.	Definición de ROS	34
3.4.	Arquitectura de ROS	35
3.5.	Infraestructura de Mavros	36
3.6.	Ejemplos de escenarios en Airsim	39
3.7.	Diagrama del comportamiento del modo de vuelo Position	44
3.8.	QGroundControl	45

4.1.	Arquitectura de los componentes utilizados en el TFG	48
4.2.	Diagrama de comunicaciones entre PX4, QGC y Airsim	49
4.3.	Salida de la terminal ejecutando PX4 SITL	50
4.4.	Diagrama de comunicaciones	51
4.5.	Visualización del entorno original ilustrando los objetos dificultando la percepción de la carretera	52
4.6.	Logotipo de ROS noetic	54
4.7.	Resultados de los pesos preentrenados del modelo YOLOP	60
4.8.	Diferentes resultados utilizando la red neuronal YOLOP en el entorno de simulación	61
4.9.	Ejemplo ilustrativo de como funciona el algoritmo de DBSCAN [13] . .	62
4.10.	Diferentes resultados utilizando el algoritmo de DBSCAN	64
4.11.	Ilustración del proceso de clasificación de líneas detectadas respecto al carril que queremos seguir	66
4.12.	Resultado de la regresión cuadrática	69
4.13.	Resultado de la interpolación	70
4.14.	Resultado del centro de masas	71
4.15.	Profiling de las partes de la percepción	72
4.16.	Entorno en la fase de entrenamiento del sigue carril basado en Q-Learning	74
4.17.	Estados definidos para el sigue carril basado en Q-Learning	75
4.18.	Gráficas de la fase de entrenamiento	80
4.19.	Distribucción de acciones en el circuito de entrenamiento	81
4.20.	Distribucción de acciones en el circuito de entrenamiento multiplicadas por el factor	83
4.21.	Comparativa realizando un trayecto entre ambos comportamientos . .	84
4.22.	Media de velocidades angulares de ambos comportamientos desarrollados	85

Listado de códigos

3.1. Ejemplo de código en Python de una función para calcular el factorial de un número	27
3.2. Ejemplo de código en Python de operaciones básicas utilizando la librería OpenCv	28
3.3. Ejemplo de código en Python de operaciones básicas utilizando la librería Numpy	28
3.4. Cargar modelo YOLOP con pesos preentrenados End-to-end.pth	32
4.1. Configuración del vehículo mediante el fichero settings.json	53
4.2. gitclone	55
4.3. bash	55
4.4. comando	55
4.5. Cargar modelo YOLOP con pesos preentrenados End-to-end.pth	56
4.6. Cargar modelo YOLOP escogiendo como dispositivo la GPU	56
4.7. Inferencia del modelo	57
4.8. Resultado de la inferencia del modelo YOLOP	58
4.9. Cargar modelo	58
4.10. Inferencia del modelo yolop-320-320.onnx	59
4.11. Algoritmo de clustering utilizando DBSCAN	63
4.12. Clasificación de clústeres según las dimensiones de la imagen	65
4.13. Función maximizada para escoger el grupo de cluster más cercano y denso respecto al punto P	65
4.14. Función del cálculo de la regresión cuadrática	68
4.15. Método de interpolación	70
4.16. Valores de las variables del PD del control de altura y del PID del controlador de velocidad angular	74
4.17. Construcción de las acciones para Q-Learning	76
4.18. Función de recompensa	77
4.19. Factores	82

Listado de ecuaciones

Índice de cuadros

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 que requerían una gran cantidad de energía y mantenimiento. 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. 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, 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

Como mencionamos anteriormente, 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.

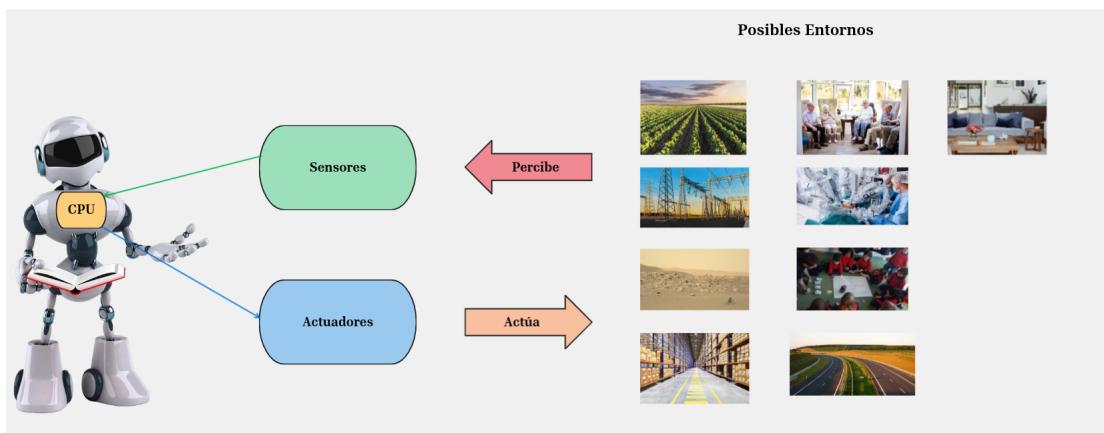


Figura 1.1: Definición de robot.

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 detención de objetos o la manipulación de objetos con sensores táctiles y de fuerza, dichas tareas 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

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

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 ser humano ha utilizado una variedad de herramientas para ampliar su capacidad de manipulación como palos utilizados para caer la fruta madura de un arbol. 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 estado evolucionando hasta el punto de incluir sistemas robóticos robustos que pueden ser controlados a distancia, permitiendo al operario poder realizar tareas en entornos peligrosos e innaccesibles 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 puede ser la exploración espacial, en donde se utiliza la teleoperación como técnica de manipulación remota como el Sojourner Rover. Como se muestra en la figura 1.2, El Sojourner Rover² es un pequeño robot móvil compuesto por 6 ruedas creado por los científicos de la NASA para estudiar la superficie de Marte con la capacidad de enviar imágenes en directo y realizar análisis del terreno del planeta. Gracias a sus ruedas podía moverse por terrenos rocosos y de difícil acceso ya que estaban equipadas materiales como de aluminio y acero inoxidable.

²<https://www.astronomy.com/space-exploration/sojourner-nasas-first-mars-rover/>

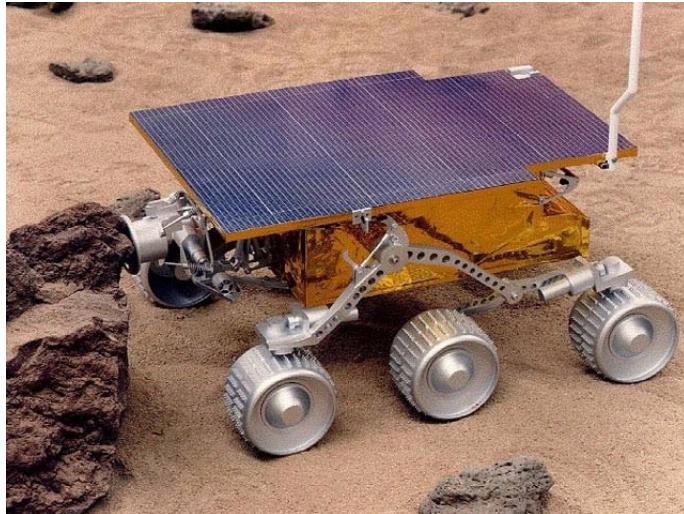


Figura 1.2: Sojourner Rover

Con esta misión espacial se pudo probar como era el entorno marciano con técnicas realizadas en los laboratorios de la NASA demostrando que se podía realizar una teleoperación en el espacio abriendo el camino a futuros rovers como el Spirit, Opportunity y más³.

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. Otro tipo de limitación puede ser la carga cognitiva que puede tener el operario 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 semiautónomo 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

³<https://spaceplace.nasa.gov/mars-spirit-opportunity/sp/>

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 ilustra en la figura 1.3. El Nereus⁴ es un vehículo submarino semiautónomo que puede ser manejado por control remoto que entró en servicio en el año 2009, su propósito fue explorar la Fosa de las Marianas, específicamente el Abismo Challenger (es el punto más profundo conocido en los océanos). Fue manejado mediante control remoto por pilotos que se encontraban en un barco en la superficie aunque el Nereus también podía cambiar al modo de vehículo autónomo pudiendo navegar libremente adaptándose a las condiciones del entorno sin intervención humana directa.

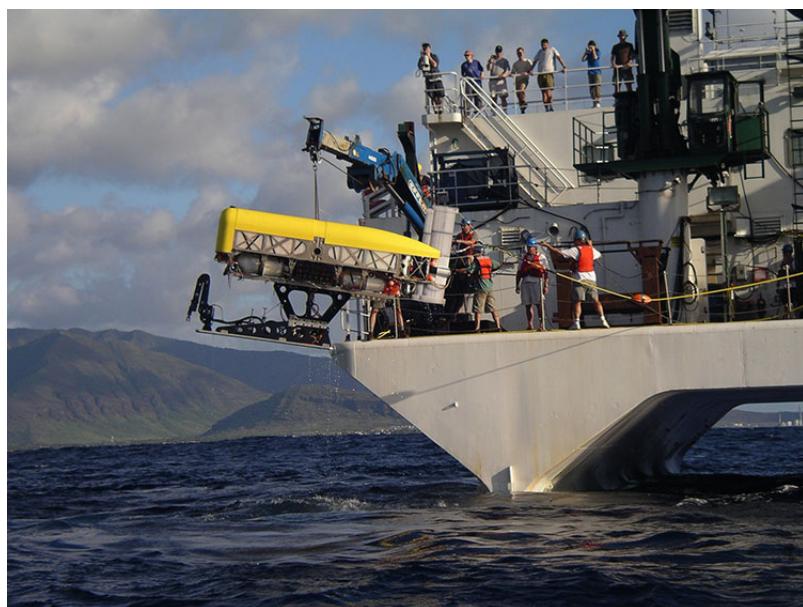


Figura 1.3: Nereus

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 pérdida, los datos que se pudieron recopilar en este robot submarino siguen siendo una fuente de conocimiento sobre las profundidades marinas. Este ejemplo de robot semiautónomo demuestra que

⁴https://www.bbc.com/mundo/ciencia_tecnologia/2009/06/090603_1541_nereus_robot_mar_mr

se pueden realizar tareas en entornos peligrosos sin poner en riesgo la vida humana aunque tenga control por un operario⁵.

En cuanto a las debilidades de la robótica semiautonómica, 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 semiautónomos, 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 semi-autónomos 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 automía 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 suizos⁶. 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 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.

⁵<https://www.elperiodico.com/es/ciencia/20140512/famoso-sumergible-nereus-pierde-fondo-mar-3271389>

⁶<https://www.swissinfo.ch/spa/ciencia/drones-suizos-al-rescate/46203902>



Figura 1.4: El dron de rescate Rega

1.2. Robótica aérea

Dentro del campo de la robótica aérea tenemos los drones. Podemos definir un 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 autonomamente incoporando software en su sistema. 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)⁷ era propulsado por un motor alternativo. Podía volar de forma autónoma hasta un punto específico, donde soltaba sus alas y caía en “caída libre”⁸. Avanzando en la historia, en 1935 se desarrolló el DH.82 Queen Bee⁹. Éste era un blanco aéreo sin piloto que era controlado por radio. De hecho, parece que el término “dron” se originó a partir del nombre, que se refiere a la abeja macho que realiza un vuelo en busca de la abeja reina y luego fallece.

⁷<https://www.nationalmuseum.af.mil/Visit/Museum-Exhibits/Fact-Sheets/Display/Article/198095/kettering-aerial-torpedo-bug/>

⁸<https://daytonunknown.com/2023/06/30/the-kettering-bug-the-worlds-first-drone/>

⁹<https://dronewars.net/2014/10/06/rise-of-the-reapers-a-brief-history-of-drones/>

Durante la Segunda Guerra Mundial, quizás el más conocido fue el V-1 "Flying Bomb"¹⁰ , el primer misil de crucero operativo del mundo, en donde su sistema de guía pre establecido incluía una brújula magnética que monitoreaba un autopiloto con giroscopios. También en este periodo, destacaremos el *Proyect 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.

Destacando más UAVs, tenemos la familia Teledyne Ryan Firebee/Firefly¹¹, 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 impulsado por estatorreactor con velocidades mayores que Mach 3¹² . En la Edad Moderna, destacamos El Condor [4], fue el primer UAS en utilizar navegación GPS y tecnología de aterrizaje automático y el Predactor¹³. 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 .

¹⁰<https://migflug.com/jetflights/the-v1-flying-bomb/>

¹¹<https://www.designation-systems.net/dusrm/m-34.html>

¹²<https://www.marchfield.org/aircraft/unmanned/d-21-drone-lockheed/>

¹³<https://www.airforce-technology.com/projects/predator-uav/?cf-view>



Figura 1.5: Historia de los drones

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 marciano¹⁴. 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 era demostrar la viabilidad del vuelo en la atmósfera de Marte, ya que su atmósfera está compuesta por el 1% de la densidad terrestre dificultando el vuelo del dron. Sin embargo, gracias a su diseño ligero y a sus

¹⁴<https://ciencia.nasa.gov/sistema-solar/finaliza-la-mision-del-helicoptero-ingenuity-en-marte/>

hélices especialmente diseñadas para crear suficiente sustentación en la atmósfera del planeta, el Ingenuity fue capaz de superar este desafío¹⁵.



Figura 1.6: El dron Ingenuity

Además, en su última fase, el Ingenuity realizó pruebas de vuelo experimentales para ampliar el conocimiento sobre cuáles eran sus límites aerodinámicos¹⁶.

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 distribución 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

¹⁵<https://www.bbc.com/mundo/noticias-56738201>

¹⁶<https://science.nasa.gov/mission/mars-2020-perseverance/ingenuity-mars-helicopter/>

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 repararlos¹⁷.



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.

¹⁷<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 MK27-2¹⁸. 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

¹⁸<https://www.europapress.es/portaltic/gadgets/noticia-amazon-prime-air-comienza-entregar-pedidos-drones-estados-unidos-20221229115034.html>

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 vencidarios.

Se tiene previsto que se llegue a probar en el año 2024 empezando por ciudades como Texas y California en Estados Unidos ¹⁹.



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* [21] que tiene como objetivo desarrollar un algoritmo de detención y seguimiento de carreteras específicas en videos capturados por vehículos aéreos no tripulados (UAV). Para la realizar la detención de carreteras utilizan un algoritmo denominado Graph-Cut²⁰, 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 GMM²¹ (Gaussian Mixture Model) para modelar las características de la imagen y representar regiones o clases en

¹⁹<https://www.forbesargentina.com/innovacion/asi-nuevo-asombroso-dron-amazon-mk30-n42612>

²⁰<https://www.sciencedirect.com/topics/engineering/graph-cut-technique>

²¹<https://builtin.com/articles/gaussian-mixture-model>

la imagen (por ejemplo, carretera, fondo, vehículos).

Una vez se realice la identificación de la carretera, se utilizará 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.

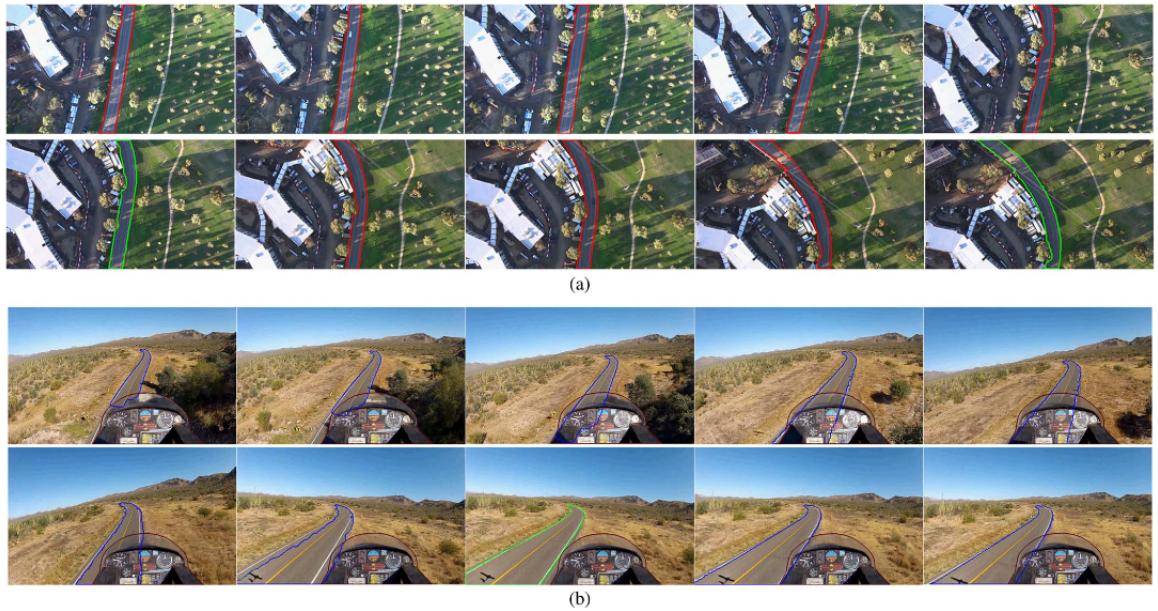


Figura 1.11: Resultados de detección y seguimiento de carreteras en Efficient Road Detection and Tracking for Unmanned Aerial [21]

Este enfoque puede tener aplicaciones como el monitoreo 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 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].

En el artículo *Automatic Damage Detection and Diagnosis for Hydraulic Structures Using Drones and Artificial Intelligence Techniques* [22], 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 la figura 1.12, demuestran la eficacia de esta detección de daños en las presas hidráulicas mediante el uso de técnicas de visión e inteligencia artificial.

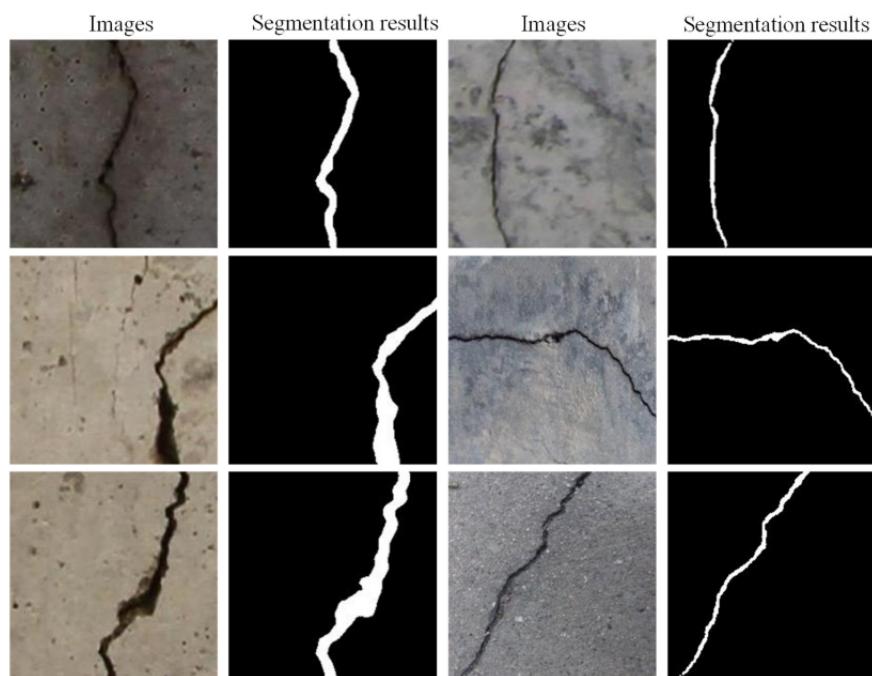


Figura 1.12: Demostración de la identificación de diferentes tipos de grietas [17]

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 autonómica, permitiéndoles tomar decisiones en tiempo real y adaptarse a entornos cambiantes de manera eficiente. Permitiendo a los drones poder aprender de sus experiencias y entender e interactuar con el entorno en el que se encuentran de una manera 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. Ésto 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.

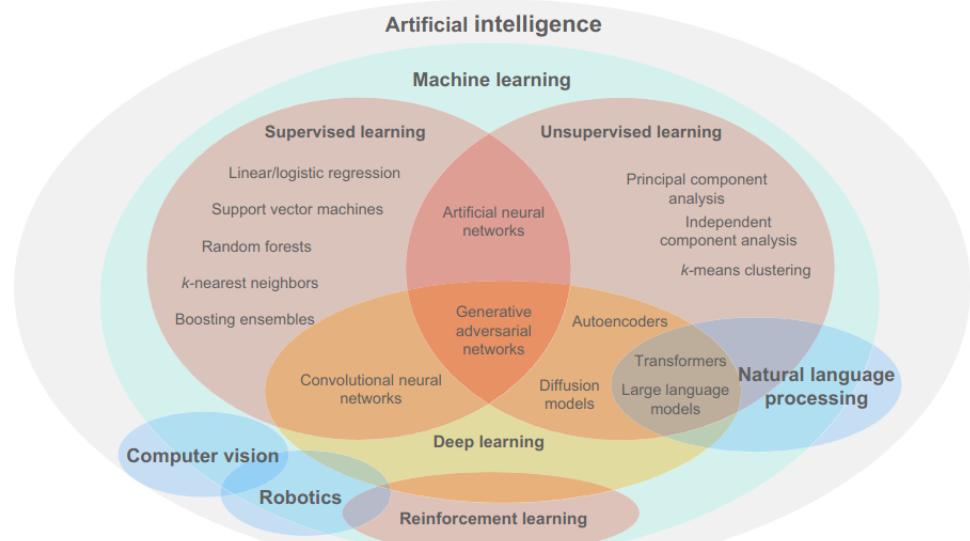


Figura 1.13: Clasificación de Inteligencia Artificial [11]

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 [18] 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 1.14.

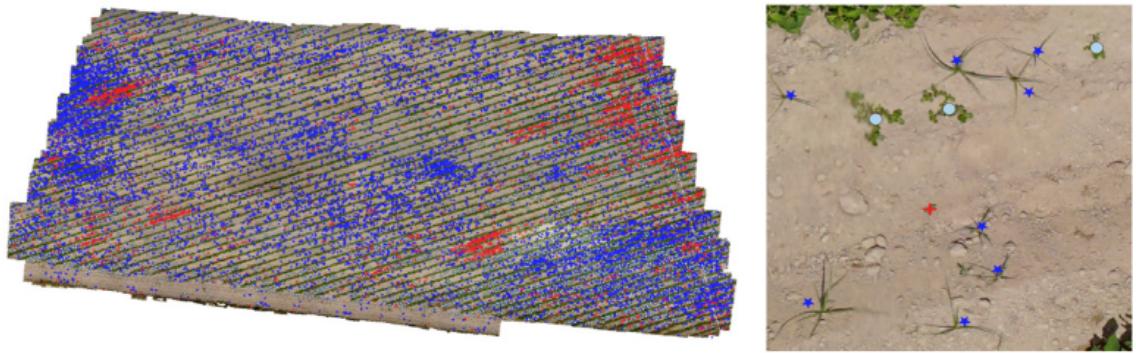


Figura 1.14: Resultados de la detección y clasificación de malas hierbas en un cultivo [10]

Por otro lado, reinforcement learning (RL) [16] es una técnica dentro del aprendizaje automático que es interesante utilizar en la navegación autonóma de drones. Esta metodología permite a los drones aprender a planificar rutas de forma autonóma, 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* [19] 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 aprender comportamientos y tomar decisiones por él mismo.

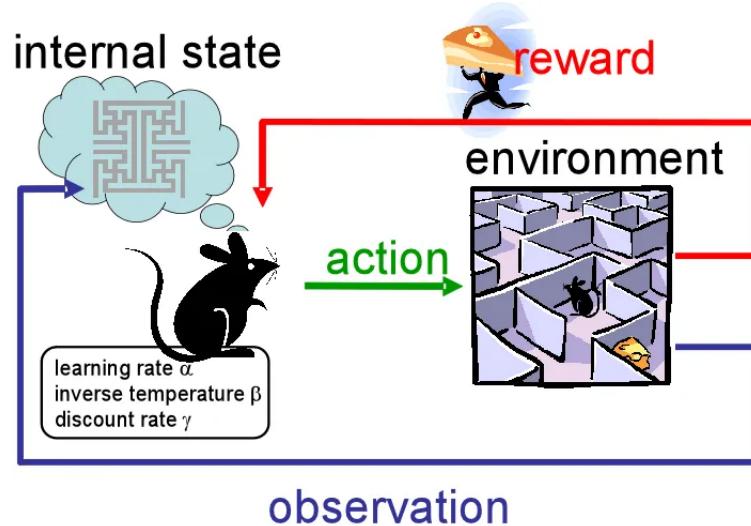


Figura 1.15: Esquema de Reinforcement Learning [7]

En conclusión, la inteligencia artificial puede ser 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 vayamos 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. Navegación autónoma en Airsim basada en inteligencia artificial y aprendizaje por refuerzo

En este trabajo realizaremos un algoritmo basado en navegación autonómica para drones por entornos de carreteras sin intervención humana. Nuestro enfoque se basa en combinar la inteligencia artificial (IA) y aprendizaje por refuerzo (RL) para lograr vuelos autónomos y seguros, además de utilizar técnicas de procesamiento de imágenes y visión para detectar y segmentar las carreteras en las imágenes capturas por el dron permitiendo establecer regiones de interés específicas para la navegación autónoma.

Este tipo de comportamientos pueden tener aplicaciones potenciales como monitorizar carreteras en la seguridad vial identificando los diferentes carriles de las carreteras y sus vehículos en tiempo real, entrega de paquetes de manera autónoma siguiendo rutas de carreteras o vigilancia de accidentes en áreas de carreteras.

En conclusión, con este trabajo de investigación buscamos impulsar el uso de la tecnología de drones en la seguridad, planificación y eficiencia en el ámbito de las carreteras.

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 aprendizaje por refuerzo e inteligencia artificial, en el que el dron sea capaz de navegar de una manera robusta sin colisiones 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 desarrollo de un sistema perceptivo basado en redes neuronales la en la navegación autonómica de drones.
3. Desarrollo de un sistema de control para drones utilizando técnicas de aprendizaje por refuerzo para lograr una navegación autónoma, segura y eficaz.
4. Analisis y desarrollo de una aplicación de navegación autónoma de drones basandonos 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 segura del vehículo dentro del circuito urbano.
4. Los sistemas desarrollados deben ser reactivos para poder reaccionar a su entorno de manera concisa y eficiente durante la navegación.
5. Emplear el algoritmo de QLearning 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 Mayo 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 via 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 seguir dentro del proyecto junto la asignación e identificación de tareas.

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/>

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 tableros 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.

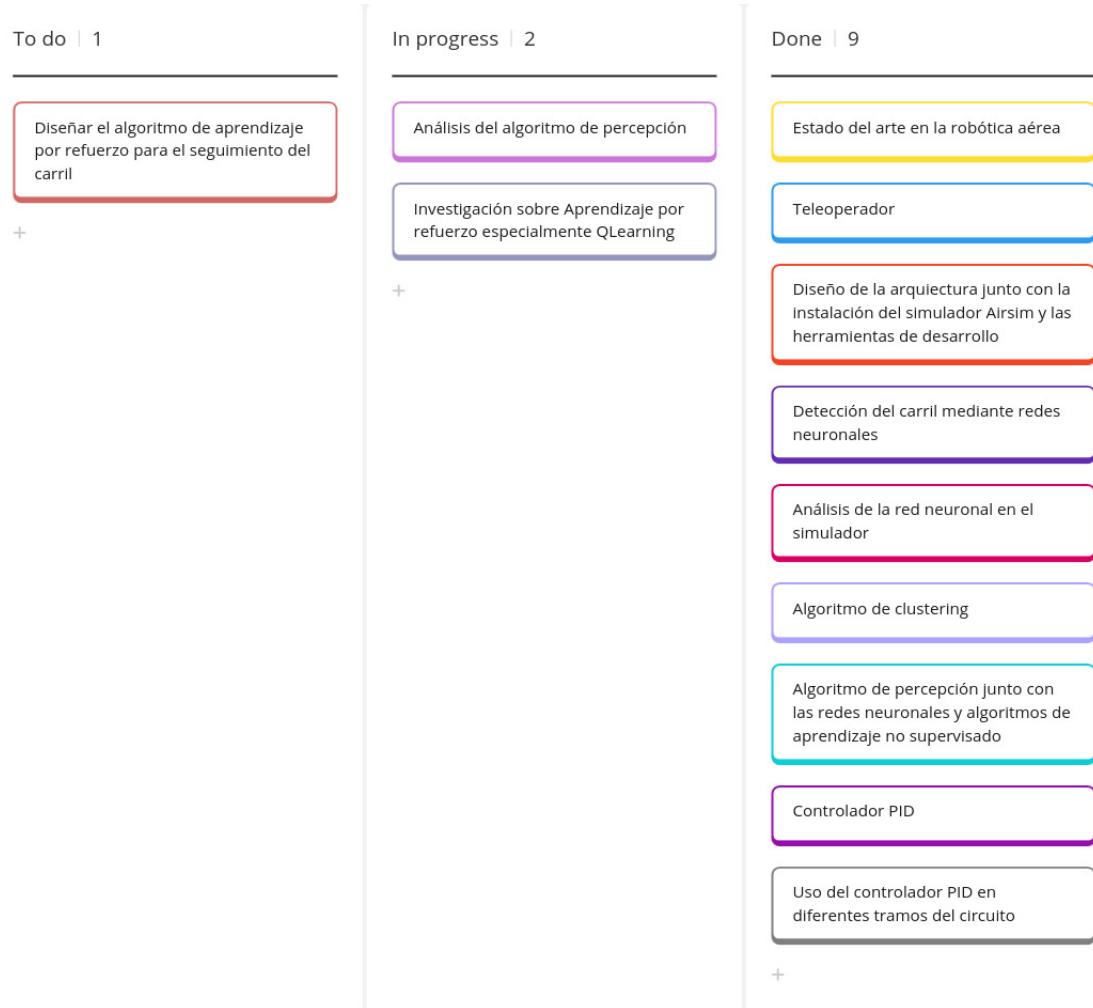


Figura 2.1: Ilustración del tablero Kanban durante el desarrollo del TFG

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.

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

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



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 desarrollo 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 detención de carril mediante redes neuronales.
 - Análisis y comparación de los resultados de los diferentes modelos que ofrece 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, fue la programación del 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 hablaremos sobre qué tecnologías hemos 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, hemos 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.

También a destacar, la facilidad que puede ser la instalación de dicho lenguaje en sistemas operativos como Linux o Windows.

En el caso de este TFG, se utiliza Python3 para todo el desarrollo del código junto

¹<https://www.python.org/>

²<https://gvanrossum.github.io/>

con el middleware robotico ROS (veáse la sección 3.3) y para el desarrollo de los diferentes algoritmos.

```
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

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

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 nuestro caso utilizaremos esta biblioteca para poder obtener la imagen mediante la cámara que va abordo del vehículo.

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/>

```
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 librería OpenCv

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

Utilizaremos 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 de Python para el aprendizaje automático 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 siendo un muy buen soporte para Nvidia. Esta disponible para varios sistemas operativos como Windows, Linux, MacOs y Cloud Platforms (Amazon Web Services, Google Cloud Platform, Microsoft Azure).

⁵<https://pytorch.org/>

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 multiples 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 portatil 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 realizaremos la inferencia de los diferentes modelos que ofrece la red neuronal YOLOP(3.2) para encontrar el mejor modelo en cuanto a rendimiento y velocidad de inferencia.

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

⁶<https://onnxruntime.ai/>

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

⁸<https://scikit-learn.org/stable/>

completa y su enfoque en la eficiencia y la simplicidad. En este trabajo utilizaremos la librería para el uso de algoritmos de clasificación 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. **Detención de objetos de tráfico:** Identifica los objetos presentes en la carretera, como otros vehículos, peatones, señales de tráfico, etc.
2. **Segmentación del área transitable:** Determina las áreas de la carretera por las que un vehículo puede conducir de manera segura.
3. **Detención de carriles:** Identifica los carriles de la carretera.

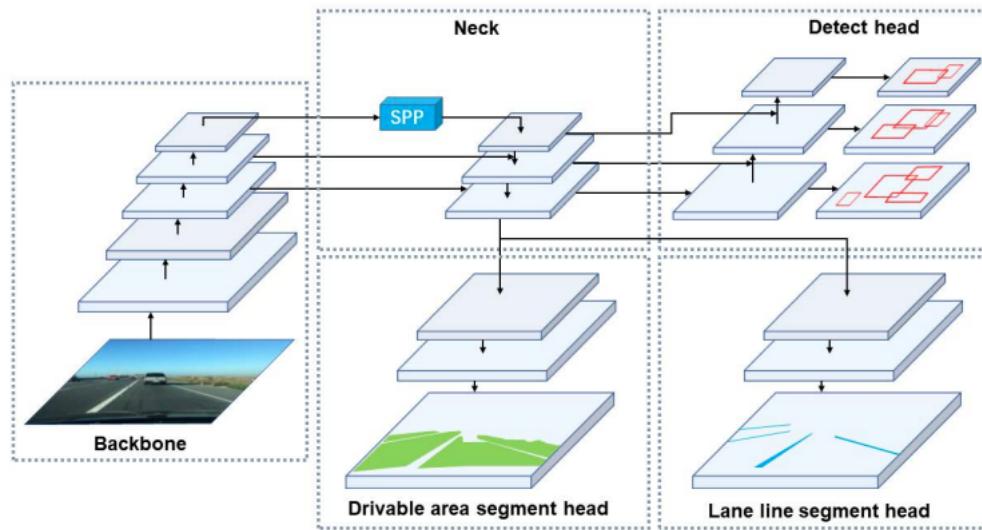


Figura 3.1: Arquitectura de YOLOP

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 extremadamente bueno en el desafiante conjunto de datos BDD100K[20], logrando el estado del arte en las tres tareas en términos de precisión y velocidad.

Nosotros utilizaremos los decodificadores de segmentación del área transitable y detención 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 pixeles. Esto significa que para cada pixel de la imagen, el decodificador de segmentación del área transitable predice si el pixel pertenece a un área transitable o no.

El decodificador de detención de carriles, también utiliza los mapas de características extraídos por el codificador. Sin embargo, en lugar de predecir si un pixel es transitable o no, este decodificador predice si un pixel pertenece a un carril de la carretera.

Es importante destacar que estos decodificadores no funcionan de manera aislada, sino que forman parte de un sistema de aprendizaje multitarea, es decir, se entranaran conjutamente para realizar sus tareas respectivas, lo que puede mejorar el rendimiento general del sistema.

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. Dicho modelo presenta unos pesos preentrenados:

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 a partir de Onnx.

Dependiendo de que pesos preentrenados queramos utilizar tendremos resultados diferentes en el ámbito de cómputo y rápidez a la hora de realizar la inferencia del modelo con los pesos.

```
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

Este ejemplo se puede encontrar en la página Pytorch dedicada a la red neuronal YOLOP⁹.

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

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

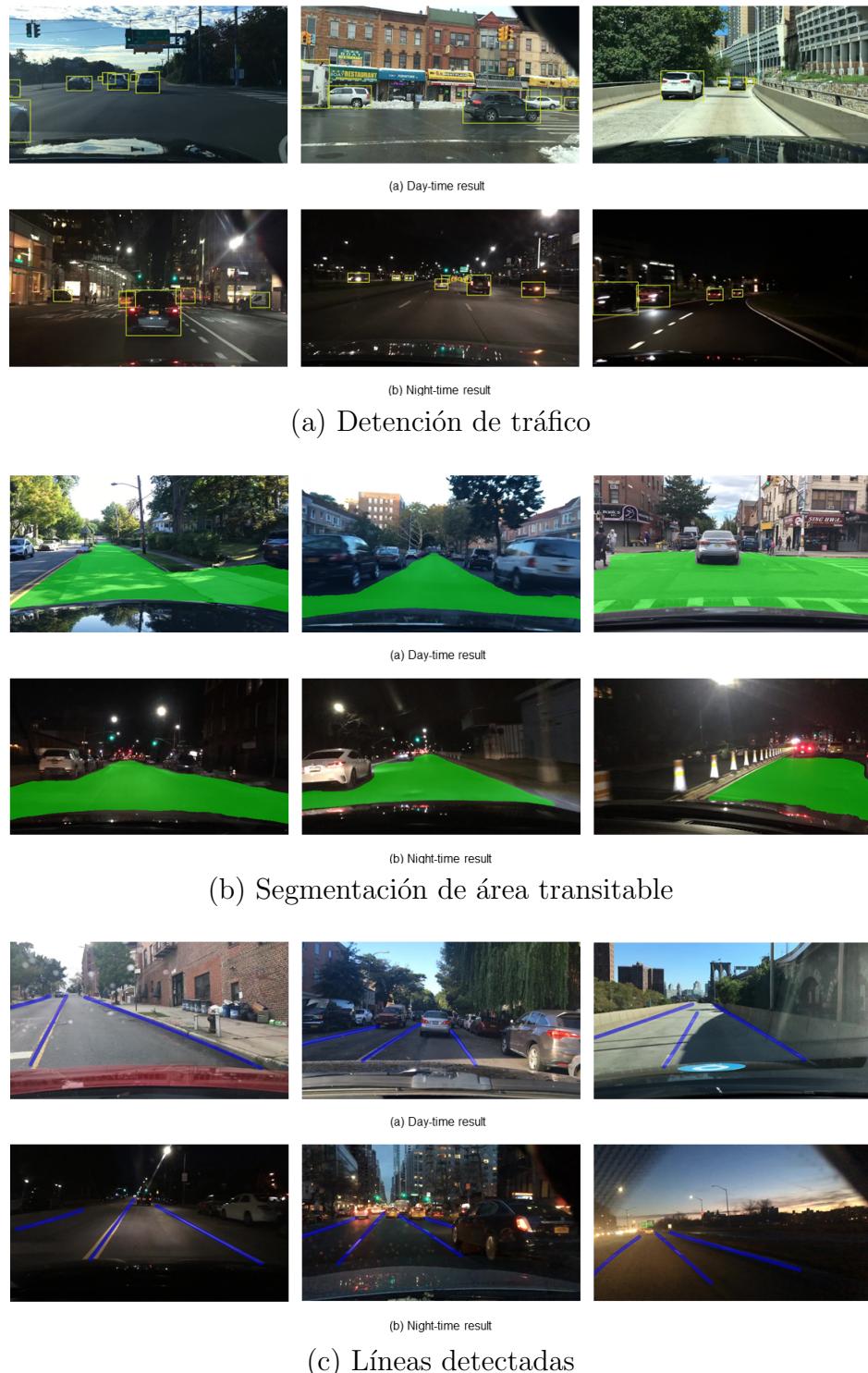


Figura 3.2: Resultados de la salida de la red neuronal YOLOP[8]

3.3. ROS

ROS (Robot Operating System)¹⁰ es un conjunto de librerías de código abierto utilizadas principalmente para aplicaciones robóticas. Podemos 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,

¹⁰<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.

Utilizamos ROS 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.

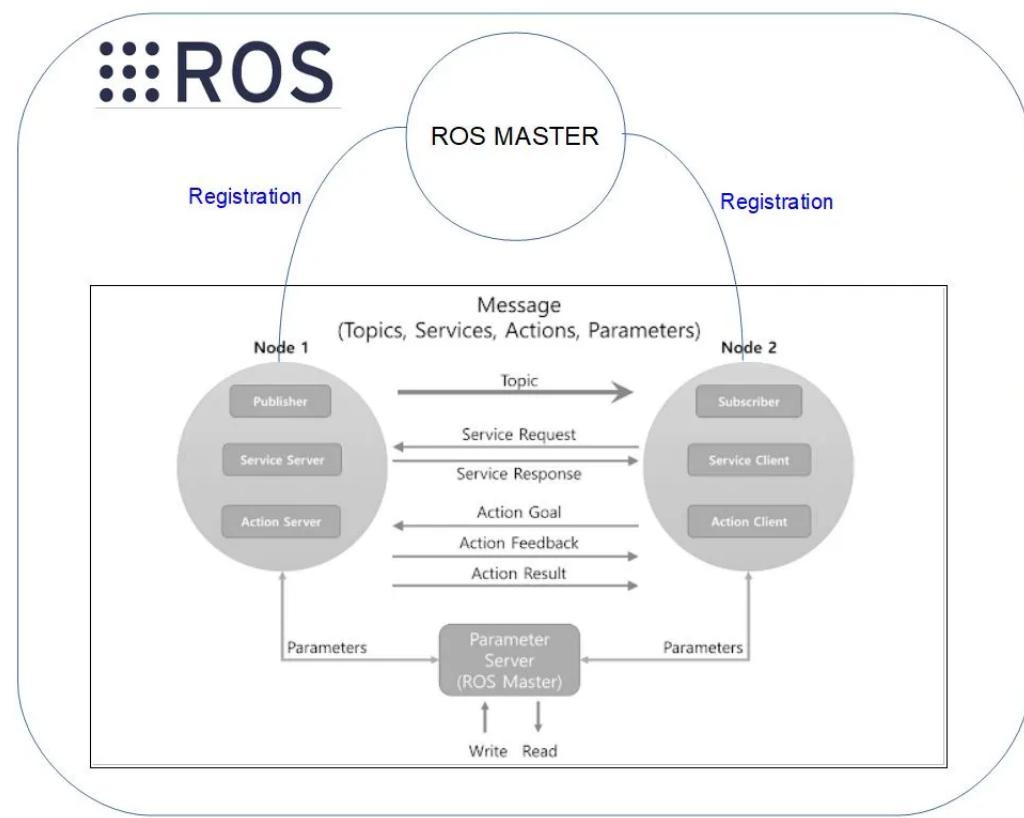


Figura 3.4: Arquitectura de ROS

3.3.1. Mavros

Mavros¹¹ es un paquete formado por **ROS** y el protocolo de comunicaciones ligero **MAVLink** (Micro Air Vehicle Link) diseñado por Lorenz Meier¹² 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 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 estudiaremos y analizaremos si **Mavros** se podría utilizar para el control del dron junto con **PX4 AutoPilot** (3.5) a través de Airsim.

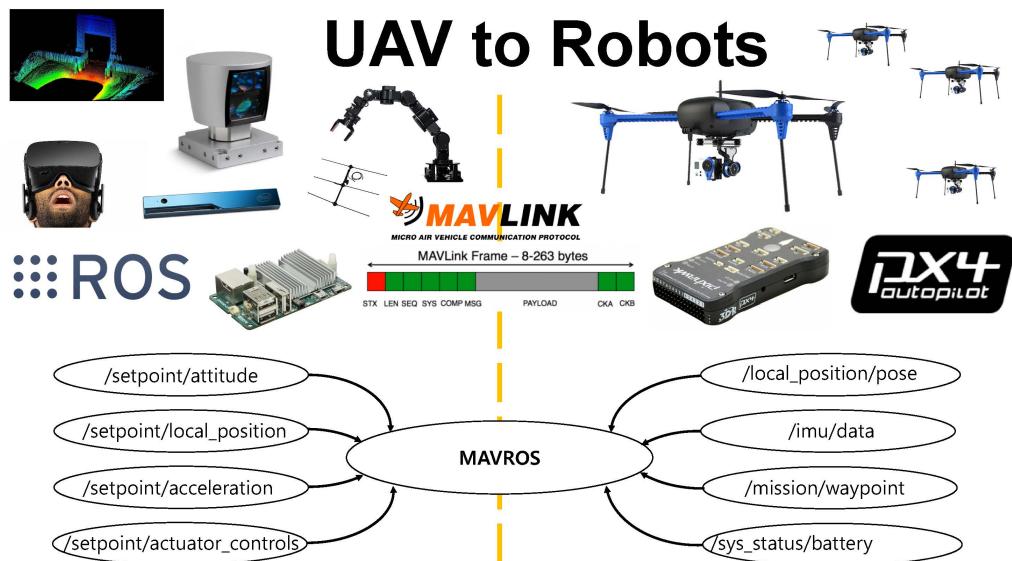


Figura 3.5: Infraestructura de Mavros

¹¹<http://wiki.ros.org/mavros>

¹²<https://www.technologyreview.es/listas/35-innovadores-con-menos-de-35/2017/inventores/lorenz-meier>

3.4. Airsim

El entorno de simulación en el que vamos a estar trabajando será **Airsim** junto con **UnRealEngine** de Epic Games¹³. **Airsim**¹⁴ 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 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 vision de computador.
2. **ClockType**: Determina qué tipo de reloj se utiliza para medir el tiempo en la simulación.

¹³<https://www.unrealengine.com/es-ES>

¹⁴<https://microsoft.github.io/AirSim/>

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 fisico si utilizamos un vehículo en un entorno real.
- **LockStep:** Es una característica importante cuando se comunica con el simulador AirSim a través de TCP.
- **UseTcp:** Para poder comunicarnos a través de TCP necesitamos habilitar esta opción a true.
- **TcpPort:** Especificamos el puerto TCP que vayamos a usar.
- **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.
- **Parameters:** Estos parametros son de PX4 y permite la configuración del vehiculo, como por ejemplo los modos de vuelo, sus configuraciones, configuraciones de velocidades y más.
- **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.

Nosotros usaremos una cámara que proporcionará una imagen RGB de dimensiones 620x620 pixeles y activaremos 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 Airsim¹⁵

¹⁵<https://microsoft.github.io/AirSim/settings/>

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.

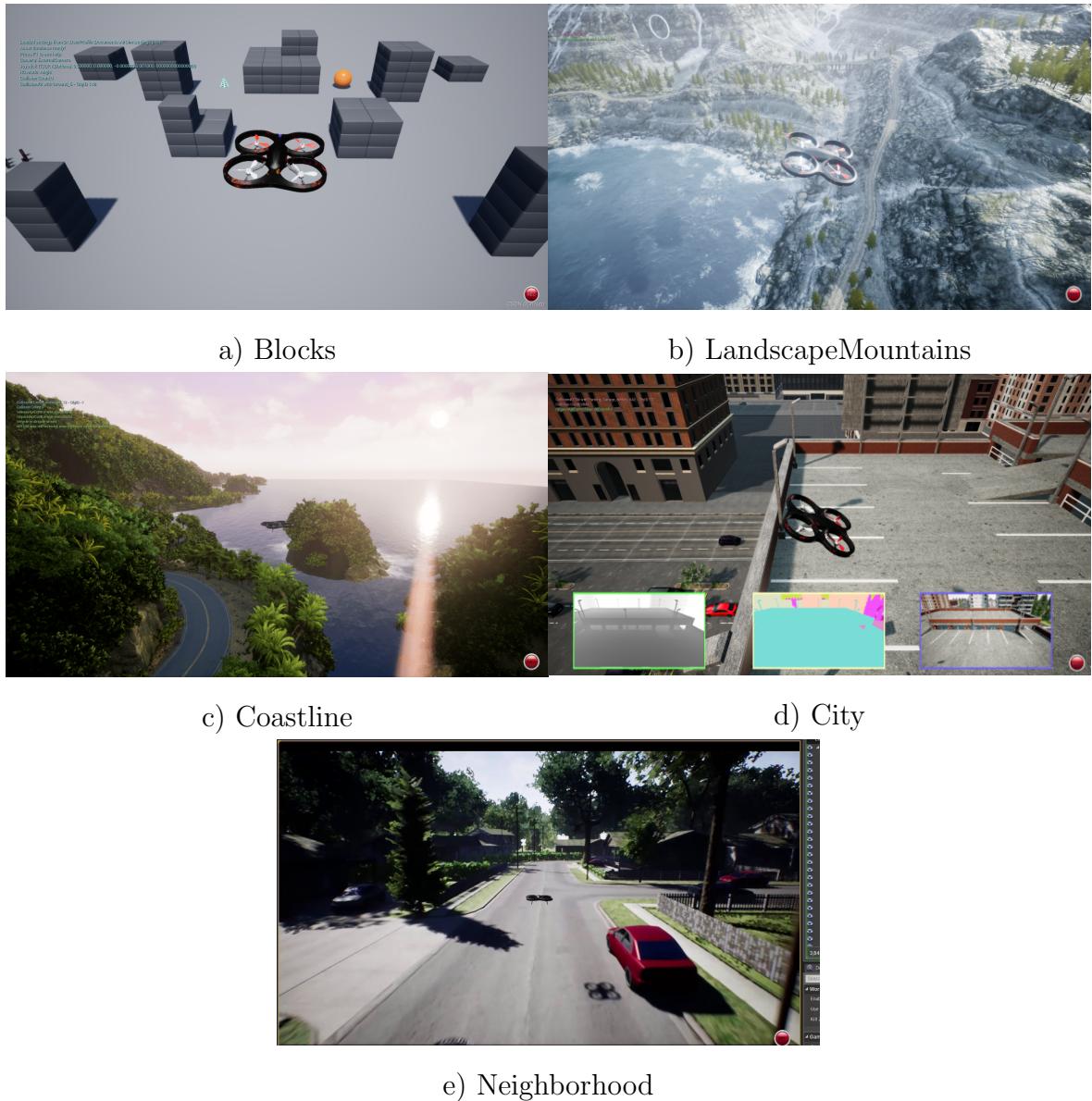


Figura 3.6: Ejemplos de escenarios en Airsim

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 nuestro caso hemos utilizado el escenario Coastline, consiste en un entorno fotorrealista de un recorrido amplio de 2 carriles con ambiente tropical. Con este entorno tendremos la información del entorno para realizar el comportamiento

sigue carril con el dron.

Todos estos escenarios se pueden encontrar en las releases de Airsim¹⁶ tanto para Windows como para Linux. En nuestro caso hemos utilizado el escenario Coastline, ya que queremos 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 nuestro caso utilizaremos 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 vehículo.

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. ArduRover¹⁷ se trata de un piloto automático de código abierto utilizado específicamente para vehículos terrestres.

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.

¹⁶<https://github.com/Microsoft/AirSim/releases>

¹⁷<https://ardupilot.org/rover/>

- **ArduCopter:** Representa un dron pero siguiendo el estándar ArduCopter. ArduCopter¹⁸ se trata de un piloto automático de código abierto utilizado para los drones

Como hemos numerado anteriormente, este entorno de simulación tiene un catálogo de vehículos, sensores y cambios climatológicos dentro del entorno. Nosotros utilizaremos un dron de tipo "SimpleFlight".

3.4.1. Airsim ROS Wrapper

Utilizaremos el paquete **Airsim ROS Wrapper**¹⁹ 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 hablaremos sobre que es un ROS Wrapper.

ROS Wrapper es un componente que facilita la integración entre dos sistemas o entornos diferentes. Si lo llevamos 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 Airsim²⁰:

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.
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**.

¹⁸<https://ardupilot.org/copter/>

¹⁹https://microsoft.github.io/AirSim/airsim_ros_pkgs/

²⁰<https://github.com/microsoft/AirSim>

3.4.2. Client Airsim

Airsim ofrece una API implementada para Python denominada Client Airsim²¹, 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 la utilizaremos sobre todo para tener el control del vehículo para realizar su navegación con los controladores PID y en el desarrollo de aprendizaje por refuerzo.

3.5. PX4 AutoPilot

PX4²² 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.

Unas 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 realizaremos un análisis respecto a la navegación del dron mediante **PX4** con el modo de simulación Software in The Loop(SITL) para tener el control el vehículo junto con **Mavros** y **Airsim**.

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

²¹<https://microsoft.github.io/AirSim/apis/>

²²<https://docs.px4.io/main/en/>

planificación de misiones.

Podemos 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 comunidad²³. En nuestro caso realizaremos la configuración PX4 SITL junto con Airsim para estudiar si es posible tener un control en el comportamiento a querer desarrollar.

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 vehículo 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 parametros tienen valores por defecto definidos por la plataforma (2.5 m y 1.5 m/s respectivamente). Antes de realizar el despegue el vehículo sera armado para poder realizarlo.

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

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

Cuando de realice este modo de vuelo por defecto se cambiará al modo de Position

3. **HOLD:** A partir de este modo de vuelo podemos parar el vehículo manteniendolo en el aire con su actual posición GPS y altitud. Este modo puede ser bastante útil para cuando queremos pausar una mision o reiniciar el comportamiento que queramos realizar.

²³<https://docs.px4.io/v1.14/en/simulation/>

4. **POSITION:** Es un modo de vuelo manual el cual puedes controlar el vehículo mediante un joystick, dicho vuelo controla la posición del vehículo cuando comandemos velocidades mediante el joystick. Este modo de vuelo lo utilizaremos para teleoperar el dron para ver el funcionamiento de la percepción por parte de la red neuronal que utilizaremos. Dicho vuelo ofrece un gran catálogo de parametros que puede afectar al vuelo.

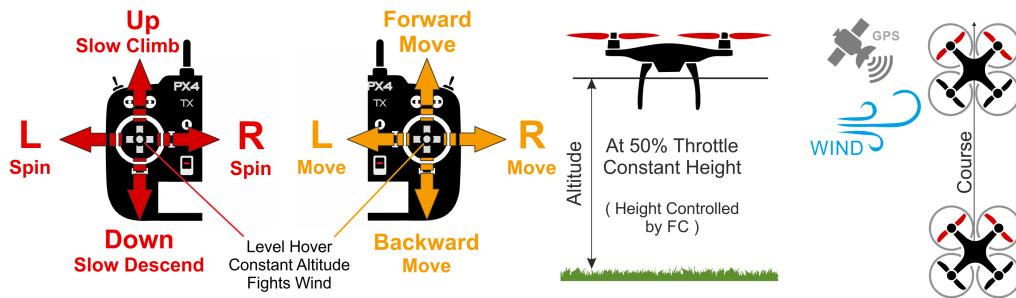


Figura 3.7: Diagrama del comportamiento del modo de vuelo Position

5. **OFFBOARD:** Con este modo de vuelo podemos 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 esta 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 Autopilot²⁴). 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

²⁴<https://docs.px4.io/v1.14/en/config/safety.html>

configuración de px4.config.yaml LOCAL_NED. Si queremos que el marco de coordenadas se mueva con el cuerpo del vehículo se tendrá que utilizar el marco de coordenadas En nuestro caso necesitamos un marco de coordenadas diferente para que el vehículo se mueva con el cuerpo del vehículo, por ello utilizaremos el marco de coordenadas BODY_NED.

Los marcos de coordenadas que ofrece Mavros se puede ver a través del servicio SetMavFrame.srv²⁵

3.6. QGroundControl

QGroundControl²⁶ 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.

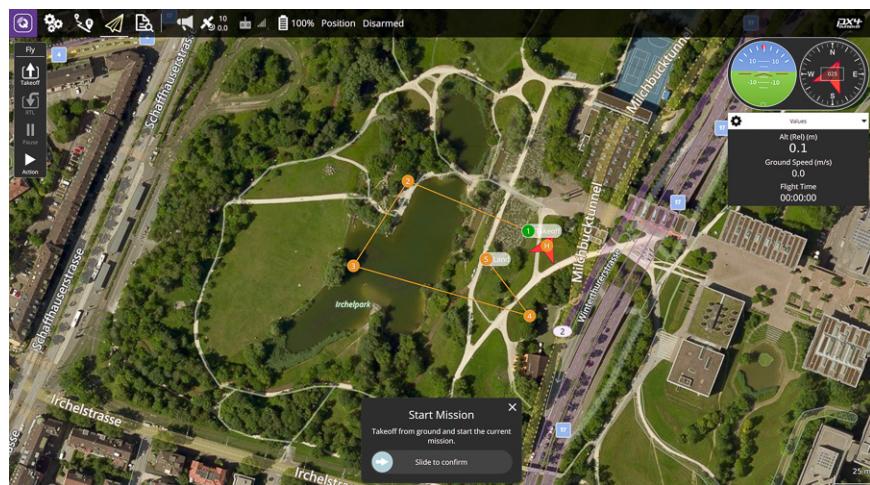


Figura 3.8: QGroundControl

²⁵https://github.com/mavlink/mavros/blob/master/mavros_msgs/srv/SetMavFrame.srv

²⁶<http://qgroundcontrol.com>

Capítulo 4

Diseño

El principal objetivo de este TFG es poder desarrollar un comportamiento de navegación autónoma en entornos de simulación con el fin de que el dron pueda alcanzar el comportamiento deseado utilizando inteligencia artificial, específicamente redes neuronales y aprendizaje por refuerzo. Se tomará como entorno de simulación Airsim junto con el motor de videojuegos UnRealEngine, adicionalmente con la comunicación de este entorno con AirSim ROS Wrapper Node y Client Airsim, además del estudio y análisis de la comunicación entre Airsim, PX4 Autopilot, Mavros y AirSim ROS Wrapper Node . Así, pudiendo evaluar si es posible la unión de Airsim junto con ROS, PX4 y Mavros para herramientas de diseño en la navegación autónoma de drones.

En primer lugar, en el contexto de la conducción autónoma de drones se evaluará el desarrollo de la percepción mediante redes neuronales y algoritmos de aprendizaje no supervisado, en especial, clustering y regresiones. Posteriormente, el control del UAV se procederá una primera aproximación con un controlador Proporcional Derivativo Integral (PID) desenvolviéndose en un entorno de carreteras de ambos carriles.

Luego una vez de tener un comportamiento de sigue carril mediante el algoritmo de percepción desarrollado, procederemos a desarrollar el algoritmo de aprendizaje por refuerzo. Junto con este aprendizaje se realizará análisis de diferentes métricas y resultados por así entonces de poder encontrar una solución robusta y eficaz al problema que queremos enfrentarnos que será la navegación autónoma de drones en entornos de carreteras.

4.1. Arquitectura

La arquitectura que se ha llevado a cabo en este trabajo se compone de varios elementos fundamentales que se integran entre sí para el desarrollo del seguimiento del carril. En primer lugar, tendremos un enfoque de simulación Airsim que ofrece la configuración directa de los modelos simulados, ofreciendo la modificación de los sensores y actuadores disponibles en el entorno de trabajo para obtener los comportamientos deseados. Así de tener una comunicación directa con los componentes intermediarios tanto para el control de los actuadores como el manejo de los sensores. Dichos componentes intermediarias se comunican tanto con los programas desarrollados durante en este trabajo como con el entorno de simulación para obtener comportamientos de navegación autónoma, facilitando la comunicación entre el entorno de simulación y los programas desarrollados.

Se realizan implementaciones de comportamientos de navegación autónoma basada en redes neuronales y algoritmos de aprendizaje automático, como control clásico basado en un controlador Proporcional-Derivativo-Integral así como comportamientos más eficientes y complejos basados en aprendizaje por refuerzo.

Así junto con el uso de herramientas como PX4 Autopilot y Mavros para así poder analizar la viabilidad junto con el entorno de desarrollo Airsim. Las principales implementaciones que se han llevado a cabo son principalmente dos como se muestra en la figura 4.1:

1. **Seguimiento de carril basado en redes neuronales, algoritmos de aprendizaje automático y control clásico:** Utiliza la red neuronal YOLOP junto con algoritmos de aprendizaje automático para y un control clásico basado en un controlador PID para el seguimiento de carriles.
2. **Seguimiento de carril basado en redes neuronales, algoritmos de aprendizaje automático y Q-learning:** Utiliza la red neuronal YOLOP junto con algoritmos de aprendizaje automático para con la diferencia de que el control se realiza con un algoritmo de aprendizaje por refuerzo en especial Q-learning para realizar el seguimiento de carriles.

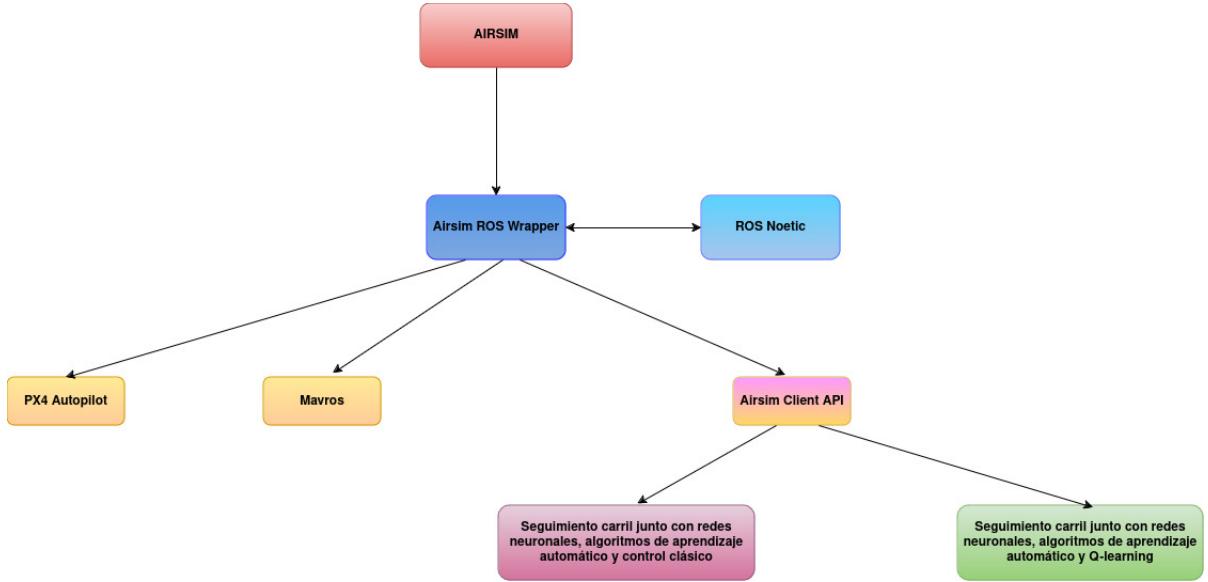


Figura 4.1: Arquitectura de los componentes utilizados en el TFG

4.1.1. Configuración de Integración Distribuida en equipos separados

En el desarrollo de este TFG, hemos tomado la decisión de tener dos enfoques:

1. El entorno de simulación Airsim estará en un ordenador con un sistema operativo Windows 10, una gráfica Nvidia RTX 2070 Super con el plugin de Airsim al cargar el entorno mediante el motor UnrealEngine.
2. Las plataformas de desarrollo como ROS, Mavros, PX4, QGC y Client Airsim API estarán en un ordenador secundario con un sistema operativo Ubuntu 20.04 con una gráfica Nvidia RTX 2070.

Esta propuesta fue tomada con la iniciativa de no tener encapsulado en un único componente el entorno de simulación y las plataformas de desarrollo, ya que desembocaba a tener un bajo rendimiento en cuanto a velocidad de procesamiento del propio entorno de simulación. La primera aproximación se llegó a construir un entorno de desarrollo con un único ordenador con una gráfica de Nvidia RTX 2070 compuesto de un sistema operativo Linux en donde se integraba el simulador Airsim junto UnrealEngine y las plataformas de desarrollo, obteniendo una media de 60 FPS (Fotogramas por segundo) conllevaron así a tener un comportamiento lento debido a la carga de procesamiento por parte del entorno de simulación. Por causa de Airsim, esta

contruido sobre un motor de videojuegos, maximizando los controladores de gráficos de NVIDIA, lo que puede afectar el rendimiento en sistemas menos optimizados. Por otro lado, si utilizamos un sistema operativo como Windows, es posible lograr un alto rendimiento debido a que Windows tiende a asignar más recursos a las aplicaciones gráficas junto con la gráfica Nvidia, lo que puede llegar a beneficiar a Airsim. Al tener Airsim en Windows llegamos tener una media 285 FPS siendo mucho más considerable que la primera opción con un sistema operativo Linux lo cual hace interesante tener Airsim en un entorno que tenga como sistema operativo como Windows.

Por otra parte, para poder utilizar las herramientas junto con el simulador necesitaremos Linux ya que estas herramientas de desarrollo como ROS funcionan con un sistema como Linux lo cual se necesitaría tener dos sistemas operativos diferentes en un mismo equipo. Como se necesitaba desarrollar una comportamiento robusto, eficiente y en tiempo real para el seguimiento de un carril con un dron se decidió la separación en dos equipos para así conseguir el mejor rendimiento por ambas partes.

El entorno de simulación Airsim se puede comunicar como hemos mencionado con PX4 y Mavros siguiendo un esquema de comunicaciones marcado por la propia página oficial de PX4¹ como se muestra en la siguiente figura:

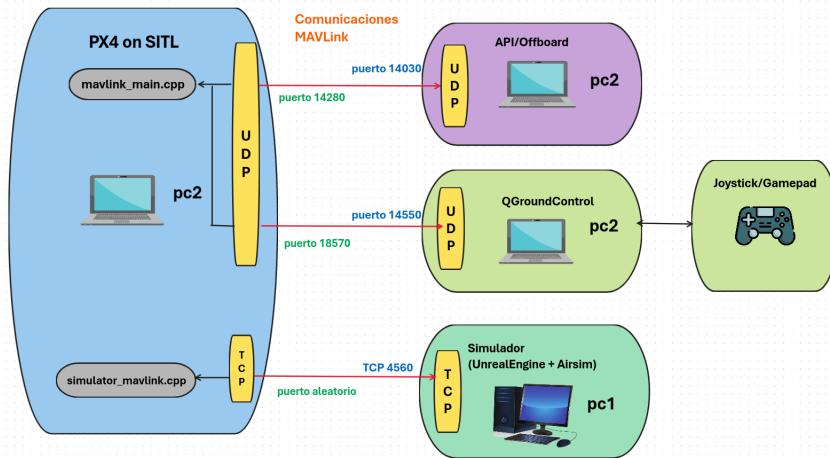


Figura 4.2: Diagrama de comunicaciones entre PX4, QGC y Airsim

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

En donde las comunicaciones internas con PX4 Autopilot se realizan mediante la ejecución del programa px4_sitl_default none_iris y los puertos asignados los realiza la propia API de Mavlink además de tener la posibilidad con PX4 de comunicarnos con QGC.

```
INFO [simulator_mavlink] using TCP on remote host 192.168.2.16 port 4560
WARN [simulator_mavlink] Please ensure port 4560 is not blocked by a firewall.
INFO [simulator_mavlink] Waiting for simulator to accept connection on TCP port 4560
INFO [simulator_mavlink] Simulator connected on TCP port 4560.
INFO [lockstep_scheduler] setting initial absolute time to 1709283013044588 us
INFO [commander] LED: open /dev/led0 failed (22)
WARN [health_and_arming_checks] Preflight Fail: ekf2 missing data
INFO [mavlink] mode: Normal, data rate: 4000000 B/s on udp port 18570 remote port 14550
INFO [mavlink] mode: Onboard, data rate: 4000000 B/s on udp port 14580 remote port 14540
INFO [mavlink] mode: Onboard, data rate: 4000 B/s on udp port 14280 remote port 14030
INFO [mavlink] mode: Gimbal, data rate: 400000 B/s on udp port 13030 remote port 13280
```

Figura 4.3: Salida de la terminal ejecutando PX4 SITL

A lo largo del desarrollo de este trabajo, nos hemos encontrado diferentes dificultades respecto a la comunicación directa entre PX4 y el simulador, ya que a menudo cuando se llegaba a comunicar las plataformas de desarrollo entre PX4 y Airsim se producían latencias afectando a la sincronización del comportamiento autónomo que queremos desarrollar. Además de que al tener una capa de comunicación extra con la propia herramienta PX4 Autopilot, Mavros el acceso no es totalmente directo y puede llegar a ser difícil. Por lo que se optó a utilizar Client Airsim por su comunicación directa con el simulador y se puede llegar a comunicar directamente con el simulador sin llegar a producir capas externas por su arquitectura, también esta pensada para realizar comportamiento en tiempo real con el simulador.

En resumen, la comunicación constará de Airsim ROS Wrapper para acceder a los diferentes sensores y Client Airsim para el control del vehículo más el entorno de simulación Airsim. El diagrama de comunicaciones que se muestra en la figura 4.4 consiste en una sencilla implementación de tener dos equipos diferentes que constan del entorno de simulación y de las plataformas en donde iremos desarrollando los diferentes comportamientos como los algoritmos de percepción y control que serán capaces de comunicarse entre sí en la mismo entorno de red mediante un router, cada equipo constará de dos direcciones IP diferentes.

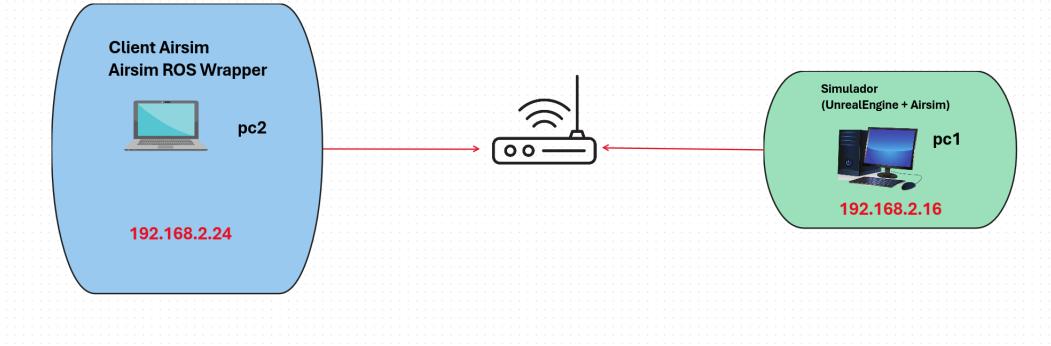


Figura 4.4: Diagrama de comunicaciones

4.2. Preparación del entorno de simulación

Como mencionamos en la sección 3.4, utilizaremos como simulador Airsim junto con el motor UnRealEngine. Para llevar acabo la construcción del entorno, primero tendremos que instalar UnrealEngine. Para ello, seguiremos las instrucciones marcadas por la página oficial de Epic Games² y trabajaremos con la versión 4.27.2.

Una vez se tenga UnRealEngine instalado, procederemos a configurar el entorno de simulación que necesitamos, para ello realizaremos la configuración del fichero settings.json con las diferentes secciones. Por defecto, cuando ejecutamos por primera vez Airsim, el propio simulador nos crea este fichero dentro de una carpeta denominada Airsim dentro de la carpeta Documentos en Windows, lo cual es bastante cómodo ya que podemos realizar la modificación de dicho archivo a nuestras necesidades.

4.2.1. Configuración del dron y del entorno

En primer lugar utilizaremos el entorno de simulación Coastline de los diferentes entornos que nos ofrece Airsim para Windows. Al descargarnos dicha carpeta, contendrá el ejecutable para poder abrir el entorno con Airsim junto con las carpetas de modelos de simulación, como las carreteras, montañas y plantas. Dentro de estas carpetas, eliminaremos dichos componentes de simulación que nos puede dificultar cuando desarrollemos la percepción como plantas y señales de tráfico que se ubican encima de las líneas de detección del carril ya que esto nos puede perjudicar en un futuro en la detección con la red neuronal YOLOP.

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

Por defecto, la primera vez que ejecutamos un entorno en Airsim, se crea el archivo settings.json en donde se almacenará por defecto en una carpeta llamada airsim ubicada en Documentos en Windows.

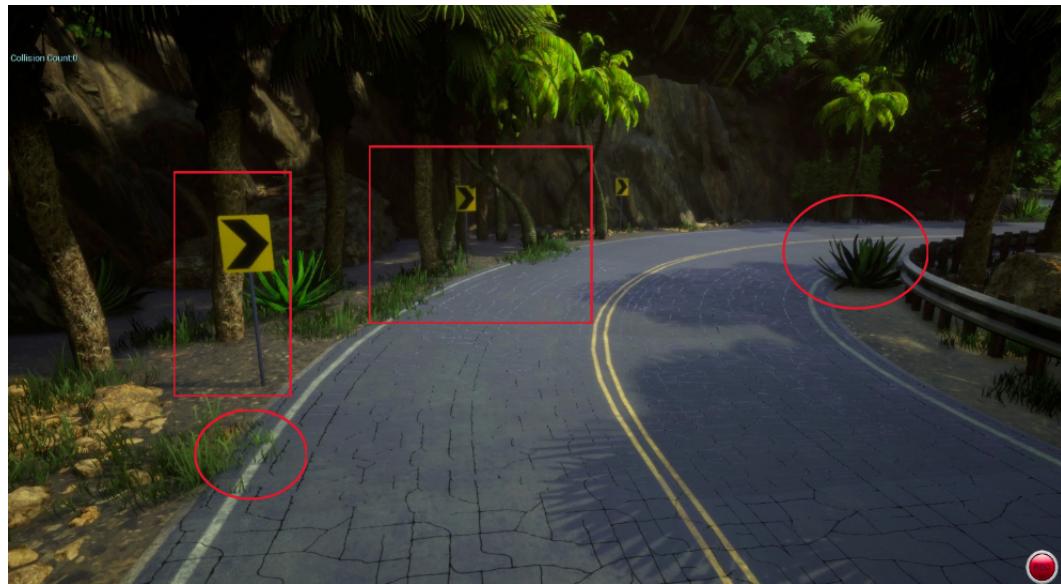


Figura 4.5: Visualización del entorno original ilustrando los objetos dificultando la percepción de la carretera

Cuando tengamos el fichero settings.json creado, ya podremos equipar al vehículo con características como qué sensores utilizará, el tipo de vehículo, el tipo de simulación, la comunicación y más. En este TFG se definirá como sensores el lidar para saber a qué altura esta volando el dron respecto al suelo, el gps para saber que localización tiene en el entorno y la cámara. Cada sensor debe llevar su propia configuración, como se muestra en el código 4.1 se configuró cada sensor siguiendo sus propios 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 parametros que se tendrán que configurar como el tamaño de la imagen, si queremos ROS, los FPS de la imagen transmitida, la posición y más.

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

```

1  {
2      "SettingsVersion":1.2,
3      "SimMode":"Multirotor",
4      "ClockType":"SteppableClock",
5      "Vehicles": [
6          "Drone": {
7              "VehicleType": "SimpleFlight",
8              "ControlIp": "remote",
9              "LocalHostIp": "192.168.2.16",
10             "Sensors": {
11                 "LidarCustom": {
12                     "SensorType": 6,
13                     "Enabled": true,
14                     "Range": 10,
15                     "NumberOfChannels": 16,
16                     "RotationsPerSecond": 10,
17                     "PointsPerSecond": 10000,
18                     "X": 0,
19                     "Y": 0,
20                     "Z": -1,
21                     "DrawDebugPoints": false,
22                     "DataFrame": "SensorLocalFrame"
23                 },
24                 "Gps": {
25                     "SensorType": 3,
26                     "Enabled": true,
27                     "EphTimeConstant": 0.9,
28                     "EpvTimeConstant": 0.9,
29                     "EphInitial": 25,
30                     "EpvInitial": 25,
31                     "EphFinal": 0.1,
32                     "EpvFinal": 0.1,
33                     "EphMin3d": 3,
34                     "EphMin2d": 4,
35                     "UpdateLatency": 0.2,
36                     "UpdateFrequency": 50,
37                     "StartupDelay": 1
38                 },
39             },
40             "Cameras": [
41                 "front_center_custom": {
42                     "CaptureSettings": [
43                         {
44                             "PublishToRos": 1,
45                             "ImageType": 0,
46                             "Width": 620,
47                             "Height": 620,
48                             "FOV_Degrees": 90,
49                             "ImageRate_FPS": 30,
50                             "TargetGamma": 1.5,
51                             "AutoExposure": true,
52                             "MotionBlur": false,
53                             "PostProcess": true
54                         }
55                     ],
56                     "X": 0.5,
57                     "Y": 0,
58                     "Z": 0.1,
59                     "Pitch": 0,
60                     "Roll": 0,
61                     "Yaw": 0
62                 },
63             ],
64             "X": 22.474933624267578,
65             "Y": -25.63629913330078,
66             "Z": 0,
67             "Pitch": 0,
68             "Roll": 0,
69             "Yaw": 30
70         }
71     }
72 }

```

Código 4.1: Configuración del vehículo 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 vehículo en el punto que decida Airsim.

4.2.2. Instalación de las herramientas de desarrollo

La instalación de las herramientas de desarrollo se realizarán en el segundo ordenador con el sistema operativo Linux.

1. **ROS:** Como se menciona en la documentación oficial de ROS [2], ROS consta de diferentes conjuntos de paquetes versionados que permiten a los desarrolladores trabajar con un código relativamente estable hasta que estén preparados y puedan publicar versiones robustas y eficientes. En nuestro caso, utilizaremos la distribución Noetic⁴ debido a que esta distribución se encuentra siendo una de las más estables para poder trabajar junto con Airsim⁵.

Para su instalación nos guiaremos en la guía oficial de ROS noetic⁶



Figura 4.6: Logotipo de ROS noetic

2. **Airsim ROS Wrapper:** Descargaremos del repositorio de Airsim a través de su página de GitHub⁷. Cuando tengamos dicho repositorio descargado, lo que realizaremos será la compilación y configuración de Airsim mediante los scripts build y setup como se pauta en la guía oficial.

⁴<http://wiki.ros.org/noetic>

⁵https://microsoft.github.io/AirSim/airsim_ros_pkgs/

⁶<https://wiki.ros.org/noetic/Installation/Ubuntu>

⁷<https://github.com/microsoft/AirSim>

```
git clone https://github.com/microsoft/AirSim.git
```

Código 4.2: Comando para poder clonar el repositorio de Airsim

Ahora para poder utilizar el Airsim ROS Wrapper que nos proporciona ROS, tendremos que irnos a la carpeta dentro del repositorio de Airsim clonado previamente y buscar la carpeta de ros y construir el paquete mediante el comando `catkin_make`. Cuando tengamos realizado dicho paso tendremos que activar el wrapper en el fichero `.bashrc` de Linux mediante la siguiente linea:

```
source /home/$USER/AirSim/roslaunch/devel/setup.bashrc
```

Código 4.3: Comando para activar el entorno de trabajo en el fichero `bashrc`

siendo `$USER` la variable de entorno que contiene el nombre del usuario que actualmente tiene iniciada sesión en el equipo.

A partir de activar el entorno de trabajo para ros, utilizaremos el nodo AirSim ROS Wrapper Node 3.4.1 junto con un launcher `airsim_node.launch` proporcionando la dirección IP en donde se encuentra Airsim.

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

Código 4.4: Lanzamiento del nodo AirSim ROS Wrapper Node especificando la dirección IP del simulador

4.3. Percepción

Para el desarrollo de la percepción una primera aproximación fue utilizar procedimientos de visión clásica, es decir, procesar la imagen y mediante métodos de la librería OpenCV poder detectar las diferentes líneas que pueden haber en la carretera. El problema que nos podemos con esta primera proximación que al tener un entorno fotorrealista, estos métodos se quedan bastante pobres desembocando una percepción muy poco robusta e ineficiente. Por lo que se optó la utilización de la red neuronal YOLOP y algoritmos de aprendizaje automático poder construir la percepción.

Para obtener la imagen del entorno, nos subscribiremos al topic /airsim-node/Drone/front_center_custom/Scene que ofrece el nodo AirSim ROS Wrapper Node y mediante un método de ROS denominado CvBridge⁸ podremos convertir dicha imagen en formato OpenCV y trabajar con ella a través de OpenCv.

4.3.1. Inferencia de YOLOP

Para realizar la detección de las líneas en la carretera nos ayudaremos de la red neuronal YOLOP3.2. A continuación hablaremos sobre la inferencia del YOLOP con sus distintos pesos preentrenados. En primer lugar, como anteriormente hemos comentado, el archivo End-to-end.pth es un archivo de pesos construido en Pytorch, cuando queremos cargar dichos pesos con el modelo podemos hacerlo de dos maneras: Podemos realizarlo mediante la CPU o la GPU. Para ello, si queremos cargar el modelo de YOLOP con los pesos preentrenados End-to-end.pth se realizará cargando el modelo de la red a través de su repositorio de github ⁹ especificando el modelo de la red neuronal y la opción 'pretrained' colocada con valor True, diciendo que cargaremos los pesos de formato pytorch del archivo End-to-end.pth.

```
import torch

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

Código 4.5: Cargar modelo YOLOP con pesos preentrenados End-to-end.pth

Una vez tengamos el modelo cargado del repositorio, podemos escoger si queremos hacer la inferencia en la CPU o GPU, tendremos que especificar como lo queremos hacer. En nuestro, elegiremos la opción de GPU para realizar la inferencia y le asignaremos al modelo que realizaremos la inferencia por GPU.

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

Código 4.6: Cargar modelo YOLOP escogiendo como dispositivo la GPU

⁸http://wiki.ros.org/cv_bridge

⁹<https://github.com/hustvl/YOLOP>

Por último, nos quedaria convertir la imagen en un tensor antes de realizar la inferencia. Para poder realizarlo, utilizaremos la funcion transforms.toTensor¹⁰ y después añadiremos una dimensión más con unsqueeze(0) , esto se debe a que el modelo de YOLOP espera una entrada específica llamada batch_size. Batch_size es el número de imágenes que se procesarán juntas (generalmente 1 para inferencia) por lo que el tensor tendrá esta forma: (batch_size, channels, height, width)

```
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 obtendremos un tensor de salida que corresponde la probabilidad de detección de la segmentación de la calzada y la detección de las líneas de la calzada. Dicho tensor habrá que convertirlo en una imagen para poder visualizar dicho resultado como una imagen. Por ello, convertiremos el tensor en un array numpy y realizaremos una transformacion para cambiar las dimensiones de (H,W,C) a (C,H,W) esto se realiza ya que en Opencv representa imágenes en formato Numpy array y se transpone las dimensiones porque las imágenes de Opencv tiene la forma de (H, W, C)¹¹.

Obtendremos un array de Numpy que normalizaremos con valores de 0-1. La normalización ayudar a igualar la escala de los pixeles de la imagen. Finalmente se podrá mostrar el resultado de la inferencia de la red en una imagen. Al normalizar los valores de 0-1, los pixeles con un valor 1 se mostrarán en la imagen final, asignándoles un color para visualiza el resultado de la segmentación de la calzada y de la detección de las líneas de la calzada.

¹⁰<https://pytorch.org/vision/main/generated/torchvision.transforms.ToTensor.html>

¹¹<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

Para poder utilizar los pesos preentrenados de Onnx, tendremos que realizar la configuración de los drivers de CUDA con la versión disponible para Onnx Runtime, dichas versiones tienen que ser compatible entre sí. Para ello nos guaremos por la tabla requisitos de la página oficial de Onnx Runtime¹².

Cuando tengamos todo preparado y configurado ya podremos realizar los pasos de la inferencia con los pesos preentrenados de Onnx. En primer lugar,cargaremos el modelo

```

import onnxruntime as ort

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

```

Código 4.9: Cargar modelo por ejemplo YOLOP-320-320.onnx

Como se observa en el código 4.9, a la hora de cargar el modelo escogemos como provider CUDAExecutionProvider. Cuando trabajamos con ONNX Runtime, podemos especificar qué proveedores de ejecución utilizar para ejecutar el modelo ONNX que escojamos, cada proveedor contiene un conjunto de núcleos optimizados para un

¹²<https://onnxruntime.ai/docs/execution-providers/CUDA-ExecutionProvider.html>

objetivo específico (por ejemplo, CPU, GPU, IoT) y se especifican como una lista en el orden de prioridad. En nuestro caso escogeremos CUDA para ejecutar mediante la GPU.

A continuación preprocesaremos las imágenes de entrada que le daremos al modelo, al escoger el modelo de yolop-320-320.onnx, las imágenes deben tener una dimensión de 320x320, para ello utilizaremos un método implementado por ellos que se puede encontrar en la siguiente página¹³, en el cuál redimensionaremos la imagen como se pide para el modelo. Una vez que tengamos la imagen preparada, daremos comienzo a la inferencia del modelo de yolop-320-320.onnx.

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

Código 4.10: Inferencia del modelo yolop-320-320.onnx

La inferencia del resto de modelos de Onnx se realizan de la misma forma pero teniendo en cuenta que se tendrá que cambiar las dimensiones de las imágenes de entrada y la ruta en donde se almacena dicho modelo.

Resultados de YOLOP

En esta sección contrastaremos los resultados de los diferentes pesos preentrenados que ofrece la red neuronal YOLOP. Como podemos observar en la figura 4.7 mostramos la media en realizar la inferencia de YOLOP utilizando los pesos preentrenados End-to-end.pth, yolop-320-320.onnx,yolop-640-640.onnx e yolop-1280-1280.onnx en segundos.

¹³<https://debuggercafe.com/yolop-onnx-inference-on-cpu>

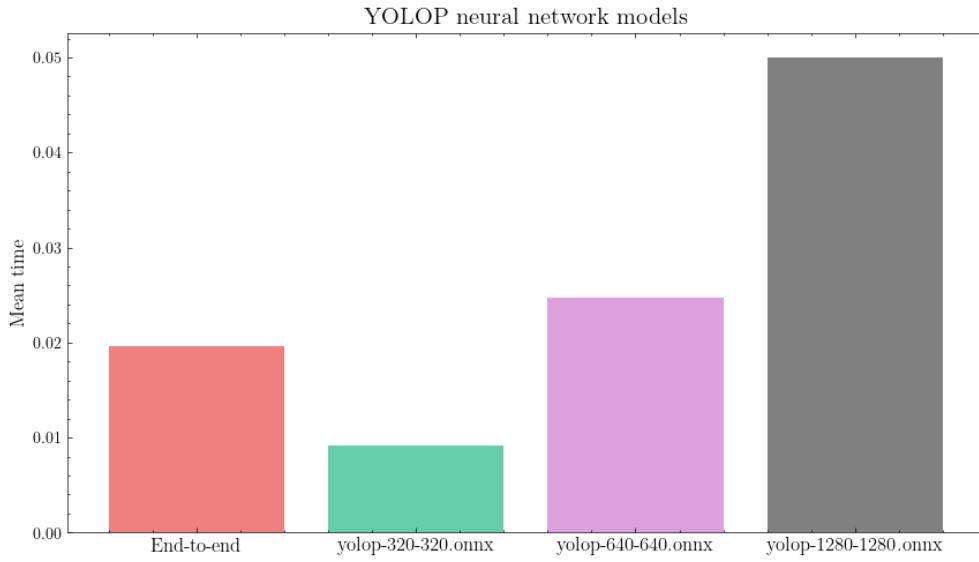


Figura 4.7: Resultados de los pesos preentrenados del modelo YOLOP

Los pesos preentrenados que tiene una inferencia menor al resto se trata de yolop-320-320.onnx, tiene un tiempo de inferencia alrededor de 0.010s, esto significa que el modelo YOLOP con estos pesos tiene aproximadamente un rate de 100 FPS. Si comparamos este resultado con los restantes pesos, 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 en cuanto Pytorch, lo que puede mejorar la velocidad y la precisión del modelo, añadiendo de tener 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, nosostros buscamos un equilibrio entre velocidad de inferencia y mejores resultados en cuanto a la detección de las líneas de la calzada. Por ello nos descantaremos por escoger el modelo yolop320-320.onnx ya que obtiene mejores resultados para poder realizar la detección de las líneas del carril que queremos seguir como se aprecia en la figura 4.8.

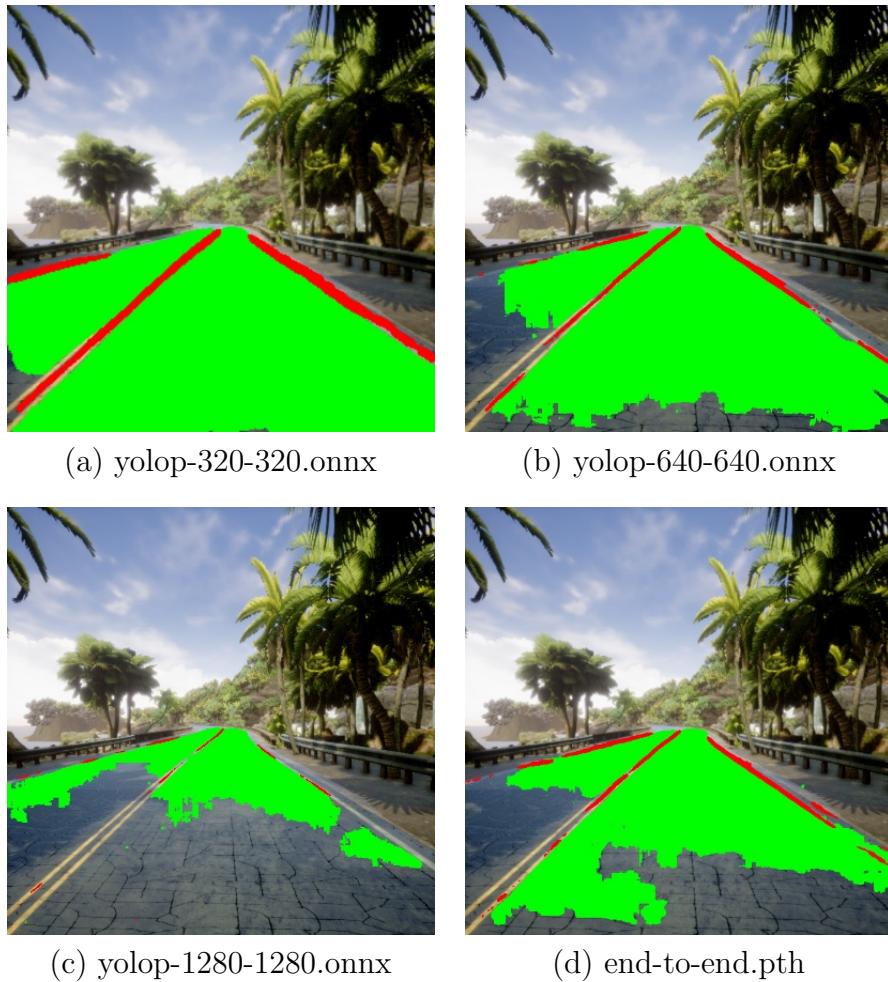


Figura 4.8: Diferentes resultados utilizando la red neuronal YOLOP en el entorno de simulación

Finalizado este proceso, procederemos a seleccionar las líneas detectadas que nos interesan mediante un algoritmo de aprendizaje no supervisado denominado clustering.

4.3.2. DBSCAN

Para saber con qué líneas detectadas por la red nos quedaremos, utilizaremos un algoritmo de aprendizaje no supervisado llamado clustering, en particular utilizaremos DBSCAN(Density-Based Spatial Clustering of Applications with Noise)[1].

Dicho algoritmo contiene varios parámetros que son importantes conocerlos y configurarlos:

- 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.

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.
3. **Metric:** La métrica utilizada para calcular la distancia entre los conjuntos de clústeres (por defecto es la distancia euclídea).

Los valores de los parámetros de la distancia máxima y el número mínimo de muestras se deben elegir cuidadosamente, es decir, si colocamos una distancia máxima alta puede provocar que los clústeres que queramos que pertenezcan a un distinto grupo pertenezcan al mismo grupo. Si min_samples se establece en un valor alto, DBSCAN encontrará clústeres más densos y si se establece en un valor bajo, los clústeres encontrados serán más dispersos.

Para entender como funciona el algoritmo, en la figura 4.9 recogida por este artículo[13] se ilustra un ejemplo teniendo un número de muestras ubicadas aproximadamente cercanas unas de otras. Aplicamos el algoritmo de DBSCAN teniendo como valor el número mínimo de muestras 3, eso significa que para que se considere una muestra a un grupo de cluster tendrá que tener una densidad de 3. El algoritmo itera por las muestras y compara el valor de la distancia máxima y el número mínimo de muestras. Si ningún de estos dos parámetros no se cumpliese ya que puede darse que una muestra se encuentra lejana del grupo de muestras será etiquetado como ruido, esto quiere decir que no pertenecerá a ningún grupo de clústeres. El resultado del algoritmo ha catalogado dos grupos de clústeres y 3 muestras.

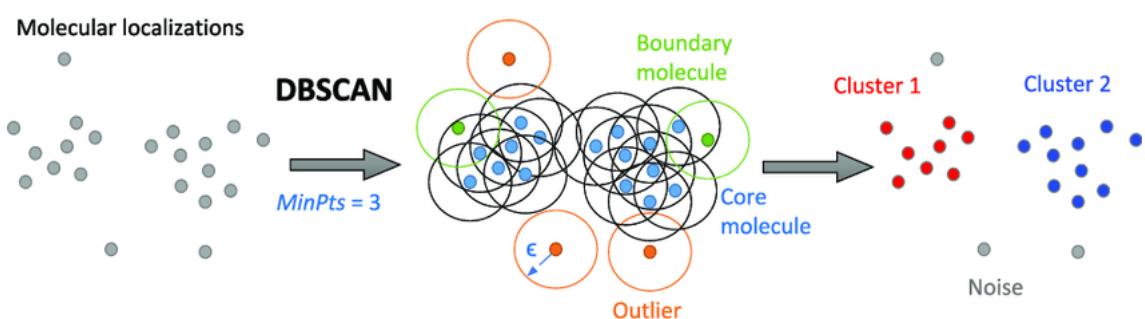


Figura 4.9: Ejemplo ilustrativo de como funciona el algoritmo de DBSCAN [13]

Por lo que para encontrar los valores de estos dos parámetros durante el desarrollo del TFG fue experimental y quedandonos con el mejor resultado. La distancia máxima tendrá un valor de 10 y el número mínimo de muestras será de 5 muestras. Lo que significa que la distancia máxima que tendrán los puntos para pertenecer al mismo

grupo de clústeres será de 10 de distancia en pixeles con un mínimo de muestras pertenecientes de 5 muestras. Para poder aplicar este algoritmo a partir del resultado de la imagen de la red neuronal se tendrá que convertir la imagen resultante de la red neuronal en un array bidimensional de coordenadas x e y de puntos con la ayuda de la función de Numpy column_stack¹⁴ , transformando un array simple de una dimensión a un array de dos dimensiones. Una vez se realice lo anterior, obtendremos un array bidimensional listo para el algoritmo DBSCAN.

```

def clustering(self,cv_image):

    --Convert image in points
    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_

        # Ignore noise if present
        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 4.11: Algoritmo de custering utilizando DBSCAN

El resultado de DBSCAN es una lista con etiquetas de 0 a n, siendo 0 el primer grupo de clusters detectado y n el último grupo de clusters detectado. De las listas de etiquetas se eliminarán los clusters que hallan sido etiquetados como ruido (-1). Para obtener el resultado solamente bastará iterar en dicha lista y mostrar los puntos correspondientes de cada etiqueta en la imagen de salida. Visualmente se puede apreciar en la figura 4.10 los diferentes resultados que tiene este algoritmo en localizaciones distintas.

¹⁴https://numpy.org/doc/stable/reference/generated/numpy.column_stack.html

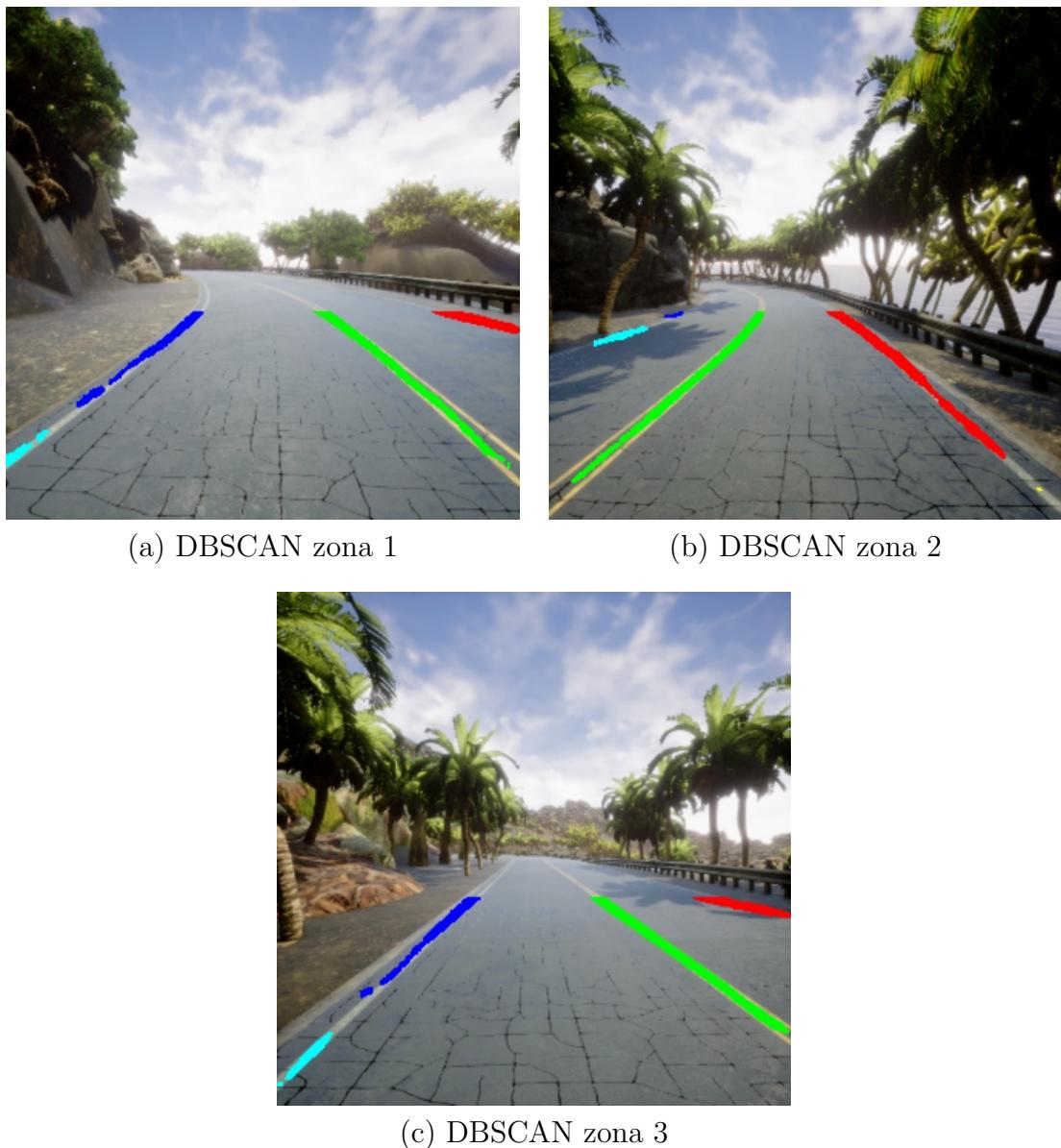


Figura 4.10: Diferentes resultados utilizando el algoritmo de DBSCAN

4.3.3. Clasificación de clústeres

Una vez realizado el algoritmo de DBSCAN necesitaremos quedarnos con el grupo de clústeres que pertenezcan al carril que queremos seguir. Para conseguirlo, se calcula los centroides de cada grupo de clústeres detectado y los clasificaremos en función de si se encuentran en la derecha o izquierda respecto al centro de la imagen. La imagen tiene unas dimensiones de 320x320 píxeles, siendo su centro (160,160) pixeles, $x = 160$ e $y = 160$. Nos fijaremos solamente en los valores de x de los centroides para poder clasificarlos en función del centro de la imagen, a partir de ello serán divididos en dos grupos distinguidos: derecha e izquierda de la imagen.

```
# Check if the centroid is within the desired lane

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 4.12: Clasificación de clústeres respecto a las dimensiones de la imagen

Cuando sean clasificado los grupos de los clústeres, necesitamos quedarnos con qué subgrupo de cada grupo de clústeres (izquierda e derecha) nos queremos quedar respecto al carril. Escogeremos dichos grupos mediante una función maximizada respecto a un punto central P predefinido con valor (220,160) siendo 220 el valor de las y e 160 el valor de las x, dichos valores son calculados respecto a las dimensiones de la imagen 320x320 píxeles. Es importante mencionar que cuando trabajamos con puntos en numpy las coordenadas estan opuestas, es decir, en vez de tener el formato (x,y) como estamos acostumbrados a trabajar tienen el formato(y,x). A parte de escoger lo sugrupos respecto a un punto central P también se realizará en función de la densidad de puntos de dicho grupo de clústeres de la derecha u izquierda detectados, con esto conseguimos que no solamente escojamos en función de la cercanía si no que también lo haremos según la cantidad de puntos.

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

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

Código 4.13: Función maximizada para escoger el grupo de cluster más cercano y denso respecto al punto P

En la figura 4.11 se muestra todo el proceso desde el uso del algoritmo de DBSCAN hasta la clasificación de clústeres respecto al carril. Después de tener la clasificación de los grupos de clústeres procederemos a realizar dos regresiones cuadráticas para construir las líneas del carril que queremos seguir.

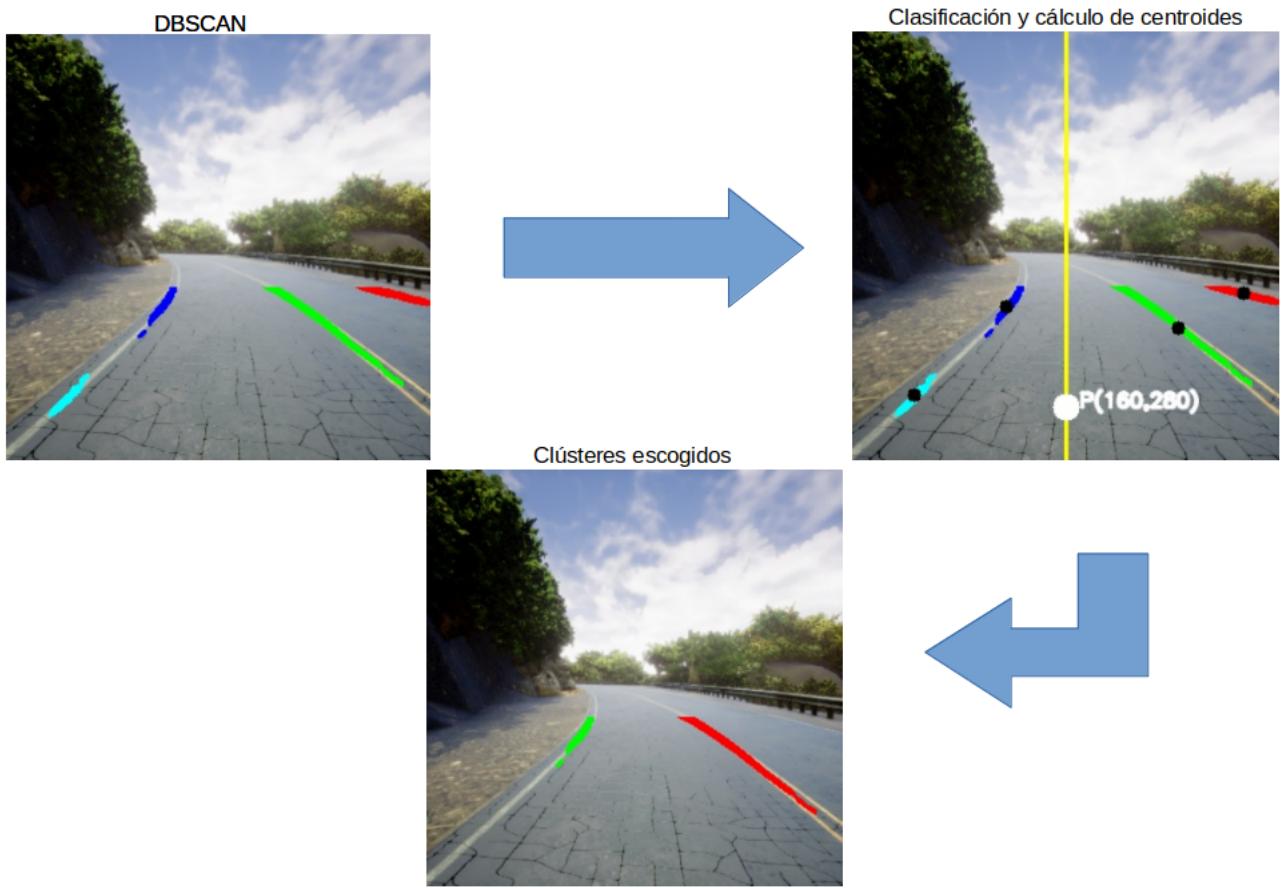


Figura 4.11: Ilustración del proceso de clasificación de líneas detectadas respecto al carril que queremos seguir

4.3.4. Regresión cuadrática

Como anteriormente hemos mencionado, cuando se clasifique los clústeres correctamente daremos pie a la construcción de dos regresiones cuadráticas.

La regresión es un método de aprendizaje supervisado el cual consiste en aproximar un número N de puntos a una recta, curva, etc, en nuestro caso hemos escogido realizar una regresión cuadrática ya que el recorrido que vamos a realizar las líneas detectadas por la red neuronal no son totalmente rectas si no que se tratan de líneas curvilíneas por lo que la regresión cuadrática en este papel puede funcionar perfectamente. Para la construcción de las regresiones cuadráticas utilizaremos las funciones de la librería Numpy denominada Polyfit¹⁵ y Polyval¹⁶. Polyfit es una función que calcula

¹⁵<https://numpy.org/doc/stable/reference/generated/numpy.polyfit.html>

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

los coeficientes del polinomio que mejor se ajusta a los datos utilizando el método de los mínimos cuadrados para la ecuación cuadrática, obtendremos tres coeficientes, denominados a,b y c.

Con dichos coeficientes realizaremos una media de los últimos diez valores y por cada cinco iteraciones dicha media se volverá a calcular, este paso lo realizamos ya que queremos disminuir las oscilaciones causadas de las detenciones de la red neuronal. Una vez calculados los coeficientes, realizaremos la regresión cuadrática mediante la función Polyval, dicha función calcula la función cuadrática con los coeficientes calculados anteriormente.

Cuando obtengamos los valores de la función cuadrática, realizaremos una clasificación para quedarnos con los puntos obtenidos de dicha función los que se encuentren en el eje y entre los valores 0 hasta el máximo de la imagen mneos una unidad que sería 319, esto se debe hacer ya que dicha función como resultado puede dar números negativos realizando dicha regresión además de que en una imagen no se puede indexar con números negativos.

```

def calculate_cuadratic_regression(self,points_cluster,cv_image):

    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]])
    line = None
    extrem_point_line = None
    a = 0
    b = 0
    c = 0

    points_cluster = np.vstack((points_cluster,point))
    coefficients = np.polyfit(points_cluster[:,0],points_cluster[:,1],2)

    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_right_coeff_a[-10:])
    b = np.mean(self.list_right_coeff_b[-10:])
    c = np.mean(self.list_right_coeff_c[-10:])

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

    self.counter_it_right += 1

    if(self.counter_it_right > 5):
        self.list_right_coeff_a.clear()
        self.list_right_coeff_b.clear()
        self.list_right_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)

    return line

```

Código 4.14: Función del cálculo de la regresión cuadrática

Este proceso se realizará 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 realizará una dilatación de los puntos

para tener un resultado más llamativo y visual al poder ver las regresiones cuadráticas como se muestra en la figura 4.12.



Figura 4.12: Resultado de la regresión cuadrática

4.3.5. Interpolación y cálculo del centro de masas del carril

Para poder saber qué puntos se encuentran dentro de ambas regresiones cuadráticas, realizaremos una interpolación. La interpolación consiste en recorrer los puntos de la imagen original y quedarnos con los puntos que se encuentren dentro de los límites de ambas regresiones. Se realizará dos interpolaciones, una interpolación para los puntos de la regresión de la derecha y otra interpolación para los puntos de la regresión izquierda. Estas funciones interpolan los valores de los puntos en y en función de los valores de los puntos en x, más adelante se evaluará los valores de las puntos que se encuentren entre ambas regresiones cuadráticas. Cuando realizemos dicha evaluación realizaremos un filtro para seleccionar el rango de puntos que representarán un fragmento del carril de color azul en la imagen final. Visualizando el resultado en la figura 4.13

```
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 4.15: Método del cálculo de las funciones de interpolación



Figura 4.13: Resultado de la interpolación

Finalmente, cuando obtengamos el cálculo del carril que queremos seguir solamente faltarán calcular el centro de masas de dicho fragmento conocido como el centroide siguiendo la ecuación del cálculo de centro de masas de una superficie:

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

Supondremos que todos los puntos tienen la misma masa ($m_i = 1$). Esto simplifica el cálculo, pero en aplicaciones del mundo real, las masas pueden variar. El segundo paso será el cálculo de la masa total, se calcula multiplicando la masa individual (m_i) por la cantidad de puntos en el carril. A continuación calcularemos la suma de las posiciones de los puntos ponderadas por su masa y dividimos por la masa total calculada anteriormente. El resultado es una centro de masas conocido como centroide con coordenada x e y como se muestra en la figura final 4.15.



Figura 4.14: Resultado del centro de masas

4.3.6. Análisis del algoritmo de percepción

Por último, realizamos un análisis de tiempos de cada parte para saber cuánto tiempo tarda en realizarse la percepción completa. Dicho análisis se llama profiling, con esta técnica nos ayuda ver el rendimiento que podemos llegar a tener análisis global del algoritmo de percepción en cuanto a eficiencia. En la figura 4.15, se muestra cuánto tarda cada parte que compone la percepción en segundos.

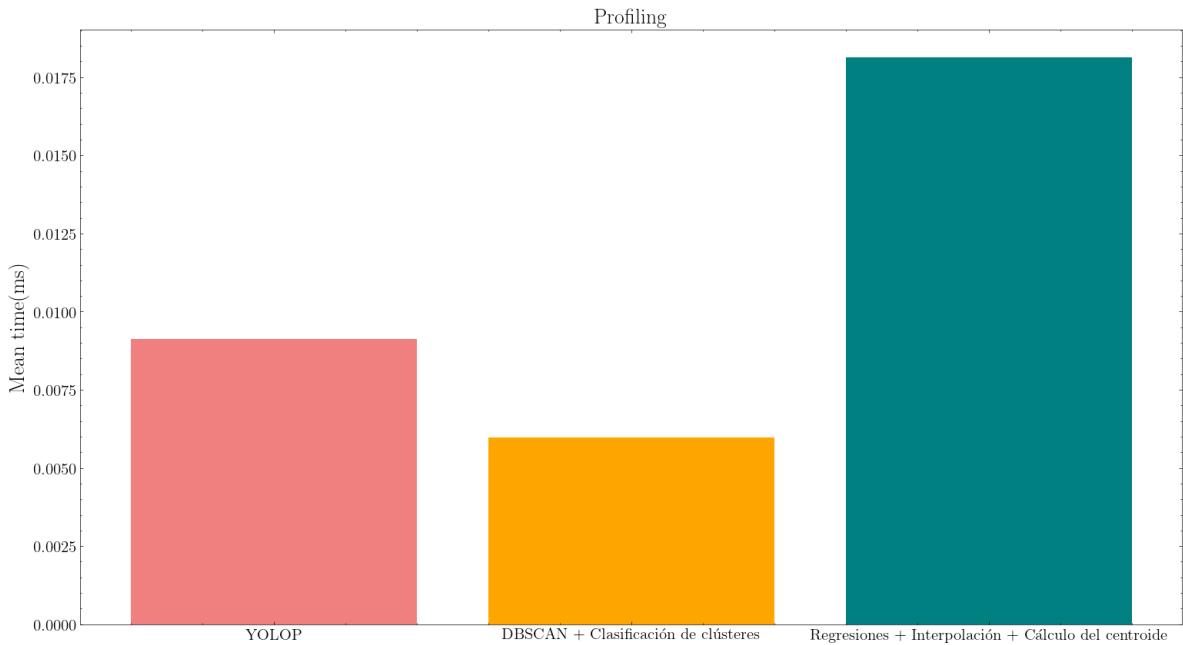


Figura 4.15: Profiling de las partes de la percepción

Como podemos apreciar, la parte de la percepción que tarda más es el cálculo de la regresiones cuadráticas, la interpolación y el cálculo del centroide. Esto se debe a que cuando se realiza el cálculo de las regresiones hay que recorrer los puntos de los grupos de clústeres y calcular sus funciones para obtener las regresiones. Respecto al tiempo que tarda el algoritmo de DBSCAN junto con la clasificación de los clústeres es el ganador en cuanto menor tiempo de cómputo lo cual es interesante trabajar con dicho algoritmo de clustering. En conclusión, sumando todos los tiempos de cada parte obtenemos un tiempo global de 0.033 segundos pasado a FPS conseguimos un rate aproximadamente de 30 FPS.

4.4. Seguimiento del carril mediante control clásico

Una vez realizado el comportamiento de la percepción, construiremos un comportamiento autónomo con el dron basandonos en un simple controlador PID. Con este primer comportamiento queremos demostrar la funcionalidad de la percepción utilizando un controlador de movimiento sencillo.

Un controlador PID (proporcional, derivativo e integral) es un sistema que es capaz de mantener una variable (como la temperatura, velocidad o posición) cercana a un valor deseado o de referencia. Cada componente de un controlador tiene un

papel importante, la componente P (proporcional) ajusta la salida del controlador en función de la diferencia entre el valor medido y el valor deseado. Cuanto mayor sea esta diferencia (error), mayor será la corrección aplicada. Sin embargo, el control proporcional solo no puede eliminar completamente el error por ello se utiliza las componentes derivativa e integral. La componente D (derivativa) considera la tasa de cambio del error, si el error cambia rápidamente, el término derivativo aplicará una corrección para evitar oscilaciones o inestabilidad. Con el término integral acumula el error a lo largo del tiempo y ajusta la salida del controlador en función de esta acumulación. Ayuda a eliminar el error persistente o constante. Si el error es pequeño pero persistente, el término integral lo corregirá gradualmente.

Lo cual con este controlador PID sencillo controlaremos las velocidades respecto al error que se produce entre posición deseada y la que obtenemos. La variable deseada para estar alineados con el carril escogeremos en el eje de las x el valor central de la imagen, ya que queremos en todo momento permanecer centrales respecto al carril. Para calcular el error realizaremos la diferencia del valor deseado que sería el valor central de la imagen y el valor del centroide del carril mencionado en la sección 4.3.5. A parte de este controlador, utilizaremos un pequeño controlador PD para controlar la altura del vehículo y tengamos una altura medianamente constante. Para saber a qué altitud nos encontramos utilizaremos el sensor del Lidar.

Para encontrar los valores de cada término que compone ambos controladores se ha realizado a base de experimentación, empezando de manera creciente, es decir, primero utilizaremos la componente proporcional para ver su comportamiento, una vez que tengamos el valor del término proporcional pasaremos al término derivativo para suavizar los movimientos que puede producir este término y por último utilizaremos la parte integral para eliminar el error. El controlador PID se usará para controlar la velocidad de giro del dron en cuanto a la velocidad lineal tendremos un valor constante.

4.5. Seguimiento del carril mediante aprendizaje por refuerzo

Como mencionamos en la sección 1.3, el aprendizaje por refuerzo consiste en enseñar un agente desempeñar un comportamiento mediante recompensas y penalizaciones. Este comportamiento se aprende a base de interacciones con el entorno de trabajo y

```

kp_height = 0.1
kd_height = 0.4

kp_speed_controller = 0.09
kd_speed_controller = 0.1
ki_speed_controller = 0.008

```

Código 4.16: Valores de las variables del PD del control de altura y del PID del controlador de velocidad angular

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. Se compone de diferentes elementos claves:

- **Agente:** El agente es una entidad o modelo que pretendemos entrenar para que aprenda a tomar decisiones (acciones) en función del estado en el que nos encontramos. Nuestro agente será el dron.
- **Entorno:** Ambiente en donde interactua el agente para que pueda aprender el comportamiento deseado. El dron estará interactuando sobre el entorno de Coastline que nos proporciona Airsim durante las fases de pruebas.



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

Dentro de este circuito, hemos escogido tres localizaciones distintas para el vehículo a la hora de realizar el entrenamiento con el algoritmo de QLearning.

Como se puede apreciar en la figura 4.16 las posiciones tienen una distancia entre ellas considerablemente.

- **Estados:** Condiciones en las que se puede encontrar el agente en ese instante de tiempo. Para la definición de los estados, dividiremos la imagen que damos como la salida de la detección del carril en 14 franjas, dichas franjas tendrán una separación de 10 pixeles. Las franjas representan cada estado en el que se puede encontrar el agente. Empezamos a definir el primer estado desde la izquierda hasta la derecha con una hasta conseguir los 14 estados correspondientes, 7 estados izquierda, 1 estado central y 6 estados derecha. Se puede observar el resultado de los estados en la figura 4.17

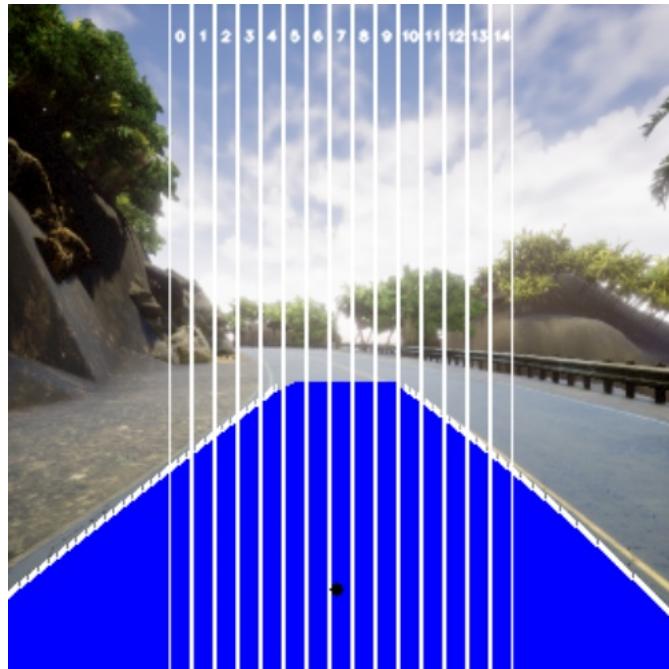


Figura 4.17: Estados definidos para el sigue carril basado en Q-Learning

- **Acciones:** Movimientos que puede realizar el agente dentro del estado en el entorno de entrenamiento. Tendremos en total 21 acciones que podrá escoger el agente en el algoritmo de Q-Learning, se componen de pares de velocidades lineales y angulares. Las velocidades lineales tendrán 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 serán formados a partir de la función que nos proporciona Numpy

llamada `linspace`¹⁷.

```
def build_actions():
    ACTIONS = []
    speeds_actions = np.linspace(0.1,2.0,11, dtype=float)
    angular_speeds = np.linspace(-25,25, 21)

    left_angular_speeds = angular_speeds[:10]
    right_angular_speeds = np.flip(angular_speeds[-10:])
    central_angular_speed = angular_speeds[10]
    speeds = speeds_actions[:10]
    central_speed = speeds_actions[10]

    for i in range(len(speeds)):
        ACTIONS.append([round(speeds[i],3),round(left_angular_speeds[i],3)])

    ACTIONS.append([round(central_speed,3),0.0])

    for i in reversed(range(len(speeds))):
        ACTIONS.append([round(speeds[i],3),round(right_angular_speeds[i],3)])

    return ACTIONS
```

Código 4.17: Construcción de las acciones para Q-Learning

- **Función de Recompensa o Penalizaciones:** Consiste en como queremos premiar o penalizar al agente con el objetivo que queremos cumplir. Las recompensas o penalizaciones se recogen en una función, dicha función es independiente en cada diseño del objetivo que quiere completar el agente. La función de recompensa constará de dos partes premiando al dron de que se permanezca centrado en el carril y además premiarle por mantener una orientación adecuada respecto al carril, asegurando que se navegue de forma paralela al mismo. Cada parte tendrá un peso de importancia, siendo el peso que se permanezca del carril del 85 % y la orientación respecto al carril de un 15 %. Tanto la recompensa por mantenerse centrado en el carril como la orientación respecto al carril será normalizada entre valores de 0-1, penalizando con un valor constante al dron cuando se salga del carril o se pierda la percepción. La implementación de la función de recompensa y las penalizaciones se puede ver el siguiente código:

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

```

def reward_function(self,cx,angle):

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

    MIN_ERROR = 0
    MAX_ERROR = 80

    MIN_ANGLE = 0
    MAX_ANGLE = 70

    CENTRE_WEIGHT = 0.85
    ANGLE_WEIGHT = 0.15

    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 4.18: Función de recompensa

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

Seguiremos la política epsilon-greedy[6], que consiste equilibrar la exploración y explotación en la fase de entrenamiento. Cuando el agente tenga que escoger la

acción que tomar tendrá en cuenta dos enfoques:

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

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 generará un número n aleatoriamente, si n es menor que la probabilidad ϵ la acción se escogerá aleatoriamente en cambio si n es mayor que la probabilidad de ϵ la acción será que mayor valor tenga en la tabla $Q(S,A)$ para dicho estado.

Dentro de esta política la probabilidad de ϵ será decayente, es decir, no ser un valor constante en cada episodio que nos encontraremos, lo que realizaremos será una disminución de esta probabilidad para conseguir que el agente con el paso del tiempo poco a poco explote lo que ha aprendido y que el modelo llegue a una convergencia óptima. Para llevar a cabo el descenso de ϵ se puede realizar de varias formas, por ejemplo podemos realizarlo linealmente, logarítmicamente, exponencial o escalonado.

El algoritmo Q-Learning se basa en seguir una función acción-recompensa formada por un tablero estado-acción la cual iremos llenando siguiendo la siguiente ecuación de Bellman[5]:

$$Q(s, a) = Q(s, a) + \alpha \cdot [R(s, a) + \gamma \cdot \max Q(s', a') - Q(s, a)] \quad (4.2)$$

en donde,

- $Q(s, a)$: Valor Q para el estado s y la acción a . Se trata de una matriz formada por estado y acción.
- 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 será 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 favorable o deseable es una acción tomada por el agente en el estado que se encuentre. Proporciona información al agente sobre qué acciones maximizan la recompensa total a lo largo del tiempo. Dicha función de recompensa es diseñada dependiendo de cuál 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 el agente valorará mucho las recompensas futuras, mientras que un valor bajo indica que se enfocará más en las recompensas inmediatas. Este parámetro tendrá un valor de 0.7.
- $\max Q(s', a')$: Valor Q máximo en el próximo estado s' para todas las acciones posibles a' .

A la medida que vayamos avanzando en la fase de entrenamiento, iremos llenando la tabla $Q(s, a)$ para más adelante indexar en ella en la fase de inferencia.

4.5.1. Fase de entrenamiento

Antes de adentrarnos en esta fase, definiremos dos conceptos importantes:

- **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 (steps)**: 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 en el 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 inicializará la tabla $Q(S, A)$ a cero todos sus valores, esta representación al comienzo es así ya que el agente desconoce por completo el entorno hasta que poco a poco vaya iterando sobre él en cada estado tomando una acción.

Basicamente, en cada iteración del algoritmo se escogerá una acción según si estamos en exploración o explotación, se calculará la recompensa obtenida e iteraremos de nuevo sobre el algoritmo.

El entrenamiento finaliza cuando el agente haya aprendido el objetivo que queremos seguir, esto se sabe cuando las iteraciones del algoritmo y la recompensa acumulada del agente en esta fase se estabilice y tenga valores constantes, es decir, no cambia significativamente con más iteraciones, esto se denomina que el modelo ha convergido y pasaremos a la fase de inferencia.

Durante esta fase, dejamos el algoritmo de Q-Learning entrenando durante aproximadamente 12 horas teniendo un rate del algoritmo de QLearning más el algoritmo de percepción aproximadamente de unos 10 FPS. De esta manera, nos podemos asegurar que el algoritmo completará el entrenamiento de manera correcta y poco a poco acabará convergiendo. En la figura 4.18 se ilustra la fase de entrenamiento durante la navegación del dron. En la primera figura se corresponde al número de iteraciones de cada episodio durante el entrenamiento, la segunda corresponde al valor que va teniendo épsilon en cada episodio y la tercera grafica se muestra el valor de la recompensa acumulativa en cada episodio.

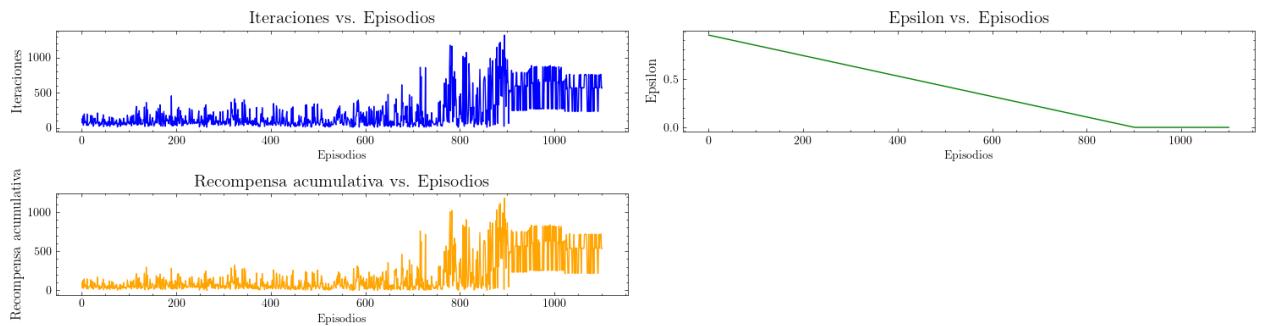


Figura 4.18: Gráficas de la fase de entrenamiento

Al comienzo del entrenamiento, se puede observar que el dron obtiene pocas iteraciones y bajas recompensas acumulativas, esto se debe a que el dron al comienzo del entrenamiento desconoce por completo el entorno. A medida que vamos avanzando en el entrenamiento, las iteraciones y los valores de la recompensa acumulativa son mayores hasta que en el episodio 900 donde acaba la fase de exploración siendo el valor de épsilon cero (las acciones no son aleatorias). A partir de ese punto, la recompensa y las iteraciones tienen una tendencia ascendente, y finalmente, a lo largo del entrenamiento, ambas se estabilizan en un valor fijo sin cambiar considerablemente. Esto se debe a que el dron completa constantemente el circuito en cualquier punto de reinicio dentro del circuito sin cambiar considerablemente la tabla Q. En este momento, se puede decir

que el algoritmo ha sido capaz de converger. Una vez se analice los resultados del entrenamiento con las diferentes métricas daremos pie a la fase de inferencia para verificar el resultado obtenido con el modelo entrenado.

4.5.2. Fase de inferencia

La fase de inferencia consiste en indexar la tabla $Q(S,A)$ que hemos ido rellenando 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 4.5.2 se puede observar las diferentes acciones que ha utilizado el dron para completar el circuito de entrenamiento utilizando cuatro de veintiuno de acciones disponibles. 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 angulo de orientación adecuado sin salirse del carril.

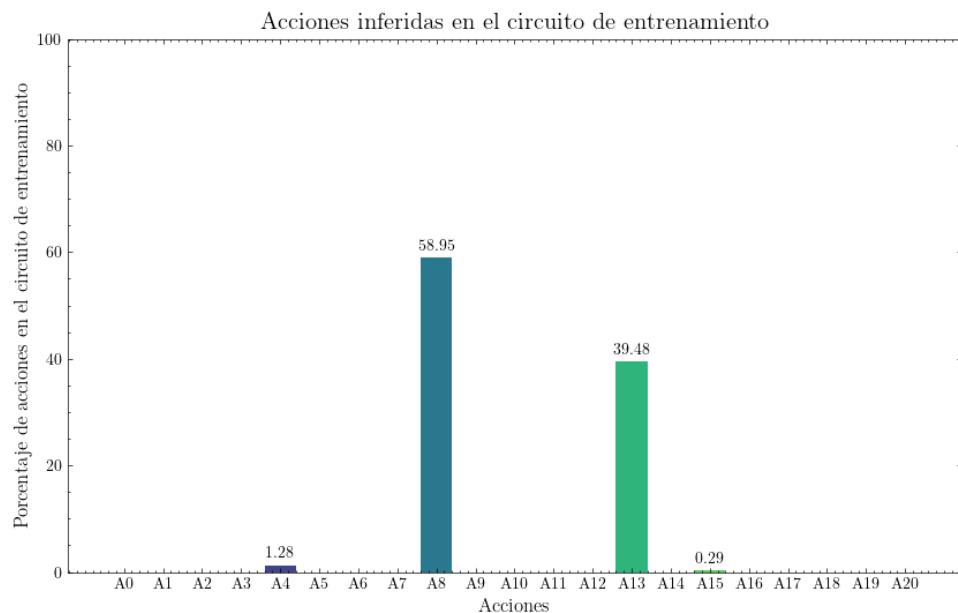


Figura 4.19: Distribución de acciones en el circuito de entrenamiento

Como resultado al utilizar el algoritmo de Q-learning, el modelo entrenado ofreció ser un resultado eficaz a la hora de navegar por el circuito, pudiendo observar el modelo entrenado final en diferentes localizaciones aleatorias en el entorno de la figura 4.16. Para poder obtener el máximo rendimiento del modelo entrenado se busco los límites de velocidad angular y lineal a la que podía ir el dron, esto significa multiplicar por un factor todas las acciones tanto las velocidades lineales y angulares. Este proceso es interesante realizarlo para maximizar el comportamiento sin la necesidad de volver a entrenar un nuevo modelo. Estos factores se pueden ver en el código 4.19 lo cual las acciones se pueden aumentar un 100 % lo que podemos decir que el modelo entrenado es efectivo al aumentar el valor de las acciones.

```
factor_speed = 2.0
factor_angular_speed = 1.8
```

Código 4.19: Valores de los factores aplicados a las acciones de Q-learning

Siendo los nuevos valores de las acciones utilizados durante las fases de comparación entre el seguimiento de carril con el PID y el seguimiento de carril mediante Q-learning. Los nuevos valores de las acciones originales como se mostraba en el código 4.17 son multiplicados por ambos factores y su gráfico de acciones en la figura 4.20, se puede observar que en la distribución de acciones 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 que se mostraba en la figura

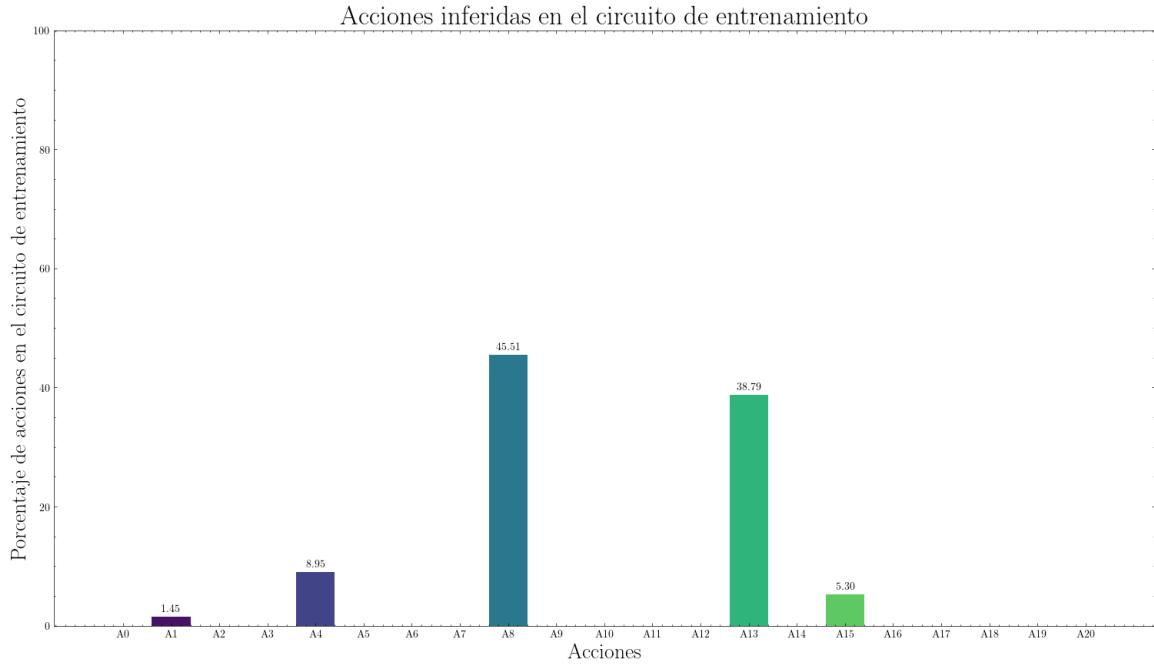


Figura 4.20: Distribución de acciones en el circuito de entrenamiento multiplicadas por el factor

4.5.3. Análisis y comparativa entre el seguimiento de carril clásico

Para comprobar la robustez y la eficacia que demuestra ser el modelo entrenado junto con el seguimiento de carril mediante el PID, se realizó varias comparativas en ambos comportamientos. Se ajustó las velocidades del controlador PID para que ambos tuvieran las mismas condiciones, para ello se utilizó un circuito en el que ambos son capaces de recorrerlo sin salirse del carril. En la figura 4.21 se muestra el resultado de ambos comportamientos, en la primera figura es todo el recorrido hasta el punto final y en la segunda figura se realiza zoom en una parte del recorrido, los datos fueron recogidos mediante la posición del dron guardándolos en un fichero tanto las coordenadas en el eje x como las coordenadas en el eje y.

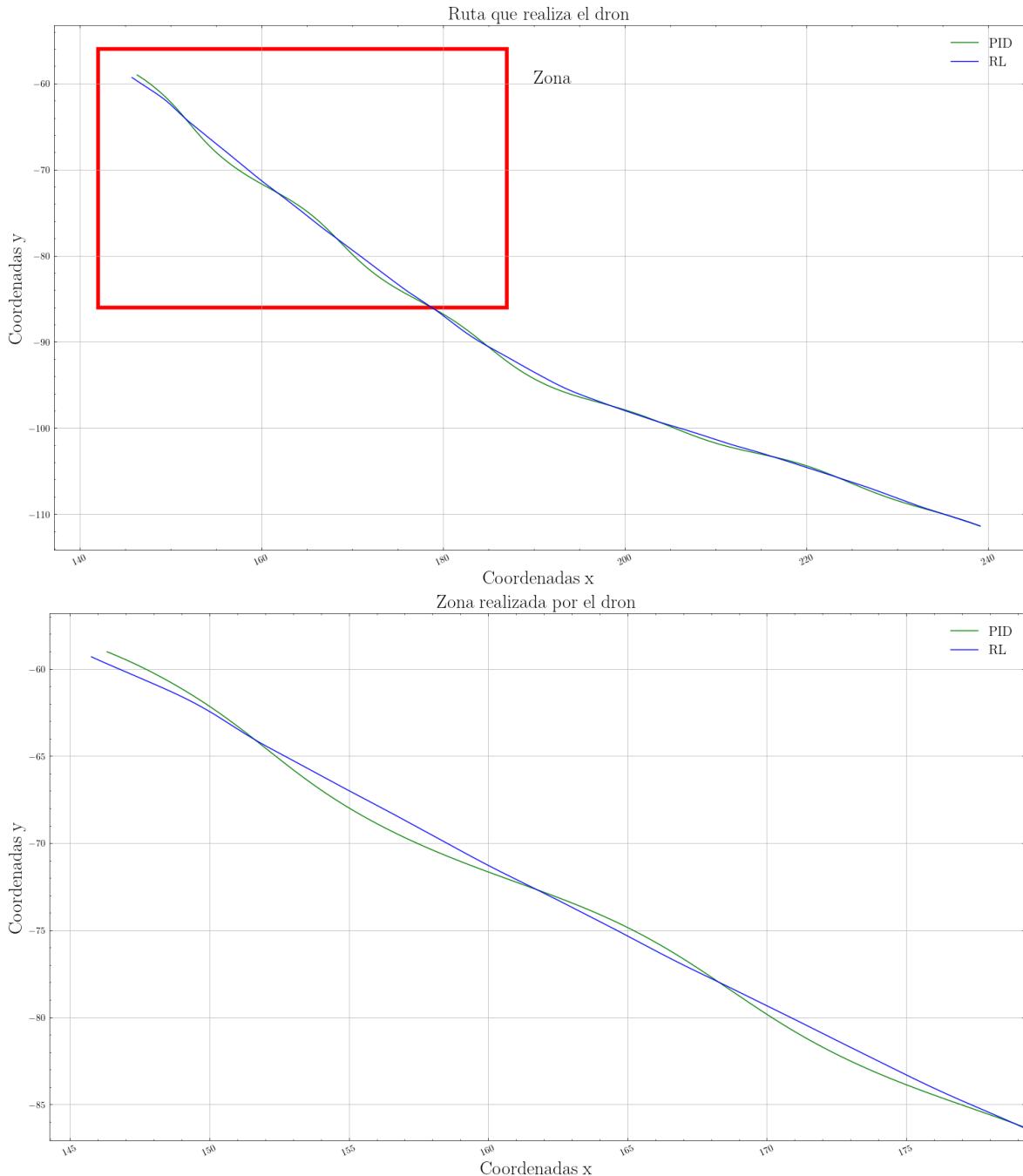


Figura 4.21: Comparativa realizando un trayecto entre ambos comportamientos

Una de las principales diferencias entre ambos comportamientos es que el comportamiento con control clásico presenta oscilaciones durante el recorrido realizando eses por ambas partes del recorrido hasta llegar al punto final del recorrido. En cambio, el comportamiento con aprendizaje por refuerzo destaca por su trayectoria constante durante todo el recorrido sin apenas realizar oscilaciones, destacando que el trayecto para realizar la comparativa se trata de un trayecto recto sin apenas curvas como se muestra en la figura .

Finalmente, en la figura se ilustra la media de velocidades angulares que han

obtenido ambos comportamientos en el recorrido que se muestra en la figura . Se puede apreciar como el comportamiento de aprendizaje por refuerzo obtiene menor media de velocidad angular respecto al controlador PID manteniendo un trayectoria segura y eficaz en un recorrido que nunca ha visto durante la fase de entrenamiento. Demostrando así que el modelo entrenado es capaz de obtener comportamientos satisfactorios tomando acciones acordes al recorrido.

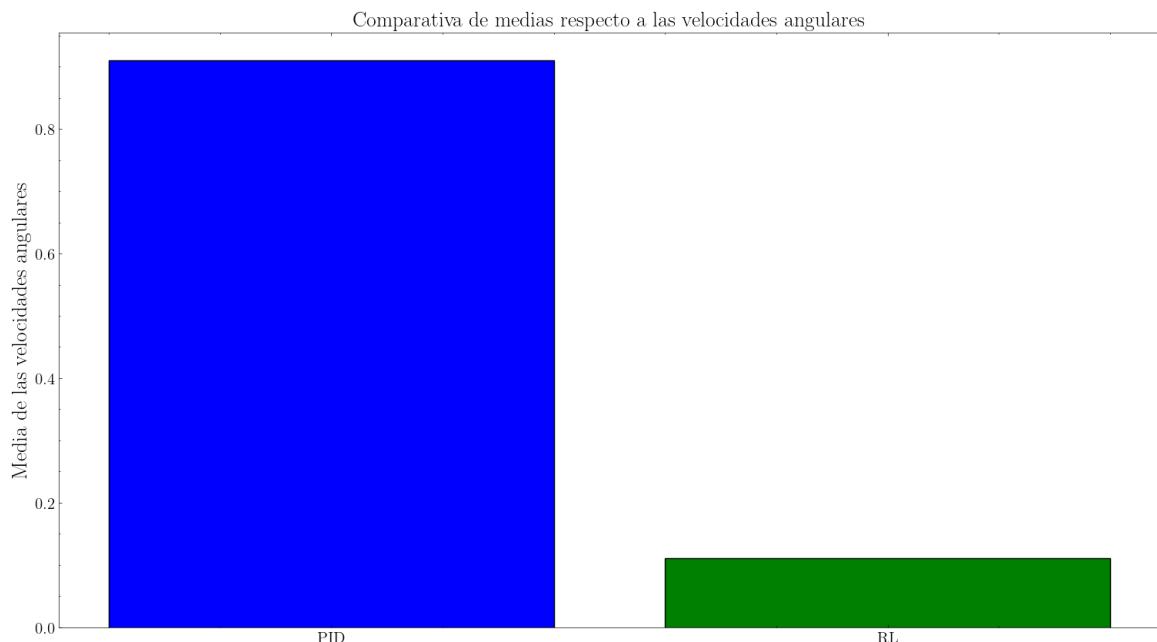


Figura 4.22: Media de velocidades angulares de ambos comportamientos desarrollados

Capítulo 5

Conclusiones

Quizás algún fragmento de libro inspirador...

Autor, Título

Escribe aquí un párrafo explicando brevemente lo que vas a contar en este capítulo, que básicamente será una recapitulación de los problemas que has abordado, las soluciones que has prouesto, así como los experimentos llevados a cabo para validarlos. Y con esto, cierras la memoria.

5.1. Conclusiones

Enumera los objetivos y cómo los has cumplido.

Enumera también los requisitos implícitos en la consecución de esos objetivos, y cómo se han satisfecho.

No olvides dedicar un par de párrafos para hacer un balance global de qué has conseguido, y por qué es un avance respecto a lo que tenías inicialmente. Haz mención expresa de alguna limitación o peculiaridad de tu sistema y por qué es así. Y también, qué has aprendido desarrollando este trabajo.

Por último, añade otro par de párrafos de líneas futuras; esto es, cómo se puede continuar tu trabajo para abarcar una solución más amplia, o qué otras ramas de la investigación podrían seguirse partiendo de este trabajo, o cómo se podría mejorar para conseguir una aplicación real de este desarrollo (si es que no se ha llegado a conseguir).

5.2. Corrector ortográfico

Una vez tengas todo, no olvides pasar el corrector ortográfico de L^AT_EXa todos tus ficheros *.tex*. En Windows, el propio editor TeXworks incluye el corrector. En Linux, usa aspell ejecutando el siguiente comando en tu terminal:

```
aspell --lang=es --mode=tex check capitulo1.tex
```

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://airandspace.si.edu/multimedia-gallery/web12070-2011640.jpg
1.3	https://www.whoi.edu/oceanrobots/robots/nereus-phone.html
1.4	https://www.swissinfo.ch/spa/ciencia/drones-suizos-al-rescate/46203902
1.5	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=AOvVaw20cBlgMDH1HVU5qsiJ9_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=AOvVaw1q3J6eh2YCyEUzy1QM9z_K&ust=1714152091161000&source=images&cd=vfe&opi=89978449&ved=OCBQQjhxqFwoTCJDyqXw3YUDFQAAAAAdAAAAABAE 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=AOvVaw2csv7vma6UxJokGv1G8j7h&ust=1714152048517000&source=images&cd=vfe&opi=89978449&ved=OCBQQjhxqFwoTCLiCtZh3YUDFQAAAAAdAAAAABAE https://www.xataka.com/espacio/helicoptero-ingenuity-ha-terrizado-lugares-marte-que-nasa-se-esta-quedando-letras-para-nombrarlos
1.6	https://www.xataka.com/espacio/helicoptero-ingenuity-ha-terrizado-lugares-marte-que-nasa-se-esta-quedando-letras-para-nombrarlos

1.7	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-drone-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.mdpi.com/2072-4292/15/3/615
1.13	https://www.researchgate.net/publication/378676909_The_prospect_of_artificial_intelligence_to_personalize_assisted_reproductive_technology
1.14	https://revistas.rcaap.pt/rca/article/view/34973/24651
1.15	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

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 Dhillon, 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 (2021). 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.
- [19] Xue, Z. and Gonsalves, T. (2021). Vision based drone obstacle avoidance by deep reinforcement learning. 2:366–380.
- [20] 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.
- [21] 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.
- [22] Zhu, Y. and Tang, H. (2023). Automatic damage detection and diagnosis for hydraulic structures using drones and artificial intelligence techniques. *Remote Sensing*, 15(3).