

Máster Universitario en Robótica y Automatización
2021-2022

Trabajo Fin de Máster

**Middleware y aplicaciones visuales para
robots aéreos con aprendizaje profundo**

Pedro Arias Pérez

Tutor/es

David Martín Gómez

José María Cañas Plaza

Lugar y fecha de presentación prevista

DETECCIÓN DEL PLAGIO

La Universidad utiliza el programa **Turnitin Feedback Studio** para comparar la originalidad del trabajo entregado por cada estudiante con millones de recursos electrónicos y detecta aquellas partes del texto copiadas y pegadas. Copiar o plagiar en un TFM es considerado una **Falta Grave**, y puede conllevar la expulsión definitiva de la Universidad.



Esta obra se encuentra sujeta a la licencia Creative Commons **Reconocimiento - No Comercial - Sin Obra Derivada**

RESUMEN

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetur adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Palabras clave:

DEDICATORIA

ÍNDICE GENERAL

| | |
|--|----|
| 1. INTRODUCCIÓN | 1 |
| 1.1. ROBÓTICA AÉREA | 1 |
| 1.2. MOTIVACIÓN | 4 |
| 1.3. PROBLEMA | 5 |
| 1.4. OBJETIVOS | 5 |
| 1.5. ESTRUCTURA DE LA MEMORIA | 6 |
| 2. HERRAMIENTAS | 8 |
| 2.1. SEGMENTO TIERRA | 8 |
| 2.1.1. ESTACIÓN DE CONTROL TERRESTRE | 9 |
| 2.1.2. TOOLKITS DE VUELO | 10 |
| 2.1.3. PLATAFORMAS ROBÓTICAS DE SOFTWARE | 11 |
| 2.1.4. APRENDIZAJE PROFUNDO | 11 |
| 2.2. SEGMENTO AIRE | 12 |
| 2.2.1. AERONAVES REALES | 13 |
| 2.2.2. AERONAVES SIMULADAS | 14 |
| 2.3. PROTOCOLO DE COMUNICACIONES | 15 |
| 3. MATERIAL Y MÉTODO | 19 |
| 3.1. HARDWARE | 19 |
| 3.1.1. AERONAVES | 19 |
| 3.1.2. Dispositivos Embebidos | 21 |
| 3.1.3. Sensores de visión | 23 |
| 3.2. SOFTWARE | 24 |
| 3.2.1. Simulación | 25 |
| 3.2.2. NVIDIA JetPack | 27 |
| 3.2.3. Aprendizaje profundo | 27 |
| 3.3. Método | 27 |
| 4. INFRAESTRUCTURA DESARROLLADA | 29 |

| | |
|--|----|
| 5. APLICACIÓN | 30 |
| 5.1. Diseño. | 30 |
| 5.1.1. Percepción | 30 |
| 5.1.2. Control | 31 |
| 5.2. Implementación | 31 |
| 5.3. Experimentos | 31 |
| 5.3.1. Base | 31 |
| 5.3.2. Sigue color | 32 |
| 5.3.3. Sigue persona. | 32 |
| 6. CONCLUSIONES Y LÍNEAS FUTURAS | 34 |
| 6.1. Conclusiones | 34 |
| 6.2. Líneas futuras. | 34 |
| BIBLIOGRAFÍA | 35 |

ÍNDICE DE FIGURAS

| | | |
|------|--|----|
| 1.1 | Clasificación de UAVs | 2 |
| 1.2 | Ejemplos de UAVs | 3 |
| 1.4 | Aeronave de vigilancia en carretera perteneciente a la Dirección General de Tráfico | 3 |
| 2.1 | Transmisor <i>Taranis Q X7S OpenTX</i> | 8 |
| 2.2 | Módulo receptor <i>X-Rock V5</i> | 9 |
| 2.3 | QGroundControl | 10 |
| 2.4 | Arquitectura perceptrón multicapa | 12 |
| 2.5 | Bucle de control de un UAV | 13 |
| 2.6 | Ejemplos de drones de fabricantes propietarios | 14 |
| 2.8 | Diferentes modelos de controladoras PixHawk | 15 |
| 2.10 | Drone simulado mediante <i>jMAVsIm</i> | 15 |
| 3.1 | DJI Tello | 20 |
| 3.2 | Drone no comercial | 21 |
| 3.3 | NVIDIA Jetson AGX Xavier | 22 |
| 3.4 | Cámara Victure AC600 | 23 |
| 3.5 | Aeronave con cámara instalada | 24 |
| 3.6 | 3DR Iris simulado mediante <i>Gazebo</i> | 26 |
| 3.7 | Esquema de PX4 sobre SITL | 26 |

ÍNDICE DE TABLAS

| | |
|---|----|
| 2.1 Principales mensajes de MAVLink. | 18 |
| 3.1 Especificaciones del DJI Tello. | 20 |
| 3.2 Especificaciones del drone no comercial de construcción propia. | 21 |
| 3.3 Especificaciones de Jetson AGX Xavier. | 23 |
| 3.4 Especificaciones de la cámara Victure AC600. | 24 |

ÍNDICE DE CÓDIGOS

| | | |
|-----|--|----|
| 2.1 | Definición del mensaje HEARTBEAT de MAVLink. | 16 |
| 2.2 | Definición CMD_NAV_WAYPOINT, comando de MAVLink. | 17 |
| 5.1 | Test | 32 |

GLOSARIO

| Entrada | Descripción |
|----------------|--|
| 3DR | 3D Robotics. |
| API | Application Programming Interface. |
| CARMEN | CARnegie MELLon robot Navigation. |
| COCO | Common Objects in Context. |
| CPU | Central Processing Unit. |
| CUDA | Compute Unified Device Architecture. |
| CVAR | Computer Vision Aerial Robotics. |
| DJI | Dà-Jiāng Innovations. |
| GCS | Ground Control Station. |
| GPS | Global Positioning System. |
| GPU | Graphics Processing Unit. |
| IEEE | Institute of Electrical and Electronics Engineers. |
| LSI | Laboratorio de Sistemas Inteligentes. |
| LTS | Long Term Support. |
| MAVLink | Micro Air Vehicle Link. |
| MAVROS | MAVLink to ROS. |
| MLP | MultiLayer Perceptron. |
| NumPy | Numerical Python. |
| OpenCV | Open Computer Vision. |
| OROCOS | Open RObot Control Software. |
| OSRF | Open Source Robotics Foundation. |
| ROS | Robot Operating System. |
| SDF | Structure Data File. |
| SDK | Software Development Kit. |

Entrada Descripción

| | |
|-------|---|
| SITL | Software In The Loop. |
| SLAM | Simultaneous Location And Mapping. |
| SOM | System On Module. |
| UAS | Unmanned Aircraft System. |
| UAV | Unmanned Aerial Vehicle. |
| UC3M | Universidad Carlos III de Madrid. |
| UDP | User Datagram Protocol. |
| UHF | Ultra High Frequency. |
| UPM | Universidad Politecnica de Madrid. |
| URJC | Universidad Rey Juan Carlos. |
| USB | Universal Serial Bus. |
| VANT | Vehículo Aéreo No Tripulado. |
| vSLAM | Visual Simultaneous Location And Mapping. |
| WiFi | Wireless Fidelity. |
| YARP | Yet Another Robot Platform. |
| YOLO | You Only Look Once. |

1. INTRODUCCIÓN

Este trabajo propone el diseño e implementación de un sistema para la programación y navegación de robots aéreos. El diseño persigue una estructura modular donde los diferentes bloques que constituyen el sistema puedan sustituirse para adaptarlos al problema, y que a su vez, permita la reutilización del código en diversas circunstancias.

Este primer capítulo recoge una introducción a la materia de estudio, la robótica aérea. Además, se expone la motivación cuyo resultado ha derivado en este trabajo, junto al problema concreto al cual se pretende dar solución y los objetivos extraídos del problema.

1.1. ROBÓTICA AÉREA

El término *robot* aparece por primera vez en 1920, en la obra teatral *Rossum's Universal Robots* del escritor checo Karel Čapek en cuyo idioma la palabra “robota” significa fuerza o servidumbre [1]. El nacimiento de la robótica y los robots surge asociado a la idea de trabajo y producción tras la Segunda Revolución Industrial y a lo largo del siglo XX. La automatización industrial de aquella época da lugar a los primeros sistemas de control automático que se extienden rápidamente a todos los sectores industriales, y que dan lugar a la robótica industrial tal como la conocemos hoy en día [2].

El término *robótica* es acuñado por Isaac Asimov, definiendo a la ciencia que estudia a los robots. El propio Asimov postuló también las Tres Leyes de la Robótica en su libro *Yo Robot* publicado en 1950 [1]. El término ha evolucionado mucho desde sus inicios, hoy entendemos la robótica como la rama de la tecnología que estudia el diseño y construcción de máquinas capaces de desempeñar tareas realizadas por el ser humano o que requieren el uso de inteligencia [3]. Como se puede entender, la visión actual es mucho más amplia que en sus inicios y abarca muchas áreas de la ingeniería.

Existen diversas clasificaciones en función de su arquitectura, de su aplicación, de su cronología, etc. Una de estas clasificaciones distingue robots en función de su morfología [4], que suele distinguir los siguientes tipos:

- **Poliarticulados:** Son artíluguos mecánicos y electrónicos destinados a realizar de forma automática determinados procesos de fabricación o manipulación. Suelen ser fijos, aunque también pueden realizar desplazamientos limitados y poseen un espacio de trabajo concreto y limitado. Los mejores ejemplos son los robots industriales, manipuladores o cartesianos.
- **Móviles:** Están provistos de algún tipo de mecanismo que les permite desplazarse autónomamente, como patas o ruedas, y reciben información de su entorno a través de sus propios sensores. Son ampliamente utilizados en el transporte de mercancías

o en la exploración de lugares de difícil acceso. Pueden ser terrestres, acuáticos, aéreos o espaciales.

- **Androides:** Intentan reproducir total o parcialmente la forma y el comportamiento del ser humano. No solo imitan la apariencia humana (antropomorfismo), si no que emulan también la conducta de forma autónoma.
- **Zoomórficos:** Son aquellos que trata de reproducir en mayor o menor grado de realismo los sistemas de locomoción de diversos seres vivos. Este tipo podría incluir también a la morfología androide, en función del autor de la clasificación. Una subclasificación distingue entre robots caminadores y no caminadores.

La robótica aérea es la rama de la robótica que se encarga del estudio del comportamiento autónomo de aeronaves no tripuladas. Se entiende como una aeronave no tripulada (UAV, *Unmanned Aerial Vehicle*, o más recientemente UAS, *Unmanned Aircraft System*) a aquella que es capaz de realizar una misión sin necesidad de tener una tripulación embarcada [5]. Otro término que también se utiliza con frecuencia es VANT, Vehículo Aéreo No Tripulado. A lo largo de esta memoria se utilizará el término *drones* en referencia a este tipo de sistemas, cuyo uso está muy extendido.

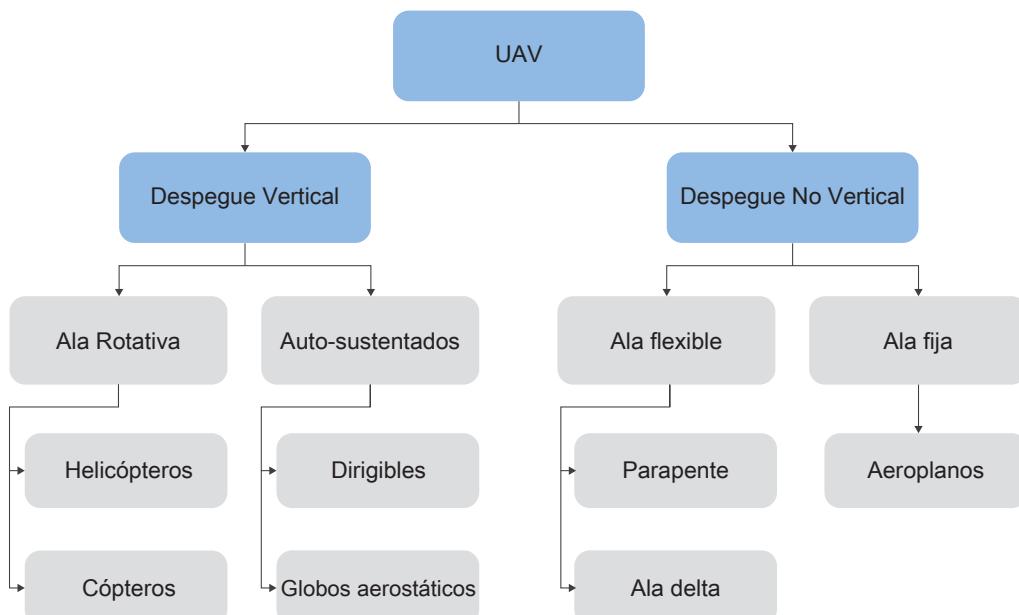


Fig. 1.1. Clasificación de UAVs [5].

A la hora de establecer una clasificación de los UAV es posible atender a diferentes criterios. Siguiendo la clasificación propuesta por Barrientos *et al.* [5] se distinguen aeronaves en función del tipo de despegue, que puede ser vertical o no. A su vez, podemos subdividir las aeronaves en función del origen de su sustentación o del tipo de ala que poseen. Esta clasificación se representa en la Figura 1.1, mientras que en la Figura 1.2 se muestran dos ejemplos de diferentes tipos de UAV.



(a) Aeroplano.



(b) Multicóptero.

Fig. 1.2. Ejemplos de UAVs.

Otros criterios de clasificación pueden responder a las capacidades de vuelo como el alcance, la altitud, la autonomía o la carga máxima. A su vez, también se pueden clasificar las aeronaves en función de la actividad que realizan. Algunas de estas clasificaciones distinguen entre uso civil y militar o aplicaciones en explotación (producto) o en desarrollo (prototipo).

Entre las principales aplicaciones civiles donde se emplean drones se encuentran el transporte (tanto de mercancías como de personas), la seguridad y/o vigilancia (Fig. 1.4), la pesca y agricultura, la cartografía o el ocio y entretenimiento, entre otros.



Fig. 1.4. Aeronave de vigilancia en carretera perteneciente a la Dirección General de Tráfico [6].

En la actualidad tiende a utilizarse el concepto de UAS frente al de UAV. La extensión del concepto de vehículo a sistema refleja que el vehículo aéreo autónomo precisa para su funcionamiento de todo un sistema y no solo de la aeronave instrumentada.

Típicamente, un sistema UAS se compone de el segmento aire, compuesto principalmente por la aeronave, y por el segmento tierra, compuesto por un computador donde se ejecuta algún software de control, habitualmente una estación terrestre. Entre ambos segmentos debe existir en todo momento una comunicación, que se realiza a través de un protocolo de comunicaciones.

Sin embargo, es posible prescindir del segmento tierra dotando de la suficiente inteligencia a la aeronave. La inmensa mayoría de los sistemas que hoy en día entendemos como *inteligentes* o *autónomos* llevan embarcado un ordenador con un software que les permite suprimir al segmento tierra de la ecuación. La complejidad del ordenador embarcado y del software de control marcará el nivel de *inteligencia* de la aeronave y la necesidad, o no necesidad, del segmento tierra.

Embarcar el segmento tierra abordo de la aeronave supone un aumento en la complejidad del sistema, por lo que puede no ser siempre interesante. Las características de la plataforma aérea y el problema a resolver definirán si es necesario embarcar o no el software de control sobre la aeronave.

1.2. MOTIVACIÓN

El ámbito de la robótica y de los vehículos aéreos no tripulados es un sector caracterizado por una fuerte expansión en los últimos tiempos, con unas expectativas de crecimiento y de demanda de personal cualificado tanto a nivel Europeo como nacional muy relevantes para los próximos años. En el año 2035 el volumen de negocio anual estimado será 10.000M€ y 90.000 puestos de trabajo (1.220M€ y 11.000 ud solo para España), pasando en el año 2050 a un volumen de 14.600M€ y 110.000 puestos de trabajo (1.520M€ y 11.500 ud caso español) [7].

La importancia del vuelo autónomo se ha visto reflejada en el alto incremento del uso de UAVs. Recientemente, los UAV se utilizan ampliamente tanto en aplicaciones militares como civiles debido a su pequeño tamaño y gran capacidad de maniobra [8]. Especialmente en aplicaciones civiles, los usos de los UAV se han expandido a casi todas las áreas. En el área de la agricultura, la rápida evolución de los UAV puede conducir a aplicaciones de agricultura de precisión, como la monitorización aérea de cultivos y las tareas de fumigación inteligente [9]. En el campo industrial, los desarrollos de los UAV mejoran la eficiencia de misiones como inspección industrial (e.g. plantas fotovoltaicas) [10], identificación de carga y entrega o logística, fuertemente ligadas a técnicas de *slam* visual (VSLAM). Además, los UAV también se pueden ver en tareas de búsqueda y rescate, de topografía o de vigilancia entre otras aplicaciones.

Dentro del ámbito nacional, el Ministerio de Ciencia e Innovación recoge en el documento Estrategia Española de Ciencia, Tecnología e Innovación 2021-2027 [11] las Líneas Estratégicas de I+D+I nacional. Entre ellas se encuentra Inteligencia Artificial y

Robótica, en el ámbito de intervención de Mundo Digital, Industria, Espacio y Defensa, ratificando la relevancia que están cobrando los UAVs en los últimos años.

Debido a la gran expansión social y económica que está sufriendo esta materia, se observa que existe la necesidad de herramientas que permitan construir aplicaciones para el uso extendido de drones. Herramientas que permitan abstraerse de las complejidades que lleva asociada una aeronave e incluso de la plataforma en sí.

Este trabajo persigue ese fin. Busca proporcionar al usuario una herramienta para la programación y navegación de robots aéreos, que permita al usuario centrar sus esfuerzos en resolver solamente la aplicación final para la que se usa el drone, y olvidarse del resto de complejidades.

1.3. PROBLEMA

El problema a resolver es realmente complejo, alcanzar un software seguro y robusto que permita el vuelo autónomo de aeronaves es inabordable en tiempo y esfuerzo para este trabajo.

Por ello, este trabajo se centra en un subconjunto del problema. En concreto, se busca un software intermedio, a modo de *middleware*, que permita conectar distintas plataformas de vuelo con diferentes aplicaciones finales, las cuales quedarían a cargo del desarrollo del usuario. Se utiliza la palabra *middleware*, aunque no sea del todo correcta, ya que el software propuesto se encuentra a medio camino entre las aplicaciones (software) y las aeronaves (hardware).

Entre las aplicaciones posibles, la herramienta a desarrollar busca ofrecer soluciones para la creación de algoritmos de navegación autónomos. Más allá del típico control por posición en exteriores, basado en GPS, se persiguen aplicaciones basadas en posición en interiores (en ausencia de GPS), p.e. algoritmos de auto-localización visual (*visual SLAM*), o aplicaciones de control visual, no basadas en posición.

Por último, con el propósito de acotar todavía más el problema, este trabajo solo se centrará en el uso de un tipo de aeronaves, los multicópteros, cuyo uso es el más extendido dentro de todos los tipos de aeronaves.

1.4. OBJETIVOS

Este trabajo presenta una serie de objetivos principales y secundarios. En primer lugar, se quiere desarrollar herramientas software que permitan la programación y navegación de multicópteros. El primer objetivo se abordará creando una solución software que ofrezca al usuario una interfaz de programación de aplicaciones (API, *Application Programming Interface*) que facilite el desarrollo de aplicaciones. Asociado a este objetivo principal, se encuentran un grupo de características del código considerados como atributos de calidad y objetivos secundarios. Se precisa un software

intuitivo y sencillo de usar, los usuarios no tienen porque ser expertos en drones, y robusto y seguro, al tratarse de un software que trabaja directamente con drones. Cabe destacar que estas aplicaciones pueden ser creadas por usuarios ajenos al resto del software e incluso a la aeronave utilizada. Además, el software que se propone está enfocado a un público, el cual, aunque sí deba tener unas nociones básicas en robótica, no tiene porque ser experto en drones. Por tanto, se consideran como objetivos secundarios ligados a este primer objetivo principal la usabilidad, la seguridad y la robustez del código.

En segundo lugar, se pretende extender dicho software para el uso de diferentes plataformas aéreas. Así, otro de los objetivos principales del software es que sea multiplataforma y se pueda utilizar con diferentes aeronaves. Para ello, se realizarán diferentes pruebas que permitan comprobar el funcionamiento del software con varias aeronaves.

Como objetivo secundario ligado a este segundo objetivo principal, se buscará utilizar aeronaves de distinta naturaleza dentro de los multicópteros. Una variedad en las aeronaves usadas durante las pruebas dará una mejor perspectiva del alcance del trabajo realizado. Aunque durante el Capítulo 2 se profundizará sobre las herramientas disponibles para este trabajo, es necesario introducir las posibles alternativas existentes para el correcto entendimiento de los objetivos. Donde los multicópteros posibles pueden ser reales o simulados, y orientados a un vuelo en interiores o en exteriores.

En tercer lugar, se realizarán diferentes aplicaciones ilustrativas similares a las desarrolladas por los futuros usuarios. Dichas aplicaciones harán uso de la solución desarrollada como primer objetivo del trabajo y se basarán en algún tipo de control reactivo visual. Dichas aplicaciones tratarán de mostrar ejemplos de aplicaciones de dificultad variable, partiendo de un uso básico de las herramientas recogidas en este trabajo, hasta el desarrollo de algoritmos navegación basado en aprendizaje profundo.

Así pues, y a modo de resumen, los objetivos contemplados en este trabajo son:

- **Desarrollo de herramientas para la programación y navegación de multicópteros.** Herramientas cuyo diseño se centrará en la usabilidad, la seguridad y robustez del software.
- **Comprobaciones sobre distintas plataformas aéreas.** Multicópteros de diversa índole, haciendo uso de aeronaves reales, simuladas, de interiores y de exteriores.
- **Desarrollo de diferentes aplicaciones de control.** Aplicaciones de distinto nivel de dificultad técnica, partiendo de usos básicos centrados en navegación hasta algoritmos complejos basados en aprendizaje profundo.

1.5. ESTRUCTURA DE LA MEMORIA

Esta última sección de la primera parte del trabajo describe la estructura de la memoria, introduce los diferentes capítulos y los temas que se tratarán en cada uno de los mismos.

En primer lugar se encuentra este capítulo introductorio (Cap. 1) que se concibe como una serie de aclaraciones previas y necesarias para el correcto entendimiento del trabajo en su conjunto. Incluye un contexto histórico de la robótica, la robótica aérea y las estaciones de tierra y sus aplicaciones, que establecen el marco en el que se sitúa este trabajo. A continuación incluye, el problema abordado en este trabajo y la motivación del mismo, es decir, ¿qué se trata de resolver con este trabajo?, y ¿por qué surge la necesidad del mismo?. Se incluyen también los objetivos principales y secundarios que ha de cumplir la solución desarrollada. Por último, se encuentra la sección actual que muestra una visión global de lo que se presenta en esta memoria.

En el Capítulo 2 se detallan las herramientas *software* y *hardware* que constituyen el estado del arte, ofreciendo una visión global de la robótica aérea actual. Para ello, se realiza un pequeño estudio de las principales herramientas presentes tanto en la comunidad científica como en la industria.

Además, este capítulo recoge en diferentes secciones cada uno de los elementos que entran en juego en un sistema aéreo. Por un lado, en una primera sección se explica el segmento tierra y su composición. Por otro lado, en la siguiente sección se detalla el segmento aire y sus componentes. Finalmente, el capítulo se centra en el protocolo de comunicaciones entre ambos lados, tierra y aire.

En el Capítulo 3 se presentan el material y método utilizados en este trabajo. Este capítulo recopila, de entre todas las herramientas existentes, los componentes seleccionados para la resolución del problema. Además, ofrece una descripción pormenorizada de los mismos, junto con la motivación de su elección.

En el Capítulo 4 se explica la infraestructura software desarrollada para dar solución al problema. En una primera sección se explica detalladamente el diseño elegido y se explican también varias decisiones relevantes a la hora de seleccionar el diseño. En la sección sucesiva se describe la implementación de cada una de las herramientas de código identificadas.

En el Capítulo 5 se desarrollan las aplicaciones propuestas que permiten comprobar la solución presentada en el anterior capítulo. A lo largo del capítulo, se introducen las diferentes aplicaciones, se explican cada uno de los diseños y sus implementaciones y se presentan los resultados obtenidos tras el desarrollo en forma de diversos ensayos experimentales con pruebas de vuelo en simulación y en real.

Por último, en el Capítulo 6 se recogen las conclusiones extraídas con la finalización del trabajo y se evalúan los distintos objetivos iniciales propuestos. A mayores, se exponen una serie de posibles vías de desarrollo futuro y de mejoras para la herramienta.

2. HERRAMIENTAS

A lo largo de este segundo capítulo se analizan las diferentes herramientas *software* y *hardware* que forman el estado del arte de la robótica aérea actual. A la hora de proceder a desgranar los diferentes agentes que entran en acción, se realiza previamente una clasificación entre el lado tierra, el lado aire y la comunicación entre estos que se reflejan en las secciones de este capítulo.

2.1. SEGMENTO TIERRA

El lado tierra del sistema se compone por los elementos del sistema que no se encuentran abordo de la aeronave. Principalmente, está compuesto por los elementos de control de vuelo. Estos son, típicamente, una emisora de vuelo y/o un ordenador. La presencia de la emisora depende en gran medida de la aeronave en cuestión y suele estar casi siempre presente, mientras que la presencia de un ordenador en el lado tierra depende de la aplicación. Ambos elementos son compatibles y el uso de uno no significa la ausencia del otro, siendo en muchos casos la frontera que los separa difusa. La Figura 2.1 muestra el aspecto típico de una emisora de vuelo.



Fig. 2.1. Transmisor *Taranis Q X7S OpenTX*.

Las emisoras radiocontrol son un control remoto que permiten controlar la aeronave a distancia. La comunicación suele ser por radio frecuencia, ya que ofrecen más libertad de movimiento a la aeronave, aunque existen modelos que se comunican mediante cable con el drone (aeronaves en vuelo cautivo), y típicamente dúplex. Las frecuencias utilizadas suelen depender de las necesidades de la aplicación. Esto es debido a que las características de la señal, fijan parámetros de la conexión tan importantes como el rango o el ancho de banda. Las bandas utilizadas se encuentran dentro de la frecuencia ultra-alta (UHF, *Ultra High Frequency*) sobre los 433MHz o sobre los 915MHz, en función la cantidad de datos que es necesario transmitir (mensajes, imágenes, vídeo, etc.). Otras frecuencias utilizadas, aunque en menor medida debido a su corto alcance, son las bandas de 2.4GHz, correspondientes con el estándar WiFi (IEEE 802.11). La Figura 2.2 contiene un módulo receptor de telemetría a 915MHz.



Fig. 2.2. Módulo receptor *X-Rock V5*.

Por otro lado, un ordenador ofrece una amplia variedad de aplicaciones con multitud de posibilidades, desde aplicaciones generalistas a específicas a un ámbito (véase por ejemplo la topografía), o desde aplicaciones comerciales a libres o incluso desarrolladas por el propio usuario.

Debido a la gran cantidad de posibles opciones, los sucesivos apartados tratarán sobre diferentes tipos de aplicaciones que resultan relevantes para este proyecto.

2.1.1. ESTACIÓN DE CONTROL TERRESTRE

Las Estaciones de Control Terrestre (GCS) son herramientas software que se utilizan para planificar y volar una misión. Generalmente proporcionan una pantalla con un mapa donde el usuario puede definir puntos de referencia para el vuelo y ver el progreso de la misión. Además, suelen disponer de una “cabina virtual”, mostrando muchos de los

mismos instrumentos que poseen los aviones tripulados. Entre otras funcionalidades, suelen también disponer de herramientas de configuración de la aeronave y herramientas de descarga y análisis de archivos de registro de vuelo.

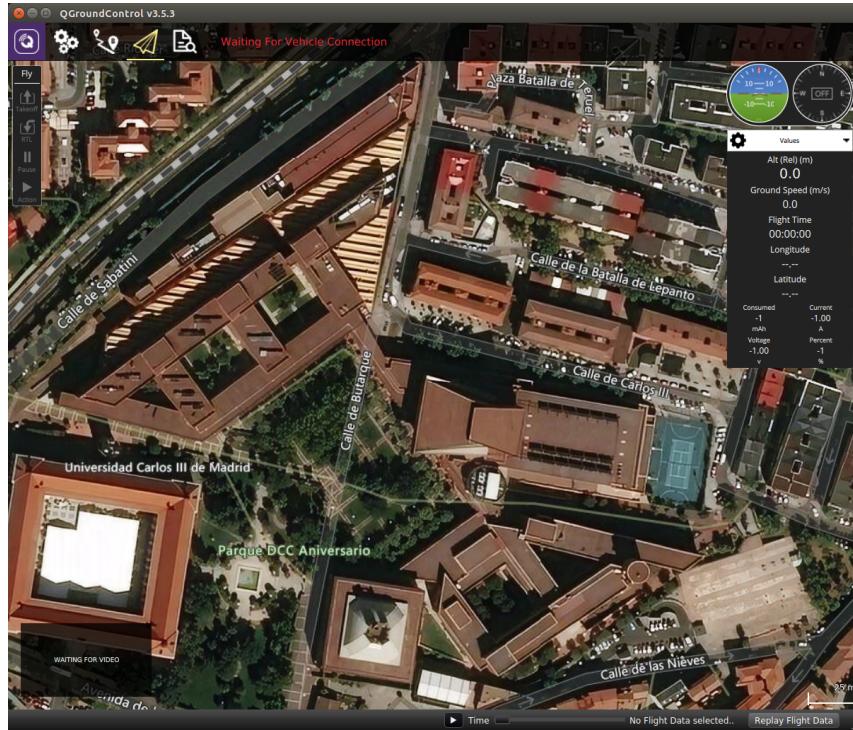


Fig. 2.3. QGroundControl.

Hoy en día, las principales GCS son, o bien software propietario de los principales fabricantes de UAVs, o software libre. Debido a su naturaleza, resultan más interesantes estas segundas. Entre ellas destacan principalmente *QGroundControl* [12] y *MissionPlanner* [13]. La Figura 2.3 muestra un ejemplo de estos programas en ejecución.

2.1.2. TOOLKITS DE VUELO

Los toolkits de vuelo incluyen bibliotecas, herramientas, drivers, etc para aplicaciones de vuelo programables. Ofrecen una interfaz de programación (API) que permite el desarrollo de aplicaciones a un nivel de abstracción muy elevado. Este tipo de software permite abstraerse de aspectos como el tipo concreto de aeronave o la comunicación con la misma. Cabe destacar que este tipo de software puede ejecutarse a bordo de la aeronave y formar parte del segmento aire o pertenecer al segmento tierra y ejecutarse en un ordenador externo.

Un ejemplo de toolkit de vuelo es *Aerostack* [14]. *Aerostack* ofrece a desarrolladores herramientas para diseñar y construir la arquitectura de control de sistemas robóticos aéreos, integrando múltiples soluciones computacionales heterogéneas como algoritmos de visión por ordenador, métodos de mapeo y auto-localización, planificadores de movimiento, etc. [15]. Ha sido desarrollada por el grupo de Visión por Computador y Robótica

Aérea de la Universidad Politécnica de Madrid (CVAR-UPM) [16] y actualmente se encuentra en su versión 5.

2.1.3. PLATAFORMAS ROBÓTICAS DE SOFTWARE

Las plataformas robóticas de software son entornos que permiten la interacción software-hardware. Una analogía que las define con sencillez es que si los datos son el torrente sanguíneo de tu robot, entonces las plataformas robóticas de software son el sistema circulatorio del mismo.

Más específicamente, estas plataformas permiten la construcción de sistemas de control de robots como una colección de programas que se comunican *peer-to-peer* con una familia extensible de tipos de conexión. A este tipo de plataformas se les conoce comúnmente como *middleware*, al encontrarse a medio camino entre el hardware y el software.

Pese a describir este software como parte del segmento tierra, es importante destacar que puede también pertenecer al segmento aire, en función de donde esté ubicado. Existen múltiples plataformas robóticas, las más comunes son ROS [17], YARP [18], [19], Orosocos [20] o CARMEN [21].

Hoy en día, la plataforma más utilizada en robótica es ROS. ROS (*Robot Operating System*) es “un meta-sistema operativo de código abierto para su robot”, mantenido por *Open Source Robotics Foundation* (OSRF) [22]. ROS proporciona servicios que se esperarían de un sistema operativo, incluyendo abstracción de hardware, control de dispositivos de bajo nivel, implementación de funcionalidades comunes, envío de mensajes entre procesos y manejo de paquetes. También brinda herramientas y librerías para obtener, construir, escribir y correr código a través y mediante varios ordenadores.

2.1.4. APRENDIZAJE PROFUNDO

El aprendizaje profundo es un conjunto de algoritmos de aprendizaje automático que intenta modelar abstracciones de datos de alto nivel usando redes neuronales. Las redes neuronales son arquitecturas computacionales de datos expresados en forma matricial o tensorial. Cada unidad, llamada *neurona*, se organiza en capas que forman la red.

La palabra “profundo” hace referencia al número de capas que posee la red, siendo redes profundas aquellas redes con más de tres capas. En la Figura 2.4 se ilustra una arquitectura de red neuronal básica, conocida como perceptrón multicapa (MLP, *MultiLayer Perceptron*). La arquitectura presentada posee una capa de entrada, una de salida y una capa oculta.

Existen diferentes marcos de programación que permiten el desarrollo de redes y algoritmos. Las principales plataformas son Keras [23], TensorFlow [24], PyTorch [25] y, en el ámbito de la visión, Darknet [26].

- **Keras** es una biblioteca de redes neuronales de código abierto escrita en Python

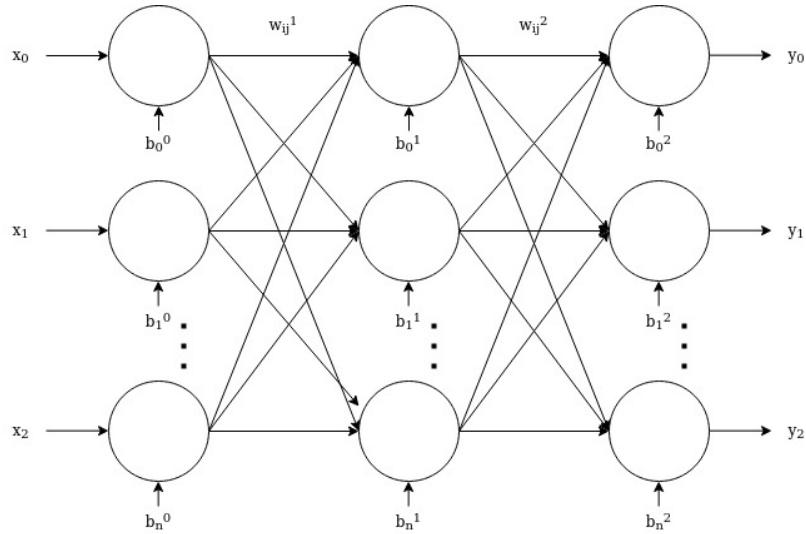


Fig. 2.4. Arquitectura perceptrón multicapa.

y desarrollada por François Chollet. Es capaz de ejecutarse sobre TensorFlow (fue integrado a mediados de 2017) y otros marcos de programación. Está diseñado para permitir una experimentación rápida con redes neuronales profundas.

- **TensorFlow** es una biblioteca de software de código abierto para la programación de redes neuronales y del flujo de datos en una variedad de tareas, desarrollado por Google y lanzado en 2015. Ofrece múltiples niveles de abstracción para construir y entrenar modelos.
- **PyTorch** es una biblioteca de aprendizaje automático de código abierto para Python, basada en Torch. Se utiliza para aplicaciones como el procesamiento del lenguaje natural y fue desarrollado por el grupo de investigación de inteligencia artificial de Facebook en 2017.
- **Darknet** es un marco de trabajo de código abierto escrito en C y CUDA. Es rápido, fácil de instalar y admite cálculos de CPU y GPU. En él se han desarrollado redes como YOLO.

2.2. SEGMENTO AIRE

El lado aéreo se compone por la plataforma aérea, es decir el drone junto a todos los elementos a bordo que componen el sistema. En esta sección se presentan dos tipos de aeronaves, reales o simuladas, que se abordarán los sucesivos apartados.

El elemento principal de una aeronave es el autopiloto, el cual se encarga del control de los sensores y los actuadores. Un autopiloto se compone de una parte hardware (controladora) que va embarcada en la aeronave y una parte software (firmware) que se ejecuta en el hardware. El funcionamiento que sigue una aeronave simulada es el mismo

que el de una real. La Figura 2.5 representa el bucle estándar de control disponible en los autopilotos más comunes.

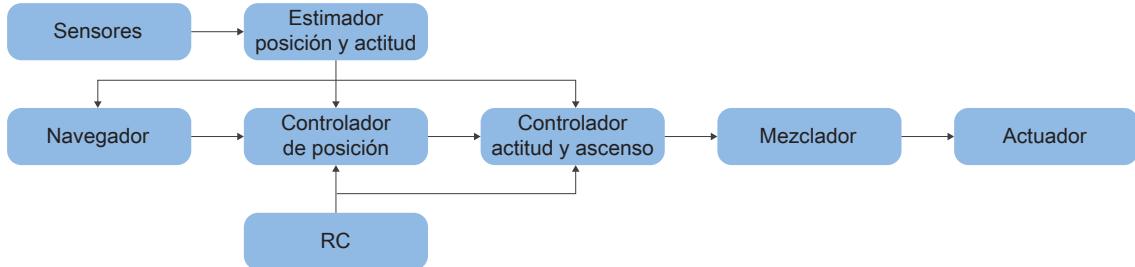


Fig. 2.5. Bucle de control de un UAV [27].

El funcionamiento de un bucle de control básico es el siguiente.

El estimador toma una o más entradas de diferentes sensores, las combina y calcula el estado del vehículo. La controladora toma un punto de ajuste y una medición o estado estimado como entradas. Su objetivo es ajustar el valor del estado estimado de modo que coincida con el punto de ajuste. La salida es una corrección para eventualmente alcanzar ese punto de ajuste. Por ejemplo, el controlador de posición toma los puntos de ajuste de posición como entradas, y según la posición estimada calcula la salida que es un punto de ajuste de actitud y empuje que mueve el vehículo hacia la posición deseada. Finalmente, el mezclador toma comandos concretos, como girar a la derecha, y los traduce a comando de motor individuales, al tiempo que garantiza que no se excedan algunos límites. Esta traducción es específica para cada vehículo y depende de varios factores, como la disposición del motor con respecto al centro de gravedad o la inercia rotacional del vehículo, entre otros.

2.2.1. AERONAVES REALES

Existen multitud de aeronaves y diversas posibles clasificaciones, como la propuesta por *Barrientos et al.* (Fig. 1.1) [5]. Otro posible criterio es la naturaleza del firmware, en base a si es propietario o libre.

En una aeronave libre el usuario tiene acceso a la controladora y al software de vuelo, que sería de código abierto y donde el mismo usuario podría modificar si lo precisa. En frente se encuentran las aeronaves propietarias, donde tanto la controladora y el firmware son privados y no accesibles al usuario.

Por un lado, entre los principales fabricantes propietarios se encuentran *DJI* [28], *Parrot* [29] o *3DR Robotics* [30] (ver Fig. 2.6). Aeronaves como el Tello de DJI (Fig. 2.7b) son ampliamente utilizadas en investigación debido a su bajo coste y tamaño. Ejemplo de ello puede ser el estudio realizado por A. Anwar y A. Raychowdhury donde proponen un nuevo algoritmo de navegación autónoma basado en técnicas de aprendizaje profundo [31] o A. Scannell et al. en su investigación sobre nuevos algoritmos de optimización de trayectorias en entornos complejos [32].



(a) 3DR Solo Drone.



(b) DJI Tello.

Fig. 2.6. Ejemplos de drones de fabricantes propietarios.

Por el otro lado, a la hora de hablar de plataformas libres es necesario distinguir entre los componentes hardware (principalmente la controladora) y el software de vuelo (firmware). Los dos principales firmware libre son *PX4* y *Ardupilot*.

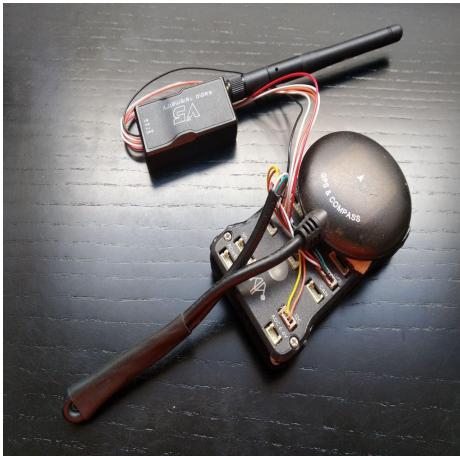
PX4 es un software de control de vuelo de código abierto para drones y otros vehículos no tripulados. Proporciona un conjunto flexible de herramientas para que los desarrolladores compartan tecnologías que consigan crear soluciones personalizadas para aplicaciones de drones [33]. *PX4* pertenece Dronecode, una organización sin ánimo de lucro de la Fundación Linux.

Ardupilot es un sistema de piloto automático confiable, versátil y de código abierto que admite muchos tipos de vehículos: multícopteros, helicópteros, aviones de ala fija, boites, submarinos, rovers y más [34]. El código fuente está desarrollado por la comunidad abierta de desarrolladores *Ardupilot-Dev Team*.

La principal controladora de vuelo utilizada con ambos firmware es *Pixhawk* [35]. Al igual que *PX4*, *Pixhawk* pertenece al grupo Dronecode. La Figura 3.2 muestra un multícoptero equipado con *PX4*, mientras que la Figura 2.8 contiene diferentes controladoras *Pixhawk*.

2.2.2. AERONAVES SIMULADAS

Las aeronaves simuladas suelen utilizar software de vuelo libre, dada la naturaleza abierta del mismo. En una simulación, la parte física de la aeronave se ve sustituida por software (SITL, *Software-In-The-Loop*). El SITL contiene el software de vuelo y permite enviar y recibir comandos de vuelo sin necesidad de tener una controladora real, haciendo de intermediario entre el autopiloto y el modelo SDF, necesario en la visualización gráfica. Dichas simulaciones se suelen acompañar de una visualización. Para ello, es necesario el uso de un simulador como *Gazebo* [36], *AirSim* [37] o *jMAVSIM* [38]. El más extendido, debido a sus grandes prestaciones y posibilidades de configuración es *Gazebo*. La Figura



(a) PixHawk v1



(b) PixHawk v2.1

Fig. 2.8. Diferentes modelos de controladoras PixHawk.

2.10 muestra un drone simulado mediante *jMAVsim* con *PX4* como autopiloto.

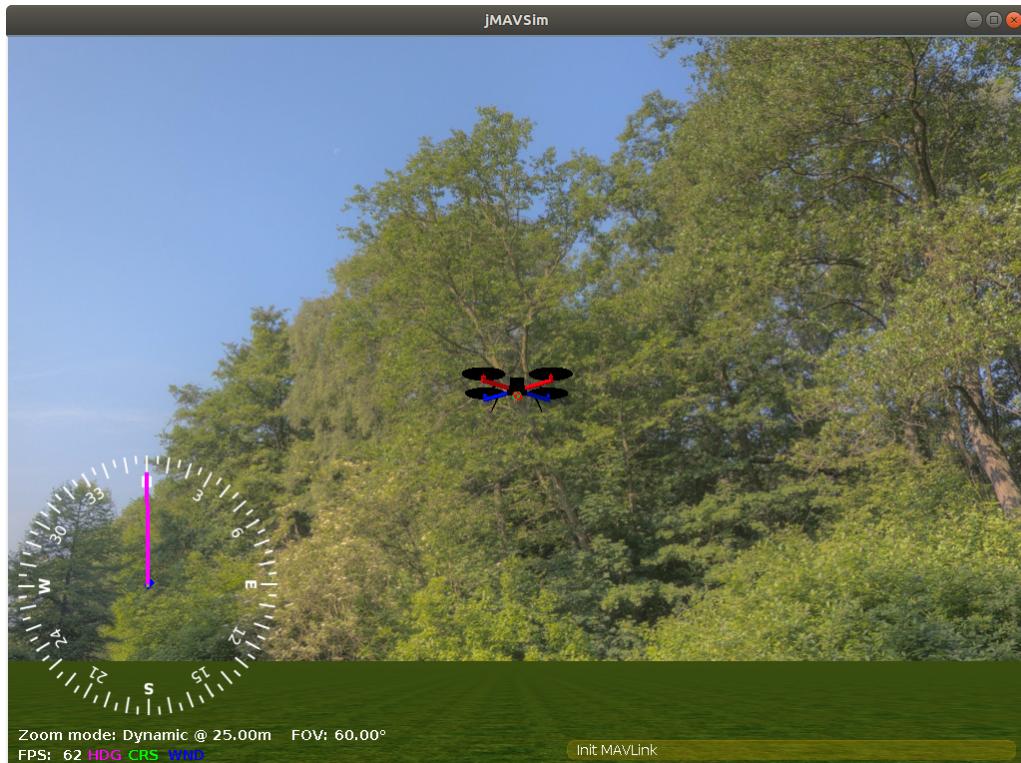


Fig. 2.10. Drone simulado mediante *jMAVsim*.

2.3. PROTOCOLO DE COMUNICACIONES

El protocolo de comunicaciones puede ser propietario, utilizado por fabricantes de aeronaves y diferentes para cada una de ellas, o libre. El protocolo de comunicaciones libre por excelencia es MAVLink [39], que se ha convertido *de facto* en el estándar del sector.

MAVLink son las siglas de *Micro Air Vehicle Link*, un protocolo de comunicaciones muy ligero para el intercambio de mensajes con un drone y sus componentes a bordo. MAVLink se sitúa entre los extremos de la comunicación, siendo el puente de comunicación entre el lado tierra y el lado aire. Es considerado como el protocolo estándar de comunicaciones en robótica aérea y multitud soluciones comerciales lo utilizan, por ejemplo las ya mencionadas GCS, *QGroundControl* y *Mission Planner*.

Se compone por cuatro tipo de elementos: mensajes, comandos, enumerados y micro-servicios.

Los mensajes son el objeto más pequeño de intercambio de información del protocolo. Sirven para transmitir el estado o información de la aeronave, principalmente datos de navegación o de posición. Son muchos los mensajes existentes y de diversa utilidad, como se puede comprobar en la documentación existente [40]. Ejemplo de ello son los mensajes *HEARTBEAT* (ver Cód. 2.1) o *ATTITUDE*.

```

1 <message id="0" name="HEARTBEAT">
2   <description>The heartbeat message shows that a system or
      component is present and responding. The type and autopilot
      fields (along with the message component id), allow the
      receiving system to treat further messages from this system
      appropriately.</description>
3   <field type="uint8_t" name="type" enum="MAV_TYPE">Vehicle or
      component type. For a flight controller component the vehicle
      type (quadrotor, helicopter, etc.). For other components the
      component type (e.g. camera, gimbal, etc.).</field>
4   <field type="uint8_t" name="autopilot" enum="MAV_AUTOPILOT">
      Autopilot type / class. Use MAV_AUTOPILOT_INVALID for
      components that are not flight controllers.</field>
5   <field type="uint8_t" name="base_mode" enum="MAV_MODE_FLAG"
      display="bitmask">System mode bitmap.</field>
6   <field type="uint32_t" name="custom_mode">A bitfield for autopilot
      -specific flags</field>
7   <field type="uint8_t" name="system_status" enum="MAV_STATE">System
      status flag.</field>
8   <field type="uint8_t_mavlink_version" name="mavlink_version">
      MAVLink version, not writable by user, gets added by protocol
      because of magic data type: t_mavlink_version</field>
9 </message>
```

Cód. 2.1. Definición del mensaje HEARTBEAT de MAVLink.

Los comandos sirven para transmitir peticiones y acciones a la aeronave desde la GCS, o viceversa. Se encapsulan en un tipo especial de mensajes (*COMMAND_INT* o *COMMAND_LONG*). Los comandos pueden ser de tres tipos: de navegación, de acción y de condición, en función del tipo de petición requerida. Los comandos utilizados en el protocolo se pueden comprobar también en la documentación de MAVLink [40]. El Código 2.2 muestra, a modo de ejemplo, la estructura de un comando de navegación a un

punto.

Los enumerados describen diferentes estados, como puede ser el modo de vuelo, el tipo de aeronave o el estado de la comunicación, y son usados como campos de los mensajes o de los comandos. Los enumerados sirven, en definitiva, para representar errores, estados o modos. Cada enumerado tiene un atributo de nombre obligatorio y puede contener una serie de elementos de entrada (con nombres exclusivos de enumeración) para los valores admitidos.

En el mensaje HEARTBEAT (Cód. 2.1) se hace referencia a varios, como *MAV_TYPE* o *MAV_AUTOPILOT*. Al igual que los mensajes y los comandos, los enumerados disponibles se pueden comprobar en la documentación de MAVLink [40].

Por último, los microservicios establecen cómo deben ser intercambiados los mensajes y comandos para un correcto funcionamiento de la comunicación. Representan protocolos de alto nivel que los sistemas MAVLink pueden adoptar para una mejor interacción. Los microservicios se utilizan para intercambiar muchos tipos de datos, incluidos: parámetros, misiones, trayectorias, imágenes y otros archivos. Los datos pueden ser mucho más grandes de lo que cabe en un solo mensaje, por lo que los microservicios definirán cómo se dividen y se vuelven a ensamblar los datos, y cómo garantizar que los datos perdidos se vuelvan a transmitir. Otros microservicios proporcionan reconocimiento de comandos, informes de errores, etc. Los diferentes microservicios se definen en la documentación de MAVLink [41].

```
1 <enum name="MAV_CMD">
2   <description>Commands to be executed by the MAV. They can be
3     executed on user request, or as part of a mission script. </
4     description>
5   <entry value="16" name="MAV_CMD_NAV_WAYPOINT">
6     <description>Navigate to waypoint.</description>
7     <param index="1">Hold time in decimal seconds. (ignored by fixed
8       wing, time to stay at waypoint for rotary wing)</param>
9     <param index="2">Acceptance radius in meters (if the sphere with
10       this radius is hit, the waypoint counts as reached)</param>
11     <param index="3">0 to pass through the WP, if > 0 radius in
12       meters to pass by WP. Positive value for clockwise orbit,
13       negative value for counter-clockwise orbit. Allows
14       trajectory control.</param>
15     <param index="4">Desired yaw angle at waypoint (rotary wing).
16       NaN for unchanged.</param>
17     <param index="5">Latitude</param>
18     <param index="6">Longitude</param>
19     <param index="7">Altitude</param>
20   </entry>
21   ...
22 </enum>
```

Cód. 2.2. Definición CMD_NAV_WAYPOINT, comando de MAVLink.

Comandos como el presentado (Cód. 2.2) son útiles a la hora navegar de forma tradicional mediante posición GPS. Junto a este comando, otros mensajes y comandos son necesarios para un correcto funcionamiento del protocolo de comunicaciones. La Tabla 2.1 recoge varios mensajes muy utilizados.

Tabla 2.1. Principales mensajes de MAVLink.

| MENSAJES | |
|---------------------|--------------|
| HEARTBEAT | VFR_HUD |
| SYS_STATUS | SET_MODE |
| HIGHRES_IMU | RADIO_STATUS |
| BATTERY_STATUS | ATTITUDE |
| GLOBAL_POSITION_INT | |

Fuente: MAVLink [39]

3. MATERIAL Y MÉTODO

Durante este capítulo se explican las diferentes herramientas que han servido de ingredientes en la realización de este trabajo. A la hora de proceder a desgranar los diferentes agentes que entran en acción, se realiza previamente una clasificación entre *software* y *hardware* que se reflejan en las secciones de este capítulo.

3.1. HARDWARE

Dentro del material hardware dispuesto, se distinguen aeronaves, dispositivos embedidos o sensores de visión. Dichos elementos se abordarán en los próximos apartados.

3.1.1. AERONAVES

En coherencia con los objetivos del proyecto, se han seleccionado tres diferentes aeronaves sobre las que realizar las pruebas del software desarrollado. La elección de estas aeronaves se ha realizado con el propósito de cubrir un amplio abanico de opciones, siendo los tres cuadri-cópteros muy diferentes entre ellos. Entre las seleccionadas se encuentran aeronaves reales y simuladas, aeronaves propietarias y libres, y aeronaves de vuelo en interiores o en exteriores. Cubrir una gran variedad de posibilidades supone dotar al software de una alta versatilidad, siendo, en un futuro, más fácilmente extensible a otras plataformas.

Las aeronaves elegidas son:

- **3DR Iris simulado.**
- **DJI Tello.**
- **PX4 de construcción propia.**

En primer lugar se encuentra una aeronave simulada, 3DR Iris de *3D Robotics*. Utilizar una aeronave simulada en las primeras fases del desarrollo supone grandes ventajas en ahorro de tiempo y coste, pues los errores en el código no suponen un daño en el material. La aeronave utiliza PX4 como autopiloto y se simula mediante *Gazebo*. Se darán más detalles acerca de este aeronave en la sección de software (Sec. 3.2.1).

La segunda aeronave seleccionada es el DJI Tello (Fig. 3.1). Una aeronave propietaria (del fabricante chino *DJI*) pensada para vuelos en interiores debido a su pequeño tamaño y peso (ver Tabla 3.1). Incluye un sensor telemétrico (*optical flow*) y un barómetro, para cálculos de odometría, un sensor de visión y una antena WiFi para las comunicaciones.

Tabla 3.1. Especificaciones del DJI Tello.

| Características | Valores |
|----------------------|----------------|
| Peso (g) | 80 |
| Tamaño (mm) | 98×92.5×41 |
| Distancia máx. (m) | 100 |
| Velocidad máx. (m/s) | 10 |
| Tiempo máx. (min) | 13 |
| Altura máx. (m) | 30 |
| Cámara (MP) | 5 |
| FOV (°) | 82.6 |
| Vídeo | HD 720p 30 fps |

Fuente: DJI



Fig. 3.1. DJI Tello.

En tercer lugar, se ha elegido un dron de construcción propia y desarrollado por el Laboratorio de Sistemas Inteligentes (LSI) de la Universidad Carlos III de Madrid (UC3M) [42]. La aeronave posee una controladora PixHawk v1, con el autopiloto PX4 en su última versión estable (v1.12.3). La aeronave lleva a bordo un ordenador de computo que permite realizar el procesamiento de la información in-situ. Además incluye una antena GPS, un receptor de telemetría y una antena WiFi, entre otros sensores. La Tabla 3.2 recoge las características principales de la aeronave, mientras que la Figura 3.2 muestra el cuadri-cóptero.

Tabla 3.2. Especificaciones del drone no comercial de construcción propia.

| Características | Valores |
|----------------------|---------|
| Peso (g) | ? |
| Tamaño (mm) | ?×?×? |
| Distancia máx. (m) | ? |
| Velocidad máx. (m/s) | ? |
| Tiempo máx. (min) | ? |
| Altura máx. (m) | ? |

Fuente: DJI

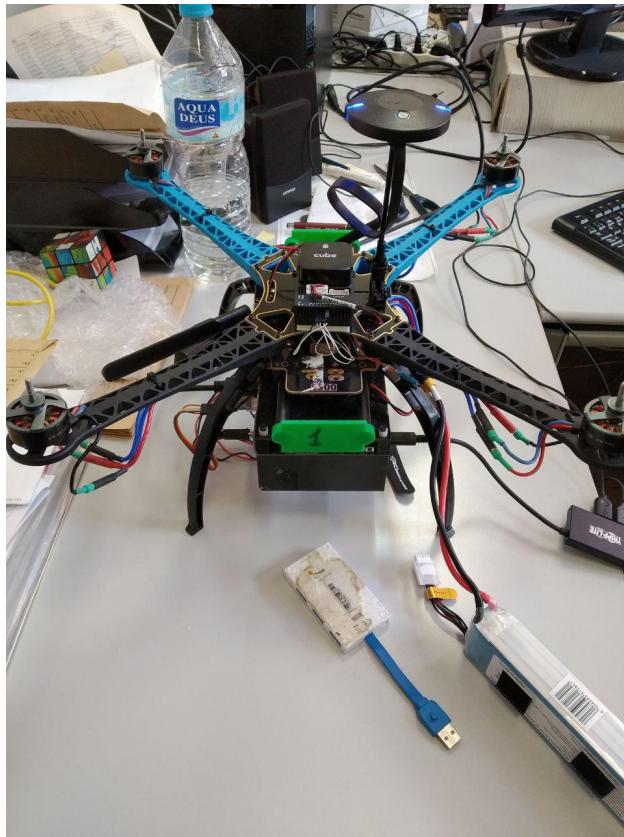


Fig. 3.2. Drone no comercial.

3.1.2. Dispositivos Embebidos

En la apartado anterior, con la última aeronave se ha presentado un dispositivo embebido. Estos equipos están presentes cuando existe interés de embarcar parte del software en la aeronave. Ejecutar ciertos procesos en la aeronave ofrece un salto en rendimiento en algoritmos en tiempo real, muy presentes en la visión por computador. Además, al tratarse de vehículos aéreos no es posible en muchos correr estos algoritmos en ordenadores

portátiles como sí se hace tradicionalmente en vehículos terrestres, debido a la latencia en las comunicaciones.

Por ello, debido al mencionado aumento de interés por aplicaciones de visión en tiempo real ha propiciado el desarrollo de dispositivos embebidos de baja potencia para su integración en sistemas robóticos móviles. Ejemplo de ello son los dispositivos como Arduino [43] o Raspberry Pi [44], utilizados en robots como PiBot [45].

Sin embargo, para ejecutar algoritmos complejos, como redes neuronales, no es suficiente con los dispositivos mencionados y es necesario utilizar dispositivos específicos. Una alternativa viable son los dispositivos *Jetson* fabricados por *NVIDIA* [46]. Cada NVI-DIA Jetson es un sistema en módulo (SOM) completo que incluye CPU, GPU, memoria, administración de energía, interfaces de alta velocidad y más, que permiten ejecutar CUDA [47], una biblioteca de computación paralela de bajo nivel, así como varios kits de herramientas (como JetPack SDK [48]) diseñados para optimizar los procesos que se ejecuten en el dispositivo.

El tamaño y consumo de estos dispositivos hacen que sea sistemas ideales para embarcar en vehículos aéreos. Existen diferentes modelos disponibles como la Jetson Nano, la Jetson TX2 o la Jetson AGX Xavier, seleccionada para embarcar en la aeronave. La Figura 3.3 muestra los principales elementos del dispositivo, mientras que la Tabla 3.3 recoge sus características.

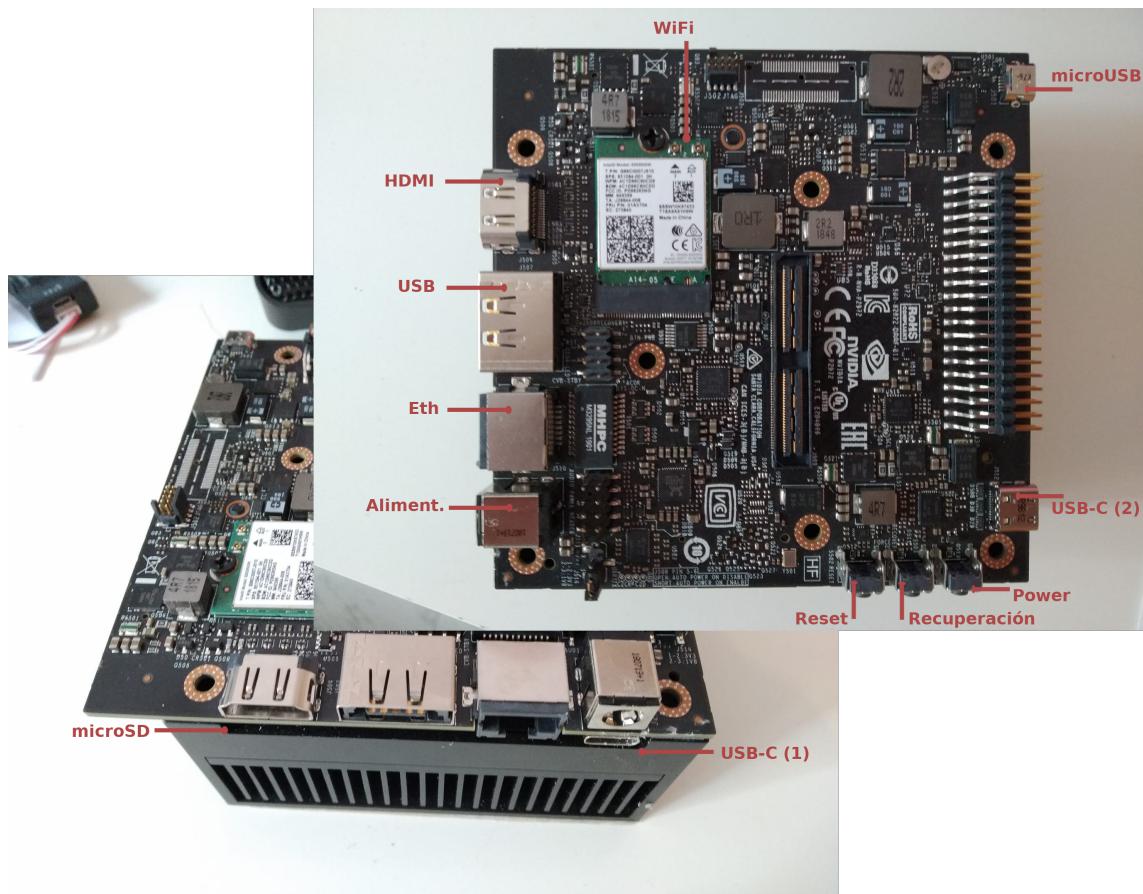


Fig. 3.3. NVIDIA Jetson AGX Xavier.

Tabla 3.3. Especificaciones de Jetson AGX Xavier.

| Características | Valores |
|----------------------|-----------------------|
| Peso (g) | 630 |
| Tamaño (mm) | 100×86 |
| Potencia (W) | 10 15 30 |
| GPU | Volta 64 Tensor Cores |
| CPU | 8-Core ARM 64-Bit |
| Rendimiento (TFLOPS) | 11 |
| Memoria (GB) | 16 |
| Almacenamiento (GB) | 32 |

Fuente: NVIDIA

3.1.3. Sensores de visión

Para el desarrollo de aplicaciones de control visual, es necesario poseer un sensor de visión a bordo de la aeronave. Tanto la aeronave simulada como el Tello poseen una cámara embarcada, en el primer caso un sensor simulado (del cual se darán más detalles en la Sección 3.2.1), mientras que en el segundo el sensor viene integrado en el propio drone.

Sin embargo, el drone de desarrollo propio es necesario seleccionar una cámara, ya que el diseño inicial no incluye ninguna. A la hora de elegir el sensor, se ha apostado por el uso de un sensor rgb básico, teniendo un sistema similar al de resto de las aeronaves.

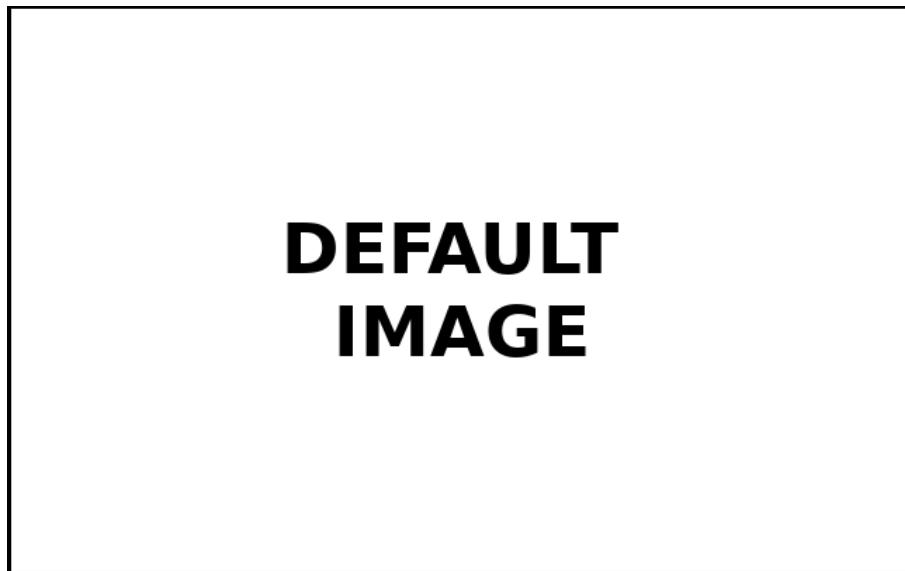


Fig. 3.4. Cámara Victure AC600.

En concreto, el sensor seleccionado es una cámara USB Victure AC600 (Fig. 3.4). Además, la Tabla 3.4 recoge las características de la cámara. Finalmente, el montaje final del sistema de la cámara junto a la aeronave se muestra en la Figura 3.5.

Tabla 3.4. Especificaciones de la cámara Victure AC600.

| Características | Valores |
|-----------------|-------------------|
| Peso (g) | 78 |
| Tamaño (mm) | 60x40x25 |
| Resolución (MP) | 16 |
| FOV (°) | 170 |
| Vídeo | 4k ultraHD 25 fps |

Fuente: Victure

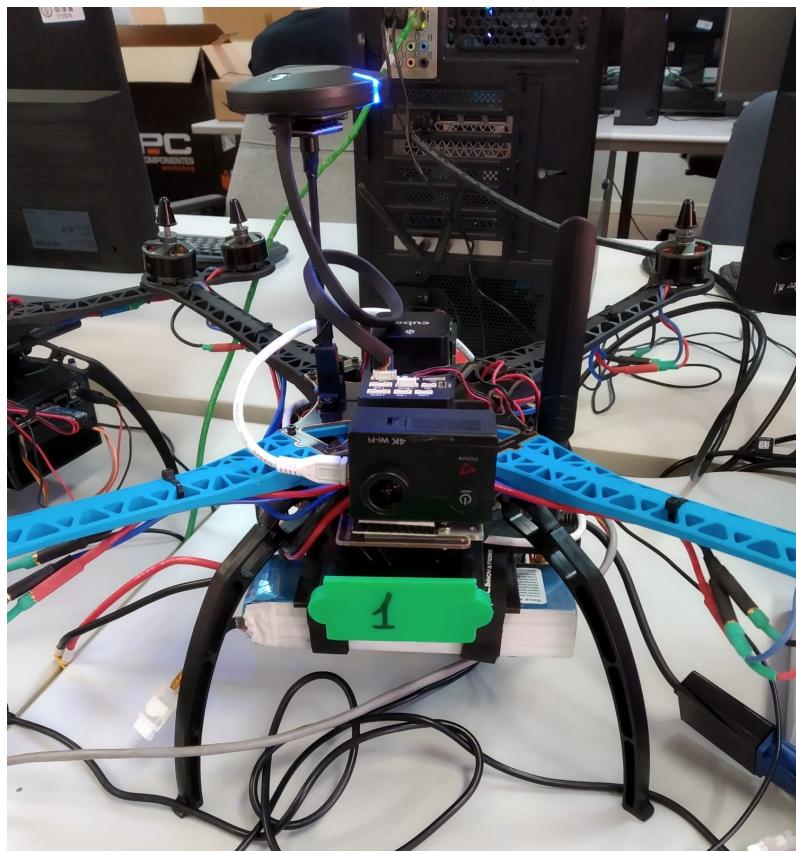


Fig. 3.5. Aeronave con cámara instalada.

3.2. SOFTWARE

El sistema operativo utilizada durante el desarrollo y las pruebas ha sido Ubuntu, la distribución de Linux basada en Debian. En concreto, la edición utilizada es la versión con soporte de largo plazo, **Ubuntu 18.04.3 LTS (Bionic Beaver)** [49]. El motivo de esta elección es debido a que Ubuntu suele ser la primera opción en aplicaciones relacionadas con el software libre y la robótica. Además, como veremos en a lo largo de la sección, las principales herramientas de simulación de drones utilizan esta plataforma.

El lenguaje elegido para el desarrollo de la aplicación ha sido Python [50]. Este lenguaje creado a principios de 1990 por Guido van Rossum en los Países Bajos es un lenguaje de programación interpretado, interactivo y orientado a objetos. Sus principales ventajas, que han motivado su elección para este proyecto, son una sintaxis muy clara y su portabilidad. En concreto, la versión utilizada es **Python v2.7.17**.

Además, para ciertos aspectos del desarrollo se ha utilizado también C++ [51]. C++ es un lenguaje de programación multiparadigma diseñado en 1979 por Bjarne Stroustrup. La intención de su creación fue extender al lenguaje de programación C mecanismos que permiten la manipulación de objetos. En concreto, la versión **C++14** junto al compilador versión **g++ 7.5.0**.

La plataforma robótica de software seleccionada es ROS. Además de ser la plataforma de uso más extendido, dispone de librerías en diferentes lenguajes que facilitan su uso. Una de ellas es *rospy* [52], un cliente para Python que permite interactuar rápidamente con los nodos, servicios y parámetros de ROS.

ROS posee diferentes paquetes que facilitan el desarrollo de software en múltiples ámbitos. Dentro de la robótica aérea, existe una colección de nodos de comunicación extendible MAVLink para ROS conocida como **MAVROS (Micro Air Vehicles ROS)** [53]. Este paquete de ROS proporciona un controlador de comunicación para varios autopilotos con protocolo de comunicación MAVLink, junto a una colección de nodos, servicios y parámetros que aseguran una correcta comunicación con la aeronave. La versión de MAVROS es **v1.9.0** con la colección de mensajes de **MAVLink v2021.3.3**.

Para el procesamiento de imágenes, se ha elegido **OpenCV (Open Source Computer Vision)** [54]. OpenCV es una biblioteca de código abierto C++/Python/Java (escrita de forma nativa en C++) utilizada en Visión por Computador. Entre los métodos clásicos y de última generación que incluye, se pueden encontrar varias funciones adecuadas para reconocimiento facial, seguimiento ocular, establecimiento de marcadores para realidad aumentada, etc. Debido a su excelente rendimiento, esta biblioteca ha logrado convertirse en el estándar *de facto* para todo tipo de usuarios. La versión utilizada es **v3.2.0**.

Otra biblioteca muy útil para el trabajo matricial con imágenes es **NumPy** [55]. NumPy es una biblioteca para Python que da soporte para crear vectores y matrices grandes multidimensionales, junto con una gran colección de funciones matemáticas de alto nivel para operar con ellas. La versión utilizada es la **v1.13.3**.

3.2.1. Simulación

Entre los drones utilizados se ha introducido en la sección anterior el 3DR Iris simulando. Tal como se ha adelantado, la simulación se realiza mediante el SITL de PX4 y sobre el simulador *Gazebo*. Este simulador de código abierto es por excelencia el más utilizado en aplicaciones de robótica y visión artificial. Debido a su gestión abierta permite la integración de múltiples vehículos, mundos, sensores, físicas, etc.

La versión utilizada durante el proyecto es **Gazebo9**. El modelo del Iris simulado en el

simulador se puede observar en la Figura 3.6.

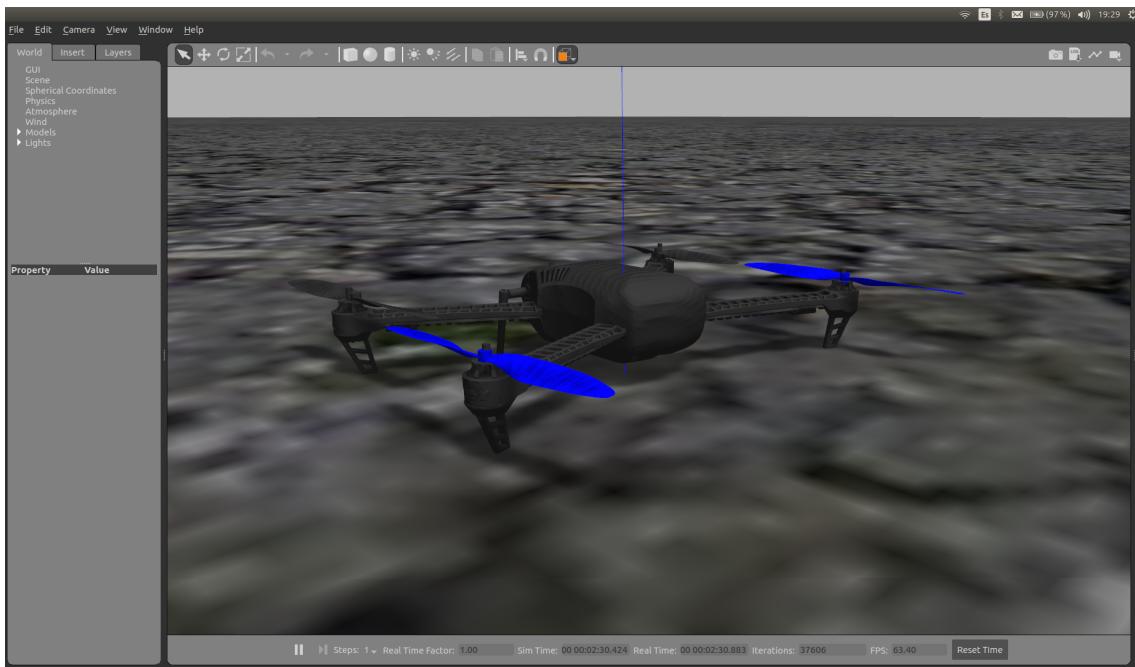


Fig. 3.6. 3DR Iris simulado mediante *Gazebo*.

El firmware utilizado por la controladora es PX4 con la versión v1.11.3, cuyo esquema de SITL se puede observar en la Figura 3.7.

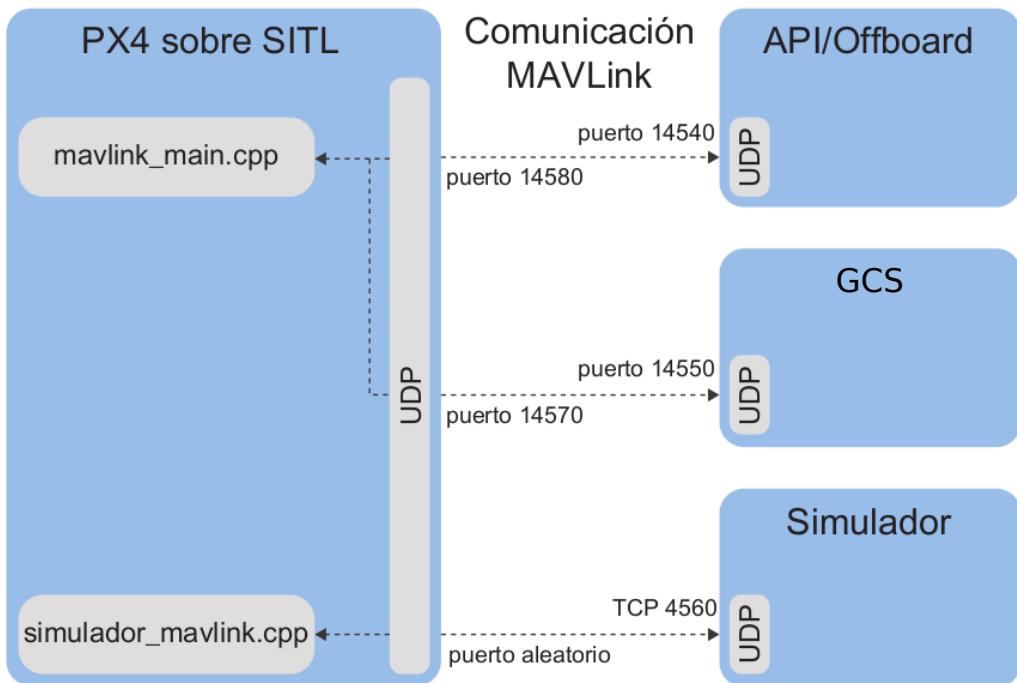


Fig. 3.7. Esquema de PX4 sobre SITL [56].

3.2.2. NVIDIA JetPack

El dispositivo embebido utilizado, la NVIDIA Jetson AGX Xavier, sigue unas pautas de diseño integradas muy optimizadas. NVIDIA desarrolla y mantiene una versión personalizada de Ubuntu Linux, llamada NVIDIA JetPack, y está disponible para descargar e instalar como firmware de la placa [48]. Esta implementación personalizada incluye interfaces de bajo nivel para implementar operaciones de computación paralelas (CUDA) y varias optimizaciones SDK (kits de desarrollo de software), como TensorRT.

Para el sistema desarrollado, la versión utilizada es **JetPack 4.6** (L4T 32.6.1).

3.2.3. Aprendizaje profundo

Para este trabajo no se ha desarrollado ninguna red neuronal, ya que no es uno de los objetivos del proyecto. En cambio, se ha utilizado ya creada y ampliamente usada, YOLO (*You Only Look Once*). YOLO es un sistema de detección de objetos en tiempo real de última generación, diseñado por Joseph Redmon hasta su tercera versión [57]. Tras el abandono del autor, fue continuada por Alexey Bochkovskiy, creador de las versiones más actuales [58].

Su enfoque, muy novedoso en su lanzamiento, permite alcanzar velocidad de ejecución muy elevadas. Con su método, la red neuronal se aplica una única vez sobre la imagen. Esta red divide la imagen en regiones y predice cuadros delimitadores y probabilidades para cada región.

La versión utilizada de la red para este trabajo **YOLOv4**. Se han utilizado la configuración y pesos por defecto de la versión YOLOv4 y YOLOv4-tiny. Dichos pesos se han obtenido entrenando con la base de datos Microsoft COCO (*Common Objects in Context*) [59], la cual posee 80 clases diferentes y más de 300 mil imágenes y un millón y medio de objetos etiquetados.

3.3. Método

La metodología seguida durante el desarrollo es la filosofía del código libre. Todo el código se encuentra disponible en abierto en *GitHub* [60]. *GitHub* es una plataforma de desarrollo colaborativo para alojar proyectos utilizando el sistema de control de versiones *git*. Esta plataforma permite además una metodología en espiral con incidencias y parches.

El código fuente del proyecto se encuentra en diferentes repositorios públicos. Parte del código se aloja en *JdeRobot/drones*¹, mientras que otra parte del código se puede encontrar en *RoboticsLabURJC/2021-tfm-pedro-arias*².

El software en el repositorio *drones* ha sido desarrollado por la asociación de robótica

¹<https://github.com/JdeRobot/drones>

²<https://github.com/RoboticsLabURJC/2021-tfm-pedro-arias>

JdeRobot [61], ligada a la Universidad Rey Juan Carlos (URJC), y entre sus desarrolladores se encuentra el autor de este trabajo. El código del segundo repositorio ha sido desarrollado en su totalidad por el autor y con motivo de este proyecto.

4. INFRAESTRUCTURA DESARROLLADA

 Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pelentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

5. APLICACIÓN

 Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pelentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

5.1. Diseño

 Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pelentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

5.1.1. Percepción

 Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pelentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

5.1.2. Control

 Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

5.2. Implementación

 Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

5.3. Experimentos

5.3.1. Base

 Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

5.3.2. Sigue color

 Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

5.3.3. Sigue persona

 Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

```
1 #include <stdio.h>
2
3 int main()
4 {
5     printf("Hola mundo\n");
6     return 0;
7 }
```

Cód. 5.1. Test

Para un control por posición visual o sin GPS (p.e. en interiores) es necesario utilizar otro tipo de mensajes. Una posible alternativa es el mensaje *SET_POSITION_TARGET_LOCAL_NED* que permite establecer la posición, velocidad, aceleración o fuerza deseada del vehículo en un marco de coordenadas local NED (*North East Down*), relativo al propio robot aéreo en cada momento.

En segundo lugar, *Drone Wrapper*, ha sido desarrollada por la asociación de robótica *JdeRobot* [61] ligada a la Universidad Rey Juan Carlos, y entre sus desarrolladores

se encuentra el autor de este proyecto. Ofrece *drivers* para diferentes aeronaves, herramientas de visualización y teleoperación, y una API que permite abstraerse del sistema y desarrollar aplicaciones sobre la misma.

6. CONCLUSIONES Y LÍNEAS FUTURAS

 Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pelentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

6.1. Conclusiones

 Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pelentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

6.2. Líneas futuras

 Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pelentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

BIBLIOGRAFÍA

- [1] F. S. Martín et al., “Historia de la robótica: de Arquitas de Tarento al robot Da Vinci (Parte I),” *Actas Urológicas Españolas*, vol. 31, n.º 2, pp. 69-76, 2007.
- [2] A. O. Baturone, *Robótica: manipuladores y robots móviles*. Marcombo, 2005.
- [3] R. A. Española, *Diccionario de la lengua española*, 23.º ed. <https://dle.rae.es>, Último acceso: 02-02-2020.
- [4] J. R. de Garibay Pascual, “Robótica: Estado del arte,” *Universidad de Deuston. Número. Fecha*, p. 54, 2006.
- [5] A. Barrientos, J. Del Cerro, P. Gutiérrez, R. San Martín, A. Martínez y C. Rossi, “Vehículos aéreos no tripulados para uso civil. Tecnología y aplicaciones,” *Universidad politécnica de Madrid, Madrid*, 2007.
- [6] D. G. de Tráfico, *Tráfico distribuye los 39 drones que vigilarán las carreteras españolas este verano*, https://www.dgt.es/prensa/notas-de-prensa/2021/Trafico_distribuye_los_39_drones_que_vigilaran_las_carreteras_espanolas_este_verano.shtml, Último acceso: 12-07-2021.
- [7] M. de Fomento, *Plan Estratégico para el desarrollo del sector civil de los drones en España 2018-2021*, <https://www.fomento.gob.es/NR/rdonlyres/7B974E30-2BD2-46E5-BEE5-26E00851A455/148411/PlanEstrategicoDrones.pdf>, Último acceso: 07-07-2021.
- [8] J. Kim, S. Kim, C. Ju y H. I. Son, “Unmanned aerial vehicles in agriculture: A review of perspective of platform, control, and applications,” *IEEE Access*, vol. 7, pp. 105 100-105 115, 2019.
- [9] P. Radoglou-Grammatikis, P. Sarigiannidis, T. Lagkas e I. Moscholios, “A compilation of UAV applications for precision agriculture,” *Computer Networks*, vol. 172, p. 107 148, 2020.
- [10] H. Yao, R. Qin y X. Chen, “Unmanned aerial vehicle for remote sensing applications—A review,” *Remote Sensing*, vol. 11, n.º 12, p. 1443, 2019.
- [11] M. de Ciencia e Innovación, *Estrategia Española de Ciencia, Tecnología e Innovación 2021-2027*, <https://www.ciencia.gob.es/stfls/MICINN/Ministerio/FICHEROS/EECTI-2021-2027.pdf>, Último acceso: 07-07-2021.
- [12] DronecodeProject, *QGroundControl*, <http://qgroundcontrol.com/>, Último acceso: 12-07-2021.
- [13] ArduPilot-Dev-Team, *Mission Planner*, <https://ardupilot.org/planner/>, Último acceso: 12-07-2021.

- [14] CVAR-UPM, *Aerostack Github*, <https://github.com/aerostack>, Último acceso: 24-12-2021.
- [15] J. L. Sanchez-Lopez et al., “Aerostack: An architecture and open-source software framework for aerial robotics,” en *2016 International Conference on Unmanned Aircraft Systems (ICUAS)*, IEEE, 2016, pp. 332-341.
- [16] U. P. de Madrid, *Grupo de Visión por Computador y Robótica Aérea*, <https://cvar-upm.github.io/>, Último acceso: 25-12-2021.
- [17] M. Quigley et al., “ROS: an open-source Robot Operating System,” en *ICRA workshop on open source software*, Kobe, Japan, vol. 3, 2009, p. 5.
- [18] P. Fitzpatrick, E. Ceseracciu, D. E. Domenichelli, A. Paikan, G. Metta y L. Natale, “A middle way for robotics middleware,” *J. Softw. Eng. Robot*, vol. 5, n.º 2, pp. 42-49, 2014.
- [19] G. Metta, P. Fitzpatrick y L. Natale, “YARP: yet another robot platform,” *International Journal of Advanced Robotic Systems*, vol. 3, n.º 1, p. 8, 2006.
- [20] H. Bruyninckx, “Open robot control software: the OROCOS project,” en *Proceedings 2001 ICRA. IEEE international conference on robotics and automation (Cat. No. 01CH37164)*, IEEE, vol. 3, 2001, pp. 2523-2528.
- [21] M. Montemerlo, N. Roy y S. Thrun, “Perspectives on standardization in mobile robot programming: The Carnegie Mellon navigation (CARMEN) toolkit,” en *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453)*, IEEE, vol. 3, 2003, pp. 2436-2441.
- [22] I. Open Source Robotics Foundation, *OSRF*, <https://www.openrobotics.org/>, Último acceso: 13-07-2021.
- [23] K. Team, *Keras*, <https://keras.io/>, Último acceso: 31-12-2021.
- [24] Google, *TensorFlow*, <https://www.tensorflow.org/>, Último acceso: 13-07-2021.
- [25] F. AI, *PyTorch*, <https://pytorch.org/>, Último acceso: 31-12-2021.
- [26] J. Redmon, *Darknet: Open Source Neural Networks in C*, <http://pjreddie.com/darknet/>, 2013–2016.
- [27] I. Dronecode Project, *Bucle de control de un UAV*, <https://dev.px4.io/master/en/concept/architecture.html>, Último acceso: 24-02-2020.
- [28] DJI, *DJI drones*, <https://www.dji.com/es>, Último acceso: 12-07-2021.
- [29] P. D. SAS, *Parrot drones*, <https://www.parrot.com/es/drones>, Último acceso: 12-07-2021.
- [30] 3. Robotics, *3DR Drone*, <https://www.3dr.com/>, Último acceso: 12-07-2021.
- [31] A. Anwar y A. Raychowdhury, “Autonomous navigation via deep reinforcement learning for resource constraint edge nodes using transfer learning,” *IEEE Access*, vol. 8, pp. 26 549-26 560, 2020.

- [32] A. Scannell, C. H. Ek y A. Richards, “Trajectory Optimisation in Learned Multi-modal Dynamical Systems Via Latent-ODE Collocation,” en *IEEE International Conference on Robotics and Automation (ICRA)*, 2021.
- [33] L. Meier, D. Honegger y M. Pollefeys, “PX4: A node-based multithreaded open source robotics framework for deeply embedded platforms,” en *2015 IEEE international conference on robotics and automation (ICRA)*, IEEE, 2015, pp. 6235-6240.
- [34] A. D. Team y Community, *Ardupilot*, <https://ardupilot.org/>, Último acceso: 11-02-2020.
- [35] L. Meier, P. Tanskanen, F. Fraundorfer y M. Pollefeys, “Pixhawk: A system for autonomous flight using onboard computer vision,” en *2011 IEEE International Conference on Robotics and Automation*, IEEE, 2011, pp. 2992-2997.
- [36] O. S. R. Foundation, *Gazebo*, <http://gazebosim.org/>, Último acceso: 11-02-2020.
- [37] M. Research, *AirSim*, <https://microsoft.github.io/AirSim/>, Último acceso: 13-07-2021.
- [38] I. Dronecode Project, *jmavsim*, <https://github.com/PX4/jMAVSIM>, Último acceso: 13-07-2021.
- [39] DronecodeProject, *MAVLink*, <https://mavlink.io/en/>, Último acceso: 12-07-2021.
- [40] I. Dronecode Project, *Documentación de MAVLink con los mensajes disponibles*. <https://mavlink.io/en/messages/common.html>, Último acceso: 26-02-2020.
- [41] ——, *Microservicios de MAVLink*, <https://mavlink.io/en/services/>, Último acceso: 02-03-2020.
- [42] U. C. I. de Madrid, *Laboratoria de Sistemas Inteligentes*, <https://lsi.uc3m.es/>, Último acceso: 26-12-2021.
- [43] A. AG, *Arduino*, <https://www.arduino.cc/>, Último acceso: 27-12-2021.
- [44] R. P. Foundation, *Raspberry pi*, <https://www.raspberrypi.org/>, Último acceso: 27-12-2021.
- [45] J. Vega y J. M. Cañas, “PiBot: An open low-cost robotic platform with camera for STEM education,” *Electronics*, vol. 7, n.º 12, p. 430, 2018.
- [46] N. Corporation, *NVIDIA Jetson*, <https://www.nvidia.com/es-es/autonomous-machines/embedded-systems/>, Último acceso: 20-07-2021.
- [47] ——, *NVIDIA CUDA*, <https://developer.nvidia.com/cuda-zone>, Último acceso: 20-07-2021.
- [48] ——, *NVIDIA JetPack SDK*, <https://developer.nvidia.com/embedded/jetpack>, Último acceso: 20-07-2021.

- [49] C. Ltd., *Ubuntu 18.04.3 LTS*, <https://ubuntu.com/download/desktop>, Último acceso: 09-02-2020.
- [50] P. S. Foundation, *Pyhton*, <https://www.python.org/>, Último acceso: 09-02-2020.
- [51] C++, C++, <https://www.cplusplus.com/>, Último acceso: 28-12-2021.
- [52] I. Open Source Robotics Foundation, *rospy*, <http://wiki.ros.org/rospy>, Último acceso: 28-12-2021.
- [53] ——, *MAVROS*, <http://wiki.ros.org/mavros>, Último acceso: 13-07-2021.
- [54] O. Team, *OpenCV*, <https://opencv.org/>, Último acceso: 19-07-2021.
- [55] N. developers, *NumPy*, <https://numpy.org/>, Último acceso: 10-02-2020.
- [56] I. Dronecode Project, *PX4*, <https://px4.io/>, Último acceso: 11-02-2020.
- [57] J. Redmon y A. Farhadi, “YOLOv3: An Incremental Improvement,” *arXiv*, 2018.
- [58] A. Bochkovskiy, C.-Y. Wang y H.-Y. M. Liao, *YOLOv4: Optimal Speed and Accuracy of Object Detection*, 2020. arXiv: [2004.10934 \[cs.CV\]](https://arxiv.org/abs/2004.10934).
- [59] T.-Y. Lin et al., “Microsoft coco: Common objects in context,” en *European conference on computer vision*, Springer, 2014, pp. 740-755.
- [60] I. GitHub, *GitHub*, <https://github.com/>, Último acceso: 26-02-2020.
- [61] JdeRobot, *JdeRobot*, <https://jderobot.github.io/>, Último acceso: 13-07-2021.