UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ETSI Aeroespacial y Diseño Industrial

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Aeroespacial y Diseño Industrial

Control embarcado de un dron DJI mediante el computador Jetson Nano

Trabajo Fin de Máster

Máster Universitario en Ingeniería Aeronáutica

AUTOR/A: Rabanal Gutiérrez, Omar

Tutor/a: Rodas Jordá, Ángel

CURSO ACADÉMICO: 2023/2024

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ETSI Aeroespacial y Diseño Industrial

Trabajo Fin de Máster

Máster Universitario en Ingeniería Aeronáutica

Control embarcado de un dron DJI mediante el computador Jetson Nano

Autor: Omar Rabanal Gutiérrez

Tutor: Ángel Rodas Jordá

Universitat Politècnica de València

Escuela Técnica Superior de Ingeniería Aeroespacial y Diseño Industrial

Curso académico: 2023/2024

Resumen

El presente TFM propone embarcar un computador Jetson Nano, caracterizado por su gran potencia de cálculo, en un dron comercial DJI. Para ello, se utilizará el software estándar de control MSDK (*Mobile System Development Kit*) para drones DJI como puente entre el Jetson y el propio dron. La ventaja de dicha disposición es la posibilidad de utilizar el conocimiento del entorno que Jetson Nano puede adquirir mediante una cámara conectada al mismo, para el control adaptativo del dron. Esto permitirá disponer de una adquisición en tiempo real de imágenes que el propio dron es incapaz de ofrecer. El único requisito necesario es que el mencionado dron pueda transportar todo el *hardware* embarcado. Aunque el TFM podría aplicarse a cualquier dron capaz de transportar la placa Jetson, las pruebas de campo se realizarán concretamente sobre el dron DJI Phantom 4 Advance+, programando en Java su controladora GL300E. Por otra parte, el código de control del Jetson se implementará en el lenguaje de programación Python utilizando librerías que permitan explotar la potencia de dicho hardware.

Palabras clave: Jetson Nano, DJI Mobile SDK, DJI Phantom, Python

Resum

Aquesta tesi de màster proposa embarcar un ordinador Jetson Nano, conegut per la seua gran potència de càlcul, en un dron comercial DJI. Per a això, s'utilitzarà el programari estàndard de control MSDK (Mobile System Development Kit) per a drons DJI com a pont entre el Jetson i el propi dron. L'avantatge d'aquesta disposició és la possibilitat d'utilitzar el coneixement de l'entorn que el Jetson Nano pot adquirir mitjançant una càmera connectada a aquest, per al control adaptatiu del dron. Això permetrà disposar d'una adquisició en temps real d'imatges que el propi dron és incapaç d'oferir. L'únic requisit necessari és que el mencionat dron puga transportar tot el *hardware* embarcat. Encara que la tesi podria aplicar-se a qualsevol dron capaç de transportar la placa Jetson, les proves de camp es realitzaran concretament sobre el dron DJI Phantom Advance+, programant en Java el seu controladora GL300E. D'altra banda, el codi de control del Jetson s'implementarà en el llenguatge de programació Python utilitzant llibreries que permeten explotar la potència de dit *hardware*.

Paraules clau: Jetson Nano, DJI Mobile SDK, DJI Phantom, Python

Abstract

This Master's Thesis proposes to onboard a Jetson Nano computer, known for its high computing power, onto a commercial DJI drone. To achieve this, the standard MSDK (Mobile System Development Kit) software for DJI drones will be used as a bridge between the Jetson and the drone itself. The advantage of this setup is the ability to leverage the environmental awareness that the Jetson Nano can acquire via a connected camera, facilitating adaptive drone control. This setup enables real-time image acquisition that the drone alone cannot provide. The only requirement is that the drone must be capable of carrying all the onboard hardware. While this thesis could apply to any drone capable of transporting the Jetson board, field tests will specifically be conducted on the DJI Phantom Advance+ drone, with its GL300E controller programmed in Java. Additionally, the Jetson control code will be implemented in Python, utilizing libraries that harness the hardware's full potential.

Keywords: Jetson Nano, DJI Mobile SDK, DJI Phantom, Python

Índice general

Índice de figuras

Índice de tablas

Índice de códigos

Nomenclatura

Latinos

c_{a}	Amortización	euros/año
f_{x}	Distancia focal en la dirección horizontal	Píxeles
f_{y}	Distancia focal en la dirección vertical	Píxeles
h	Horas trabajadas al año	h
n_{a}	Período de amortización	años
t_{h}	Tasa horaria	euros/h
VC	Valor de compra	euros
VR	Valor residual al cabo del período de amortización	euros
AFOVx	Campo de visión angular horizontal	rad
AFOVy	Campo de visión angular vertical	rad
H	Alto de la imagen	Píxeles
K	Matriz de parámetros intrínsecos	Píxeles
s	Coeficiente de sesgo	–
W	Ancho de la imagen	Píxeles

Siglas

AESA	Agencia Estatal de Seguridad Aérea
AFOV	*Angular Field of View*
AMC	*Acceptable Means of Compliance*
API	*Application Programming Interface*

AR *Augmented Reality*

ArUco *Augmented Reality University of Cordoba*

CAGR *Compound Annual Growth Rate*

CCD *Charge-Coupled Device*

CMOS *Complementary Metal-Oxide-Semiconductor*

CNN *Convolutional Neural Network*

CSI *Camera Serial Interface*

DGAC Dirección General de Aviación Civil

EASA *European Union Aviation Safety Agency*

GCS *Ground Control Station*

GIL *Global Interpreter Lock*

GM *Guidance Material*

GS *Ground Speed*

HDMI *High-Definition Multimedia Interface*

HDMI *High-Definition Multimedia Interface*

IA Inteligencia Artificial

IA Inteligencia artificial

IDE *Integrated Development Environment*

JSON *JavaScript Object Notation*

JVM *Java Virtual Machine*

MAVLink *Micro Air Vehicle Link*

MSDK *Mobile Software Development Kit*

ODS Objetivos de Desarrollo Sostenible

OpenCV *Open Source Computer Vision Library*

OSDK *Onboard Software Development Kit*

PSDK *Payload Software Development Kit*

ROS *Robot Operating System*

RPA *Remotely Piloted Aircraft*

RPAS *Remotely Piloted Aircraft System*

RTF *Ready-to-Fly*

RTK *Real Time Kinematic*

SDK *Software Development Kit*

SES *Single European Sky*

SSD *Solid-State Drive*

SSH *Secure Shell*

TFM Trabajo Fin de Máster

TOPS *Tera Operations Per Second*

UAS *Unmanned Aerial System*

UAV *Unmanned Aerial Vehicle*

UE Unión Europea

USB *Universal Serial Bus*

USD *United States Dollar*

VNC *Virtual Network Computing*

VS *Vertical Speed*

VxC Visión por computador

WiFi *Wireless Fidelity*

Capítulo 1

Introducción, motivación y objetivos

En los últimos años, los drones han experimentado un auge significativo, revolucionando diversos sectores industriales y comerciales. Este crecimiento se debe a la combinación de avances tecnológicos, la reducción de costos y la expansión de sus aplicaciones. Según el informe de Fortune Business Insights [1], una empresa de investigación de mercado y consultoría que ofrece análisis profundos y recomendaciones estratégicas que permite a las compañías del sector navegar entornos competitivos y lograr crecimientos notables en los mismos, el mercado global de drones comerciales fue valorado en USD[1] 8,77 mil millones en 2022 y se espera que crezca a una tasa compuesta anual (CAGR)[2] del 25,82 % hasta 2030, alcanzando un valor en USD de 54,81 mil millones.

En consonancia con la información expuesta en el párrafo anterior, a continuación, se muestra un gráfico, Figura 1.1, que pone de manifiesto el inminente crecimiento del mercado de drones y la proyección a futuro del mismo. El gráfico, basado en un informe de Drone Industry Insights [3] [2], también desglosa la participación del mercado

[1] USD es la abreviatura de *United States Dollar* (dólar estadounidense). Es la moneda oficial de los Estados Unidos y una de las monedas más utilizadas en transacciones internacionales.

[2] CAGR significa *Compound Annual Growth Rate* (tasa compuesta anual de crecimiento). Es una medida utilizada para describir la tasa de crecimiento de una inversión o un mercado durante un período específico de tiempo, asumiendo que el crecimiento se produce a una tasa constante cada año. Aplicado al caso descrito en la presente memoria, si se espera que el mercado de drones crezca a una tasa compuesta anual (CAGR) del 25,82 % hasta 2030, significa que, en promedio, el mercado crecerá un 25,82 % cada año durante ese período.

[3] Drone Industry Insights (Droneii) es una empresa de investigación de mercado y consultoría especializada exclusivamente en drones comerciales. Fue fundada en 2015 en Hamburgo, Alemania, y se ha convertido en la principal fuente global de inteligencia de mercado sobre drones comerciales. Los clientes de Droneii incluyen a Airbus, Boeing, EY, Hexagon, Intel, PwC, Lufthansa, Sony e incluso el gigante de los drones DJI. Además, ha sido destacada en publicaciones reconocidas como The New

por segmentos en 2023, donde los servicios representan el 80,1 % del mercado, el hardware el 16 %, y el software el 3,9 %. Esto indica que la mayor parte de los ingresos en la industria de drones proviene de los servicios, que incluyen operaciones internas de drones, operaciones de drones para terceros, integración de sistemas, asesoría, educación, simulación y entrenamiento.

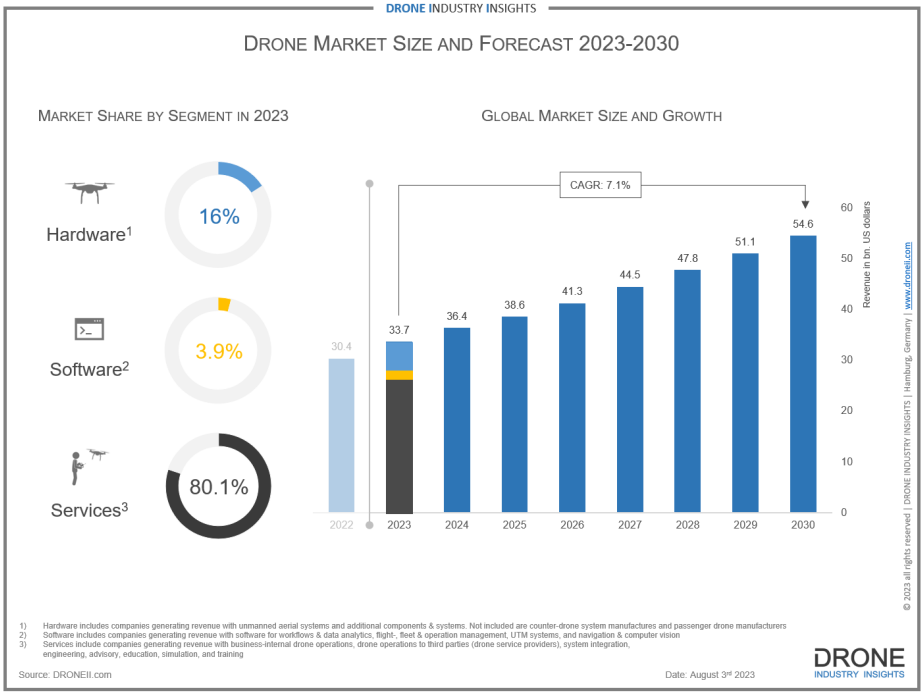


Figura 1.1: Crecimiento y proyección a futuro del mercado de drones comerciales [2].

Respecto a la información y los datos expuestos anteriormente, es interesante notar que las tasas de crecimiento proyectadas para el mercado de drones varían significativamente entre distintas fuentes. Estas diferencias se deben a diversos factores, como las metodologías empleadas, los alcances del mercado considerados y las suposiciones económicas y regulatorias adoptadas. Por ejemplo, Drone Industry Insights proyecta una tasa compuesta anual (CAGR) del 7,1 % hasta 2030, mientras que Fortune Business Insights estima un CAGR del 25,82 % en el mismo período. Esta variabilidad se debe a diferencias en la definición del mercado (inclusión de drones de consumo vs. solo drones comerciales), los modelos predictivos utilizados y los factores externos contemplados, como las políticas gubernamentales y avances tecnológicos. A pesar de estas discrepancias, lo que subyace es el consenso sobre el crecimiento y expansión sos-

York Times, The Economist, The Wall Street Journal, Harvard Business Review, Bloomberg y The Washington Post, entre otras.

tenidos del mercado de drones, destacando su potencial transformador y su creciente adopción en múltiples sectores industriales y comerciales.

Una vez presentado el crecimiento del mercado de los drones y sus prometedoras proyecciones a futuro, es interesante preguntarse qué factores han impulsado este notable desarrollo y por qué se está produciendo precisamente en este momento histórico. En primer lugar, los avances tecnológicos han mejorado significativamente las capacidades de los drones, haciéndolos más accesibles y versátiles. La integración de cámaras de alta resolución, sistemas de estabilización, y tecnologías de inteligencia artificial y aprendizaje automático ha permitido que los drones sean utilizados en una amplia gama de aplicaciones, desde la agricultura de precisión (Figura 1.2a) hasta la inspección de infraestructuras (Figura 1.2b) y la entrega de paquetes (Figura 1.2c). Además, la reducción de los costes de producción y la creciente disponibilidad de los drones han facilitado su adopción tanto por parte de consumidores como de empresas. La normativa regulatoria también ha evolucionado, proporcionando un marco más claro y seguro para la operación de drones comerciales, lo que ha incentivado a más industrias a incorporar esta tecnología en sus operaciones diarias. Otro factor importante es el aumento de la demanda en sectores específicos como la agricultura, donde los drones permiten monitorear cultivos y optimizar recursos, y la logística, donde se están explorando soluciones innovadoras para la distribución de mercancías. Durante la pandemia de COVID-19, los drones adquirieron un papel crucial en la entrega de medicamentos y vacunas en lugares remotos. También se ha visto un uso creciente en sectores de entretenimiento y medios, donde los drones abren nuevas oportunidades para la creación cinematográfica y la captura de imágenes aéreas (Figura 1.2d).

En el contexto del crecimiento y la expansión del mercado de drones, es imprescindible mencionar a DJI, una empresa que ha desempeñado, y desempeña, un papel fundamental en esta industria. DJI, con sede en Shenzhen, China, es el líder indiscutible en el mercado global de drones, dominando una participación significativa en diversos sectores. DJI ha mantenido su liderazgo gracias a su constante innovación y mejora de productos. La compañía lanza regularmente nuevos modelos que integran las últimas tecnologías, como sistemas de estabilización de tres ejes, cámaras de alta resolución y avanzados sistemas de evitación de obstáculos. Estas mejoras no solo han hecho que los drones de DJI sean más accesibles y fáciles de usar, sino que también han expandido sus aplicaciones en campos anteriormente mencionados como la cinematografía, la agricultura de precisión, la logística y la inspección de infraestructuras. Para poner en contexto estas ideas, en el artículo [7] se detalla de manera exhaustiva el camino recorrido por DJI a lo largo de los años, destacando las constantes innovaciones y mejoras tecnológicas que le han permitido alcanzar y mantener su posición de liderazgo en el mercado global de drones. En definitiva, DJI no solo lidera el mercado de drones por su innovación y calidad de productos, sino que también contribuye significativamente al crecimiento y desarrollo de esta industria, manteniéndose en la vanguardia del progreso tecnológico y adaptándose a las cambiantes demandas del mercado global.

(a) Agricultura de precisión [3].

(b) Revisión de infraestructuras [4].

(c) Operaciones logísticas [5].

(d) Creación cinematográfica [6].

Figura 1.2: Ejemplos del uso de drones en diversos sectores industriales.

Por suerte o por desgracia, los drones de DJI son drones privativos. Un dron privativo es un dispositivo cuyo *hardware* y *software* están cerrados al acceso y modificación por parte de los usuarios. Esto significa que los usuarios no pueden alterar el código fuente del *software* que controla el dron ni realizar cambios significativos en su *hardware*.

Una de las ventajas de los drones privativos, como los de DJI, es la seguridad y estabilidad que ofrecen. Al tener un control estricto sobre su *software* y *hardware*, DJI puede garantizar una experiencia de usuario consistente y minimizar problemas técnicos, lo que resulta en productos confiables y fáciles de usar para una amplia gama de usuarios, desde aficionados hasta profesionales.

Por otro lado, una de las principales desventajas de los drones privativos es la limitación en la personalización y el desarrollo comunitario. Debido a que el *software* no puede ser modificado por los usuarios, la comunidad de desarrolladores no puede crear aplicaciones personalizadas o añadir funcionalidades específicas que podrían mejorar o adaptar el dron a necesidades particulares. Esto restringe la flexibilidad y la innovación dentro de este ámbito, generando una dependencia del fabricante para el soporte y las actualizaciones.

Para abordar la limitación en la personalización y permitir que los desarrolladores creen sus propias aplicaciones, DJI lanzó su *Software Development Kit* (SDK) en 2014. El

Figura 1.3: Ejemplo de drones fabricados por DJI [8].

SDK de DJI es un conjunto de herramientas que proporciona a los desarrolladores acceso a las capacidades de los drones de DJI, permitiéndoles crear aplicaciones personalizadas que amplíen las funcionalidades de estos dispositivos. Con el SDK, los desarrolladores pueden controlar aspectos del vuelo, la cámara, la transmisión en vivo y más, resolviendo así el problema de la falta de personalización en los drones privativos.

El DJI SDK se presenta en varias versiones [9], cada una diseñada para diferentes propósitos y tipos de aplicaciones, como se describe a continuación:

- ***Mobile* SDK (MSDK):** Permite a los desarrolladores crear aplicaciones móviles tipo Android que interactúan con los drones de DJI.
- **UX SDK:** Facilita el desarrollo mediante el uso de elementos de interfaz de usuario predefinidos, cubriendo las funcionalidades básicas, lo que acelera significativamente el tiempo de desarrollo.
- **Windows SDK:** Permite desarrollar aplicaciones para Windows que controlen drones DJI, integrando capacidades de vuelo y transmisión de datos en un entorno Windows.
- ***Onboard* SDK (OSDK):** Expande las capacidades de las plataformas aéreas permitiendo la integración de computación a bordo y sensores externos para tareas avanzadas.

- ***Payload* SDK (PSDK):** Facilita la integración de sensores personalizados en los drones industriales de DJI, permitiendo aplicaciones especializadas en sectores como la agricultura, la inspección y la construcción.
- ***Cloud* API:** Ofrece acceso fácil a plataformas en la nube de terceros, permitiendo la gestión y control de flotas de drones a gran escala.
- ***Edge* SDK:** Proporciona un centro de datos eficiente y seguro en los *docks* de DJI [4], mejorando la transmisión de vídeo y la comunicación en operaciones prolongadas.

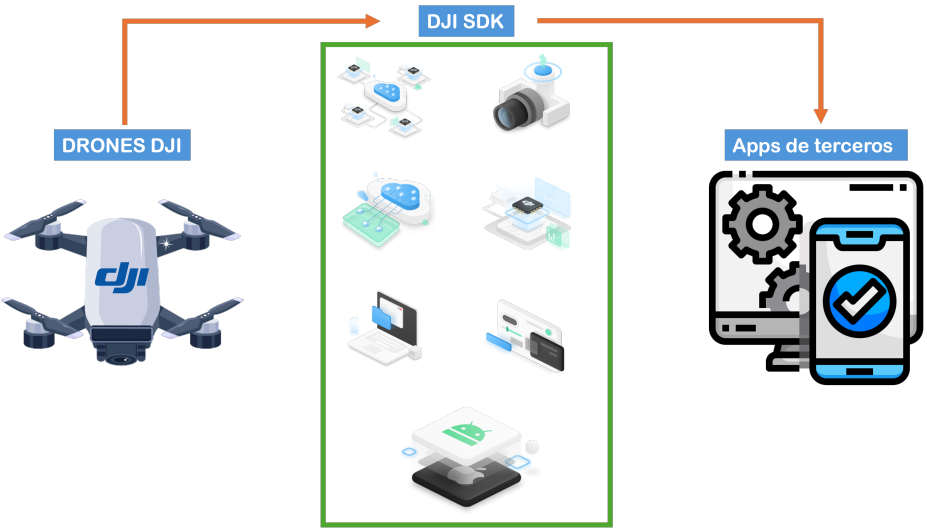


Figura 1.4: Fundamento del DJI SDK [Elaboración Propia].

El lanzamiento del SDK de DJI ha sido crucial para ampliar el ecosistema de aplicaciones disponibles para sus drones, permitiendo a los desarrolladores abordar necesidades específicas que el software nativo de DJI no cubre. Esto ha potenciado la flexibilidad y la innovación, permitiendo a terceros aportar sus soluciones y mejoras. Sin embargo, no todo son ventajas con el SDK. A pesar de otorgar mayor poder de personalización a los desarrolladores, sigue siendo un ecosistema cerrado. Cualquier modificación en los drones de DJI debe realizarse a través del SDK, lo que puede limitar la creatividad y las posibilidades. En muchos casos, los desarrolladores deben encontrar soluciones ingeniosas para sortear estas restricciones y lograr desarrollar las aplicaciones que desean,

[4]Los *docks* de DJI son estaciones base que permiten el despliegue, carga y almacenamiento automatizados de drones. Están diseñados para operar drones de forma remota, facilitando misiones de vuelo prolongadas sin necesidad de intervención humana. Estas estaciones proporcionan un entorno seguro y eficiente para el mantenimiento y lanzamiento de drones, integrándose con herramientas de software para la gestión y control de las operaciones.

lo que puede ser un desafío significativo, de hecho, una de las motivaciones principales que da vida al presente proyecto es explorar enfoques más allá del SDK de DJI. Utilizando el ecosistema proporcionado por el SDK, se busca desarrollar soluciones alternativas que permitan la creación de aplicaciones personalizadas para drones privativos como los de DJI. Esta iniciativa pretende identificar métodos innovadores que, apoyándose en las herramientas del SDK, logren superar sus limitaciones y ofrezcan una mayor flexibilidad y creatividad en el desarrollo de aplicaciones específicas, ampliando así las posibilidades de uso y adaptación de estos drones a diversas necesidades y contextos.

Hasta el momento, en esta introducción se han abordado dos de los cuatro pilares fundamentales sobre los que se sustenta este proyecto, siendo estos dos primeros participantes los drones en general y los drones de DJI en particular. Sin embargo, aún quedan por presentar los otros dos pilares esenciales para el desarrollo del proyecto, los cuales serán descritos a continuación, ofreciendo una visión completa de los elementos clave que fundamentan este TFM.

La diferencia fundamental entre el aeromodelismo y un dron radica en que un dron está diseñado con un propósito específico, generalmente para ejecutar una aplicación concreta, más allá del simple vuelo recreativo. Como se ha mencionado anteriormente, los drones se utilizan en una amplia gama de aplicaciones, incluyendo la agricultura de precisión, la inspección de infraestructuras, la cinematografía y la logística, entre otras. Una característica constructiva común en la mayoría de los drones actuales es la presencia de un sistema de visión. Prácticamente todos los drones modernos están equipados con cámaras, lo que refleja la importancia crucial de estos sistemas de visión. Estas cámaras no solo permiten la captura de imágenes y vídeos, sino que también son esenciales para la navegación, el mapeo y la ejecución de tareas automatizadas. Este aspecto pone de manifiesto el papel fundamental que juegan los sistemas de visión en la funcionalidad y versatilidad de los drones, destacando su relevancia en la mayoría de las aplicaciones actuales.

Los sistemas de visión desempeñan un papel crucial en los drones modernos. No obstante, las imágenes obtenidas por estos sistemas suelen utilizarse únicamente para la navegación en tiempo real, proporcionando al usuario solo una visualización del entorno del dron. Generalmente, estas imágenes no se emplean para procesamientos posteriores como el seguimiento de objetos, el reconocimiento de formas o incluso el manejo autónomo del dron. Para realizar estos procesamientos adicionales, el usuario debe desarrollar una aplicación utilizando el DJI SDK, siendo el Mobile SDK la opción más común. Esta aplicación se instala en un dispositivo móvil que se conecta al mando del dron mediante un cable USB. Las imágenes captadas por la cámara del dron se transmiten vía radiofrecuencia al mando, y posteriormente se transfieren a la aplicación desarrollada e instalada en el móvil conectado por USB. El principal problema que surge de este enfoque es que la calidad final de las imágenes que llegan a la aplicación del usuario a menudo es deficiente, con retrasos y latencias que impiden un tratamiento y procesamiento eficaz de estas en tiempo real.

Esta situación representa una singularidad en el ámbito de los drones y la visión por computador (VxC). A pesar de que DJI, mediante su SDK, proporciona herramientas para la explotación y el desarrollo de aplicaciones avanzadas, las limitaciones inherentes a la transferencia de las imágenes impiden que estas alternativas sean viables en la práctica. La calidad deficiente, el retardo y el *lag* en la transmisión de imágenes dificultan el procesamiento en tiempo real, lo que obliga a los desarrolladores a buscar enfoques alternativos. Para abordar estos desafíos, se requiere la exploración de nuevas estrategias y tecnologías que permitan superar estas barreras y mejorar la eficiencia del procesamiento de imágenes captadas por drones.

Antes de continuar, cabe destacar que los modelos más avanzados de DJI, como el Matrice 300 RTK[5] y otros de la serie Enterprise, ya vienen equipados con software nativo que incorpora funciones de inteligencia artificial (IA). Estas capacidades permiten el reconocimiento de objetos y formas, facilitando aplicaciones más avanzadas sin necesidad de postprocesamiento externo. Además, algunos drones de DJI de gamas más altas admiten el desarrollo de aplicaciones mediante el Windows SDK. Este SDK proporciona herramientas para la comunicación directa con el dron a través de WiFi en lugar de radiofrecuencia. La ventaja principal de esta conexión es que permite la transmisión de imágenes vía *streaming* con una calidad suficientemente alta para permitir un procesamiento posterior efectivo. No obstante, el Windows SDK ha sido algo descuidado y su uso está limitado a drones de gama muy alta, los cuales no son accesibles para la mayoría de la comunidad de desarrolladores y usuarios. El *Onboard* SDK surge precisamente como respuesta a las limitaciones y dejadez del Windows SDK. Como se mencionó anteriormente, este SDK permite el desarrollo de aplicaciones para la computación a bordo de los drones y sensores externos embarcados. Si se pretende solucionar el problema de la calidad en la transferencia de imágenes a través de radiofrecuencia, este SDK sería la opción más efectiva y lógica a seguir, ya que proporciona todas las herramientas necesarias para abordar los problemas anteriormente expuestos. De hecho, si se considerara la adopción de esta vía, este TFM tendría un enfoque totalmente distinto, ya que muchos de los problemas se resolverían automáticamente y el ecosistema de trabajo quedaría prácticamente cerrado y definido. Sin embargo, el principal inconveniente del *Onboard* SDK es que solo está disponible para los drones de DJI de gamas más altas, más modernos y, consecuentemente, más caros. Esto representa una gran limitación, ya que para el presente TFM se utilizarán drones comerciales de gamas medias y bajas, lo que hace inviable la utilización del *Onboard* SDK. Por esta razón, se ha descartado esta opción para el desarrollo del proyecto, enfocándose en soluciones alternativas que sean accesibles y viables con los recursos disponibles.

Por lo tanto, y en base a las razones expuestas anteriormente, el presente TFM se centra en el uso del Mobile SDK y el desarrollo de aplicaciones móviles. Sin embargo,

[5]RTK (*Real-Time Kinematic*) es una técnica de posicionamiento satelital que proporciona correcciones en tiempo real, aumentando la precisión de los datos GPS a nivel centimétrico. Esta tecnología es especialmente útil en aplicaciones que requieren alta precisión, como la topografía, la agricultura de precisión y la navegación de drones.

este enfoque presenta el problema ya mencionado de la transmisión de imágenes por radiofrecuencia, lo que impide un procesamiento en tiempo real debido a la calidad y latencia de las imágenes recibidas. Aquí es donde la ingeniosidad y creatividad de los desarrolladores entran en juego, explorando soluciones fuera del ecosistema tradicional del SDK. Los desarrolladores buscan alternativas innovadoras para la transferencia de imágenes que no dependan de la radiofrecuencia. Estas soluciones pueden incluir el uso de redes WiFi dedicadas o sistemas de almacenamiento y transmisión en la nube, que permiten mantener la calidad de las imágenes y reducir la latencia, facilitando así un procesamiento más eficiente y efectivo.

En la actualidad, gran parte de los esfuerzos orientados a resolver el problema de la transferencia de imágenes desde los sistemas de visión de los drones vía radiofrecuencia, con el fin de que estas sean utilizadas en procesamientos posteriores en aplicaciones desarrolladas por terceros mediante el DJI SDK, se centran en la solución conocida como *Computer Vision On-Board*. Esta estrategia implica incorporar un pequeño ordenador al dron, equipado con su propia cámara. Este ordenador a bordo captura las imágenes a través de su cámara integrada y realiza el procesamiento directamente, sin necesidad de transferir las imágenes al mando vía radiofrecuencia. De este modo, se solventan los problemas de calidad, retardo y *lag* previamente mencionados.

El procesamiento de imágenes realizado en el ordenador a bordo incorporado permite obtener una serie de datos precisos, como errores de posición, que pueden ser transmitidos a una aplicación desarrollada por terceros mediante el DJI SDK. Esta aplicación (instalada en un teléfono móvil conectado vía USB al mando del dron), a alto nivel, se encarga de enviar comandos al dron, permitiendo su manejo en tiempo real. Este enfoque no solo mejora la eficiencia del procesamiento de imágenes, sino que también garantiza una respuesta más rápida y precisa en las operaciones del dron, optimizando su rendimiento en aplicaciones avanzadas.

Una opción muy atractiva para los ordenadores a bordo en drones son los dispositivos Jetson de NVIDIA. Los módulos NVIDIA Jetson, como Jetson Nano, Jetson Xavier NX y Jetson AGX Xavier, están diseñados para ofrecer capacidades avanzadas de computación y aceleración de IA en aplicaciones embebidas y en aplicaciones de tipo *Edge Computing* que son aquellas en las que los datos son procesados cerca del lugar donde se generan, en lugar de enviarlos a un centro de datos centralizado para su procesamiento. Estos dispositivos proporcionan varias ventajas significativas para el procesamiento de imágenes a bordo. Primero, son extremadamente compactos y eficientes en términos de energía, lo que los hace ideales para integrarse en drones. Por ejemplo, el Jetson Nano es un ordenador pequeño pero potente, capaz de manejar cargas de trabajo de IA modernas con un consumo de energía mínimo. Segundo, los módulos Jetson están equipados con una serie de núcleos CUDA y Tensor, lo que permite ejecutar múltiples redes neuronales en paralelo y soportar sensores de alta resolución. Esto es esencial para tareas como la detección de obstáculos, el mapeo y la planificación de rutas, que son críticas en aplicaciones avanzadas de drones. Además, los dispositivos Jetson utilizan el software NVIDIA JetPack, que incluye bibliotecas

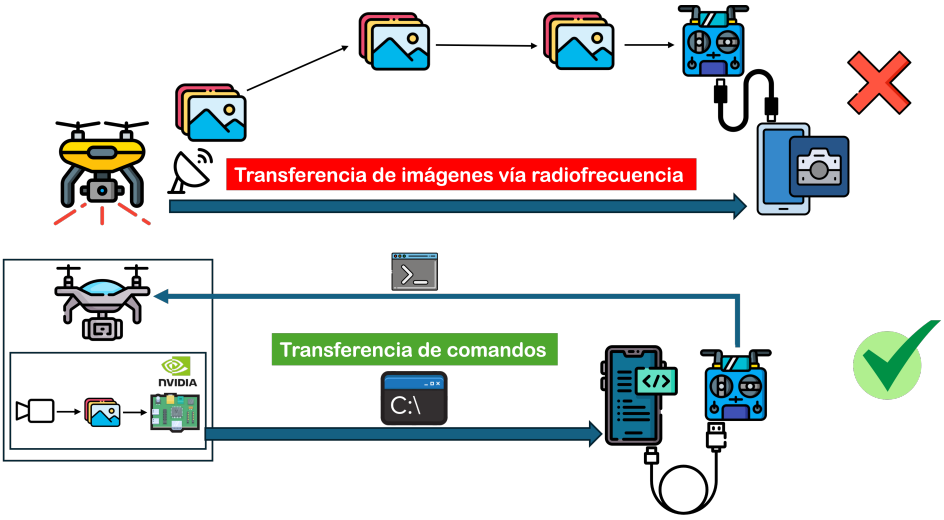


Figura 1.5: Problema, y solución al mismo, de la transferencia y procesamiento de imágenes en drones vía radiofrecuencia [Elaboración Propia].

aceleradas para aprendizaje profundo, visión por computadora, gráficos y multimedia. Esta pila de software facilita a los desarrolladores el despliegue rápido de aplicaciones de IA en dispositivos embebidos, reduciendo significativamente el tiempo de desarrollo y mejorando la eficiencia del sistema. En resumen, los dispositivos Jetson de NVIDIA ofrecen una solución robusta y eficiente para el procesamiento de imágenes en drones, permitiendo realizar análisis en tiempo real y mejorando la capacidad operativa del dron sin los problemas asociados a la transferencia de imágenes vía radiofrecuencia.

En el epígrafe anterior, se han introducido y cohesionado los elementos fundamentales sobre los que se sustenta el proyecto. Al tratarse de una introducción, no se ha profundizado en exceso en los conceptos presentados, ya que este nivel de detalle será abordado en secciones posteriores de la presente memoria. El objetivo principal ha sido ofrecer al lector una comprensión general de los componentes del proyecto y cómo estos se interrelacionan entre sí. Como conclusión de esta introducción, en la Figura 1.6 se muestra un gráfico que ilustra estos pilares y su integración en el contexto general del proyecto. Este gráfico servirá como referencia visual para facilitar la comprensión de la estructura del trabajo que se desarrollará en los siguientes capítulos.

Figura 1.6: Pilares sobre los que se sustenta el proyecto [Elaboración Propia].

1.1 Motivación del proyecto y ecosistema de trabajo

En la sección precedente, se ofreció una visión global de los temas fundamentales que abarca el presente proyecto. Además, se presentó uno de los principales problemas en el estado del arte actual relacionado con los drones: la transferencia y el procesamiento de las imágenes captadas por los mismos. El objetivo de dicho epígrafe era simplemente introducir los conceptos, nombres e ideas esenciales que dan vida al proyecto.

En este nuevo epígrafe, se pretende avanzar de lo general a lo particular, transformando esos conceptos e ideas generales en una aplicación tangible y real. Para entender las motivaciones que sustentan este proyecto y su justificación dentro del marco de trabajo general, es crucial presentar el ecosistema de trabajo previo al mismo. Esto implica materializar las ideas generalistas anteriormente mencionadas en una aplicación concreta. Solo de esta manera, mediante el análisis de una aplicación específica, es posible comprender el problema que surge y cómo este proyecto está diseñado para abordar, cubrir y resolver dicho problema.

En el ecosistema que se presentará a continuación, varios elementos ya están desarrollados y definidos con precisión, lo que implica que el punto de partida del presente proyecto se basa en trabajo previamente realizado. Sin embargo, para comprender plenamente el problema que se pretende abordar y la solución que este TFM propone, es esencial introducir y analizar adecuadamente todo este trabajo previo. Esta comprensión integral permitirá apreciar la base sobre la que se edifica el proyecto y cómo se estructura la solución propuesta para resolver el problema identificado.

Tal y como se mencionó anteriormente, los drones de DJI son dispositivos privativos, comúnmente designados como *Ready-to-Fly* (RTF). Esto significa que estos drones vienen equipados con todos los elementos necesarios para su operación, permitiendo al usuario utilizarlos tras una mínima configuración inicial.

En primer lugar, es fundamental comprender la comunicación entre el dron y el usuario a través del mando. En el vuelo de un dron, intervienen dos componentes principales:

el propio dron y el mando que el usuario emplea para controlarlo. Cuando el usuario mueve los *joysticks* del mando, estos movimientos se convierten en comandos específicos que son enviados al dron mediante señales de radiofrecuencia. De manera inversa, el dron puede enviar constantemente al mando datos de telemetría y de posición. Esta retroalimentación permite al usuario monitorear variables críticas de control, asegurando un vuelo más seguro y eficiente. Este flujo bidireccional de información es esencial para mantener al usuario informado sobre el estado y la posición del dron en todo momento. El esquema de este tipo de comunicación se muestra en la Figura 1.7.

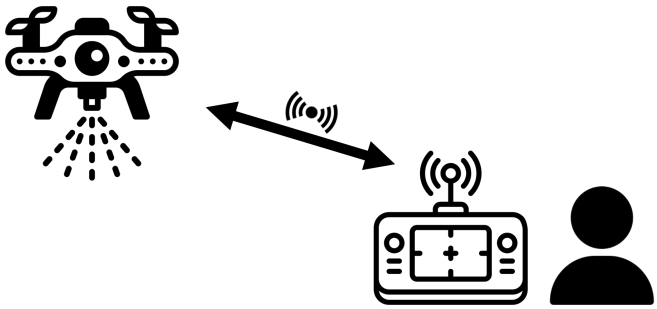


Figura 1.7: Esquema de comunicación de un dron con un mando [Elaboración Propia].

Una vez comprendida la comunicación entre el dron y el mando, es pertinente preguntarse cuáles son los pasos a seguir si un usuario desea desarrollar una aplicación propia para comandar el dron. Para lograr este objetivo, el usuario debe recurrir al DJI SDK. En el presente proyecto, se explorará únicamente el desarrollo de aplicaciones móviles, las cuales pueden ser creadas utilizando el *Mobile* SDK o el UX SDK. Otros SDKs y tipos de aplicaciones no serán abordados en este trabajo. Por lo tanto, cada vez que se mencione el desarrollo de aplicaciones, se hará referencia a aplicaciones móviles desarrolladas con cualquiera de estos dos SDKs.

Un usuario con los conocimientos adecuados y utilizando el *Mobile* SDK puede crear una aplicación móvil personalizada que permita pilotar y monitorear el dron en tiempo real. Para utilizar esta aplicación, solo se necesita disponer de un teléfono móvil, generalmente con sistema operativo Android, en el cual se haya instalado la aplicación. El móvil se conectaría vía cable USB al mando del dron, que actúa como intermediario. La comunicación debe entenderse de la siguiente manera: el usuario utiliza su aplicación para enviar comandos al dron o para monitorear sus variables en tiempo real. Estos comandos se transmiten vía cable USB al mando, que luego los envía al dron a través de radiofrecuencia. De manera inversa, el dron envía datos de telemetría mediante radiofrecuencia al mando, el cual los recibe y los transmite finalmente a la aplicación del usuario vía cable USB. En la Figura 1.8 se muestra un esquema de este tipo de comunicación.

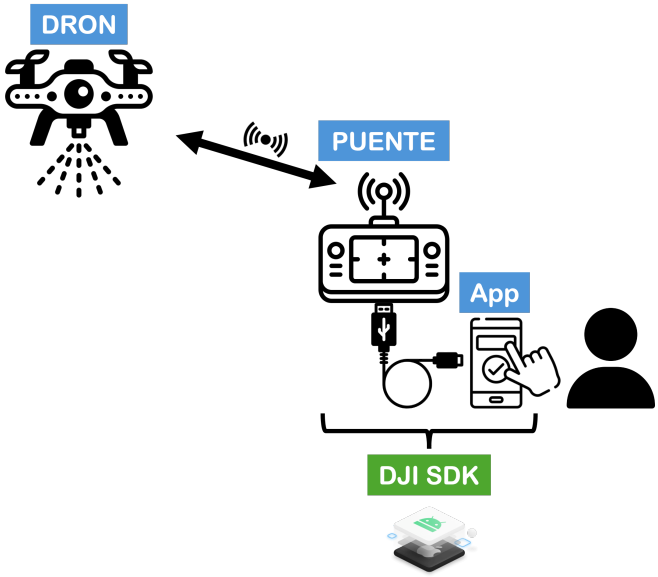


Figura 1.8: Esquema de la comunicación de un dron con una App desarrollada por un usuario mediante el DJI SDK [Elaboración Propia].

El esquema de comunicaciones planteado anteriormente, Figura 1.8, es la solución más adoptada por la extensa comunidad de desarrolladores, ya que permite la creación de aplicaciones con un alto grado de personalización de manera sencilla. Actualmente, existen múltiples aplicaciones desarrolladas bajo este enfoque, incluso estudiantes de esta misma Universidad han explorado estos métodos y creado sus propias aplicaciones. Por lo tanto, se podría concluir que esta vía de desarrollo ya está bien establecida y ampliamente explorada. Sin embargo, este enfoque no resuelve los problemas de transmisión de imágenes mencionados en la introducción de esta memoria. Si un usuario desea utilizar las imágenes captadas por el dron para algún tipo de procesamiento, el recorrido de estas imágenes hasta llegar a la aplicación del usuario sería el siguiente: las imágenes son captadas por el sistema de visión del dron y enviadas al mando a través de radiofrecuencia, como cualquier otro comando proveniente del dron. Una vez en el mando, se transmiten vía cable USB a la aplicación móvil. No obstante, la calidad con la que las imágenes llegan a la aplicación es muy deficiente, lo que hace imposible realizar un procesamiento efectivo sobre las mismas.

Antes de iniciar este proyecto, se exploraron diversas alternativas para mejorar la calidad de las imágenes transmitidas por radiofrecuencia. Sin embargo, ninguna de estas opciones logró ofrecer los resultados esperados, dejando sin resolver el problema de la transferencia de imágenes. **La motivación principal de este proyecto, y su razón de ser, es explorar nuevas vías de desarrollo que vayan más allá del DJI SDK,**

con el objetivo de encontrar soluciones que permitan capturar imágenes con una calidad suficiente para realizar algún tipo de procesamiento posterior sobre las mismas, lo cual es esencial para tareas como el control del dron, el reconocimiento de objetos y el seguimiento de objetivos. A través de este enfoque, se busca superar las limitaciones actuales y abrir nuevas posibilidades para el uso avanzado de drones en diversas aplicaciones.

Como se mencionó en la introducción, una de las soluciones más eficaces y actualmente más explotadas es embarcar un ordenador a bordo del dron que tenga su propia cámara. Esta cámara se encarga de la captura de imágenes, las cuales son transmitidas directamente al ordenador a bordo al que está conectada. El ordenador embarcado realiza el procesamiento de estas imágenes, obteniéndose como resultado una serie de comandos de control que deben ser enviados al dron. La principal ventaja de esta configuración es que las imágenes no necesitan ser transmitidas vía radiofrecuencia para su procesamiento. En su lugar, las imágenes capturadas por la cámara del ordenador embarcado son procesadas directamente por este, eliminando la necesidad de transferirlas. Esto no solo mejora la calidad del procesamiento, sino que también reduce la latencia y el riesgo de pérdida de datos, proporcionando una solución más robusta y eficiente para el control y operación del dron. En la Figura 1.5 se esquematiza el problema presentado y la solución al mismo adoptada.

Llegados a este punto, podría pensarse que siguiendo este enfoque todos los problemas estarían resueltos. Sin embargo, existe un aspecto crucial de esta estrategia que representa un gran desafío y que debe ser abordado previamente para alcanzar el objetivo final. Este aspecto, aunque a priori secundario y aparentemente irrelevante, implica salirse del ecosistema del DJI SDK y versa sobre la resolución de la comunicación entre el nuevo ordenador embarcado y el propio dron. Como se mencionó anteriormente, el procesamiento de las imágenes captadas por la cámara conectada al ordenador embarcado genera una serie de errores de posición que deben ser corregidos mediante el envío de comandos al dron. En la configuración inicial, el envío de comandos al dron era relativamente sencillo mediante la creación de una aplicación móvil usando el DJI SDK, y utilizando el mando como intermediario entre la aplicación del usuario y el dron, ver Figura 1.8. No obstante, es importante destacar que el ordenador embarcado en el dron es, efectivamente, un ordenador. Esto significa que no es posible desarrollar una aplicación móvil para este ordenador que, utilizando el mando como puente, permita el envío de comandos al dron. Según el planteamiento del ecosistema de DJI, los comandos al dron solo pueden ser enviados a través de aplicaciones móviles, no mediante ordenadores. Este desafío requiere una solución innovadora que permita la integración del ordenador embarcado con el sistema de control del dron, superando las limitaciones impuestas por el ecosistema de DJI. Resolver este problema es esencial para aprovechar las ventajas del procesamiento a bordo y lograr un control eficiente y preciso del dron basado en el procesamiento de imágenes en tiempo real.

Para resolver el problema de comunicaciones planteado, que es esencial para establecer un enlace entre el ordenador embarcado en el dron y la aplicación móvil que permite

comandar el mismo, se ha tomado como fuente de inspiración el Trabajo de Fin de Máster de Leya San Jose Moralejo [10], exalumna de esta universidad. En [10] surgió un problema similar al planteado en el presente proyecto. La manera en que [10] resolvió este problema, cuyo enfoque se ha adoptado en el presente TFM, se detalla a continuación.

En primer lugar, es importante señalar que se mantiene la estructura de comunicaciones inicial, ver Figura 1.8. Es decir, el ecosistema de trabajo seguirá teniendo un dron que, mediante radiofrecuencia, envía y recibe información hacia y desde el mando, que continúa actuando como puente. El mando seguirá conectado vía cable USB a un teléfono móvil, en el cual se ha instalado una aplicación desarrollada por terceros utilizando el DJI SDK. Esta aplicación continuará teniendo la función de enviar comandos al dron y recibir telemetría del mismo a través del puente.

Hasta este punto, la estructura de las comunicaciones permanece inalterada. Sin embargo, se debe encontrar una manera de establecer la comunicación entre el ordenador embarcado y la aplicación móvil. Si se logra establecer esta comunicación, el problema estaría resuelto, ya que el ordenador podría enviar comandos a la aplicación móvil, que a su vez, utilizando el puente, los transmitiría al dron. De manera similar, en sentido contrario, el dron enviaría telemetría a la aplicación móvil a través del puente, y esta retransmitiría los datos al ordenador embarcado, que es en sí el que lleva a cabo el procesamiento de las imágenes, permitiendo así el control del dron, el seguimiento de objetos, o la implementación de cualquier tipo de aplicación que se desee diseñar.

La forma de resolver la comunicación entre el ordenador embarcado y la aplicación móvil es mediante la modificación del código fuente de la aplicación móvil. En términos generales, se convierte la aplicación móvil en un servidor, lo que permite resolver el problema de comunicaciones a través de un sencillo esquema servidor-cliente. En este esquema, la aplicación móvil actúa como un servidor, mientras que el ordenador embarcado en el dron opera como un cliente.

La comunicación entre el cliente (ordenador embarcado) y el servidor (aplicación móvil) se efectúa vía WiFi, es decir, a través de Internet. Para el correcto funcionamiento del sistema, tanto el cliente como el servidor deben estar conectados a la misma red local. Esto se puede lograr conectando la aplicación móvil y el ordenador embarcado al mismo *router*, o bien compartiendo un punto de acceso a Internet desde un dispositivo externo y conectando tanto el cliente como el servidor a este punto de acceso.

A continuación, en la Figura 1.9, se esquematiza el ecosistema global de trabajo, incluyendo los diferentes tipos de comunicaciones y los elementos presentes en el mismo.

Dado el planteamiento del ecosistema global de trabajo, resulta evidente la necesidad de desarrollar una aplicación utilizando el DJI SDK. Para el desarrollo de la aplicación móvil, se ha tomado como punto de partida el trabajo realizado en [10], TFM con-

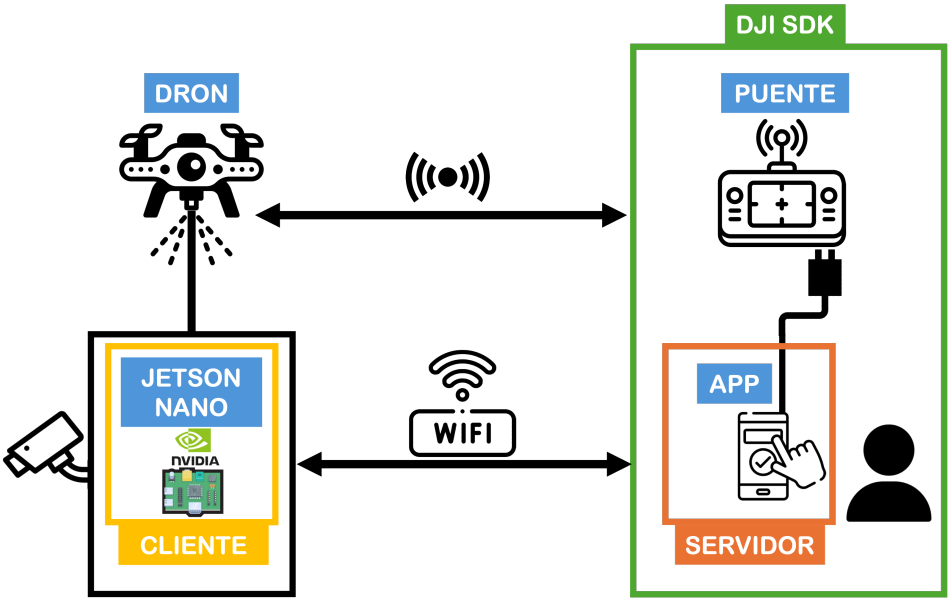


Figura 1.9: Ecosistema de trabajo, comunicaciones y elementos que intervienen [Elaboración Propia].

sistente en el desarrollo de una aplicación móvil que permite comandar un dron DJI Mavic Mini. En [10] se llevó a cabo el desarrollo de esta aplicación haciendo uso del UX SDK de DJI. En el presente TFM, se ha tomado como base la aplicación desarrollada en [10]. A pesar de basarse en su trabajo inicial, se ha realizado una migración del código fuente de la aplicación desde el UX SDK al *Mobile* SDK. Esta migración permite que la aplicación móvil utilizada en este proyecto se adapte a las directrices y capacidades del *Mobile* SDK, ofreciendo así mayores posibilidades de personalización y funcionalidad. Además de esta migración, se han implementado una serie de mejoras y cambios que no estaban presentes en la aplicación original, y que serán detallados en capítulos posteriores de esta memoria. Todos estos cambios, migraciones y mejoras sobre la aplicación inicial desarrollada por [10] se han llevado a cabo en colaboración con el tutor del presente TFM.

Como inciso, es importante destacar que la resolución del problema de comunicaciones entre el ordenador embarcado y la aplicación móvil ha permitido desvincularse del SDK de DJI. Como se mencionó anteriormente, el SDK de DJI solo permite el control de sus drones a través de aplicaciones móviles. En el ecosistema de trabajo actual, el control del dron se realiza a partir de un ordenador, no de una aplicación móvil. Aunque la aplicación móvil sigue presente, su función ha cambiado; ya no es el punto de origen o destino de los comandos enviados o recibidos desde el dron, sino que ahora actúa como intermediaria entre el dron y el ordenador embarcado. Este avance abre un

abanico de posibilidades, ya que se ha encontrado una forma de controlar drones de DJI mediante ordenadores y no exclusivamente a través de aplicaciones móviles, desafiando así las restricciones impuestas por DJI para el control de sus drones mediante su SDK. Aunque en el presente proyecto el ordenador esté embarcado en el dron, el enfoque y la solución planteada permiten el control de drones desde otros ordenadores en tierra, que no necesariamente deben estar embarcados en el dron.

Por último, es fundamental concretar el tipo de procesamiento que se realizará con las imágenes captadas por la cámara del ordenador embarcado. El procesamiento a efectuar se centra en el reconocimiento de códigos ArUco, los cuales serán descritos y explicados en detalle en secciones posteriores de este documento. Básicamente, la aplicación que se pretende desarrollar operará de la siguiente manera: la cámara del ordenador embarcado tomará imágenes continuamente, procesándolas y buscando la presencia de estos códigos. En caso de detectar alguno de estos códigos, y dependiendo del tipo de código identificado, el ordenador embarcado (cliente) enviará comandos a la aplicación móvil (servidor). El servidor, en función del tipo de comando recibido, transmitirá las órdenes adecuadas al dron a través del puente. El desarrollo de la aplicación encargada del procesamiento de imágenes, la detección de códigos ArUco y el envío de los comandos pertinentes según la situación específica del vuelo, será explicada de manera exhaustiva y detallada en las secciones venideras del presente documento. En la Figura 1.10 se esquematiza el fundamento de la aplicación que se pretende desarrollar, junto con su integración con el resto de las partes del sistema de trabajo.

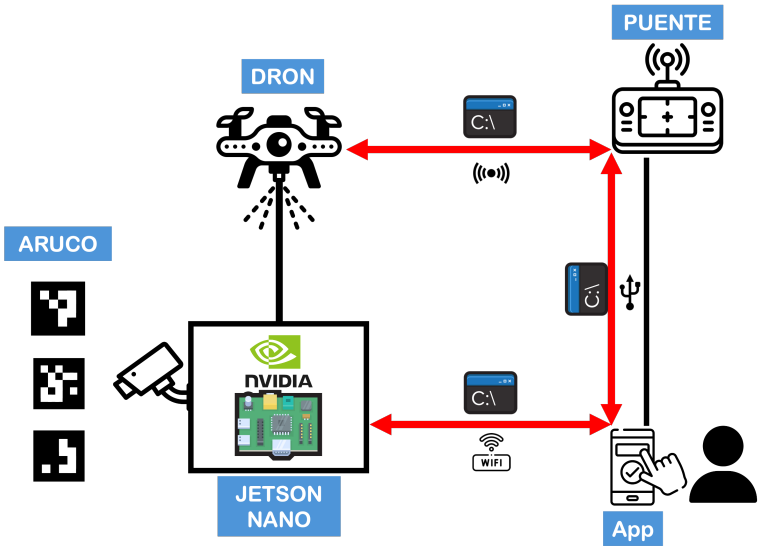


Figura 1.10: Esquema de la aplicación del procesamiento de imágenes [Elaboración Propia].

Por último, es importante mencionar que el presente TFM se enmarca dentro de un conjunto de proyectos que coexisten y que son liderados por el tutor del presente trabajo, todos ellos orientados al campo de la visión por computador y el manejo automático de drones. A continuación, se describen brevemente estos proyectos para proporcionar al lector una visión más amplia del estado del arte y de las diferentes aproximaciones a algunos de los problemas que este TFM pretende abordar:

- Proyecto [11]: Representa un ejemplo de soluciones alternativas de procesamiento de imágenes en drones comerciales. Este proyecto emplea un mini dron para procesar imágenes, ofreciendo una perspectiva diferente al uso de drones de mayor envergadura. A pesar de la utilización de Python, se depende del *streaming* proporcionado por la librería correspondiente, lo cual no siempre ofrece una fluidez óptima.
- Proyecto [12]: Ejemplo de drones *Open Source* con Raspberry Pi. En este proyecto, se emplea una Raspberry Pi con una cámara embarcada en el dron. La infraestructura para el procesamiento de imágenes se estableció, pero no se llegó a realizar un procesamiento avanzado de reconocimiento.
- Proyecto [13]: Otro ejemplo de soluciones alternativas de procesamiento de imágenes en drones comerciales. Similar al primer proyecto, se exploran diferentes métodos de procesamiento de imágenes en mini drones, destacando las limitaciones y desafíos del *streaming* de video.
- Proyecto [14]: Ejemplo de drones *Open Source* con cámaras analógicas. En este proyecto, la cámara analógica transfiere imágenes a un ordenador en tierra, donde son digitalizadas y procesadas. Aunque se creó la infraestructura necesaria, el procesamiento avanzado de imágenes no se llevó a cabo.
- Proyecto [15]: Otro ejemplo de drones *Open Source* con cámaras analógicas. Este proyecto sigue una metodología similar al anterior, utilizando cámaras analógicas y procesamiento en tierra. La infraestructura se preparó, pero faltó continuar con el reconocimiento avanzado de imágenes.

Estos proyectos destacan diferentes enfoques y soluciones en el campo del manejo automático de drones y la visión por computador, ofreciendo al lector una perspectiva más amplia sobre las diversas técnicas y desafíos en este ámbito.

1.2 Objetivos del Trabajo Fin de Máster

En las secciones anteriores de este capítulo, se ha realizado, en primer lugar, una introducción al entorno general de trabajo y, en segundo lugar, se ha descrito el ecosistema en el que se enmarca el proyecto, ver Sección 1.1. Al describir este ecosistema, se han identificado componentes previamente desarrollados y claramente definidos que, aunque son cruciales para el funcionamiento adecuado del proyecto, no forman parte de los objetivos del presente TFM.

Para definir con precisión el alcance de este TFM, en el siguiente epígrafe se enumerarán los objetivos específicos que se pretenden alcanzar. La consecución de estos objetivos permitirá abordar y resolver los problemas identificados en la arquitectura actual del proyecto. Este TFM busca satisfacer los siguientes objetivos:

1. **Modificación y puesta a punto de la aplicación móvil Android desarrollada con el SDK de DJI:** Como se introdujo anteriormente, este Trabajo de Fin de Máster (TFM) se basa en la aplicación móvil desarrollada por Leya San Jose Moralejo [10]. El fundamento de este objetivo es adaptar y acondicionar la aplicación móvil para que cumpla con las necesidades específicas del presente TFM. Este proceso es fundamental para garantizar que la aplicación móvil pueda soportar de manera robusta y eficiente las operaciones del dron, incluyendo la recepción y procesamiento de datos en tiempo real, así como el envío de comandos precisos al dron durante el vuelo. A su vez, este objetivo puede descomponerse en objetivos más pequeños, los cuales se citan y explican brevemente a continuación:

    a) **Migración del código fuente de la aplicación del UX SDK al Mobile SDK:** Leya desarrolló la aplicación utilizando el UX SDK, que está más orientado al desarrollo de aplicaciones móviles con una interfaz gráfica avanzada. Sin embargo, en este TFM no se pretende explotar toda esta parte de las interfaces gráficas, por lo que se ha decidido migrar el código fuente de la aplicación al Mobile SDK. Esta migración optimiza el funcionamiento de la aplicación, permitiendo un desarrollo más eficiente y enfocado en las funcionalidades necesarias para este proyecto.

    b) **Depuración del código de la aplicación:** Se realizará una depuración exhaustiva del código de la aplicación para mejorar su rendimiento general. Esto incluye identificar y corregir errores, así como optimizar los procesos internos para asegurar que la aplicación funcione de manera fluida y sin interrupciones durante su uso en el dron.

    c) **Implementación de mejoras en la aplicación:** Se implementarán una serie de mejoras relacionadas con la forma en que la aplicación envía los comandos al dron. Además, se añadirá la recepción de datos de telemetría relativos a la distancia a los obstáculos. Leya no implementó esta funciona-

lidad en su aplicación original, ya que trabajó con un dron que no contaba con sensores de distancia a obstáculos. Sin embargo, en el presente TFM se trabajará con un dron que sí dispone de estos sensores, por lo que es necesario modificar la aplicación para incluir la recepción y procesamiento de estas nuevas variables de telemetría.

2. **Puesta a punto del ordenador embarcado y desarrollo de la aplicación de procesamiento de imágenes en Python:** En el presente TFM, para solucionar el problema de la transmisión y procesamiento de imágenes, se embarcará un ordenador a bordo del dron que llevará a cabo el procesamiento de imágenes de manera autónoma, sin necesidad de enviar estas imágenes vía streaming. El objetivo principal es tanto la puesta a punto del ordenador embarcado como el desarrollo de la aplicación de procesamiento de imágenes en Python. Dentro de este objetivo principal, se destacan los siguientes objetivos secundarios:

   a) **Instalación del sistema operativo en el ordenador a embarcar:** Se instalará un sistema operativo compatible que soporte las necesidades del proyecto, proporcionando un entorno estable para el desarrollo y ejecución de la aplicación.

   b) **Instalación y configuración de la cámara encargada del procesamiento de las imágenes en el ordenador embarcado:** Se instalará y configurará la cámara, asegurando que esta esté correctamente integrada con el sistema y lista para capturar y procesar imágenes.

   c) **Compilación de la librería `OpenCV` desde el código fuente:** Se compilará la librería `OpenCV` desde el código fuente de la misma para garantizar que todas las características necesarias estén disponibles y optimizadas para el hardware específico del ordenador embarcado.

   d) **Instalación y puesta a punto de VNC:** Se instalará y configurará VNC (*Virtual Network Computing*), un programa necesario para el control remoto del ordenador auxiliar cuando este se encuentre en vuelo. Esto permitirá gestionar y supervisar el reComputer J3011 desde un ordenador en tierra, facilitando el acceso y el manejo de la aplicación de procesamiento de imágenes en tiempo real.

   e) **Desarrollo en Python de la aplicación de procesamiento de imágenes y detección de códigos ArUco:** Se desarrollará una aplicación en Python que utilizará OpenCV para el procesamiento de imágenes y la detección de códigos ArUco, permitiendo al dron tomar decisiones basadas en la interpretación visual del entorno.

   f) **Testeo de los códigos desarrollados mediante pruebas en vuelo con un dron de menor tamaño (DJI Mavic Mini):** Esto permitirá identificar

y corregir posibles errores en un entorno controlado antes de implementar las soluciones en el dron principal del proyecto, el DJI Phantom 4 Advanced+.

3. **Embarcado del ordenador auxiliar reComputer J3011 en el dron DJI Phantom 4 Advanced+:** Este objetivo se centra en analizar la viabilidad técnica del embarque del ordenador auxiliar, asegurando que su integración no comprometa la estabilidad ni el rendimiento del dron. Además, se busca optimizar la distribución del peso y la colocación del reComputer J3011 para mantener el equilibrio del dron durante el vuelo.

Como nota final, se adelanta que el ordenador embarcado será un Jetson Nano de NVIDIA. Los drones utilizados para las pruebas, el testeo de las comunicaciones y el diseño de los códigos de procesamiento de imágenes serán un DJI Mavic Mini y un DJI Phantom 4 Advanced. La elección de estos elementos se discutirá más adelante en este documento, donde se explicarán las razones técnicas y operativas detrás de su selección.

Capítulo 2

Estado del arte

En el capítulo anterior de este documento se presentó una introducción al contexto general del Trabajo de Fin de Máster (TFM). Inicialmente, se expusieron los pilares fundamentales sobre los cuales se construye el proyecto, ver Figura 1.6. Posteriormente, estas ideas, que al comienzo eran meramente conceptos teóricos, se concretaron en una aplicación práctica y un ejemplo de uso real, ver Sección 1.1. Además, se describió en detalle la aplicación, explicando los componentes involucrados, los problemas identificados en el estado actual del proyecto y cómo estos problemas constituyeron las motivaciones principales para la realización de este TFM. Finalmente, se definió con mayor precisión el ámbito de trabajo y se enumeraron y justificaron brevemente los objetivos del TFM, Sección 1.2, los cuales serán abordados a lo largo de este documento. No obstante, en el capítulo anterior se mencionaron varios conceptos y temas sin profundizar en su definición y descripción. Por ejemplo, se hizo referencia a los drones y a los drones privativos, pero no se explicó qué son los drones, qué tipologías existen ni bajo qué normativas se regulan. Además, se introdujo el ambiente de trabajo y los componentes que lo conforman, pero sin detallar las especificaciones de cada componente. Se mencionó, por ejemplo, un ordenador embarcado, pero no se describieron los tipos de ordenadores embarcados disponibles en el estado del arte actual, ni se especificó cuál se eligió para este proyecto y la justificación de dicha elección. Además, se mencionaron los códigos ArUco y se explicó que la aplicación de procesamiento de imágenes que se pretende desarrollar tiene como núcleo principal estos códigos y su reconocimiento.

El presente capítulo tiene como objetivo responder a todas estas cuestiones y proporcionar un contexto más amplio y definido para estos conceptos. Se explicará el fundamento teórico subyacente a los pilares del proyecto y se discutirán las razones detrás de muchas de las decisiones tomadas, los caminos elegidos y los elementos seleccionados, especialmente en lo que respecta al hardware, el ordenador embarcado y el tipo de dron.

2.1 Drones

2.1.1 Clasificación y definiciones básicas

Antes de proceder con cualquier tipo de clasificación, es esencial presentar una serie de definiciones básicas. Aunque todas ellas se refieren a conceptos relacionados, es importante señalar que no todas tienen el mismo significado. Comprender estas definiciones con precisión permite un uso correcto y riguroso de los términos que hacen referencia al mundo de los drones.

Definiciones básicas

- **Aeronave no tripulada:** Cualquier aeronave diseñada para operar sin un piloto a bordo, ya sea de forma autónoma, automática, o pilotada a distancia.
- **Dron:** Término coloquial utilizado para referirse generalmente a todas las aeronaves no tripuladas.
- **Aeromodelo:** Aeronave de tamaño reducido, pilotada a distancia, utilizada principalmente para vuelos deportivos o experimentales. La principal diferencia entre un dron y un aeromodelo radica en la aplicación a la que están destinados cada uno de ellos. Un dron se desarrolla para cumplir una función específica y sin esta aplicación pierde su propósito, convirtiéndose en un aeromodelo. En resumen, todos los drones son aeromodelos, pero no todos los aeromodelos son drones.
- **UAV (*Unmanned Aerial Vehicle*)**: Se refiere específicamente a la aeronave que opera sin un piloto a bordo. Este término se utiliza para describir cualquier tipo de aeronave que puede volar de manera autónoma, automática o ser pilotada a distancia.
- **UAS (*Unmanned Aerial System*)**: El término UAS engloba no solo a la aeronave no tripulada, sino también al equipo necesario para su operación remota. Esto incluye el control remoto y cualquier otro componente necesario para la operación del sistema. Los UAS pueden ser pilotados a distancia, operar de forma automática o de forma autónoma. Notar que UAS es el término que engloba tanto a los UAV (la aeronave) como a los componentes necesarios para su operación, incluyendo los sistemas de control y comunicación. En la Figura 2.1 se muestra la arquitectura general de un UAS.
- **RPA (*Remotely Piloted Aircraft*)**: Se refiere a una categoría específica de aeronave no tripulada (UAV) que es controlada a distancia por un operador humano. A diferencia de los UAV completamente autónomos, los RPA requieren la intervención continua de un piloto que los controla desde una estación remota.

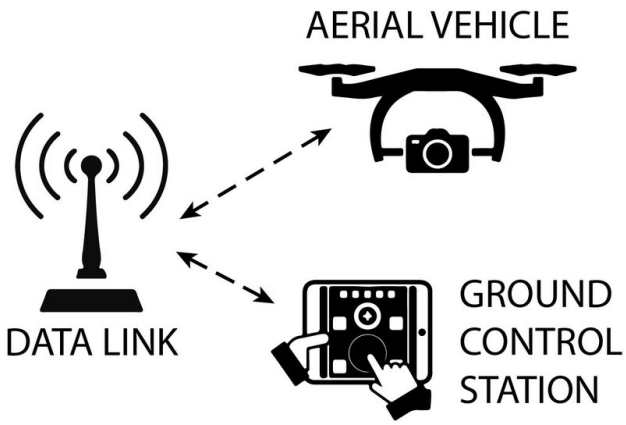


Figura 2.1: Arquitectura general de un UAS (*Unmanned Aerial System*) [16].

- **RPAS (*Remotely Piloted Aircraft System*)**: Se refiere a los sistemas de aeronaves pilotadas a distancia. En un RPAS, el piloto controla la aeronave desde una estación remota a través de enlaces de mando y control específicos. A diferencia de los UAS autónomos, el RPAS requiere intervención humana continua para su operación. Destacar que RPAS es una subcategoría específica de UAS que se centra en aeronaves pilotadas a distancia (RPA), destacando la necesidad de intervención humana continua durante su operación. Cabe señalar nuevamente que los términos UAS y RPAS se refieren al sistema completo, que incluye la aeronave, los enlaces de comunicaciones y la estación de control en tierra. En contraste, los términos UAV y RPA se refieren únicamente a la aeronave en sí. Para comprender adecuadamente todos estos conceptos, es útil tener en cuenta la siguiente afirmación: «Todos los RPAS son UAS, pero no todos los UAS son RPAS». Este concepto se ilustra en la Figura 2.2.

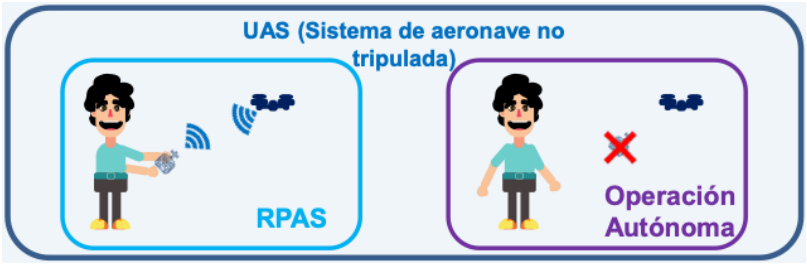


Figura 2.2: Relación entre UAS y RPAS [17].

En esta memoria, se emplea de manera general el término dron/es para referirse a todos los conceptos previamente definidos. Aunque este no es el término más preciso

ni el más adecuado desde una perspectiva técnica, es el más comúnmente utilizado por la sociedad, especialmente entre quienes no son expertos en el sector. Por esta razón, y con el objetivo de hacer la memoria accesible a un público más amplio, se ha decidido utilizar el término dron/es para abarcar todos los conceptos descritos anteriormente.

Clasificación de los drones

Existen diversas maneras de clasificar los drones, cada una basada en distintos criterios. Sin embargo, desde un enfoque didáctico y en el contexto de este TFM, las clasificaciones que revisten mayor importancia son las siguientes:

1. **Clasificación según el sistema de operación y forma de pilotaje**

    a) **UAS (*Unmanned Aerial System*)**

        1) **Autónomos:** Estas aeronaves operan sin intervención humana directa durante el vuelo. Utilizan algoritmos avanzados y sistemas de navegación para realizar misiones específicas de manera completamente autónoma. El piloto a distancia no puede intervenir durante el vuelo.
        2) **Automáticos:** Estas aeronaves también operan sin intervención humana directa durante el vuelo utilizando sistemas previamente configurados y algoritmos de navegación. Sin embargo, a diferencia de los autónomos, un humano puede intervenir en cualquier momento durante el vuelo para tomar el control manual si es necesario.
        3) **Pilotados a distancia (RPAS):** En estos sistemas, un piloto controla la aeronave mediante un control remoto. Este tipo de operación es común en aplicaciones comerciales y recreativas.

    b) **RPAS (*Remotely Piloted Aircraft System*)**

        1) **Controlado manualmente:** El piloto utiliza un control remoto para dirigir todas las funciones de vuelo de la aeronave.
        2) **Semi-Autónomo:** Aunque requieren intervención humana, estos sistemas pueden ejecutar ciertas maniobras o misiones de manera autónoma, basándose en configuraciones previas.

2. **Clasificación según el diseño estructural (Figura 2.6)**

    a) **De ala fija (Figura 2.3a):** Los drones de ala fija, diseñados para vuelos largos y eficientes, son ideales para misiones de vigilancia, mapeo y monitoreo de grandes áreas, ya que su estructura, similar a la de los aviones tradicionales, cuenta con alas que generan sustentación a medida que la aeronave se desplaza por el aire.

b) **Multirrotores (Figura 2.3b):** Los drones multirrotores, conocidos por su maniobrabilidad y estabilidad en vuelo, son ideales para aplicaciones que requieren precisión y control en espacios reducidos, ya que cuentan con varios rotores (generalmente cuatro, seis u ocho) que les permiten despegar y aterrizar verticalmente.

(a) Dron de ala fija [18].

(b) Dron multirrotor [19].

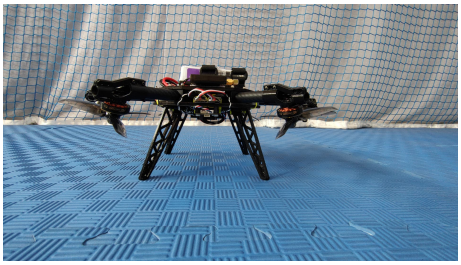
Figura 2.3: Clasificación de los drones según el diseño estructural.

3. **Clasificación según el tipo de *software* (Figura 2.4)**:

a) **Drones privativos (Figura 2.4a):** Estos drones utilizan *software* propietario, desarrollado y controlado por una empresa específica. Los usuarios tienen acceso limitado a las funcionalidades del *software* y no pueden modificarlo, lo que es común en aplicaciones comerciales donde la seguridad y la integridad del sistema son prioritarias. Al estar gestionados por una empresa, estos drones suelen contar con soporte técnico dedicado y recibir actualizaciones regulares, aunque la capacidad de personalización es limitada.

b) **Drones *Open Source* (Figura 2.4b):** Estos drones utilizan *software* de código abierto, que puede ser modificado y personalizado por los usuarios. Esta característica proporciona una gran flexibilidad para adaptar el dron a necesidades específicas y experimentar con nuevas funcionalidades. Los drones de código abierto se benefician de una comunidad activa que contribuye continuamente al desarrollo y mejora del *software*, ofreciendo recursos y soporte colaborativo.

(a) Dron con *software* privativo [20].



(b) Dron con *software Open Source* [21].

Figura 2.4: Clasificación de los drones según el tipo de *Software*.

2.1.2 Normativa Regulatoria

Los drones, a pesar de no contar con un piloto a bordo, son aeronaves y su uso inadecuado puede poner en riesgo otros elementos del entorno en el que operan, como la aviación tripulada, las personas y las construcciones. Aunque no es uno de los objetivos principales de este TFM, omitir una mención a la normativa y las instituciones encargadas de su regulación sería un error. Por ello, a continuación se ofrece una breve introducción a este importante tema.

Cabe mencionar que la normativa relacionada con los drones es un tema complejo debido a su constante evolución y cambio. El rápido crecimiento del mercado de drones, significativo en la última década, ha llevado a múltiples instituciones a esforzarse por adaptarse a los avances tecnológicos y la evolución del sector. Su objetivo es establecer una normativa actualizada, segura y eficiente. Sin embargo, aún queda mucho camino por recorrer para alcanzar el nivel de seguridad y estabilidad de los estándares que rigen la aviación tripulada. El auge del sector de los drones y la constante adaptación de la normativa conllevan cambios y modificaciones frecuentes, lo que significa que las leyes y regulaciones vigentes hoy en día podrían no estarlo en unos años y ser reemplazadas por otras diferentes. La breve introducción a la normativa que se presenta a continuación refleja el estado del arte actual sobre el que se basa el marco regulativo de los drones.

Concepto normativo - Agencias e Instituciones reguladoras

En el marco regulatorio de los drones, es fundamental comprender las funciones y responsabilidades de las instituciones que se encargan de la elaboración de las normativas y regulaciones. A continuación, se presentan las principales entidades involucradas:

- **EASA – Agencia de la Unión Europea para la Seguridad Aérea [22]**
    - La EASA es la entidad responsable de unificar los estándares de seguridad aérea entre los Estados miembros de la Unión Europea. Su objetivo

principal es garantizar la seguridad de las operaciones en la aviación civil mediante la creación de normas comunes aplicables en todos los estados miembros. Para la regulación de los drones, la EASA ha desarrollado los Reglamentos (UE) 2019/947 y (UE) 2019/945, que establecen las bases para la estandarización en el uso de UAS (*Unmanned Aerial Systems*).

- **AESA – Agencia Estatal de Seguridad Aérea [23]**

  - La AESA es el organismo estatal encargado de velar por el cumplimiento de las normas de aviación civil en España. Esta agencia promueve el desarrollo y la aplicación de la legislación aeronáutica a nivel nacional e internacional, asegurando la seguridad, calidad y sostenibilidad del sistema de aviación civil español. En caso de incumplimiento de las normas, la AESA es la autoridad sancionadora en el territorio nacional.

- **DGAC – Dirección General de Aviación Civil [24]**

  - La Dirección General de Aviación Civil (DGAC) tiene como misión planificar, regular y proveer servicios de aviación civil de manera eficiente y segura. La DGAC es responsable de la elaboración de propuestas normativas, la aprobación de circulares aeronáuticas, y la representación de España en organismos internacionales de aviación. Esta entidad coordina la política aeronáutica y gestiona la normativa relacionada con aeronaves no tripuladas, garantizando la seguridad operacional y el desarrollo sostenible del sector [24].

Reglamentos UE y Reales Decretos

Hasta 2014, el uso de RPAS en España no estaba regulado. La Ley 18/2014 introdujo los primeros requisitos, seguida del Real Decreto 1036/2017 que reguló el uso civil de RPAS en España. En 2019, entraron en vigor los Reglamentos (UE) 2019/947 y 2019/945, armonizando la normativa en toda la UE. El nuevo Real Decreto 517/2024 complementa esta regulación proporcionando un marco más actualizado y adaptado a las necesidades del sector.

- **Reglamento de Ejecución (UE) 2019/947 [25]**

  - Este reglamento establece las normas y procedimientos para la operación de UAS en la Unión Europea. Cubre aspectos como la certificación de operadores, requisitos de seguridad, y las categorías de operaciones (abierta, específica y certificada), ver Figura 2.5. Ha sido modificado por varios reglamentos para ajustar fechas de implementación y requisitos específicos debido a la pandemia de COVID-19 y otros factores.

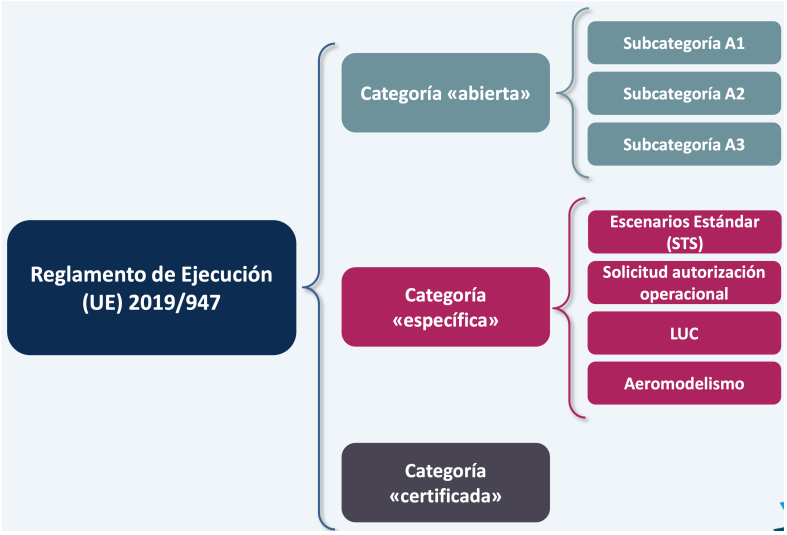


Figura 2.5: Categorías establecidas por el Reglamento de Ejecución (UE) 2019/947 [17].

- **Reglamento Delegado (UE) 2019/945 [26]**
  - Este reglamento define los requisitos para el diseño y fabricación de UAS, así como las normas para su comercialización y mantenimiento. Establece las clases de UAS (C0 a C4) y las normas aplicables a operadores de terceros países dentro del espacio aéreo del cielo único europeo (SES). Este reglamento ha sido modificado por el Reglamento Delegado (UE) 2020/1058 para introducir, entre otras actualizaciones, dos nuevas clases de sistemas de aeronaves no tripuladas (Clases C5 y C6), que se utilizarán en los escenarios estándar europeos.
- **Real Decreto 1036/2017 [27]**
  - El Real Decreto UAS establece el régimen jurídico para los aspectos no cubiertos por los Reglamentos de Ejecución y Delegado, permitiendo a los Estados miembros regular ciertos aspectos específicos de las operaciones con UAS.
- **Nuevo Real Decreto 517/2024 en España [28]**
  - El nuevo Real Decreto 517/2024, que entró en vigor el 25 de junio de 2024, desarrolla el régimen jurídico para la utilización civil de sistemas de aeronaves no tripuladas (UAS) en España. Este decreto complementa y actualiza la normativa nacional en consonancia con las regulaciones europeas.

Incluye disposiciones sobre el control a la importación de determinados productos, demostraciones aéreas civiles, lucha contra incendios, búsqueda y salvamento, y requisitos de aeronavegabilidad y licencias. Este nuevo marco normativo busca proporcionar estabilidad y seguridad jurídica al sector de los drones en España, adaptando las regulaciones nacionales a las necesidades actuales y futuras del sector, y facilitando la integración de nuevas tecnologías y aplicaciones de UAS en diversas áreas, tanto civiles como de interés público.

- **Material Guía (GM) y Medios Aceptables de Cumplimiento (AMC)**

  - Los Reglamentos de Ejecución (UE) 2019/947 y Delegado (UE) 2019/945 desarrollados por la EASA establecen las normativas para la utilización de UAS en la Unión Europea. Para facilitar la comprensión y correcta implementación de estos reglamentos, la EASA también ha publicado Material Guía (GM) y Medios Aceptables de Cumplimiento (AMC). El Material Guía proporciona orientación adicional y clarificaciones sobre la normativa, mientras que los AMC presentan métodos específicos que, si se siguen, aseguran el cumplimiento de los requisitos reglamentarios. Es importante destacar que, aunque los Reglamentos son vinculantes en su totalidad para todos los Estados miembros, el Material Guía y los AMC no son obligatorios. Esto significa que los operadores pueden optar por diferentes métodos de cumplimiento, siempre y cuando puedan demostrar que cumplen con los requisitos de seguridad establecidos en los Reglamentos. Esta flexibilidad permite a los operadores adaptar sus procedimientos a sus circunstancias específicas, fomentando la innovación y eficiencia dentro del marco regulatorio seguro. En resumen, los Reglamentos proporcionan el marco legal obligatorio, mientras que el GM y los AMC ofrecen herramientas y métodos recomendados para facilitar su aplicación y asegurar un alto nivel de seguridad operacional.

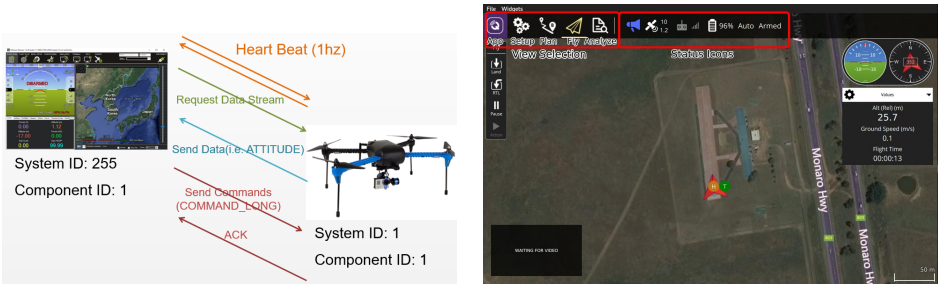
2.2 Proyecto Rosetta Drone [29]

Rosetta Drone es un proyecto de código abierto que proporciona un marco para desarrollar y probar software para drones DJI. Este proyecto se centra en crear una capa de compatibilidad MAVLink para el SDK de DJI, permitiendo que los drones DJI sean controlados mediante estaciones de control en tierra que utilizan el protocolo MAVLink, como QGroundControl.

Antes de continuar, es interesante explicar brevemente qué son MAVLink y QGroundControl.

- **MAVLink (Micro Air Vehicle Link)** es un protocolo de comunicación ligero, utilizado en sistemas de vehículos aéreos no tripulados (UAV) para intercambiar datos entre el vehículo y la estación de control en tierra. Este protocolo permite la transmisión de información crítica como telemetría, comandos de control, y datos de misión, asegurando una comunicación fiable y eficiente entre los componentes del sistema, ver Figura 2.6a.

- **QGroundControl**, por otro lado, es una estación de control en tierra (GCS) de código abierto que proporciona una interfaz gráfica para controlar UAVs. Ofrece funcionalidades avanzadas como la planificación de misiones, el monitoreo de telemetría en tiempo real, y el control de la cámara y otros sistemas a bordo del dron. QGroundControl es compatible con varios protocolos de comunicación, incluido MAVLink, y es ampliamente utilizado en la comunidad de drones por su flexibilidad y capacidad de personalización, ver Figura 2.6b.



(a) Esquema del flujo de mensajes mediante el protocolo de comunicación MAVLink [30].

(b) Aspecto visual de la interfaz gráfica de QGroundControl [31].

Figura 2.6: Proyecto Rosetta Drone. MAVLink y QGroundControl.

2.2.1 Objetivos y funcionalidades [32]

El principal objetivo del proyecto Rosetta Drone es integrar los drones DJI dentro del ecosistema de software de código abierto MAVLink. Esto proporciona a los usuarios de DJI una mayor flexibilidad y opciones, al poder utilizar aplicaciones y herramientas desarrolladas en la comunidad de MAVLink. Además, Rosetta Drone facilita la implementación y prueba de *scripts* de inteligencia artificial, permitiendo a los desarrolladores experimentar y mejorar las capacidades autónomas de los drones.

2.2.2 Características técnicas del proyecto [33, 34]

- **Funcionalidades:** Entre las principales funcionalidades se incluyen la capacidad de realizar misiones de waypoint, la transmisión de video en tiempo real, y la integración con herramientas de procesamiento de vídeo como GStreamer y OpenCV para desarrollar funciones de inteligencia artificial.
- **Compatibilidad con Software:** Funciona con aplicaciones como QGroundControl, Mission Planner, ArduCopter SITL, DroneKit-Python y MAVSDK.
- **Compatibilidad con Drones:** Rosetta Drone ha sido probado con varios modelos de DJI, incluyendo el DJI Mini, Mini SE, Mini 2, Air 2S, Mavic 2 y Matrice 210 V2.

2.2.3 Implementación y uso

Para utilizar Rosetta Drone, los usuarios deben conectar su teléfono Android a un transmisor DJI y encender el dron. La aplicación Rosetta Drone facilita la comunicación con el dron mediante el SDK de DJI y establece una conexión de telemetría con aplicaciones como QGroundControl a través del protocolo MAVLink. Los usuarios pueden controlar el dron, recibir telemetría en tiempo real y ejecutar *scripts* de inteligencia artificial para diversas aplicaciones avanzadas.

2.2.4 Discusión

En el presente epígrafe del estado del arte, se ha decidido mencionar el proyecto Rosetta Drone debido a que representa una forma alternativa de aprovechar las capacidades del SDK de DJI, explorando más allá de sus limitaciones estándar. Este enfoque se alinea con el objetivo del presente TFM, que también busca extender las posibilidades del SDK de DJI y superar sus restricciones convencionales.

Tal vez se podría pensar que la forma más sencilla de explorar opciones alternativas fuera del ecosistema del SDK de DJI sea la utilización del proyecto Rosetta Drone. Sin embargo, esta alternativa no soluciona los problemas de transferencia de imágenes a los que el presente TFM pretende dar solución. En el proyecto Rosetta Drone, la imagen captada por el sistema de visión del dron se transfiere mediante un flujo de video que, aunque funcional, sufre de limitaciones significativas en términos de calidad y latencia. La imagen final resultante es de menor calidad y presenta un retardo perceptible, lo cual impide un procesamiento eficiente y preciso sobre la misma.

Actualmente, se están haciendo muchos esfuerzos para mejorar esta situación. Los desarrolladores y la comunidad en general están trabajando arduamente para reducir la latencia y mejorar la calidad de la imagen transmitida. No obstante, hasta el día de hoy, no se ha logrado ofrecer una solución estable y robusta que elimine completamente

estos problemas. Por ello, mientras el proyecto Rosetta Drone muestra un camino prometedor, aún no puede proporcionar las capacidades necesarias para satisfacer las necesidades específicas de procesamiento de imágenes en tiempo real que plantea este TFM.

Otra de las razones que han llevado a no utilizar el proyecto Rosetta Drone en el presente TFM es la configuración específica del dron DJI Phantom 4 Advanced+, que, tal y como se explica en la Subsección 3.1.2, es el dron sobre el que se llevará a cabo el embarcado del Jetson Nano. A diferencia de otros drones del fabricante DJI, este modelo integra el teléfono móvil en el propio mando, haciendo que mando y móvil sean una única unidad, como se puede apreciar en la Figura 2.7. Esta configuración ofrece varias ventajas, como una mayor estabilidad y comodidad de uso al evitar conexiones externas adicionales y una interfaz optimizada para el control del dron.

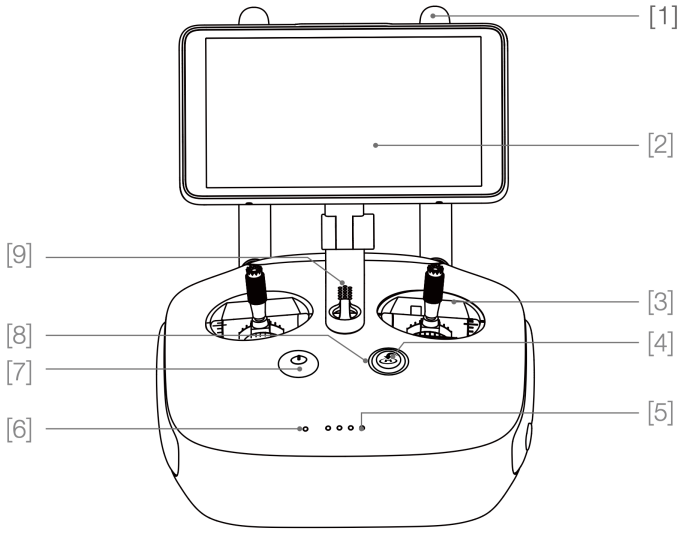


Figura 2.7: Esquema del control remoto utilizado para comandar el DJI Phantom 4 Advanced+ [35].

Sin embargo, presenta un inconveniente significativo: la integración rígida entre el móvil y el mando hace imposible la instalación de aplicaciones y programas de terceros. Incluso las aplicaciones desarrolladas con el SDK de DJI enfrentan estas restricciones. Muchos expertos consideran que esta tarea es prácticamente imposible debido a las restricciones del sistema operativo y la falta de acceso administrativo al dispositivo, ver [36].

No obstante, para el presente TFM, se logró encontrar una solución ingeniosa para introducir una aplicación de terceros desarrollada con el SDK de DJI en este móvil integrado. Esto era esencial no solo para desarrollar la aplicación móvil, sino también

para poder ejecutarla efectivamente en el dispositivo integrado y aprovechar el abanico de posibilidades ofrecido por el SDK de DJI. Sin esta capacidad para introducir la aplicación desarrollada con el SDK de DJI, el proyecto no habría podido avanzar debido a la imposibilidad de utilizar el SDK de DJI y, por lo tanto, no se habría podido llevar a cabo el control del dron desde una aplicación externa desarrollada por un usuario.

Dada la complejidad y las dificultades encontradas al introducir aplicaciones de terceros en el móvil del mando del DJI Phantom 4 Advanced+, y considerando que Rosetta Drone es una aplicación de terceros no desarrollada con el SDK de DJI, se ha descartado la utilización del Proyecto Rosetta Drone. Este proyecto, al no estar basado en el SDK de DJI, enfrenta las mismas restricciones y problemas de compatibilidad, lo que hace inviable su uso en el contexto del presente TFM.

2.3 ROS (*Robot Operating System*)

A pesar de que en el presente TFM no se ha empleado ROS, es crucial mencionar su relevancia en el desarrollo actual de aplicaciones para drones. El *Robot Operating System* (ROS) se ha convertido en una herramienta indispensable en la robótica moderna, y su uso en drones está en constante crecimiento debido a sus múltiples ventajas.

ROS es un conjunto de bibliotecas y herramientas de *software* de código abierto que facilitan el desarrollo de aplicaciones robóticas. A diferencia de los sistemas operativos tradicionales, ROS proporciona servicios como abstracción de *hardware*[6], control de dispositivos de bajo nivel y comunicación entre procesos. Estos servicios permiten la integración eficiente de componentes mecánicos y tecnológicos, facilitando la implementación de algoritmos avanzados de navegación, control y procesamiento de datos. Esta capacidad de integración optimiza el rendimiento y la funcionalidad de los drones, permitiendo desarrollos más rápidos y eficaces.

En el ámbito de los drones, ROS facilita la creación de sistemas autónomos y semiautónomos gracias a sus capacidades de modularidad y reutilización de código. Esto permite a los desarrolladores implementar rápidamente nuevas funcionalidades y adaptarse a los avances tecnológicos del sector. Además, la activa comunidad y la vasta cantidad de recursos disponibles en ROS potencian su adopción y evolución continua.

La creciente popularidad de ROS en el desarrollo de drones se debe a su capacidad para manejar la complejidad de los sistemas UAV modernos, ofreciendo soluciones eficientes para la comunicación entre sensores, actuadores y sistemas de control. Mediante ROS,

[6]Los servicios de abstracción de *hardware* en ROS proporcionan una capa intermedia entre el *hardware* del robot y el *software* de control. Esto permite a los desarrolladores interactuar con el *hardware* sin preocuparse por los detalles específicos de cada dispositivo. Por ejemplo, ROS puede proporcionar interfaces estándar para controlar diferentes tipos de motores, sensores y actuadores, independientemente del fabricante. Esta abstracción simplifica el desarrollo de aplicaciones robóticas, permitiendo a los desarrolladores centrarse en la lógica y funcionalidad del sistema en lugar de en los detalles de implementación del *hardware*.

los desarrolladores pueden diseñar aplicaciones que optimizan el rendimiento de los drones, mejorando su capacidad para realizar tareas complejas como la navegación autónoma, el mapeo 3D y el procesamiento en tiempo real de datos sensoriales. De manera sencilla, ROS se puede comparar con un «cerebro» que coordina todas las funciones y comunicaciones entre las diferentes «partes del cuerpo» del dron, como los sensores, cámaras y motores. De manera más compleja, en ROS, los «nodos» son pequeñas aplicaciones que realizan tareas específicas, como controlar un sensor o procesar datos. Estos nodos se comunican entre sí mediante «mensajes» que se envían a través de «*topics*». Un nodo puede «publicar» un mensaje en un «*topic*», mientras que otros nodos pueden «suscribirse» a ese *topic* para recibir los mensajes. Además, ROS utiliza «servicios» para la comunicación bidireccional, permitiendo que un nodo solicite información o acciones de otro nodo y reciba una respuesta. En la Figura 2.8 se esquematiza este tipo de funcionamiento.

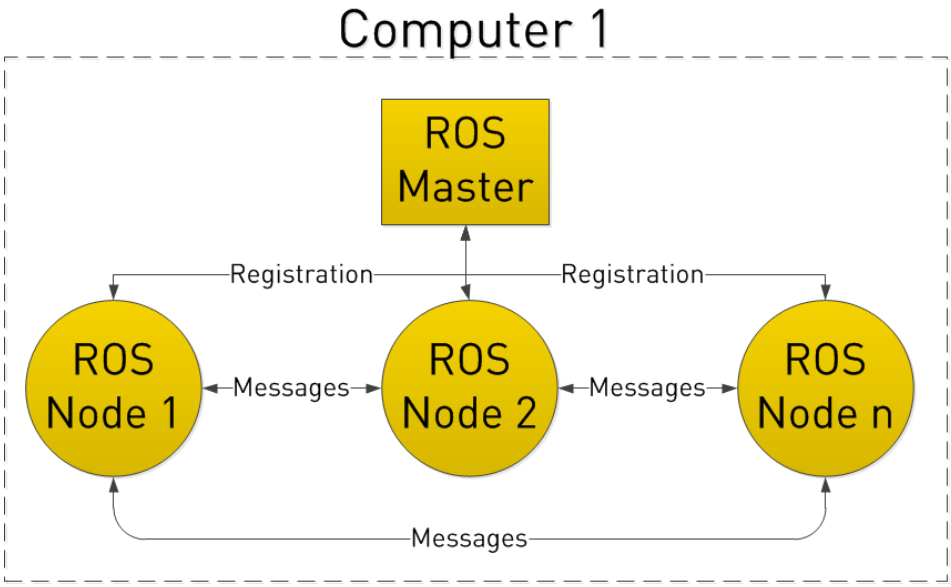


Figura 2.8: Funcionamiento básico de ROS [37].

En resumen, aunque ROS no se ha utilizado directamente en este TFM, su importancia en el desarrollo de aplicaciones para drones justifica su mención, destacando cómo este sistema operativo se ha integrado profundamente en la robótica aérea y sigue siendo un pilar fundamental en la innovación y evolución de los UAVs.

2.4 Códigos ArUco

Como se mencionó en el capítulo introductorio de este TFM, ver Sección 1.1, la aplicación de procesamiento de imágenes que se pretende desarrollar tiene como objetivo capturar imágenes continuamente a través de una cámara conectada al ordenador embarcado en el dron. Estas imágenes son analizadas para detectar la presencia de códigos ArUco en el entorno del dron. Dependiendo de si se detectan o no estos códigos, se enviarán comandos específicos al dron para que actúe en consecuencia. Este proceso permite al dron realizar acciones automáticas basadas en la detección de los marcadores visuales, mejorando así su capacidad para interactuar de manera autónoma con su entorno.

Dado que la comprensión de este tema puede no ser generalizada en el ámbito de los drones, a continuación se presenta una introducción detallada a los códigos ArUco, abarcando su origen, funcionamiento, aplicaciones y otros aspectos relevantes necesarios para la realización del presente TFM.

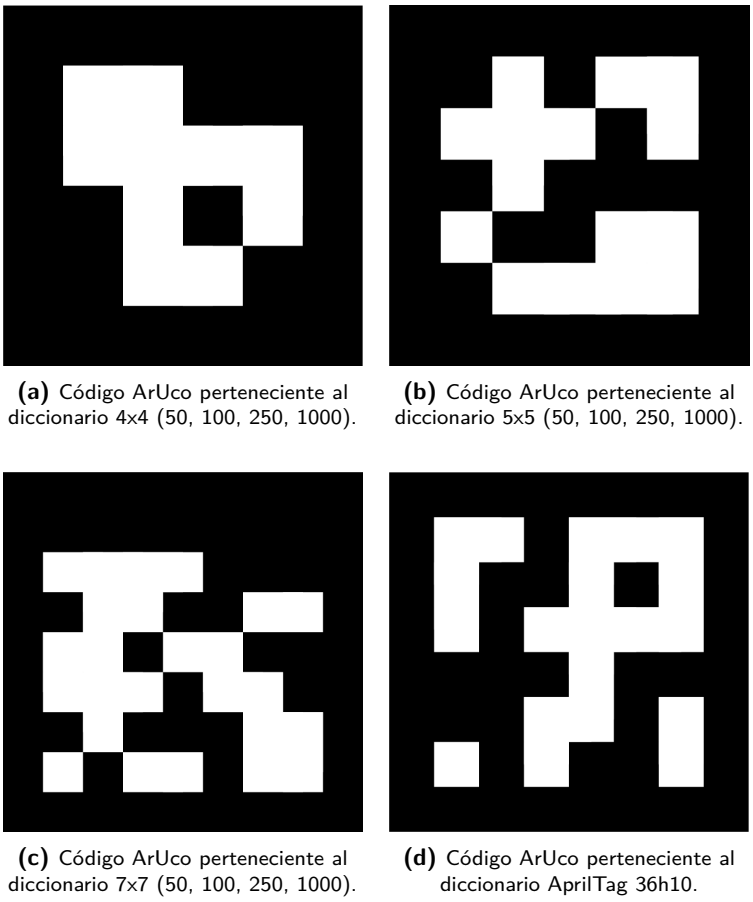
2.4.1 Definición

Los códigos ArUco son marcadores visuales bidimensionales utilizados en aplicaciones de visión por computadora, desarrollados principalmente para resolver problemas de estimación de pose[7] de cámara en realidad aumentada y otras aplicaciones similares. Estos marcadores consisten en una disposición de cuadros blancos y negros que forman patrones únicos, permitiendo una detección y reconocimiento robustos por algoritmos de visión por computadora. En la Figura 2.9 se muestran cuatro códigos ArUco de diferentes tamaños y pertenecientes a diccionarios distintos.

2.4.2 Origen y desarrollo

El término «ArUco» proviene de «*Augmented Reality University of Cordoba*», donde se desarrollaron inicialmente. Se diseñaron para proporcionar una solución eficaz y precisa a la detección y seguimiento de patrones en imágenes, facilitando aplicaciones de realidad aumentada (AR) y robótica.

[7] La estimación de pose de cámara es el proceso de determinar la posición y orientación de una cámara en el espacio tridimensional, a partir de imágenes capturadas de marcadores visuales u otros puntos de referencia.



(a) Código ArUco perteneciente al diccionario 4x4 (50, 100, 250, 1000).

(b) Código ArUco perteneciente al diccionario 5x5 (50, 100, 250, 1000).

(c) Código ArUco perteneciente al diccionario 7x7 (50, 100, 250, 1000).

(d) Código ArUco perteneciente al diccionario AprilTag 36h10.

Figura 2.9: Ejemplos de códigos ArUco pertenecientes a diferentes diccionarios [38].

2.4.3 Funcionamiento de los códigos ArUco

Cada marcador ArUco tiene un patrón único que lo identifica. Los algoritmos de detección pueden localizar estos patrones en una imagen, determinar sus bordes y calcular la orientación y posición de la cámara respecto al marcador. Este proceso se realiza a través de técnicas de visión por computadora que identifican las esquinas del marcador y utilizan estas referencias para calcular la pose, ver Figura 2.10.

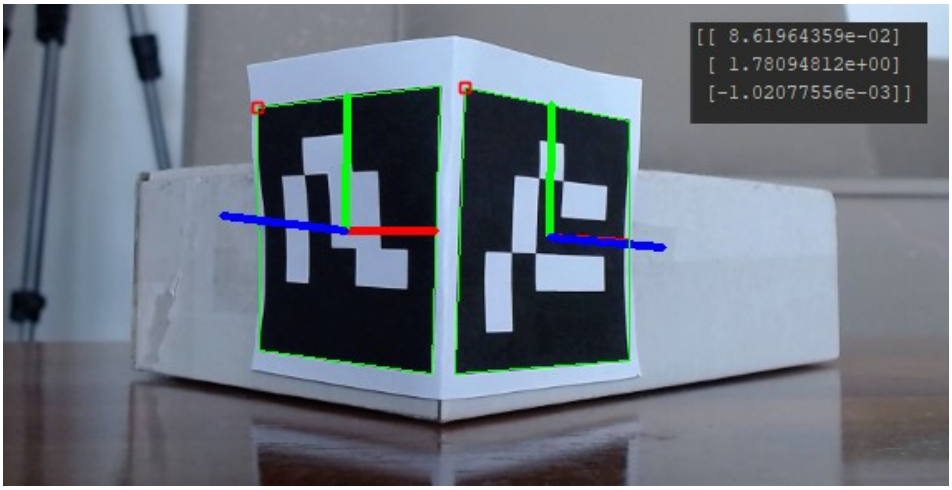


Figura 2.10: Identificación de las esquinas en un marcador ArUco [39].

2.4.4 Campos de utilización de los códigos ArUco

Los códigos ArUco se utilizan en una variedad de aplicaciones, incluyendo:

- **Realidad aumentada:** Para superponer contenido digital en el mundo real.
- **Robótica:** Para navegación y localización de robots.
- **Calibración de cámaras:** Para mejorar la precisión en la captura de imágenes.
- **Seguimiento de objetos:** En sistemas de seguimiento y control.

2.4.5 Ventajas y desventajas de los códigos ArUco

Ventajas

- **Simplicidad:** Son fáciles de generar, imprimir y detectar.
- **Robustez:** Funcionan bien en condiciones variadas de iluminación y ángulos.
- **Precisión:** Ofrecen alta precisión en la estimación de la pose.
- **Flexibilidad:** Se pueden utilizar en una amplia gama de aplicaciones como se expuso anteriormente.

Desventajas

- **Limitaciones en distancia:** Su detección es menos efectiva a grandes distancias.

- **Sensibilidad a la oclusión:** La detección puede fallar si el marcador está parcialmente oculto.
- **Resolución dependiente:** Requieren una resolución adecuada de la cámara para una detección precisa.

2.4.6 Implementación con OpenCV

La biblioteca OpenCV[8] proporciona herramientas para generar y detectar estos marcadores de manera sencilla. Utilizando el módulo «cv2.aruco», los desarrolladores pueden crear diccionarios de marcadores, detectar estos marcadores en imágenes y calcular su pose. Este proceso implica la detección de las esquinas del marcador, la identificación del patrón y la estimación de la posición y orientación del marcador respecto a la cámara.

Para obtener más información sobre los códigos ArUco, se recomienda consultar las siguientes fuentes bibliográficas [39-41].

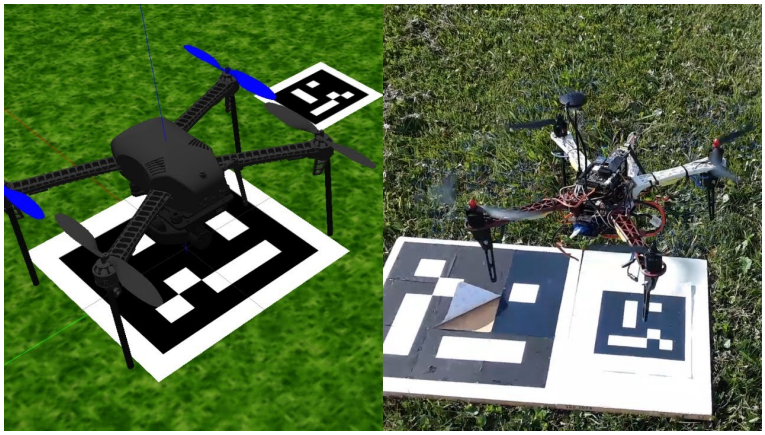


Figura 2.11: Ejemplo de utilización de códigos ArUco para realizar aterrizajes automáticos con drones [42].

[8]OpenCV (*Open Source Computer Vision Library*) es una biblioteca de software de código abierto destinada a aplicaciones de visión por computadora y aprendizaje automático.

2.5 *Companion Computers* - Gama Jetson NVIDIA

En la introducción de esta memoria se mencionó la necesidad de un ordenador embarcado para resolver el problema de la transferencia de imágenes vía radiofrecuencia. Existen múltiples opciones en el ámbito de los *Companion Computers*. Sin embargo, con el avance de la inteligencia artificial, NVIDIA ha tomado la delantera en este sector. Para apoyar el desarrollo de aplicaciones basadas en IA, NVIDIA ha lanzado la gama de módulos Jetson, dispositivos compactos diseñados para funcionar como *Companion Computers*.

A continuación, se detallan las características de estos dispositivos y se explica por qué se consideran la opción preferida para embarcar en drones:

Los dispositivos NVIDIA Jetson son plataformas de computación embebida diseñadas específicamente para aplicaciones de inteligencia artificial (IA) y para procesar datos localmente, sin necesidad de enviarlos a centros de datos remotos.

2.5.1 Características y ventajas de los dispositivos NVIDIA Jetson

1. **Alto rendimiento computacional:** Los módulos Jetson, como el Jetson Nano, Jetson TX2 y Jetson Xavier, ofrecen un rendimiento computacional significativo. Por ejemplo, el Jetson Nano cuenta con una GPU Maxwell de 128 núcleos, una CPU ARM Cortex-A57 de cuatro núcleos y 4GB de RAM, proporcionando 472 GFLOPS de rendimiento, lo cual es suficiente para diversas cargas de trabajo de IA y tareas de aprendizaje profundo, ver [43] y [44].

2. **Eficiencia energética:** Estos dispositivos están diseñados para ofrecer un alto rendimiento mientras consumen una mínima cantidad de energía, lo que los hace ideales para aplicaciones móviles y embebidas. Por ejemplo, el Jetson Nano opera entre 5 a 10 vatios, mientras que la serie más potente Jetson Xavier puede operar dentro de un rango de 10 a 30 vatios, ver [45].

3. **Capacidades versátiles de IA:** Los dispositivos Jetson son compatibles con una amplia gama de *frameworks* y modelos de IA, incluyendo TensorFlow, PyTorch y Caffe. Esta versatilidad permite a los desarrolladores ejecutar diversas redes neuronales para tareas como reconocimiento de imágenes, detección de objetos y estimación de poses. El Jetson Nano puede procesar múltiples flujos de video HD en tiempo real, lo cual es particularmente útil para aplicaciones como la vigilancia y cámaras inteligentes, ver [46].

4. **Ecosistema robusto y soporte de desarrollo:** NVIDIA proporciona un soporte integral de software a través del SDK JetPack, que incluye bibliotecas para aprendizaje profundo, visión por computadora, computación GPU, procesamiento multimedia y más. Además, el NVIDIA Transfer Learning Toolkit y

los modelos de IA preentrenados facilitan el rápido despliegue y optimización de aplicaciones de IA.

5. **Formato y compatibilidad:** Los módulos Jetson están diseñados para ser compactos y son compatibles con diversos tamaños, lo que facilita su integración en diferentes sistemas. Por ejemplo, el Jetson Xavier NX, que mide solo 70 x 45 mm, puede integrarse en sistemas de visión de alta resolución, máquinas autónomas y drones.

2.6 Discusión final

2.6.1 Tipología del Dron seleccionado

En este Trabajo de Fin de Máster (TFM), se han elegido drones de tipo *software* privativo de DJI en lugar de drones del tipo *Open Source*. Esta elección se debe a varias ventajas específicas que estos drones privativos ofrecen, particularmente en el contexto de la investigación y el desarrollo de soluciones avanzadas. Los drones de DJI, aunque menos personalizables que sus contrapartes de código abierto, proporcionan una plataforma robusta y confiable con un rendimiento probado en condiciones variadas. Una de las principales razones de esta elección es el objetivo central del TFM de resolver el problema de la transferencia de imágenes vía radiofrecuencia. Embarcando un ordenador a bordo del dron, se consigue una mayor autonomía y capacidad de procesamiento de imágenes, permitiendo salirse del ecosistema limitado del SDK de DJI y demostrando que los drones privativos también pueden admitir un cierto grado de personalización y adaptación a aplicaciones específicas.

Además, los drones seleccionados para este proyecto son de tipo multirrotor en lugar de ala fija. Los drones multirrotor ofrecen una serie de ventajas que son cruciales para la investigación y las pruebas necesarias en este TFM. Estos drones proporcionan una mayor maniobrabilidad y estabilidad en vuelo estacionario, lo cual es esencial para la captura precisa y continua de imágenes en entornos controlados. A diferencia de los drones de ala fija, los multirrotor pueden despegar y aterrizar verticalmente, operar en espacios reducidos y mantener una posición fija en el aire, lo que facilita la realización de pruebas repetitivas y controladas. Estas características hacen que los drones multirrotor sean la opción más adecuada para los objetivos de este TFM, ya que los drones de ala fija no permitirían la flexibilidad y precisión necesarias para las pruebas y demostraciones requeridas.

2.6.2 Decisión de utilizar los códigos ArUco

Este Trabajo de Fin de Máster (TFM) ha optado por utilizar códigos ArUco en la aplicación de procesamiento de imágenes a bordo del ordenador embarcado en el dron. La decisión de centrarse en los códigos ArUco, en lugar de otros métodos como el seguimiento de objetos o la detección de obstáculos, se fundamenta en varias ventajas clave que estos códigos ofrecen.

Los códigos ArUco proporcionan una solución robusta y precisa para la identificación y localización de marcadores en el entorno del dron. Su diseño simple y distintivo, compuesto por patrones de cuadros blancos y negros, permite una detección precisa incluso en condiciones de iluminación variable y ángulos desafiantes. Esta robustez es crucial para aplicaciones en las que la precisión y la fiabilidad son esenciales, como en el caso de la navegación autónoma y la interacción con el entorno.

Además, los códigos ArUco permiten una estimación precisa de la pose de la cámara, es decir, la posición y orientación del dron en relación con los marcadores. Esta capacidad es fundamental para aplicaciones que requieren un control preciso del dron y su alineación con objetos o áreas específicas. A diferencia de otros métodos de procesamiento de imágenes, los códigos ArUco facilitan una integración directa con algoritmos de control y navegación, proporcionando datos exactos y consistentes que mejoran la estabilidad y precisión del sistema.

Otra razón para elegir los códigos ArUco es su facilidad de generación y uso. Los marcadores pueden ser impresos y colocados en el entorno de manera sencilla, lo que reduce la complejidad y el costo de implementación. Esto contrasta con otros métodos, como el seguimiento de objetos, que a menudo requieren entrenar modelos de aprendizaje profundo y disponer de grandes cantidades de datos de entrenamiento, aumentando significativamente el tiempo y los recursos necesarios.

La decisión de emplear códigos ArUco en este Trabajo de Fin de Máster (TFM) se fundamenta en una serie de ventajas clave: robustez, precisión en la estimación de pose, facilidad de implementación y versatilidad. Estas características hacen de los códigos ArUco una solución óptima para el procesamiento de imágenes en aplicaciones avanzadas a bordo de drones DJI. Además, estos códigos proporcionan una base sólida para el desarrollo y la mejora de sistemas autónomos y semi-autónomos, facilitando la integración de tecnologías avanzadas y mejorando la eficiencia operativa del dron.

2.6.3 Decisión de utilizar dispositivos NVIDIA Jetson

En este Trabajo de Fin de Máster (TFM), se ha optado por utilizar el Jetson Nano de NVIDIA como *Companion Computer* para el procesamiento de imágenes y la realización de diversas tareas a bordo de un dron DJI. Esta elección se basa en varias ventajas clave que ofrece el Jetson Nano frente a otros tipos de *Companion Computers*:

Economía del producto

- El Jetson Nano es una solución económica que proporciona un excelente equilibrio entre coste y rendimiento. Comparado con otros *Companion Computers*, el Jetson Nano ofrece un precio accesible sin comprometer la capacidad de procesamiento, lo que lo convierte en una opción ideal para proyectos de investigación y desarrollo con presupuestos limitados.

Versatilidad

- La versatilidad del Jetson Nano es otra de sus grandes fortalezas. Este dispositivo es compatible con una amplia variedad de periféricos y sensores, permitiendo su uso en múltiples aplicaciones de inteligencia artificial y visión por computadora. Además, su soporte para diversas bibliotecas y *Frameworks* de software, como CUDA, TensorRT y ROS, facilita el desarrollo e implementación de soluciones avanzadas.

Robustez

- El Jetson Nano ha demostrado ser robusto y fiable en condiciones operativas exigentes. Su diseño compacto y resistente permite su integración en drones, asegurando un rendimiento estable incluso en entornos adversos. Esta robustez es crucial para aplicaciones que requieren una alta disponibilidad y fiabilidad del sistema.

Capacidad de procesamiento

- Equipado con una GPU de 128 núcleos NVIDIA Maxwell, el Jetson Nano ofrece una capacidad de procesamiento superior para tareas de inteligencia artificial y procesamiento de imágenes. Esta potencia de cómputo permite ejecutar algoritmos complejos en tiempo real, mejorando significativamente la eficiencia y efectividad de las aplicaciones a bordo del dron.

Peso y Tamaño

- El peso y tamaño del Jetson Nano son factores determinantes para su uso en drones. Con un peso de aproximadamente 70 gramos y un diseño compacto, este dispositivo minimiza la carga útil adicional en el dron, optimizando su rendimiento de vuelo y autonomía. Esta característica es particularmente importante en aplicaciones donde el peso y el espacio son limitaciones críticas.

En resumen, la elección del Jetson Nano de NVIDIA como *Companion Computer* para este TFM se fundamenta en su economía, versatilidad, robustez, capacidad de procesamiento y diseño ligero y compacto. Estas características hacen del Jetson Nano una solución ideal para abordar los desafíos del procesamiento de imágenes y la ejecución de tareas avanzadas a bordo de drones DJI.

Capítulo 3

Materiales y recursos empleados

En el Capítulo 1 de esta memoria, ver Capítulo 1, se introdujo el entorno de trabajo sobre el cual se desarrolla este TFM. Posteriormente, en el capítulo 2, ver Capítulo 2, se presentó un estado del arte que abarca todos los elementos presentes en el proyecto. El propósito de este capítulo fue definir y explicar con rigor y precisión los conceptos e ideas previamente mencionados en la introducción, que no habían sido abordados de manera suficientemente clara y didáctica. Esta explicación detallada es especialmente importante para aquellos lectores que no están familiarizados con el ecosistema de trabajo del TFM, ya que sin esta clarificación, dichos conceptos podrían resultarles confusos y complejos.

El presente capítulo tiene como objetivo introducir y explicar todos los materiales y recursos empleados en la realización del TFM. Esto incluye, desde recursos de *hardware*, como los drones utilizados, el ordenador embarcado, la cámara conectada al ordenador en el dron, y las baterías necesarias para alimentar el *Companion Computer* (Jetson Nano) durante el vuelo, hasta recursos de *software*, como los lenguajes de programación, *Frameworks* de trabajo y programas utilizados.

Se busca con este capítulo detallar todas las herramientas de trabajo empleadas para construir el TFM. Además, en aquellos casos en los que sea necesario, se explicarán las razones por las cuales se han seleccionado ciertas herramientas sobre otras que, aunque cumplen funciones similares, provienen de diferentes marcas o distribuidores. Esto permitirá una comprensión completa de las decisiones tomadas en cuanto a la elección de materiales y recursos, proporcionando una base sólida para la replicabilidad y evaluación del proyecto.

3.1 Hardware

En la presente sección se detallan todos los recursos y elementos de hardware empleados para la realización del TFM.

3.1.1 Ordenador embarcado - reComputer J3011

En el presente Trabajo de Fin de Máster, se ha optado por utilizar el reComputer J3011 de Seeed Studio, ver [47], en lugar del Jetson Nano estándar, debido a las numerosas ventajas significativas que ofrece este dispositivo, las cuales se comentarán a continuación.

¿Qué es el reComputer J3011 y cual es su Relación con el Jetson Nano?

El reComputer J3011 es un dispositivo destinado al desarrollo de aplicaciones de inteligencia artificial del tipo *Edge* AI, que incorpora el módulo NVIDIA Jetson Orin Nano de 8GB. Este módulo ofrece hasta 40 TOPS de rendimiento en inteligencia artificial, duplicando el rendimiento del Jetson Xavier NX. La principal diferencia entre el reComputer J3011 y el Jetson Nano estándar es que el reComputer es una solución completa, preinstalada con el sistema JetPack 5.1.1, y equipada con una variedad de puertos y conectividades adicionales que facilitan su uso en aplicaciones industriales y de investigación, como es el caso del presente TFM.

Antes de continuar, es importante definir los conceptos de TOPS y JetPack, ya que pueden resultar complejos y difusos para el lector:

- TOPS (*Tera Operations Per Second*) es una medida de rendimiento que indica la capacidad de procesamiento de un dispositivo para realizar un billón (10^12) de operaciones por segundo. En el contexto de la inteligencia artificial y el aprendizaje automático, TOPS se utiliza para evaluar la eficiencia de un chip en ejecutar operaciones matemáticas intensivas, como las necesarias para el entrenamiento y la inferencia de redes neuronales profundas. Un mayor número de TOPS implica un mayor rendimiento y una capacidad superior para manejar cargas de trabajo de inteligencia artificial en tiempo real, lo cual es crucial para aplicaciones avanzadas como el procesamiento de imágenes y el reconocimiento de patrones.
- JetPack es un conjunto de herramientas y bibliotecas de *software* desarrollado por NVIDIA para sus plataformas de inteligencia artificial Jetson. JetPack incluye el sistema operativo Linux basado en Ubuntu, bibliotecas de desarrollo como CUDA, cuDNN, y TensorRT, así como herramientas de visión por computadora y aprendizaje profundo. Este paquete proporciona un entorno de desarrollo completo para crear y desplegar aplicaciones de inteligencia artificial en dispositivos Jetson, facilitando la implementación de soluciones de IA desde la prototipación hasta la producción. JetPack permite a los desarrolladores aprovechar al

máximo las capacidades de *hardware* de los dispositivos Jetson, optimizando el rendimiento y la eficiencia de las aplicaciones de inteligencia artificial y robótica.

Figura 3.1: reComputer J3011 [47].

Ventajas del reComputer J3011 [47]

1. **Rendimiento superior:** El reComputer J3011 ofrece un rendimiento de hasta 40 TOPS en IA, gracias a su módulo Jetson Orin Nano de 8GB, lo que lo hace ideal para aplicaciones que requieren un procesamiento intensivo de datos en tiempo real.

2. **Conectividad versátil:** Incluye una amplia gama de puertos como 4 puertos USB 3.2, HDMI 2.1, M.2 para SSD y WiFi, entre otros, lo que permite una integración flexible con diversos periféricos y sensores.

3. **Diseño compacto y robusto:** El dispositivo cuenta con un diseño compacto y una carcasa de aluminio con un disipador de calor pasivo, lo que asegura un enfriamiento eficiente sin necesidad de ventiladores. Esto reduce el riesgo de fallos por polvo y otros contaminantes y minimiza el ruido, haciéndolo adecuado para entornos sensibles al ruido.

4. **Facilidad de implementación:** El reComputer J3011 viene preinstalado con el sistema JetPack, lo que simplifica su configuración y desarrollo. Además, su compatibilidad con las bibliotecas y herramientas de NVIDIA, como CUDA y TensorRT, facilita el desarrollo de aplicaciones avanzadas de IA.

Prestaciones [47]

- **Almacenamiento:** Incluye un SSD NVMe de 128GB, que proporciona un almacenamiento rápido y suficiente para aplicaciones complejas.
- **Conectividad:** Soporta múltiples opciones de conectividad, incluyendo Ethernet Gigabit, WiFi y Bluetooth, así como puertos para expansión adicional como PCIe y SIM para 4G/5G.
- **Soporte para vídeo:** Puede manejar múltiples flujos de vídeo 4K, lo que lo hace ideal para aplicaciones de análisis de vídeo y procesamiento de imágenes, aspectos que son precisamente relevantes para el ámbito de este TFM.

Posibles inconvenientes

- **Coste:** El reComputer J3011 es más costoso que el Jetson Nano estándar, lo que puede ser un factor limitante en proyectos con presupuestos ajustados.
- **Consumo energético:** Aunque eficiente, su alto rendimiento en IA puede implicar un mayor consumo energético en comparación con modelos menos potentes.

A pesar de explorar otros dispositivos similares, se ha decidido utilizar el reComputer J3011 en este TFM debido a su rendimiento superior, su conectividad versátil y su diseño robusto y compacto. Estas características lo hacen ideal para las necesidades de procesamiento de imágenes y tareas avanzadas a bordo del dron DJI. Además, su facilidad de implementación y soporte para múltiples flujos de trabajo de IA proporcionan una base sólida para futuras aplicaciones. Aunque en el presente TFM no se han explorado las vías de la inteligencia artificial, la elección del reComputer J3011 es una decisión estratégica que prepara la infraestructura necesaria para dotar de aplicaciones de IA a drones DJI en el futuro. Esto permitirá el desarrollo y la implementación de soluciones avanzadas de inteligencia artificial a medida que el proyecto evolucione, aunque estos cambios y evoluciones no se aborden en el presente TFM.

Para obtener más información sobre el reComputer J3011, incluyendo especificaciones técnicas, imágenes, prestaciones, configuración y otros detalles relacionados con este dispositivo, se recomienda al lector consultar las siguientes fuentes bibliográficas [47] y [48].

3.1.2 Drones empleados

En el presente Trabajo de Fin de Máster, se han empleado dos drones distintos: el DJI Phantom 4 Advanced y el DJI Mavic Mini. El dron final sobre el cual se embarcará el reComputer J3011 es el DJI Phantom 4 Advanced. Sin embargo, la mayor parte de las pruebas durante la fase de desarrollo, diseño y testeo de los algoritmos se han realizado con un DJI Mavic Mini.

La razón de utilizar dos drones diferentes radica en las dimensiones y las características de vuelo de cada uno. El Phantom 4 Advanced es un dron de gran tamaño, cuyo vuelo en interiores no es recomendable por razones de seguridad. Este dron requiere un entorno controlado y amplio para su operación segura, cumpliendo siempre con la normativa vigente que regula su uso en exteriores, la cual fue comentada anteriormente en esta memoria, ver Subsección 2.1.2.

Durante la fase de desarrollo, es crucial realizar pruebas continuas y obtener retroalimentación inmediata para ajustar y mejorar el programa de procesamiento de imágenes. Esto incluye la resolución previa del problema de comunicaciones planteado en la introducción, que requiere múltiples pruebas para asegurar una conexión estable y eficiente. Utilizar únicamente el Phantom 4 en esta fase habría alargado significativamente la duración del TFM, debido a las limitaciones impuestas por su necesidad de operar en entornos exteriores.

Por este motivo, se decidió emplear un dron de desarrollo como el DJI Mavic Mini, un dron de tamaño reducido que permite su vuelo en el interior de una habitación sin comprometer la seguridad de las personas presentes. El Mavic Mini facilita la realización de pruebas rápidas y frecuentes, permitiendo un ciclo de desarrollo más ágil y eficiente. Esta estrategia no solo optimiza el tiempo de desarrollo, sino que también garantiza que los algoritmos y el sistema de procesamiento de imágenes sean rigurosamente probados antes de su implementación final en el Phantom 4 Advanced.

En resumen, la utilización de dos drones distintos en este TFM responde a la necesidad de equilibrar la seguridad y la eficiencia durante el desarrollo y la validación de los algoritmos y sistemas de procesamiento de imágenes. El DJI Mavic Mini proporciona un entorno seguro y controlado para las pruebas iniciales, mientras que el DJI Phantom 4 Advanced ofrece la plataforma definitiva para la implementación del reComputer J3011 y la realización de tareas avanzadas en exteriores.

A continuación, se presenta una descripción más detallada de las especificaciones de estos dos drones.

DJI Mavic Mini

El DJI Mavic Mini es un dron ultraligero, con un peso de solo 249 gramos, diseñado para ser accesible tanto para principiantes como para usuarios más experimentados. Su diseño compacto facilita el transporte y su uso en diversas aplicaciones, inclu-

yendo vuelos en interiores, gracias a su tamaño reducido y facilidad de maniobra. A continuación, se presentan algunas de sus especificaciones más destacadas:

1. **Cámara y gimbal:** El Mavic Mini está equipado con una cámara que puede capturar vídeos en 2.7K a 30 fps y en 1080p a 60 fps. También soporta la captura de imágenes de 12 megapíxeles. La cámara está montada en un gimbal de 3 ejes que asegura una estabilización suave y estable del vídeo, ideal para obtener tomas cinematográficas de alta calidad.

2. Modos de vuelo: Ofrece tres modos de vuelo principales:

   - **P-Mode (Positioning):** Utiliza GPS y sistemas de visión para localizar y estabilizar el dron.
   - **S-Mode (Sport):** Maximiza la velocidad y agilidad del dron.
   - **C-Mode (Cinesmooth):** Suaviza los movimientos del dron para obtener tomas más fluidas y cinematográficas.

3. **Distancia y tiempo de vuelo:** El Mavic Mini puede volar hasta 30 minutos con una sola carga de batería y tiene una distancia máxima de transmisión de vídeo de 4 km en condiciones óptimas, usando una conexión mejorada por WiFi

4. **Control remoto y conectividad:** El control remoto se conecta a un Smartphone a través de la aplicación DJI Fly, proporcionando una interfaz intuitiva y fácil de usar. El controlador incluye funciones básicas como despegue y aterrizaje automáticos, y botones para captura de fotos y vídeos.

Figura 3.2: DJI Mavic Mini [49].

A continuación, se citan algunas de las ventajas del DJI Mavic Mini:

1. **Portabilidad:** Su diseño ultraligero y compacto facilita su transporte y uso en diversas ubicaciones.
2. **Facilidad de uso:** La interfaz de la aplicación DJI Fly es intuitiva y está diseñada para facilitar el vuelo y la captura de imágenes, incluso para usuarios novatos.
3. **Calidad de imagen:** La cámara de 2.7K y el gimbal de 3 ejes aseguran una calidad de vídeo estable y clara, adecuada para aplicaciones recreativas y algunas profesionales.
4. **Tiempo de vuelo:** Hasta 30 minutos de tiempo de vuelo con una sola carga, lo cual es significativo para su tamaño y categoría.

Ahora se especifican algunos de los posibles inconvenientes de este dron:

1. **Falta de sensores de evitación de obstáculos:** A diferencia de otros modelos más avanzados de DJI, el Mavic Mini no cuenta con sensores de evitación de obstáculos, lo que puede representar un riesgo para pilotos inexpertos.
2. **Capacidades limitadas de la cámara:** Aunque la cámara es adecuada para la mayoría de las aplicaciones recreativas, no soporta la captura de imágenes RAW ni ofrece ajustes manuales extensivos, lo que puede limitar su uso para fotógrafos y videógrafos profesionales.
3. **Sensibilidad al viento:** Debido a su peso ligero, el Mavic Mini puede tener dificultades para mantenerse estable en condiciones de viento fuerte.

A pesar de explorar otros dispositivos similares, se ha decidido utilizar el DJI Mavic Mini en este TFM debido a su excelente equilibrio entre portabilidad, facilidad de uso y calidad de imagen. El uso del Mavic Mini permite un desarrollo ágil y eficiente en espacios interiores, facilitando las pruebas y ajustes necesarios sin comprometer la seguridad. Estas características lo convierten en una elección estratégica para el desarrollo de aplicaciones avanzadas a bordo de drones DJI.

La información utilizada para la redacción de este epígrafe ha sido obtenida de las siguientes referencias bibliográficas, [50-53].

DJI Phantom 4 Advanced / Advanced+

El DJI Phantom 4 Advanced / Advanced+, ver Figura 3.3, es un dron de alta gama diseñado por DJI, conocido por su avanzada tecnología de vuelo y capacidades de captura de imágenes. A continuación, se presenta una revisión detallada de sus especificaciones, características, ventajas e inconvenientes, basada tanto en el manual del usuario como en información adicional obtenida de diversas fuentes que pueden ser consultadas en los siguientes enlaces.

Figura 3.3: DJI Phantom 4 Advanced / Advanced+ [54].

En primer lugar, las características más destacables a mencionar son las siguientes:

1. **Cámara y gimbal:** El Phantom 4 Advanced está equipado con una cámara que cuenta con un sensor CMOS de 1 pulgada, capaz de capturar vídeo en 4K a hasta 60 fotogramas por segundo y fotos de 20 megapíxeles. El gimbal[9] de 3 ejes asegura la estabilidad de la cámara, permitiendo obtener imágenes nítidas y vídeos estables. Además, la cámara tiene un obturador mecánico para eliminar la distorsión por *rolling shutter*.

2. **Controlador de vuelo:** El controlador de vuelo del Phantom 4 Advanced incluye modos de vuelo inteligentes como TapFly y ActiveTrack, que permiten al dron volar hacia cualquier punto visible en la pantalla con solo un toque y seguir automáticamente a sujetos en movimiento. También incluye un sistema de evitación de obstáculos que utiliza sensores visuales delanteros y un sistema de posicionamiento visual para mantener la estabilidad en interiores o en entornos sin GPS.

3. **Batería y tiempo de vuelo:** El Phantom 4 Advanced utiliza una batería de vuelo inteligente de 5870 mAh, proporcionando hasta 30 minutos de tiempo de vuelo. Esta batería incluye funciones avanzadas de gestión de energía y protección contra sobrecargas y sobredescargas.

4. **Control remoto:** El control remoto del Phantom 4 Advanced+ incluye una pantalla de alta luminosidad de 5,5 pulgadas integrada con la aplicación DJI GO,

[9]Un gimbal es un dispositivo mecánico que estabiliza la cámara montada en un dron, utilizando varios ejes de rotación para contrarrestar los movimientos del dron y mantener la cámara nivelada y estable, lo que permite capturar imágenes y videos nítidos y sin vibraciones.

que permite un control total del dron. Además, tiene un rango de transmisión de vídeo HD de hasta 7 kilómetros gracias a la tecnología DJI Lightbridge.

Antes de continuar, se definen los siguientes conceptos, ya que fueron mencionados anteriormente y pueden ser desconocidos para el público general. La siguiente información ha sido obtenida de las siguientes referencias bibliográficas [55-57]:

- **Rolling Shutter:** El «*rolling shutter*» es una técnica de captura de imágenes en la que cada fotograma de vídeo se captura de manera progresiva en lugar de hacerlo en un solo instante. En lugar de exponer todo el sensor de la cámara a la vez, el *rolling shutter* expone líneas sucesivas de píxeles, lo que puede generar distorsiones en la imagen si el objeto o la cámara se mueven rápidamente. Este efecto es común en cámaras con sensores CMOS y puede causar problemas como la distorsión en objetos en movimiento rápido, conocido como «efecto gelatina» o «*jello effect*».
- **Sensor CMOS:** Un sensor CMOS (*Complementary Metal-Oxide-Semiconductor*) es un tipo de sensor de imagen utilizado en cámaras digitales que convierte la luz en señales eléctricas. Los sensores CMOS son conocidos por su bajo consumo de energía y alta velocidad de procesamiento, lo que los hace ideales para una variedad de aplicaciones, desde fotografía y videografía hasta dispositivos móviles y cámaras de seguridad. A diferencia de los sensores CCD (*Charge-Coupled Device*), los sensores CMOS permiten leer y procesar las imágenes más rápidamente, aunque pueden estar más sujetos a efectos de *rolling shutter* debido a su método de captura de imagen lineal.

A continuación, se destacan algunas de las ventajas más distintivas del dron Phantom 4 Advanced:

1. **Calidad de imagen superior:** La cámara de 1 pulgada y 20 megapíxeles proporciona imágenes de alta calidad y vídeos en 4K, ideales para aplicaciones profesionales de fotografía y videografía aérea.
2. **Evita obstáculos:** Los sensores visuales delanteros permiten al dron detectar y evitar obstáculos, mejorando la seguridad durante el vuelo.
3. **Modos de vuelo inteligentes:** Funciones como TapFly y ActiveTrack facilitan la operación del dron, incluso para usuarios menos experimentados.
4. **Largo tiempo de vuelo:** La batería de alta capacidad ofrece hasta 30 minutos de vuelo, lo cual es significativo para tareas prolongadas de captura de datos.

Finalmente, se recogen algunos de los posibles inconvenientes que puede presentar este dron:

1. **Tamaño y peso:** El Phantom 4 Advanced es más grande y pesado que otros drones de DJI, lo que puede limitar su portabilidad y facilidad de uso en espacios reducidos.

2. **Precio:** Es uno de los drones más caros de la gama Phantom, lo que puede no ser ideal para todos los presupuestos.

3. **Dependencia de GPS:** Aunque tiene un sistema de posicionamiento visual, su rendimiento óptimo se logra con una señal GPS fuerte, lo que puede ser un problema en áreas con señal débil.

Para el propósito del presente TFM, es necesario un dron con las características y dimensiones del Phantom 4 Advanced. Este dron es capaz de levantar el peso del ordenador embarcado (reComputer J3011) y de la batería adicional necesaria para su funcionamiento durante el vuelo. Estas capacidades son esenciales para poder realizar el procesamiento de imágenes y otras tareas avanzadas directamente a bordo del dron. El DJI Mavic Mini, por razones obvias, no podría llevar a cabo esta tarea debido a sus limitaciones en cuanto a capacidad de carga y tamaño, lo que imposibilitaría el embarcado del reComputer J3011.

Para más información acerca de este dron, se recomienda la consulta de las siguientes fuentes bibliográficas, [35, 58-60].

3.1.3 Sistema de visión a embarcar - Raspberry Pi NoIR Camera V2

La Raspberry Pi NoIR Camera V2 es un módulo de cámara diseñado para la Raspberry Pi que cuenta con un sensor Sony IMX219 de 8 megapíxeles. Este sensor permite capturar imágenes estáticas con una resolución de 3280 x 2464 píxeles y soporta múltiples modos de vídeo, incluyendo 1080p a 30 fps, 720p a 60 fps y 640x480p a 90 fps. Una característica distintiva de esta cámara es la ausencia de un filtro infrarrojo (NoIR), lo que la hace ideal para fotografía en condiciones de poca luz y para aplicaciones de visión nocturna mediante iluminación infrarroja.

Características principales

- **Resolución:** 8 megapíxeles, capaz de capturar imágenes de alta resolución.
- **Modos de vídeo:** Soporta 1080p30, 720p60 y 640x480p90, proporcionando flexibilidad en la grabación de vídeos.
- **Tamaño y peso:** Compacta y ligera, con dimensiones de 25 x 24 x 9 mm y un peso de apenas 3 gramos.
- **Conectividad:** Se conecta a la Raspberry Pi a través de la interfaz CSI, diseñada específicamente para cámaras.

- **Visión nocturna:** La ausencia del filtro infrarrojo permite que la cámara capture imágenes en condiciones de poca luz cuando se utiliza con iluminación infrarroja.

Ventajas de la Raspberry Pi NoIR Camera V2

1. **Calidad de imagen:** La alta resolución y los modos de vídeo avanzados permiten obtener imágenes y vídeos de alta calidad.
2. **Flexibilidad en condiciones de poca luz:** Su capacidad para capturar imágenes en condiciones de poca luz es una ventaja significativa para aplicaciones de vigilancia y monitoreo nocturno.
3. **Compatibilidad y facilidad de uso:** Es compatible con todas las versiones de Raspberry Pi que tienen un conector CSI y puede ser utilizada con varias librerías de software, como Picamera y libcamera.
4. **Tamaño compacto y peso ligero:** Su pequeño tamaño y ligereza facilitan su integración en proyectos donde el espacio y el peso son limitados.

Inconvenientes

1. **Falta de autoenfoque:** La lente de enfoque fijo puede ser una limitación para aplicaciones que requieren un enfoque dinámico.
2. **Dependencia de iluminación infrarroja:** Para aplicaciones nocturnas, se necesita una fuente de luz infrarroja externa.
3. **Limitaciones en condiciones de alta luz:** La ausencia del filtro infrarrojo puede resultar en imágenes menos precisas en condiciones de alta iluminación diurna.

Justificación de la elección

Para el presente TFM, se ha seleccionado la Raspberry Pi NoIR Camera V2, ver Figura 3.4, debido a varias razones. Primero, su alta calidad de imagen y su capacidad para funcionar en condiciones de poca luz la hacen ideal para aplicaciones de monitoreo y procesamiento de imágenes a bordo del dron. En comparación con una webcam tradicional, la NoIR Camera V2 ofrece una mayor resolución y flexibilidad en modos de vídeo, lo que es crucial para la precisión y detalle en las imágenes capturadas.

Además, la integración de esta cámara con el reComputer J3011 proporciona una plataforma robusta y personalizable para el desarrollo y testeo de algoritmos de procesamiento de imágenes. El reComputer J3011 cuenta con dos puertos CSI para cámaras, lo que facilita la conexión y optimización de dispositivos de visión avanzada como la NoIR Camera V2. Aunque las webcams tradicionales pueden ser más económicas, la NoIR Camera V2 ofrece un equilibrio óptimo entre coste y rendimiento, siendo una

elección estratégica para proyectos que requieren capacidades avanzadas de captura y procesamiento de imágenes. Esto es particularmente importante para las aplicaciones de drones, donde se necesita una alta calidad de imagen y fiabilidad en condiciones variadas, asegurando un monitoreo eficaz y preciso durante el vuelo.

La información utilizada para la redacción de este epígrafe ha sido obtenida de las siguientes referencias bibliográficas, [61-65].

Figura 3.4: Cámara Raspberry Pi NoIR Camera V2 [66].

3.1.4 Batería para la alimentación del reComputer J3011

Antes de pasar a explicar el tipo de batería seleccionada, es interesante hacer un pequeño análisis sobre las formas de alimentar el reComputer J3011.

El reComputer J3011 se alimenta exclusivamente a través del conector *Barrel Jack*. Aunque el dispositivo cuenta con un puerto USB-C, este está configurado en «*Device Mode*», lo que significa que no puede ser utilizado para la carga, sino para la conexión a otros dispositivos.

En cuanto a las especificaciones de voltaje y corriente, el reComputer J3011 puede operar en un rango de voltaje de 9V a 19V. La fuente de alimentación recomendada es de 12V y 5A, utilizando un conector *Barrel Jack* de 5.5/2.5mm. La potencia máxima que el dispositivo puede demandar es de 60W (calculado como 12V * 5A).

El consumo de potencia del reComputer J3011, y consecuentemente la corriente, varía según la carga de trabajo del dispositivo. A mayor carga, mayor será la potencia demandada, y a menor carga, menor será la potencia requerida.

Para alimentar el reComputer J3011, se necesita una *Powerbank* que pueda proporcionar un voltaje de 12V y una corriente de al menos 5A. La capacidad de la *Powerbank* debe ajustarse al tiempo de operación deseado. Por ejemplo, para alimentar el dispositivo durante 30 minutos bajo una demanda de 60W, se necesita una *Powerbank* con una capacidad mínima de 2500mAh. Este cálculo se basa en la suposición de que el reComputer J3011 está operando a su máxima potencia de manera continua.

Para calcular la capacidad mínima de 2500 mAh necesaria para alimentar el reComputer J3011 durante 30 minutos, se debe considerar que el dispositivo consume una potencia máxima de 60W. Para un funcionamiento de 30 minutos, equivalente a 0.5 horas, la energía necesaria se calcula multiplicando la potencia por el tiempo, resultando en 30 Wh (vatios-hora). Dado que la batería opera a un voltaje de 12V, la capacidad en amperios-hora se obtiene dividiendo la energía entre el voltaje, lo que da 2.5 Ah. Finalmente, al convertir amperios-hora a miliamperios-hora, se multiplica por 1000, obteniendo una capacidad mínima de 2500 mAh.

La capacidad de la batería de la *Powerbank*, expresada en mAh, determina cuánto tiempo puede funcionar el reComputer J3011. En la Tabla 3.1 se muestra un cálculo simplificado para diferentes tiempos de operación.

CAPACIDAD MÍNIMA DE LA BATERÍA NECESARIA		
Tiempo de operación	**Capacidad mínima [mAh]**	**Condiciones de operación**
15 minutos	1250	Potencia máxima demandada: 60W
30 minutos	2500	Potencia máxima demandada: 60W
45 minutos	3750	Potencia máxima demandada: 60W
60 minutos	5000	Potencia máxima demandada: 60W

Tabla 3.1: Capacidad mínima de la batería necesaria para el reComputer J3011 [Elaboración Propia].

En conclusión, para asegurar una alimentación adecuada del reComputer J3011, es esencial contar con una Powerbank que pueda suministrar 12V y al menos 5A. La capacidad de la Powerbank debe seleccionarse según el tiempo de operación requerido, entendiendo que el consumo de potencia puede variar en función de la carga de trabajo del dispositivo.

Para alimentar el reComputer J3011, se ha seleccionado la batería LiPo U-TECH PRO 4s 14.8V 1550mAh 95C, ver Figura 3.5. Esta batería proporciona un voltaje de 14.8V, adecuado para el rango operativo del dispositivo (9V a 19V), una corriente máxima de 147.25A, que supera ampliamente los 5A requeridos, y una capacidad de 1550mAh, lo que permite un tiempo de vuelo aceptable para el dron. Sin embargo, el principal

inconveniente de esta batería es su peso, un factor crítico en aplicaciones de drones donde la relación carga-peso es crucial. A pesar de su mayor peso en comparación con una *powerbank*, se ha decidido utilizar esta batería debido a su accesibilidad y menor coste. Las *powerbanks* que cumplían con las especificaciones necesarias eran considerablemente más costosas y no se ajustaban al presupuesto limitado de este TFM. Por lo tanto, como diseño preliminar y para avanzar en el desarrollo del TFM, se ha optado por esta batería.

Figura 3.5: Batería LiPo U-TECH PRO 4s 14.8V 1550mAh 95C [67].

3.2 Software

En esta sección se detallan todos los componentes de software necesarios para la realización del TFM. Esto incluye lenguajes de programación, entornos de desarrollo integrados (IDE), librerías más importantes y cualquier otro programa que haya sido esencial para llevar a cabo las tareas relacionadas con el proyecto.

3.2.1 Lenguajes de programación

Para la realización del presente TFM se han empleado dos lenguajes de programación distintos, cada uno seleccionado por sus ventajas y capacidades específicas para diferentes componentes del proyecto.

En el desarrollo de aplicaciones móviles utilizando el SDK de DJI, se pueden emplear tanto Kotlin como Java. Aunque Kotlin ofrece una sintaxis concisa y expresiva, así como una interoperabilidad total con Java, se ha optado por utilizar Java en este TFM. Esta decisión se debe a la estabilidad y la amplia adopción de Java en aplicaciones empresariales y servidores, lo que proporciona una base robusta para el desarrollo de la aplicación móvil. Además, Java cuenta con una comunidad de soporte más grande y recursos de aprendizaje extensivos, lo que facilita la resolución de problemas y la integración de diversas funcionalidades. En base a lo anterior, se concluye que, para el

presente proyecto, todas las partes relacionadas con la aplicación móvil y su conversión en un servidor se han llevado a cabo utilizando el lenguaje de programación Java.

Finalmente, se ha utilizado Python para todo el desarrollo de la aplicación de procesamiento de imágenes instalada en el Jetson embarcado en el dron. Python es conocido por su simplicidad y legibilidad, lo que acelera el desarrollo y la depuración del código. Además, su extensa biblioteca de módulos y herramientas específicas para el procesamiento de imágenes y aprendizaje automático, como OpenCV y TensorFlow, lo hacen ideal para este tipo de aplicaciones. Python permite una rápida prototipación y prueba de algoritmos complejos, facilitando la implementación de soluciones avanzadas en el procesamiento de imágenes y visión por computadora.

3.2.2 Entornos de desarrollo integrados (IDE)

Para el desarrollo de este TFM, se han empleado tres entornos de desarrollo integrados (IDE) distintos, cada uno seleccionado por sus ventajas específicas y capacidades optimizadas para las tareas requeridas.

Para las tareas de programación en Java, se han utilizado dos IDEs diferentes. Para el desarrollo y diseño de la aplicación móvil se ha empleado Android Studio. Android Studio es el IDE oficial para el desarrollo de aplicaciones Android, desarrollado por Google. Ofrece una integración completa con las herramientas y librerías de Android, un editor visual intuitivo, un emulador de Android y un sistema de depuración avanzado, lo que facilita enormemente el desarrollo de aplicaciones móviles.

Además, para la programación en Java, se ha utilizado NetBeans como IDE de escritorio. NetBeans es ampliamente utilizado para el desarrollo en Java, conocido por su robustez y extensibilidad. Ofrece herramientas integradas para la gestión de proyectos y una interfaz de usuario intuitiva, lo que facilita la escritura, edición y depuración de código Java. En este TFM, NetBeans se utilizó específicamente para desarrollar una serie de *scripts* en Java que sirvieron como base y punto de partida para el posterior desarrollo de la aplicación de procesamiento de imágenes en Python. Inicialmente, se empleó Java programando en NetBeans para asentar las ideas y el fundamento de la aplicación de procesamiento de imágenes. Estos *scripts* en Java proporcionaron una referencia conceptual y una guía práctica para el desarrollo futuro de la aplicación en Python.

Para la programación en Python, se ha utilizado Visual Studio Code. Visual Studio Code es un editor de código fuente ligero pero potente, con soporte extensible para múltiples lenguajes de programación a través de sus extensiones. Su integración con herramientas de depuración, control de versiones y la capacidad de personalizar el entorno de desarrollo hacen de Visual Studio Code una opción ideal para la programación en Python. Además, su amplia comunidad y la disponibilidad de extensiones específi-

cas para Python, como Pylint y Jupyter, facilitan el desarrollo y la implementación de aplicaciones de procesamiento de imágenes en el Jetson embarcado en el dron.

3.2.3 Librerías más importantes

Para la realización del presente TFM, se han empleado varias librerías esenciales, entre las cuales destacan el SDK de DJI y OpenCV. Estas librerías han sido fundamentales para el desarrollo de las aplicaciones móviles y de procesamiento de imágenes, respectivamente.

El SDK de DJI (*Software Development Kit*) es un conjunto de librerías y herramientas que permiten a los desarrolladores crear aplicaciones personalizadas para drones DJI. Este SDK proporciona acceso a una amplia gama de funcionalidades del dron, incluyendo el control de vuelo, la captura de imágenes y vídeos, y la transmisión de datos en tiempo real. En el contexto de este TFM, el SDK de DJI ha sido crucial para el desarrollo de la aplicación móvil, permitiendo una integración eficiente con el dron y facilitando el control y la monitorización de sus operaciones.

OpenCV (*Open Source Computer Vision Library*) es una librería de código abierto que contiene más de 2500 algoritmos optimizados para el procesamiento de imágenes y visión por computadora. Esta librería es ampliamente utilizada en aplicaciones de reconocimiento de objetos, detección de movimiento y análisis de imágenes en tiempo real. Para el desarrollo de la aplicación de procesamiento de imágenes en el reComputer J3011, OpenCV ha sido indispensable debido a su robustez, flexibilidad y amplia comunidad de soporte. Su capacidad para manejar tareas complejas de visión por computadora ha permitido la implementación de algoritmos avanzados, esenciales para las tareas de monitoreo y análisis a bordo del dron.

Un componente particularmente relevante de OpenCV en este TFM es el módulo de ArUco, que ofrece una amplia variedad de funciones y soporte para el reconocimiento y procesamiento de códigos ArUco. Estos códigos son patrones binarios utilizados en visión por computadora para la detección de objetos y el cálculo de la pose en aplicaciones de realidad aumentada y robótica. OpenCV facilita la detección y decodificación de estos códigos, así como el cálculo preciso de la pose, permitiendo una integración eficiente con las tareas de navegación y posicionamiento del dron. Esto es crucial para la aplicación de procesamiento de imágenes desarrollada en el reComputer J3011, ya que uno de sus objetivos principales es el reconocimiento y procesamiento de códigos ArUco, asegurando una operación fiable y precisa del dron en su entorno operativo.

3.2.4 Programas de diseño 3D

Para la tarea de embarcar el reComputer J3011, que será analizada en capítulos posteriores de esta memoria, aunque finalmente se optó por un diseño más artesanal mediante la adaptación de piezas prefabricadas cuya impresión en 3D no era necesaria, en las primeras aproximaciones para resolver el problema del embarcado, se realizaron diferentes diseños de piezas en 3D. Todos estos diseños 3D se realizaron utilizando el programa de diseño FreeCAD.

FreeCAD es un *software* de diseño asistido por computadora (CAD) de código abierto y gratuito, que permite a los usuarios crear modelos 3D precisos y detallados. FreeCAD es altamente modular, lo que significa que puede ser ampliado mediante el uso de *plugins* para adaptarse a necesidades específicas. Su interfaz intuitiva y la capacidad de trabajar con múltiples formatos de archivo CAD hacen que sea una herramienta versátil para diseñadores e ingenieros.

Una de las razones por las que se eligió FreeCAD para este proyecto, en lugar de otros programas de *software* privativo como Fusion 360, AutoCAD, CATIA o SolidWorks, es su accesibilidad y flexibilidad. Al ser un software de código abierto, FreeCAD permite una mayor personalización y adaptación a las necesidades específicas del proyecto sin incurrir en los altos costes de licencias asociadas con el *software* privativo. Además, la comunidad activa y el continuo desarrollo de FreeCAD aseguran un flujo constante de mejoras y nuevas funcionalidades, lo que lo convierte en una opción robusta y confiable para el diseño 3D en proyectos de investigación y desarrollo.

Para más información sobre este programa, se recomienda al lector consultar la documentación oficial, accesible en [68].

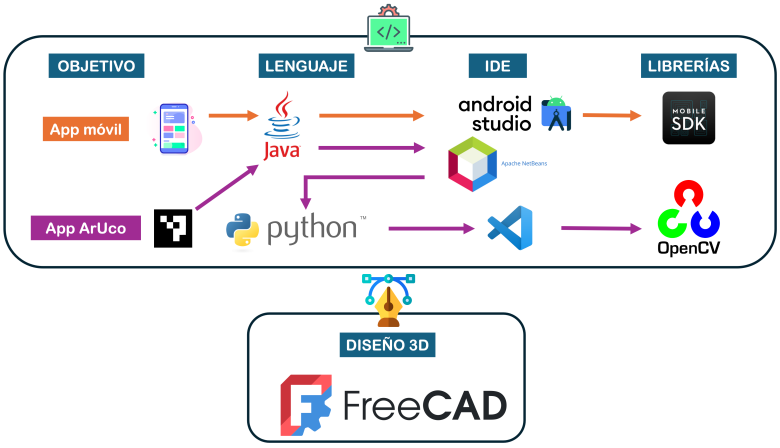


Figura 3.6: *Software* empleado en el TFM [Elaboración Propia].

Capítulo 4

Desarrollo del Proyecto

Antes de abordar los conceptos y temas que se tratarán en el presente capítulo, es conveniente hacer un breve repaso del contenido de los capítulos anteriores. Este recordatorio tiene como objetivo situar al lector en el contexto adecuado y facilitar la comprensión del propósito de este capítulo. Además, proporciona una visión integral tanto del proyecto como de la presente memoria.

En primer lugar, en el capítulo introductorio, ver Capítulo 1, se expusieron los pilares fundamentales sobre los cuales se construye el proyecto. Se argumentó la motivación detrás del mismo y se presentó el ecosistema global de trabajo en el que se desarrollará el presente TFM, ver Sección 1.1. Finalmente, se definieron los objetivos que este TFM pretende alcanzar, Sección 1.2.

En segundo lugar, el capítulo 2, ver Capítulo 2, estuvo dedicado al estado del arte. En este capítulo se ofreció una explicación detallada y rigurosa de los conceptos e ideas introducidos en el primer capítulo, con el fin de proporcionar al lector una base teórica sólida necesaria para comprender el TFM.

Finalmente, en el capítulo 3, Capítulo 3, se describieron los recursos y materiales necesarios para la realización del TFM, tanto a nivel de *hardware* como de *software*, explicando las razones detrás de la elección de cada uno de ellos.

El presente capítulo detalla el desarrollo del proyecto, así como los cambios, mejoras y avances realizados para alcanzar los objetivos establecidos en el TFM. Dado que se trata de un proyecto multidisciplinar, intervienen numerosos elementos, muchos de los cuales son independientes entre sí. Por ejemplo, el desarrollo de la aplicación móvil en Java no está directamente relacionado con la aplicación de procesamiento de imágenes en Python, aunque el funcionamiento de una depende de la otra. Es decir, aunque sean componentes distintos, forman parte del mismo conjunto.

Esta naturaleza multidisciplinar ha llevado a estructurar el capítulo en bloques. El orden en el que estos bloques van apareciendo en el capítulo sigue la línea temporal de las acciones ejecutadas en el TFM. Sin embargo, dado que el proyecto está en constante cambio y evolución, la secuencialidad de las tareas es compleja. A pesar de que en esta memoria se expongan las tareas de manera secuencial, muchas de ellas se llevaron a cabo de forma paralela. Por ejemplo, aunque el primer bloque de este capítulo trata sobre los cambios y mejoras realizados en la aplicación móvil, durante las tareas y pruebas de embarque en el dron, que se abordan en el último bloque, todavía se estaban realizando ajustes en la aplicación móvil. Esto evidencia la paralelización de las tareas en el proyecto.

4.1 Desarrollo, cambios y mejoras en la aplicación móvil Android

En la configuración del presente TFM, es fundamental contar con una aplicación móvil desarrollada con el SDK de DJI. Esta aplicación móvil actuará como servidor, encargándose de enviar y recibir comandos hacia y desde el ordenador reComputer J3011. Como se mencionó en la introducción de este TFM, el punto de partida para la aplicación móvil es el Trabajo de Fin de Máster de Leya [10], quien desarrolló una aplicación móvil utilizando el UX SDK.

Este punto de partida permite aprovechar la estructura y funcionalidad básica de la aplicación existente, aunque será necesario adaptar y mejorar la aplicación para cumplir con los requisitos específicos del presente proyecto. A continuación, se detallan todos los cambios, mejoras y modificaciones realizados en la aplicación móvil para adaptarla a las necesidades del presente TFM.

En primer lugar, partiendo del código fuente de la aplicación desarrollada en [10], el primer gran cambio llevado a cabo es la migración del código fuente de la aplicación móvil del UX SDK al Mobile SDK.

Las razones que motivan esta migración son diversas. En primer lugar, el UX SDK está orientado al desarrollo de interfaces de usuario muy avanzadas, las cuales no se iban a emplear en este proyecto. Por otro lado, el Mobile SDK es considerado el principal estado del arte en el desarrollo de aplicaciones móviles hoy en día, ofreciendo una amplia gama de funcionalidades específicas para la interacción con drones DJI. Además, el Mobile SDK proporciona mejoras significativas en términos de rendimiento, lo que es crucial para asegurar una operación fluida y eficiente de la aplicación en diferentes condiciones de vuelo. La migración al Mobile SDK permite aprovechar estas ventajas, optimizando así tanto el desarrollo como el funcionamiento de la aplicación móvil dentro del contexto del presente TFM.

Una vez completada la migración entre SDKs, el siguiente paso fue la depuración del código fuente de la aplicación. Este proceso es crucial porque la migración de un SDK a otro no es simplemente una tarea de copiar y pegar. Las funcionalidades disponibles en el UX SDK no siempre están presentes en el Mobile SDK, lo cual implica que se deben realizar modificaciones en la lógica del código para adaptarlo a las nuevas herramientas y funciones disponibles.

Durante esta depuración, fue necesario identificar y reestructurar las partes del código que dependían de funcionalidades específicas del UX SDK. Esto incluyó la adopción de soluciones alternativas proporcionadas por el Mobile SDK para garantizar que la aplicación continuara funcionando correctamente. Este trabajo de depuración y ajuste fue esencial para asegurar que todas las características críticas de la aplicación fueran compatibles con el nuevo SDK y que el rendimiento general de la aplicación mejorara, conforme a los objetivos del presente TFM.

Una vez completada la migración del código fuente al Mobile SDK y depurada la aplicación para garantizar su correcto funcionamiento bajo las nuevas directrices, se implementó la primera mejora significativa. Esta mejora consistió en la implementación de un temporizador (*timer*) para el envío de comandos, ver Código 4.1 y Código 4.2.

Anteriormente, tal y como había sido desarrollada en [10], la aplicación requería que el usuario enviara comandos múltiples veces para que estos tuvieran un efecto continuo. Por ejemplo, si se quería que el dron ascendiera, el usuario debía enviar el comando «GO UP» repetidamente. Enviar el comando una sola vez solo hacía que el dron subiera un poco, sin mantener el movimiento ascendente de manera sostenida.

Con la nueva implementación del temporizador, este problema se resolvió. Ahora, el temporizador permite reenviar el comando cada 5 milisegundos de manera indefinida hasta que el usuario envía un comando diferente. Esto libera al usuario de tener que presionar el botón continuamente, ya que el temporizador se encarga de enviar el comando repetidamente, asegurando una operación continua y fluida del dron.

En el estado del arte actual, este enfoque basado en temporizadores es ampliamente utilizado. No solo mejora la experiencia del usuario al facilitar el control del dron, sino que también hace que la operación de la aeronave sea más fluida y eficiente.

```
1 public void control() {
2     if (null == sendVirtualStickDataTimer) {
3         sendVirtualStickDataTask = new SendVirtualStickDataTask();
4         sendVirtualStickDataTimer = new Timer();
5         sendVirtualStickDataTimer.schedule(sendVirtualStickDataTask,
              0, 5);
6     }
7 }
```

Código 4.1: Código en Java del *Timer* para el envío de comandos de manera continua.

```
private class SendVirtualStickDataTask extends TimerTask {
    @Override
    public void run() {
        if (fc != null) {
            fc.sendVirtualStickFlightControlData(new FlightControlData
                (Roll, Pitch, Yaw, Throttle), new CommonCallbacks.
                CompletionCallback() {
                @Override
                public void onResult(DJIError djiError) {
                    if (djiError != null) {
                        ToastUtils.setResultToToast(djiError.
                            getDescription());
                    }
                }
            });
        }
    }
}
```

Código 4.2: Código de la implementación del *Timer* dentro de la lógica general del aplicación móvil.

La siguiente gran modificación y mejora realizada sobre la aplicación móvil de partida fue la implementación de la recepción de la telemetría relativa a las distancias de los sensores a los obstáculos en el entorno, ver Código 4.3 y Código 4.4. Esta mejora se hizo necesaria debido a que el dron utilizado en el presente TFM, un DJI Phantom 4 Advanced+, está equipado con sensores de distancia a obstáculos, a diferencia del dron utilizado en [10], un DJI Mavic Mini, que no cuenta con este tipo de sensorización.

La implementación de esta mejora consistió en modificar la parte del código fuente de la aplicación móvil que se encarga de la recepción de la telemetría por parte del dron. Ahora, además de las variables estándar de telemetría, se incluyen las nuevas variables que contienen las distancias a los obstáculos captadas por los sensores. Este cambio permite recibir y procesar la telemetría completa proporcionada por el Phantom 4 Advanced+, mejorando así la capacidad de la aplicación para manejar situaciones en las que la detección de obstáculos es crítica.

Conviene destacar que, para la implementación de esta mejora, se ha definido un vector «`distDCS`» que almacena las distancias a los obstáculos en cuatro sectores horizontales de 15 grados cada uno.

```
public CockpitServer(Context mcontext) {
    running = true;
    mContext = mcontext;
    distDCS = new float[4]; // 4 sectores horizontales de 15 grados de
        FOV (60 / 4 o 70)

    DJISampleApplication.getAircraftInstance().getFlightController().
        getFlightAssistant().setVisionDetectionStateUpdatedCallback(
        new VisionDetectionState.Callback() {
        @Override
        public void onUpdate(@NonNull VisionDetectionState
            visionDetectionState) {
            ObstacleDetectionSector[] visionDetectionSectorArray =
                visionDetectionState.getDetectionSectors();

            for (int i = 0; i < visionDetectionSectorArray.length; i
                ++) {
                distDCS[i] = visionDetectionSectorArray[i].
                    getObstacleDistanceInMeters();
            }
        }
    });
}
```

Código 4.3: Código Java para la implementación de la sensorización del dron en la telemetría.

```
while (DJISampleApplication.isAircraftConnected() && this.running) {
    Log.d(TAG, msg: "************************Aircraft connected
        *************************");
    status();

    dOut.writeFloat(GS);
    dOut.writeFloat(VS);
    dOut.writeFloat(alt);
    dOut.writeDouble(heading);
    dOut.writeDouble(roll);
    dOut.writeDouble(pitch);
    for (int i = 0; i < distDCS.length; i++) {
        dOut.writeFloat(distDCS[i]);
    }
}
```

Código 4.4: Código Java del bucle principal de la recepción de la Telemtría.

4.2 Desarrollo de los *scripts* preliminares en Java

Una vez implementadas todas las mejoras en la aplicación móvil, se procedió a verificar su correcto funcionamiento frente a las nuevas modificaciones introducidas. En colaboración con el tutor del presente TFM, se desarrollaron *scripts* sencillos en Java con dos propósitos principales.

En primer lugar, estos *scripts* demostraron efectivamente el retardo, la latencia y la deficiente calidad asociadas a la transferencia de imágenes vía radiofrecuencia. En segundo lugar, establecieron las bases y la estructura general de lo que sería la futura aplicación de procesamiento de imágenes desarrollada en Python.

Como se ha mencionado a lo largo de esta memoria, la comunicación entre el dron y su mando se realiza vía radiofrecuencia, mientras que la comunicación entre el dron y la aplicación móvil desarrollada por un tercero e instalada en un dispositivo Android se lleva a cabo en dos etapas. La primera etapa implica la transferencia de datos por radiofrecuencia entre el dron y su mando. En la segunda etapa, una vez que los datos han llegado al mando del dron, estos se transfieren vía cable USB al dispositivo Android donde se encuentra instalada la aplicación móvil.

Dado este esquema de comunicaciones, es evidente que los datos están siempre sujetos a la transferencia vía radiofrecuencia. Entre los datos transmitidos se encuentra la imagen captada por los sistemas de visión del dron, que es enviada al mando por radiofrecuencia y luego transferida al dispositivo Android mediante cable USB. En la aplicación móvil, el usuario puede realizar diversos tipos de procesamiento sobre estas imágenes. Sin embargo, estos procesamientos no son efectivos ni eficientes debido a la baja calidad y el lag que presentan las imágenes transmitidas.

Uno de los tipos de procesamiento que el usuario puede realizar es la retransmisión de las imágenes a un servidor de *streaming*. Para este TFM, se desarrolló en Java, en un ordenador externo, un servidor de *streaming* capaz de recibir las imágenes transmitidas desde la aplicación móvil del usuario. Este *script* recibía las imágenes retransmitidas con el fin de realizar algún tipo de procesamiento posterior sobre las mismas. El objetivo fundamental de este *script* era demostrar que, debido a la baja calidad de las imágenes, los procesamientos posteriores eran inviables.

Este escenario pone de manifiesto la necesidad de explorar alternativas que permitan el procesamiento efectivo de las imágenes captadas, ya que la calidad y la eficiencia actuales son insuficientes. El presente TFM surge precisamente para abordar y solucionar estos problemas de transferencia de imágenes, buscando métodos más robustos y eficientes para el procesamiento de las mismas.

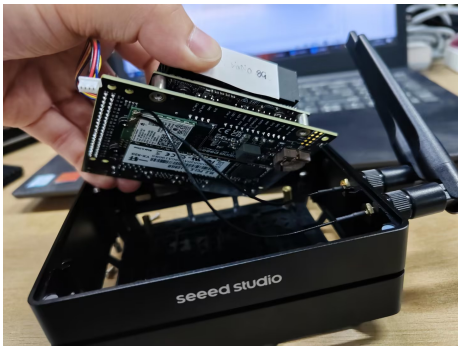
4.3 Instalación y puesta a punto del Jetson Nano

En esta sección se describe la instalación y configuración del Jetson Nano, preparándolo como entorno de desarrollo para el diseño de una aplicación de procesamiento de imágenes y detección de códigos ArUco en Python.

4.3.1 Primeros pasos e instalación del sistema operativo

Es importante destacar que el dispositivo utilizado para este TFM no es un Jetson Nano estándar, sino un reComputer J3011. El Jetson Nano es un módulo de procesamiento de IA de NVIDIA, diseñado para proporcionar capacidades de computación avanzadas en un formato compacto y accesible, ver Figura 4.1a. Por otro lado, el reComputer J3011 es una solución integrada que incluye el módulo Jetson Nano, pero ofrece un diseño más robusto y modular, facilitando la expansión y personalización con diversos periféricos y accesorios, ver Figura 4.1b. Ambos dispositivos están estrechamente relacionados, ya que el reComputer J3011 se basa en el Jetson Nano, aprovechando su capacidad de procesamiento y su arquitectura de inteligencia artificial. Sin embargo, el reComputer J3011 mejora la experiencia del usuario al proporcionar un entorno más versátil y preparado para aplicaciones de desarrollo y pruebas en proyectos más complejos.

(a) Jetson Nano Developer Kit [69].



(b) reComputer J3011 [70].

Figura 4.1: Jetson Nano Developer Kit vs. reComputer J3011.

El primer paso consiste en desempaquetar el reComputer J3011. Al abrir la caja, se encuentran varios componentes: el reComputer J3011, un transformador con un conector *Barrel Jack* que se acopla al dispositivo, un par de antenas para mejorar el rango de la conexión inalámbrica WiFi, y una serie de instrucciones y folletos informativos. Es importante destacar que la caja no incluye el cable de alimentación que conecta el transformador a una toma de corriente. Por lo tanto, el primer paso para encender y configurar el reComputer J3011 es adquirir un cable de alimentación de 3 pines, ver Figura 4.2, que permita enchufarlo a la red eléctrica.

Figura 4.2: Cable de 3 pines necesario para alimentar el reComputer J3011 [71].

Antes de continuar, es importante mencionar que el reComputer J3011 viene preinstalado con JetPack 5.1.1. A diferencia del Jetson Nano Developer Kit, que requiere que el usuario descargue y escriba una imagen de sistema en una tarjeta microSD para poder iniciar el dispositivo, el reComputer J3011 facilita el proceso de configuración inicial al venir ya preparado con el software necesario. Esto simplifica considerablemente la configuración del entorno de desarrollo, haciéndolo más accesible y rápido de utilizar. El Jetson Nano Developer Kit, por otro lado, necesita que el usuario realice varios pasos antes de estar listo para su uso. Esto incluye descargar la imagen de JetPack, escribirla en una microSD, e insertar la tarjeta en el dispositivo para su arranque inicial, ver [72, 73]. Este proceso puede ser más laborioso y propenso a errores, especialmente para aquellos con menos experiencia técnica. Esta característica fue precisamente una de las razones por las que se optó por la adquisición del reComputer J3011 para el presente TFM, ya que ofrece la ventaja de estar listo para usar y ser más accesible e intuitivo para usuarios novatos.

JetPack 5.1.1 es un conjunto integral de herramientas y bibliotecas desarrollado por NVIDIA para sus plataformas de computación embebida, como el Jetson Nano. Este paquete incluye un sistema operativo basado en Linux, conocido como Ubuntu 20.04 LTS, que ha sido específicamente optimizado y adaptado para trabajar con las bibliotecas y herramientas incluidas en JetPack. Este sistema operativo modificado es conocido como Linux for Tegra (L4T), diseñado para maximizar el rendimiento del hardware de NVIDIA.

JetPack 5.1.1 proporciona una gama de bibliotecas y herramientas esenciales para el desarrollo de aplicaciones de inteligencia artificial, visión por computadora y procesamiento multimedia. Entre estas herramientas se encuentran CUDA para la computación paralela, cuDNN para la aceleración de redes neuronales, TensorRT para la optimización de inferencias de IA, y OpenCV para el procesamiento de imágenes. Es-

tas bibliotecas permiten a los desarrolladores crear y optimizar aplicaciones avanzadas de IA y visión por computadora con mayor facilidad y eficiencia.

El reComputer J3011, utilizando JetPack 5.1.1, viene preinstalado con esta versión optimizada de Ubuntu 20.04 LTS. Esta versión adaptada está específicamente configurada para trabajar en armonía con el hardware de NVIDIA, aprovechando al máximo las capacidades de las bibliotecas y herramientas proporcionadas por JetPack. Esto simplifica el proceso de configuración inicial y facilita el desarrollo de proyectos, proporcionando un entorno listo para usar desde el primer momento en que se enciende el dispositivo.

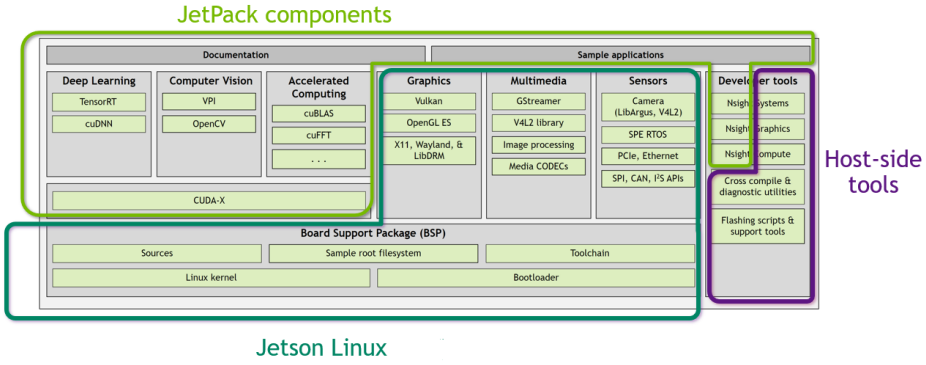


Figura 4.3: Jetson Software Architecture [74].

Una vez descrito el contenido de la caja del reComputer J3011 y el entorno de trabajo a nivel de *software*, el siguiente y primer paso es encender el dispositivo conectándolo a una toma de corriente. Aunque el reComputer J3011 viene preinstalado con el sistema operativo optimizado mencionado anteriormente, es necesario realizar una configuración inicial al encenderlo por primera vez. Durante este proceso, se deberán establecer parámetros básicos como la región horaria, el nombre y la contraseña del dispositivo, y la conexión a una red inalámbrica. Estas configuraciones iniciales son sencillas y pueden ser completadas sin necesidad de conocimientos avanzados.

Una vez completada la configuración inicial, el dispositivo está listo para su uso. Como se mencionó anteriormente, el sistema operativo del reComputer J3011 es Linux, específicamente una distribución de Ubuntu. Ubuntu es una de las distribuciones de Linux más accesibles y es comparada con sistemas operativos como macOS y Windows en términos de interfaz gráfica y facilidad de uso. Sin embargo, al tratarse de Linux, la filosofía de utilización del sistema cambia significativamente.

En Linux, muchas operaciones, como la instalación de programas, la gestión de paquetes y la actualización del sistema, se realizan principalmente a través de la terminal de comandos. Este enfoque puede resultar novedoso y algo complejo para usuarios

acostumbrados a entornos gráficos predominantes en Windows y macOS. En estos sistemas, las operaciones suelen llevarse a cabo mediante interfaces gráficas y asistentes visuales, mientras que en Linux se valora la flexibilidad y el control que ofrece la línea de comandos.

Este cambio de paradigma puede requerir un periodo de adaptación para los nuevos usuarios. Por ejemplo, en lugar de descargar un instalador y hacer clic en varios botones, en Linux se puede instalar software mediante comandos específicos como **«sudo apt-get install [nombre_del_paquete]»**. Aunque al principio este método puede parecer intimidante, una vez familiarizados, muchos usuarios aprecian la eficiencia y el poder que ofrece el control directo sobre el sistema. Este enfoque más técnico y directo es uno de los aspectos que diferencian a Linux y que, a pesar de su curva de aprendizaje, ofrece grandes beneficios en términos de personalización y flujo de trabajo en el desarrollo de aplicaciones.

4.3.2 Control remoto del reComputer J3011

Antes de pasar a la configuración del entorno de trabajo que permitirá el desarrollo de la aplicación en Python, es interesante plantearse la siguiente cuestión: ¿Cómo se utiliza el reComputer J3011 una vez embarcado en el dron? En tierra, esta pregunta carece de sentido, ya que la respuesta es obvia: se puede utilizar a través de un teclado y un ratón conectados a uno de los cuatro puertos USB del reComputer J3011 y mediante un monitor externo conectado por HDMI. En este contexto, el funcionamiento del reComputer es trivial, similar al de un ordenador de sobremesa. Sin embargo, una vez embarcado, se pierde la capacidad de conectar un teclado, ratón y monitor externo.

Existen dos posibles vías para solucionar este problema. La primera consiste en utilizar un programa que permita el control remoto del reComputer J3011, como VNC o TeamViewer. Estas aplicaciones proporcionan una interfaz gráfica a distancia, facilitando la gestión del dispositivo como si se estuviera operando directamente en el mismo.

La segunda opción es conectarse vía SSH (*Secure Shell*), un protocolo diseñado para operar en red de forma segura. El reComputer J3011 está pensado para ser manejado principalmente a través de SSH. Esta opción no proporciona una interfaz gráfica, lo que significa que todas las operaciones se realizan mediante la línea de comandos. Aunque puede resultar menos intuitivo para algunos usuarios, ofrece un control robusto y eficiente del dispositivo.

Antes de continuar, es útil definir brevemente qué son VNC y SSH:

- **VNC (*Virtual Network Computing*)**: Es una tecnología que permite visualizar y controlar un ordenador de forma remota utilizando una conexión de red. Proporciona una interfaz gráfica que replica la pantalla del dispositivo controlado, permitiendo interactuar con él como si se estuviera físicamente presente.

- **SSH (Secure Shell)**: Es un protocolo de red que permite a los usuarios acceder y administrar un sistema remoto de manera segura. SSH se basa en la línea de comandos, lo que significa que las interacciones se realizan mediante texto, proporcionando un método eficiente y seguro para la gestión remota de sistemas.

Como se mencionó anteriormente, una vez embarcado en el dron, el reComputer J3011 está diseñado para ser utilizado mediante conexión SSH (*Secure Shell*). Esta opción es preferible y más robusta en comparación con aplicaciones con interfaces gráficas como VNC. SSH permite un control remoto seguro y eficiente a través de la línea de comandos, evitando los problemas de latencia y cuello de botella que pueden surgir con VNC.

Las aplicaciones como VNC, al incluir una interfaz gráfica y depender de una conexión a Internet, pueden presentar un control remoto complejo debido al retraso (lag) en la transmisión de datos. Esto es especialmente problemático cuando en el dispositivo se ejecuta alguna aplicación de procesamiento de imágenes o algún programa con alto contenido gráfico, ya que estos requieren un considerable ancho de banda y capacidad de procesamiento. Utilizar VNC para controlar estas aplicaciones puede resultar ineficaz, ya que la interfaz gráfica consume recursos significativos y la transmisión de datos puede volverse un cuello de botella, dificultando el control efectivo y casi imposible de dichas aplicaciones.

En el presente TFM, sin embargo, se ha optado por utilizar VNC en lugar de SSH. Esta decisión se fundamenta en que las conexiones vía SSH requieren conocimientos técnicos avanzados, los cuales excedían los límites temporales y didácticos de este proyecto. A pesar de los problemas de lag mencionados, VNC ofrece una opción más intuitiva y fácil de usar que SSH. Para una aproximación inicial a la utilización del dispositivo reComputer J3011, VNC resulta más accesible y adecuado, permitiendo un control remoto básico que es suficiente para los propósitos de este TFM.

Otro aspecto a considerar que facilita la instalación y uso de VNC es que los dispositivos de NVIDIA, respaldados por JetPack, están diseñados y preparados para el uso de manera remota vía VNC. De manera nativa, todos estos dispositivos incluyen una serie de archivos de configuración en formato .txt, que se encuentran en una carpeta ubicada, por defecto, en el escritorio del usuario. Estos archivos contienen configuraciones y guías que ayudan a los usuarios a instalar y configurar ciertos programas, como VNC. En esta carpeta, hay un archivo llamado **«README-vnc.txt»** que proporciona instrucciones detalladas, paso a paso, para configurar el reComputer J3011, de manera que pueda ser controlado remotamente mediante VNC.

Dada la existencia del archivo **«README-vnc.txt»**, se omite en esta memoria la explicación detallada sobre cómo configurar VNC en el reComputer J3011. Por lo tanto, se recomienda a cualquier lector interesado en esta información que consulte y siga las instrucciones proporcionadas en el mencionado archivo.

4.3.3 Instalación de los IDEs necesarios

El siguiente paso en el proceso de configuración y puesta a punto del reComputer J3011 consiste en la instalación de los entornos de desarrollo integrado (IDE) que se utilizarán. Como se mencionó en la Subsección 3.2.2, en el presente TFM se han utilizado tres IDE distintos. Sin embargo, en el reComputer J3011 solo se instalarán dos de ellos: Visual Studio Code (VSCode) para el desarrollo de la aplicación de procesamiento de imágenes en Python y Netbeans para el desarrollo en Java.

En sistemas Linux, la instalación de software y paquetes generalmente se realiza a través de la terminal integrada del sistema operativo. Los comandos necesarios para instalar estos dos IDEs son:

- **Para instalar VSCode: «sudo apt update»** y **«sudo apt install code»**
- **Para instalar Netbeans: «sudo apt update»** y **«sudo apt install netbeans»**

No obstante, para aquellos usuarios que prefieren no utilizar la línea de comandos, las páginas web oficiales de ambos IDE ofrecen archivos ejecutables y las correspondientes instrucciones de instalación, lo que permite un proceso más tradicional y guiado.

4.3.4 Instalación de la cámara Raspberry Pi NoIR Camera V2

La cámara Raspberry Pi NoIR Camera V2 no requiere ninguna instalación de software específica en el reComputer J3011. Simplemente se conecta a uno de los dos puertos CSI del reComputer, y estará lista para su uso, ver Figura 4.4. Para verificar si el reComputer ha reconocido correctamente la cámara, se puede utilizar el siguiente comando en la terminal, **«ls -la /dev/»**. Si la cámara ha sido detectada correctamente por el sistema, el comando debería mostrar una salida similar a «video0».

El siguiente paso, una vez confirmada la correcta detección de la cámara por parte del sistema, es verificar su adecuado funcionamiento. Es importante destacar que el manejo de cámaras tipo CSI (Camera Serial Interface) difiere del de una webcam convencional. Mientras que una webcam puede ser probada utilizando programas de captura de imágenes como Cheese, las cámaras CSI no son compatibles con este tipo de software de prueba. En su lugar, se deben emplear herramientas específicas para interactuar con dispositivos CSI, como las librerías «picamera» para Python o los comandos de «libcamera».

Durante el proceso de investigación enfocado en la configuración y puesta a punto de la cámara CSI, se encontró un repositorio en GitHub, ver [75], con una amplia documentación sobre la configuración y testeo de cámaras CSI. El propietario del repositorio, además, resultó ser un usuario de un dispositivo Jetson Nano, el mismo que el utilizado en el presente TFM. Además de la documentación, el repositorio incluye varios *scripts* en Python que permiten realizar procesamientos sencillos con cámaras

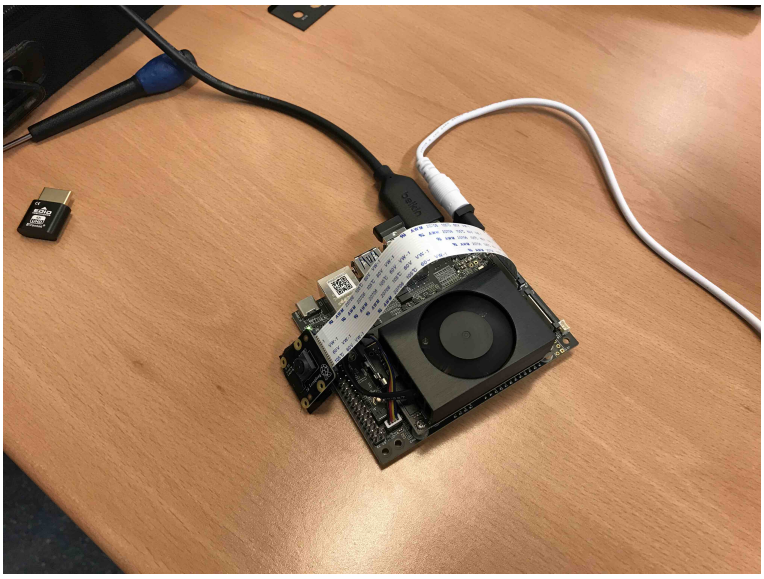


Figura 4.4: Conexión de la cámara Raspberry Pi NoIR Camera V2 al reComputer J3011 [Elaboración Propia].

CSI, como la apertura de la cámara, la captura de imágenes y el reconocimiento facial. Adicionalmente, el autor del repositorio cuenta con una página web y un canal de YouTube, cuyos enlaces se adjuntan a continuación, [76] y [77]. Todos estos recursos han sido de gran inspiración y ayuda para la realización de este Trabajo de Fin de Máster.

Según lo indicado en [75], para verificar el correcto funcionamiento de la cámara Raspberry Pi NoIR Camera V2, solamente es necesario abrir la terminal del sistema e introducir el comando Código 4.5.

```
$ gst-launch-1.0 nvarguscamerasrc sensor_id=0 ! \
   'video/x-raw(memory:NVMM),width=1920, height=1080, framerate=30/1'
       ! \
   nvvidconv flip-method=0 ! 'video/x-raw,width=960, height=540' ! \
   nvvidconv ! nvegltransform ! nveglglessink -e
```

Código 4.5: Comando para testear la cámara Raspberry Pi NoIR Camera V2.

Si todo funciona correctamente, al ejecutar el comando, se abrirá automáticamente una interfaz gráfica que mostrará los fotogramas capturados por la cámara.

4.3.5 Compilación de OpenCV desde el código fuente

Como se mencionó anteriormente, para desarrollar la aplicación de procesamiento de imágenes en Python se utilizará la biblioteca de visión por computadora OpenCV. Es importante destacar que el reComputer J3011, gracias al entorno Jetpack, ya viene con OpenCV preinstalado. Además, esta versión de OpenCV está precompilada con todos los elementos y funcionalidades necesarias para el desarrollo de aplicaciones en este tipo de dispositivos.

En contraste, si un usuario desea utilizar OpenCV en su ordenador personal, primero debe descargar una versión precompilada de la biblioteca, lo cual se puede hacer utilizando el siguiente comando, **«pip install opencv-python»**. El problema es que estas versiones precompiladas están generalmente diseñadas para aplicaciones más comunes y no para tareas avanzadas de inteligencia artificial o procesamiento de imágenes. Por lo tanto, si el usuario necesita todas las funcionalidades avanzadas, tendría que compilar OpenCV desde su código fuente.

Dado que el reComputer J3011 está orientado al desarrollo de aplicaciones avanzadas de procesamiento de imágenes, la versión de OpenCV incluida con el sistema ya está precompilada con todas las funcionalidades avanzadas. Por ello, el usuario no necesita preocuparse por realizar esta compilación adicional desde el código fuente.

En el desarrollo de aplicaciones, es muy común trabajar con entornos virtuales. Un entorno virtual es una herramienta que permite crear un espacio aislado dentro del sistema operativo, en el cual se pueden instalar paquetes y dependencias específicas sin afectar el resto del sistema. Esta práctica ofrece varias ventajas, como la capacidad de gestionar diferentes versiones de paquetes para distintos proyectos, evitando conflictos de dependencias. Además, facilita la reproducción del entorno de desarrollo en otros sistemas, lo cual es crucial para la colaboración y la replicación de resultados en proyectos científicos, como el presente TFM.

Por las razones mencionadas anteriormente, especialmente para evitar conflictos de dependencias con los paquetes ya instalados en el reComputer J3011, en el presente TFM se ha optado por desarrollar la aplicación de procesamiento de imágenes dentro de un entorno virtual. Sin embargo, esta decisión implica el desafío de utilizar la biblioteca OpenCV con todas las funcionalidades necesarias para el diseño de la aplicación. Las versiones precompiladas de OpenCV que se pueden descargar mediante **«pip install opencv-python»** no incluyen estas funcionalidades avanzadas. Debido a la necesidad de evitar conflictos de dependencias, no se puede utilizar la versión de OpenCV que viene preinstalada en el reComputer J3011 dentro del entorno virtual. Por lo tanto, es necesario compilar OpenCV desde el código fuente para asegurar que todas las funcionalidades requeridas estén disponibles en el entorno virtual.

A continuación se detallan los pasos a seguir para, en primer lugar, crear un entorno virtual en el reComputer J3011 y, en segundo lugar, compilar OpenCV desde el código

fuente dentro de ese entorno virtual, con soporte para todas las funcionalidades avanzadas necesarias. Es importante destacar que la guía que se presenta a continuación es descriptiva y debe ser utilizada más como una orientación que como una instrucción definitiva. El software libre está en constante cambio y evolución, por lo que es posible que los pasos descritos funcionen en el momento de redacción de este documento, pero puedan requerir ajustes en el futuro.

Para crear y activar el entorno virtual, se puede utilizar la terminal integrada que viene instalada por defecto en el IDE Visual Studio Code. Los pasos a seguir son los siguientes:

1. **Abrir la terminal en Visual Studio Code:** En la barra de menú, seleccionar «Terminal» y luego «New Terminal».
2. **Actualizar el sistema e instalar virtualenv: «sudo apt update» $\Longrightarrow$ «sudo apt install python3-venv»**.
3. **Crear un entorno virtual:**
   - Navegar al directorio donde se desea crear el entorno virtual.
   - Ejecutar el siguiente comando para crear el entorno virtual: **«python3 -m venv nombre_del_entorno»**.
4. **Activar el entorno virtual:**
   - Ejecutar el siguiente comando: **«source nombre_del_entorno/bin/activate»**.
5. **Instalar las dependencias necesarias para compilar OpenCV:**

```
sudo apt install build-essential cmake git libgtk-3-dev
    libavcodec-dev libavformat-dev libswscale-dev
sudo apt install python3-dev python3-numpy libtbb2 libtbb-dev
    libjpeg-dev libpng-dev libtiff-dev
sudo apt install libdc1394-22-dev
```

Código 4.6: Instalación de las dependencias necesarias para compilar OpenCV.

6. **Descargar el código fuente de OpenCV:**
   - Clonar el repositorio de OpenCV y OpenCV contrib:

```
git clone https://github.com/opencv/opencv.git
git clone https://github.com/opencv/opencv_contrib.git
```

Código 4.7: Clonación del repositorio de OpenCV y OpenCV contrib.

7. **Compilar OpenCV con las funcionalidades avanzadas:**

   - Navegar al directorio de OpenCV y crear un directorio de compilación:

```
cd opencv
mkdir build
cd build
```

Código 4.8: Creación de un directorio para la compilación de OpenCV.

   - Configurar la compilación con CMake[10]:

```
sudo chown -R $USER:$USER /home/omar/Escritorio/TFM/ArUco

python --version

PYTHON3_EXECUTABLE=$(which python)

PYTHON3_INCLUDE_DIR=$(python -c "from distutils.sysconfig
    import get_python_inc; print(get_python_inc())")

PYTHON3_PACKAGES_PATH=$(python -c "import site; print(site.
    getsitepackages()[0])")

cmake -D CMAKE_BUILD_TYPE=RELEASE \
-D CMAKE_INSTALL_PREFIX=/home/omar/Escritorio/TFM/ArUco/
    VirtualEnvironments/EntVirtP310 \
-D INSTALL_C_EXAMPLES=OFF \
-D INSTALL_PYTHON_EXAMPLES=ON \
-D OPENCV_GENERATE_PKGCONFIG=ON \
-D OPENCV_EXTRA_MODULES_PATH=/home/omar/Escritorio/TFM/ArUco/
    OpenCV/OpenCV-V4_8/opencv_contrib/modules \
-D BUILD_EXAMPLES=ON \
-D WITH_OPENGL=ON \
-D WITH_V4L=ON \
-D WITH_FFMPEG=ON \
-D WITH_QT=ON \
-D WITH_GTK=ON \
-D WITH_COCOA=ON \
-D WITH_GSTREAMER=ON \
-D BUILD_opencv_python3=TRUE \
-D HAVE_opencv_python3=ON \
-D PYTHON3_EXECUTABLE=$PYTHON3_EXECUTABLE \
-D PYTHON3_INCLUDE_DIR=$PYTHON3_INCLUDE_DIR \
-D PYTHON3_LIBRARY=/usr/local/lib/libpython3.10.so \
-D PYTHON3_PACKAGES_PATH=$PYTHON3_PACKAGES_PATH \
..
```

Código 4.9: Configuración del tipo de compilación de OpenCV deseada con CMake.

[10]CMake es una herramienta de código abierto diseñada para gestionar el proceso de compilación de software de manera eficiente y multiplataforma. Proporciona un mecanismo sencillo para definir configuraciones de compilación a través de archivos de *script*, conocidos como CMakeLists.txt, que especifican cómo se deben configurar, compilar y vincular los componentes del proyecto.

- Compilar e instalar OpenCV:

```
make -j4
sudo make install
sudo ldconfig
```

Código 4.10: Compilación e instalación de OpenCV.

De todos los pasos descritos a continuación, quizás el más crítico e importante sea la configuración de CMake. Este paso es primordial porque las opciones especificadas en CMake determinarán cómo se compilará OpenCV, es decir, qué librerías y funcionalidades estarán soportadas en la compilación final. La configuración adecuada de CMake define las capacidades de OpenCV, asegurando que incluya el soporte necesario para las diversas librerías y características avanzadas requeridas por la aplicación. Por lo tanto, una configuración incorrecta o incompleta en esta etapa puede resultar en una compilación de OpenCV que no cumpla con los requisitos del proyecto.

El Código 4.9 corresponde a la configuración de CMake utilizada en este Trabajo de Fin de Máster para satisfacer los requisitos específicos y objetivos del proyecto. Es importante destacar que algunas de las rutas presentes en este código deberán ser modificadas y adaptadas según las necesidades de cada usuario. Una vez más, el objetivo de proporcionar esta guía no es ofrecer una solución definitiva, sino servir como una referencia útil para los lectores y futuros colaboradores del proyecto.

Una vez que el entorno virtual ha sido debidamente creado y activado, y que se ha completado la compilación de OpenCV con todas las funcionalidades necesarias para el diseño de la aplicación, el entorno de desarrollo queda configurado. En este punto, se puede proceder al siguiente bloque, que aborda el desarrollo de la aplicación propiamente dicha.

4.4 Desarrollo de la aplicación de procesamiento de imágenes con OpenCV y Python

En la presente sección se explicará todo lo relacionado con la aplicación de procesamiento de imágenes desarrollada en Python, cuyo objetivo final es ser ejecutada en el ordenador reComputer J3011, que será embarcado en el Phantom 4 Advanced +. La estructuración de esta sección ha sido particularmente compleja y laboriosa, dado que, al momento de redactar esta memoria, la aplicación ya ha sido desarrollada y funciona como un todo. Sin embargo, detrás de este desarrollo hay numerosos aspectos, matices y dificultades que se han tenido que superar para alcanzar el objetivo final. Poner por escrito todo este proceso supone un desafío en sí mismo, ya que durante el desarrollo de la aplicación, debido a diversos problemas que surgieron, se tuvieron que redefinir ciertos objetivos y, en ocasiones, la filosofía inicial de la aplicación cambió conforme avanzaba el proyecto.

A pesar de estos retos, se ha procurado que la redacción de esta sección sea lo más didáctica y fluida posible, facilitando al lector no solo la comprensión de lo que hace y cómo se ejecuta la aplicación, sino también ofreciendo una visión de otros aspectos muchas veces ocultos en el desarrollo de este tipo de proyectos, como la motivación detrás de su desarrollo, la filosofía de programación empleada y las razones detrás de muchas de las decisiones y alternativas adoptadas.

4.4.1 Punto de partida y fundamento de la aplicación

En primer lugar, es importante recordar que la razón para embarcar un ordenador (reComputer J3011) en un dron (Phantom 4 Advanced+) es solucionar el problema de la transferencia de imágenes vía radiofrecuencia, que impedía su utilización en procesamientos posteriores. En la configuración inicial de este proyecto, el dron captura imágenes a través de su sistema de visión. Si un usuario desea utilizar estas imágenes para un procesamiento posterior, deben ser transmitidas vía radiofrecuencia al mando. Desde el mando, las imágenes se transfieren mediante un cable USB a una aplicación móvil desarrollada por el usuario utilizando el SDK de DJI.

El problema en este proceso radica en que la transferencia de imágenes por radiofrecuencia resulta en imágenes de baja calidad y con cierto retraso, lo que imposibilita su procesamiento en tiempo real. Para resolver este problema, se concibió la idea de un ordenador embarcado. Este ordenador, montado directamente en el dron, se encarga de capturar y procesar las imágenes sin necesidad de transferirlas a la aplicación móvil del usuario, eliminando así los problemas de calidad y latencia.

Cabe destacar que, para establecer una conexión entre el ordenador embarcado y la aplicación móvil, fue necesario convertir la aplicación móvil en un servidor. De este modo, mediante un sencillo esquema cliente-servidor, donde el reComputer J3011 actúa como cliente y la aplicación móvil como servidor, se resolvió el problema de comunicaciones.

En este punto, se tiene una aplicación móvil que, al actuar como servidor, puede recibir comandos vía WiFi y, según el tipo de comando recibido, ordenar al dron ejecutar una acción específica. De manera más didáctica, anteriormente se disponía de una aplicación móvil con una interfaz gráfica que incluía una serie de botones. Por ejemplo, había un botón de «*Take-Off*» que, al ser presionado por el usuario, enviaba al dron la orden de despegue a través del mando, que actuaba como puente.

Ahora, con la aplicación convertida en un servidor, si el usuario quiere enviar la orden de despegue al dron, en lugar de presionar un botón, envía una cadena de texto predefinida a este servidor. Cuando la aplicación recibe esta cadena de texto, la reconoce y envía la orden de despegue al dron a través del mando. La filosofía sigue siendo similar: las cadenas de texto que el usuario debe enviar ahora estaban previamente ocultas bajo la lógica de los botones de la interfaz gráfica. Es decir, antes, al presionar un

botón, se enviaba una cadena de texto interna; ahora, esa cadena de texto se envía directamente.

Entendida la idea anterior, resulta evidente que para controlar el dron es necesario enviar al servidor una serie de cadenas de texto. El fundamento de la aplicación de procesamiento de imágenes que se pretende diseñar es sencillo: ante el reconocimiento de un objeto, código, persona u otro elemento de interés, la aplicación enviará las cadenas de texto correspondientes al servidor, lo que permitirá comandar el dron.

Para ilustrar esta idea de manera más didáctica, consideremos algunos ejemplos. Se podría diseñar una aplicación que reconozca a una persona específica, de tal manera que, cuando el sistema detecte a esa persona en la imagen, envíe al servidor la cadena de texto que activa la orden de despegue del dron. Así, siempre que se quiera despegar el dron, bastaría con que esa persona se sitúe dentro del campo de visión del dron (concretamente de la cámara asociada al ordenador embarcado, que realiza todo el procesamiento).

Otro ejemplo podría ser una aplicación que, al detectar un coche rojo, envíe al servidor una cadena de texto que indique a la aplicación móvil que el dron debe girar a la derecha. De esta manera, cada vez que un coche rojo entre en el campo de visión del dron, este girará a la derecha.

Estas explicaciones son generalizadas y tienen como objetivo introducir al lector en el ecosistema de trabajo, proporcionando un contexto para comprender cómo se integran el procesamiento de imágenes y el control del dron.

En el contexto del presente TFM, tal y como se mencionó en la introducción, se ha decidido centrar la tarea del procesamiento de imágenes en el reconocimiento de códigos ArUco. El objetivo último, alineado con el estado del arte actual, es lograr el control automático del dron mediante estos códigos. Esto permitiría, por ejemplo, disponer de una serie de códigos distribuidos en un área para que el dron navegue entre ellos como si fueran *waypoints*, o bien tener un único código ArUco que, al ser detectado durante una exploración del terreno, permita al dron posicionarse y efectuar un aterrizaje en ese punto.

Como se ha demostrado, existen múltiples aplicaciones relacionadas con el reconocimiento de códigos ArUco y el control de drones. En el presente TFM, la aplicación diseñada permite dos modalidades de funcionamiento, es decir, dos aplicaciones distintas integradas en una sola, donde el comportamiento del dron varía según la modalidad seleccionada. A continuación, se explican estas dos modalidades de funcionamiento de manera didáctica. En esta parte de la memoria, se describirá únicamente el tipo de procesamiento de imágenes realizado y la respuesta del dron ante dicho procesamiento. En secciones posteriores, se proporcionarán explicaciones más detalladas sobre cómo se han programado estas aplicaciones.

Antes de continuar, es importante mencionar que en este TFM se han utilizado tres códigos ArUco diferentes: uno para el despegue, ver Figura 4.5a, otro para el aterrizaje, ver Figura 4.5b, y un tercero para el control del dron, ver Figura 4.5c.

Las acciones que ejecuta el dron ante los códigos ArUco de despegue y aterrizaje son sencillas y directas. La cámara conectada al reComputer J3011 captura continuamente fotogramas del entorno. Cuando, por ejemplo, en uno de estos fotogramas aparece un código ArUco de despegue, automáticamente se envía al servidor (la aplicación móvil) una cadena de texto correspondiente a la orden de despegue. La aplicación móvil recibe esta cadena de texto y, a través del mando que actúa como puente, envía la orden de despegue al dron. El funcionamiento del código ArUco de aterrizaje es similar; la única diferencia es que la cadena de texto enviada al servidor corresponde a la orden de aterrizaje, lo que hace que la aplicación móvil ordene al dron que aterrice.

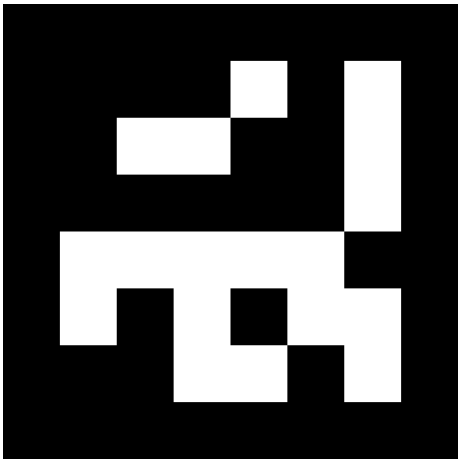
El funcionamiento de la aplicación de procesamiento de imágenes y el comportamiento del dron ante estos dos códigos ArUco se esquematiza en la Figura 4.6.

El procesamiento y los comandos enviados por el programa al servidor ante la detección del código ArUco de control son un tanto diferentes y requieren de una explicación más detallada, la cual se proporciona a continuación. En primer lugar, es importante tener en cuenta que los fotogramas capturados por la cámara tienen una resolución de 640x480 píxeles. Aunque la cámara puede capturar fotogramas a una resolución mucho mayor, se ha optado por esta resolución más baja para agilizar el funcionamiento del programa y el procesamiento de los fotogramas por parte de los algoritmos de OpenCV. Si los fotogramas se capturan a una mayor resolución, los algoritmos de OpenCV encuentran más difícil procesar las imágenes, lo que provoca un retraso en el funcionamiento del programa.

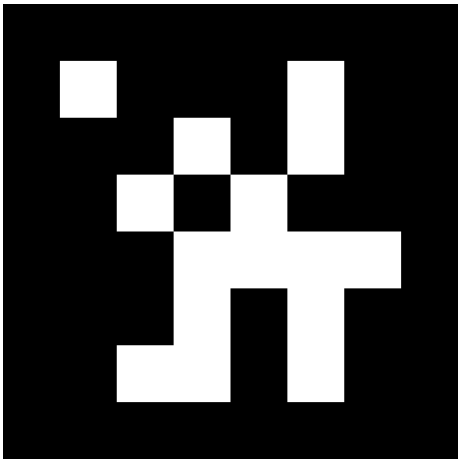
Sabiendo que la resolución de los fotogramas es de 640x480 píxeles, es sencillo calcular el centro de la imagen. Utilizando un sistema de coordenadas x e y situado en la esquina superior izquierda, con el eje x apuntando hacia la derecha y el eje y apuntando hacia abajo, el centro de la imagen se encuentra en las coordenadas (320, 240) píxeles.

OpenCV, a través del módulo «`cv2.aruco`», proporciona una serie de funciones que permiten el reconocimiento de códigos ArUco y el cálculo de las coordenadas de las esquinas, en píxeles, de los códigos ArUco detectados en un fotograma. Utilizando estas funciones, una vez que el sistema detecta la presencia de un código ArUco en un fotograma, se pueden calcular las coordenadas x e y de los cuatro vértices que conforman las esquinas del código ArUco. A partir de las coordenadas de los vértices de las esquinas, se pueden realizar operaciones matemáticas sencillas para calcular las coordenadas (x, y) del centro, en píxeles, del marcador ArUco detectado.

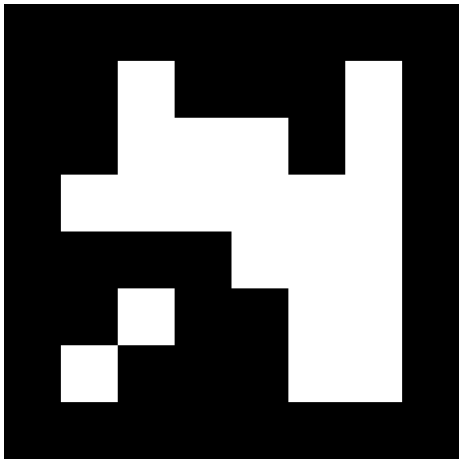
Finalmente, conociendo tanto el centro de la imagen como el centro del marcador ArUco, se puede determinar cuánto se desvía el centro del marcador ArUco del centro de la imagen y, por lo tanto, cuánto error se está cometiendo.



(a) Código ArUco TAKE-OFF_DICT_6X6_250-Marker_id_2 [Elaboración Propia].



(b) Código ArUco LANDING_DICT_6X6_250-Marker_id_7 [Elaboración Propia].



(c) Código ArUco CONTROL_DICT_6X6_250-Marker_id_10 [Elaboración Propia].

Figura 4.5: Códigos ArUco empleados en el TFM.

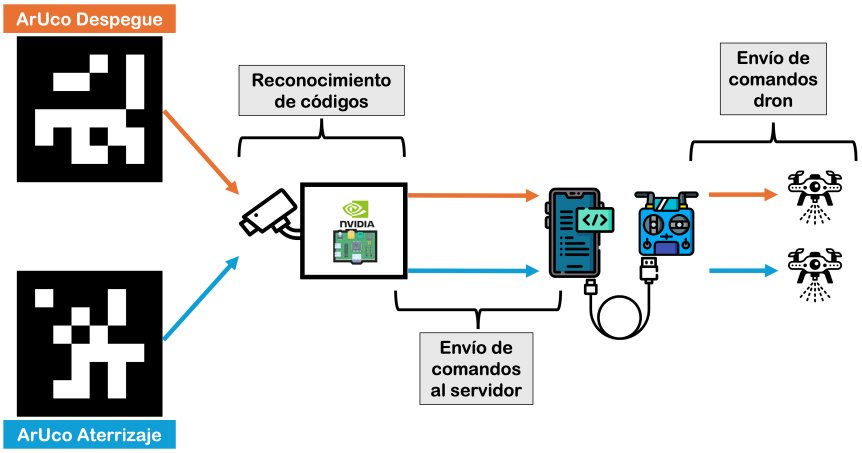


Figura 4.6: Esquema conceptual del funcionamiento de los códigos ArUco de despegue y aterrizaje [Elaboración Propia].

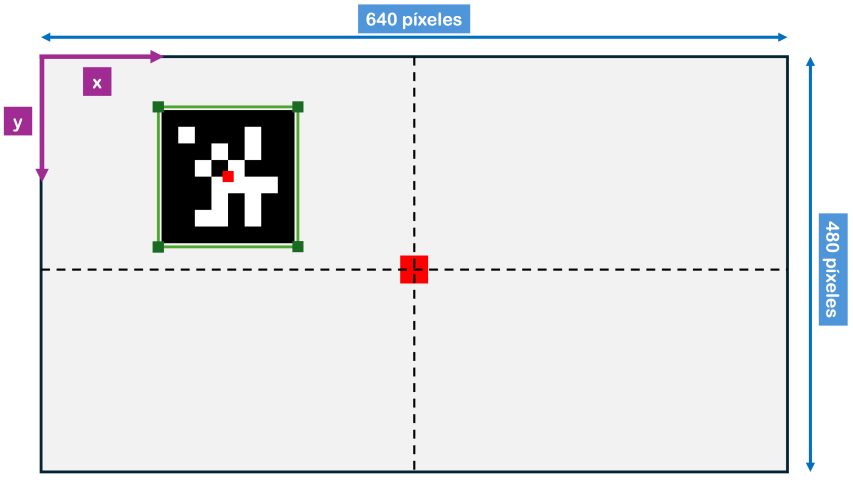


Figura 4.7: Relación entre los centros del *frame* y del código ArUco detectado [Elaboración Propia].

En el presente TFM se han considerado cuatro tipos de errores distintos: `errorPitch`, `errorRoll`, `errorYaw` y `errorThrottle`. A continuación, se explica cada uno de estos errores en detalle.

Para evaluar los errores de *pitch* y *roll*, el fotograma capturado se divide en cuatro cuadrantes, numerados en el sentido de las agujas del reloj. Si el centro del marcador ArUco detectado se encuentra en el primer cuadrante (superior izquierdo), entonces el `errorPitch` >0 y el `errorRoll` <0. Si el centro del marcador ArUco está en el segundo cuadrante (superior derecho), el `errorPitch` >0 y el `errorRoll` >0. Si se encuentra en el tercer cuadrante (inferior derecho), el `errorPitch` <0 y el `errorRoll` >0. Finalmente, si está en el cuarto cuadrante (inferior izquierdo), el `errorPitch` <0 y el `errorRoll` <0. Estos errores indican la desviación del centro del marcador respecto al centro de la imagen, permitiendo ajustar la orientación del dron en los ejes de *pitch* y *roll*.

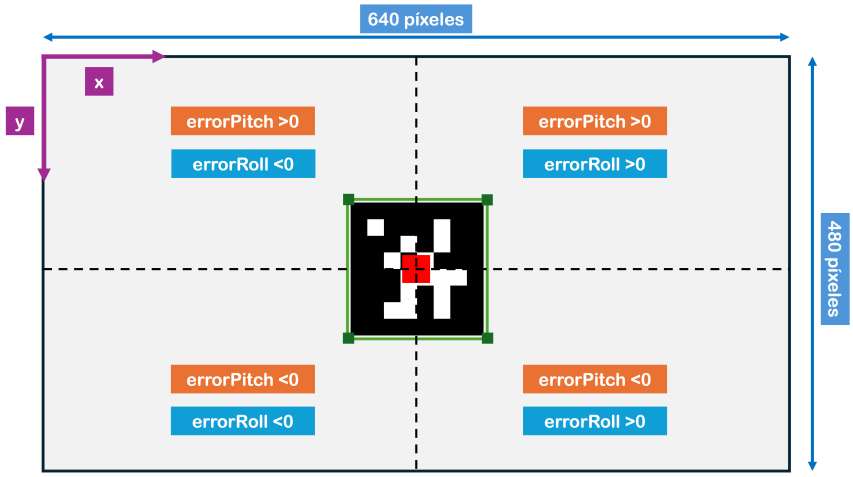


Figura 4.8: Esquema de los errores en *Pitch* y *Roll* [Elaboración Propia].

El errorYaw se calcula utilizando las funciones de OpenCV que proporcionan la rotación del código ArUco detectado. Estas funciones generan vectores de rotación que indican cómo está orientado el marcador en relación con la cámara. Utilizando estos vectores, se puede determinar la desviación angular (*yaw*) del marcador respecto a la posición deseada, ajustando así la orientación del dron en el eje de guiñada (*yaw*).

El `errorThrottle` se relaciona con la distancia entre la cámara y el marcador ArUco. Las funciones de OpenCV requieren conocer el tamaño físico del marcador, y con este dato, junto con otros parámetros que se detallarán posteriormente, pueden calcular la distancia del marcador a la cámara. Se establece una distancia de trabajo, por

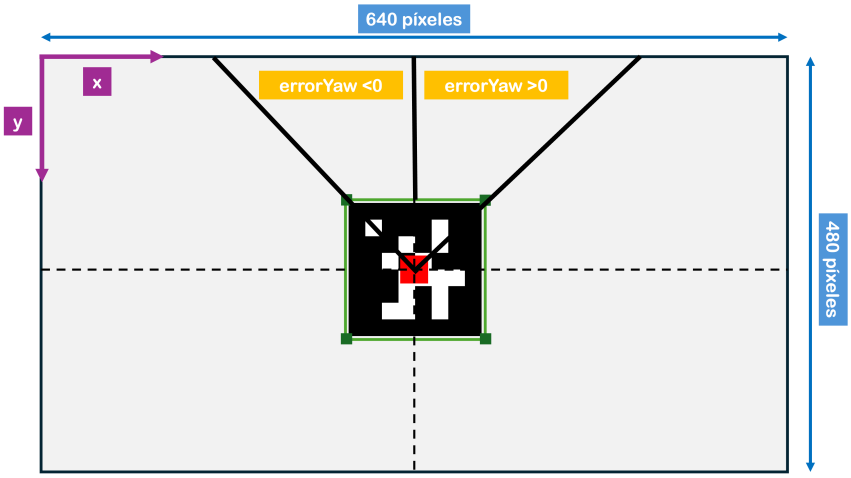


Figura 4.9: Esquema del error en *Yaw* [Elaboración Propia].

ejemplo, 30 cm. Si el marcador se mantiene a esta distancia, el `errorThrottle` es cero. Si la distancia se reduce a menos de 30 cm (el marcador se acerca a la cámara), el `errorThrottle` >0. Si la distancia aumenta a más de 30 cm (el marcador se aleja de la cámara), el `errorThrottle` <0. Este error permite ajustar la posición del dron en el eje vertical para mantener una distancia constante al marcador.

Es importante notar que el proceso de cálculo de los cuatro errores mencionados anteriormente se lleva a cabo en tiempo real para cada fotograma capturado, siempre y cuando el código ArUco de control esté presente en el fotograma. La lógica interna de la aplicación desarrollada funciona de manera que, en cada fotograma capturado donde se detecta el código ArUco de control, se calculan los cuatro errores (`errorPitch`, `errorRoll`, `errorYaw` y `errorThrottle`). Debido a la alta velocidad a la que la cámara captura los fotogramas, estos errores se calculan continuamente, permitiendo ajustes en tiempo real. Esto significa que, si la cámara detecta de manera continua el código ArUco de control, los errores se actualizan en cada fotograma, proporcionando una respuesta inmediata y precisa del dron.

La siguiente pregunta que surge es cómo, a partir del cálculo de estos cuatro errores, se puede controlar el dron. Aunque la respuesta no es trivial, es relativamente sencilla de entender. Anteriormente, la lógica interna de la aplicación era tal que, cuando en un fotograma capturado se detectaba la presencia de los códigos ArUco de despegue o aterrizaje, el cliente (reComputer J3011) enviaba automáticamente una cadena de texto al servidor (aplicación móvil), el cual la recibía y enviaba al dron la acción pertinente.

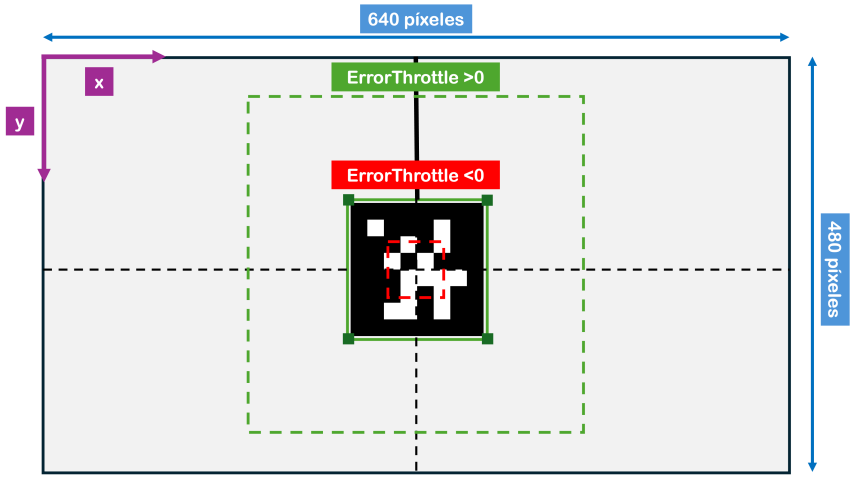


Figura 4.10: Esquema del error en *Throttle* [Elaboración Propia].

Ahora, la lógica interna de la aplicación ha evolucionado de modo que, si se detecta el código ArUco de control en un fotograma, el programa calcula los cuatro errores de la manera previamente explicada. Una vez calculados, y dependiendo de si cada uno de ellos es positivo, negativo o nulo, la aplicación envía una cadena de texto a la aplicación móvil para corregir individualmente cada uno de estos errores. Por ejemplo, si el `errorYaw` >0, la aplicación de procesamiento de imágenes (el cliente) envía una cadena de texto a la aplicación móvil para que ordene al dron girar a la derecha y compensar este error. Si el `errorPitch` >0, la aplicación (el reComputer J3011) envía al servidor una cadena de texto para que el dron avance y corrija ese error.

Es importante destacar que la corrección de los errores, tal como está implementada en la aplicación móvil, se realiza de manera individual. Esto significa que, cada vez que se envía una cadena de texto al servidor, este ordena al dron ejecutar la acción correspondiente, interrumpiendo la acción que se estaba ejecutando anteriormente. Este hecho ha condicionado el diseño de los dos tipos de aplicaciones de procesamiento de imágenes que se han desarrollado.

Es comprensible que este último comentario no se entienda completamente en este punto del documento, pero más adelante, cuando se detallen las aplicaciones desarrolladas y la forma en que se han implementado, se volverá a abordar este aspecto para que quede completamente claro.

Una vez entendido el papel de cada uno de los tres códigos ArUco dentro del contexto de la aplicación de procesamiento de imágenes, se presentan a continuación las

motivaciones que dieron origen a los dos modos de funcionamiento del programa. En este punto de la memoria, se busca explicar de manera general, sin entrar en demasiados detalles, los dos modos de funcionamiento y las razones detrás de cada uno. En secciones posteriores, como en la guía de usuario o en la filosofía de programación, se proporcionarán explicaciones más detalladas y concretas sobre cómo utilizar estos modos de funcionamiento y cómo están programados.

En la presente memoria, para evitar confundir al lector, cada modo de funcionamiento se denominará como modo simulador y modo vuelo real.

Modo de funcionamiento como simulador

Para comprender el origen y la motivación de este modo de funcionamiento, conviene recordar que, en el estado del arte actual, el objetivo de los códigos ArUco es permitir el control automático del vuelo del dron, el seguimiento automático de objetos, y la realización de tareas de seguimiento de códigos ArUco, entre otras aplicaciones. Al intentar replicar este tipo de aplicaciones en el presente TFM, surgieron dos grandes problemas.

El primero fue la complejidad del diseño de estas aplicaciones. Al tratarse de un proyecto multidisciplinar, no solo se debía considerar el desarrollo del software, sino también muchos otros elementos como el dron, las comunicaciones, y la forma en que está desarrollada la aplicación móvil que actúa como servidor y recibe los comandos del cliente. Además, este TFM representa la primera aproximación al uso de códigos ArUco, por lo que plantearse el diseño de una aplicación avanzada desde el inicio excedía los límites de dificultad del proyecto.

El segundo problema fue la necesidad de resolver el embarcado del reComputer J3011, que presentó muchas más dificultades de las esperadas. Sin embarcar el reComputer J3011, no se podían probar los códigos desarrollados, lo que implicaba trabajar de manera ineficiente y a ciegas.

El modo simulador se creó para superar estas dificultades iniciales y permitir que el TFM pudiera seguir su curso. La creación de este modo de funcionamiento pone de manifiesto la capacidad de adaptación y la flexibilidad, cualidades a menudo necesarias en el campo de la ingeniería.

El modo de funcionamiento como simulador está orientado al control del dron con el reComputer J3011 en tierra, es decir, sin necesidad de que este esté embarcado. Este modo permite pilotar el dron utilizando el código ArUco de control (ver Figura 4.5c) mencionado anteriormente, haciendo que dicho código se comporte como un volante y un acelerador.

Para entender cómo funciona, se puede imaginar el reComputer J3011 en tierra y, delante de su cámara, un código ArUco impreso en un papel. Al mover el papel frente a la cámara, el dron responde como si se estuvieran utilizando controles de un

simulador. Si el papel se gira a la derecha, el dron girará a la derecha; si se desplaza hacia arriba, el dron avanzará; y si se acerca el papel a la cámara, el dron ascenderá, como si se estuviera accionando el acelerador.

Estas acciones se basan en los errores calculados previamente. Siempre que el código ArUco no esté perfectamente centrado en la imagen capturada de 640x480 píxeles, los errores de *pitch*, *roll*, *yaw* y *throttle* no serán nulos. Mientras estos errores sean no nulos, se envían una serie de comandos al dron a través de la aplicación móvil, lo que hace que el dron se mueva en respuesta a los movimientos del papel.

El término «simulador» se utiliza porque este modo de funcionamiento permite pilotar el dron de manera similar a como se haría en un simulador, proporcionando un control intuitivo y directo del dron sin necesidad de que el reComputer esté embarcado.

Es relevante retomar el tema mencionado anteriormente sobre la corrección de errores de uno en uno. En el diseño actual de la aplicación móvil, que es la encargada de recibir las cadenas de texto procedentes del cliente (reComputer J3011) basadas en los errores calculados, solo es posible corregir un único error a la vez. Para ilustrar este punto, se presenta el siguiente ejemplo.

Supongamos que se tiene un `errorYaw` >0 mientras que los otros tres errores son nulos. Debido a este `errorYaw` >0, la aplicación de procesamiento de imágenes detecta el error y envía una cadena de texto al servidor (la aplicación móvil) para que ordene al dron que se mueva y corrija dicho error. Mientras el dron está corrigiendo este error, imaginemos que antes de finalizar la corrección, uno de los otros tres errores se vuelve no nulo, por ejemplo, `errorPitch` <0. Automáticamente, el sistema de control de la aplicación de procesamiento de imágenes detecta este nuevo error y envía una nueva cadena de texto al servidor para que el dron corrija este nuevo error.

El problema surge cuando el servidor envía esta segunda orden de movimiento, ya que automáticamente la primera orden se anula. Como resultado, el dron comienza a moverse para corregir el segundo error sin haber completado la corrección del primero. Esta particularidad implica que en el modo simulador solo se pueden ejecutar movimientos individuales y no combinados. Por ejemplo, se puede girar el «volante» para girar el dron, pero no se puede girar el volante y acelerar simultáneamente.

Además, en el modo de funcionamiento como simulador, la única manera de detener el movimiento del dron es devolver el «volante» a su posición original, es decir, centrar el código ArUco en la imagen para que todos los errores sean nulos.

Modo de vuelo real

Este modo de funcionamiento de la aplicación de procesamiento de imágenes está diseñado para ser ejecutado con el reComputer J3011 embarcado en el dron, representando así una evolución respecto al modo de funcionamiento anterior y acercándose más a las aplicaciones avanzadas del estado del arte actual. Este modo permite el despegue,

vuelo entre *waypoints* y aterrizaje del dron de forma automática. A continuación, se describe de manera general su funcionamiento, dejando los detalles específicos para secciones posteriores.

En primer lugar, se presenta al sistema de visión del reComputer J3011 el código ArUco de despegue, ver Figura 4.5a. Como consecuencia, se enviará al servidor una cadena de texto que hará que la aplicación móvil ordene al dron que despegue. Una vez el dron ha despegado, se posiciona y centra automáticamente en torno a este código ArUco de despegue. Tras posicionarse, enviará al servidor una cadena de texto para que la aplicación móvil ordene al dron que avance en busca de otro código ArUco.

El nuevo código ArUco detectado puede ser de control o de aterrizaje. Si es un código ArUco de control, el dron se posicionará y centrará en torno a él, y una vez hecho esto, enviará nuevamente una cadena de texto al servidor para que la aplicación móvil ordene al dron avanzar en busca de otro código ArUco de control o de aterrizaje. Si el código ArUco detectado es de aterrizaje, el dron se posicionará en torno a él y, al ser un código de aterrizaje, la aplicación de procesamiento de imágenes enviará una cadena de texto al servidor (la aplicación móvil), que ordenará al dron que aterrice.

Finalmente, es interesante plantearse la siguiente cuestión ¿Cómo es posible que el dron sea capaz de posicionarse en torno a un código ArUco si anteriormente se mencionó que solo se pueden corregir los errores de uno en uno? La solución a este problema se ha logrado precisamente utilizando la idea de corregir los errores de uno en uno. En cada fotograma capturado, se calcula cuál de los cuatro errores (*pitch*, *roll*, *yaw* y *throttle*) es el más grande, y se envía la acción correspondiente para corregir este error mayor.

De esta manera, manteniendo la premisa de que solo se puede corregir un error a la vez, se logra corregir todos los errores mediante un proceso de minimización consecutiva del error más grande hasta que todos los errores sean cero. Este enfoque permite que el dron se posicione y se centre en torno al código ArUco de manera eficiente y precisa.

4.4.2 Carpeta de trabajo del proyecto

Este epígrafe tiene como objetivo presentar la estructura de carpetas del proyecto. Detrás de cualquier aplicación de *software* hay una variedad de *scripts*, archivos de configuración y otros archivos esenciales que aseguran su correcto funcionamiento. La aplicación de procesamiento de imágenes desarrollada en el presente TFM se ha concebido como un bloque integral. En lugar de desarrollar *scripts* independientes que realicen tareas de procesamiento distintas, se ha optado por cohesionar todos estos *scripts* en un todo interdependiente.

El desarrollo de la aplicación se ha llevado a cabo de la manera más profesional posible, asegurando que los *scripts* que componen la aplicación estén relacionados entre sí, ejecutando cada uno acciones diferentes pero dependientes. Esto significa

que el funcionamiento de los *scripts* es interdependiente, de modo que si uno falla, toda la aplicación se ve afectada.

Este epígrafe pretende describir el contenido de la carpeta raíz del proyecto, ver Código 4.11, utilizada para el desarrollo de la aplicación de procesamiento de imágenes. Esta carpeta se ha usado como la carpeta raíz en Visual Studio Code (VSCode) durante el desarrollo de la aplicación.

```
TFM_Proyecto/
|
|-- .vscode/
|   |-- README.txt
|   |-- settings.json
|
|-- CameraFiles/
|   |-- cameraDistortion.txt
|   |-- cameraMatrix.txt
|
|-- images/
|   |--CONTROL_DICT_6X6_250-Marker_id_10.pdf
|   |--CONTROL_DICT_6X6_250-Marker_id_10.png
|   |--CONTROL_DICT_6X6_250-Marker_id_10.svg
|   |--DICT_4X4_100-Marker_id_35.pdf
|   |--DICT_4X4_100-Marker_id_35.png
|   |--DICT_4X4_100-Marker_id_35.svg
|   |--DICT_6X6_250-Marker_id_2.pdf
|   |--DICT_6X6_250-Marker_id_2.png
|   |--DICT_6X6_250-Marker_id_2.svg
|   |--LANDING_DICT_6X6_250-Marker_id_7.pdf
|   |--LANDING_DICT_6X6_250-Marker_id_7.png
|   |--LANDING_DICT_6X6_250-Marker_id_7.svg
|   |--TAKE-OFF_DICT_6X6_250-Marker_id_2.pdf
|   |--TAKE-OFF_DICT_6X6_250-Marker_id_2.png
|   |--TAKE-OFF_DICT_6X6_250-Marker_id_2.svg
|
|-- LogsTelemetria/
|   |-- telemetria.txt
|
|-- modulos/
|   |-- __pycache__
|   |-- __init__.py
|   |-- ColoresEstilosFormatos.py
|   |-- GlobalVariables.py
|   |-- ImageProcessing.py
|   |-- ImageProcessingArUco_README.txt
|   |-- ImageProcessingArUcoSimulador.py
|   |-- ImageProcessingArUcoVuelo.py
|   |-- isRotationMatrix.py
|   |-- moveSimulador.py
|   |-- moveVuelo.py
|   |-- NombresDiccionariosArUco.txt
|   |-- Predefined_ArUco_Dictionaries.py
|   |-- RasPi_Capabilities.txt
|   |-- RasPi_gstreamer_pipeline.py
```

```
|   |-- RasPi_TakePhotos.py
|   |-- rotationMatrixToEulerAngles.py
|   |-- socket_client_Proyecto.py
|   |-- socket_client.py
|   |-- socket_server.py
|   |-- Telemetry.py
|   |-- webcam_capabilities.txt
|   |-- webcam_gstreamer_pipeline.py
|   |-- webcam_TakePhotos.py
|
|-- .env
|
|-- EstructuraProyecto.txt
|
|-- main.py
```

Código 4.11: Estructura de carpetas del proyecto.

A continuación, se describirá únicamente el contenido de las carpetas, sin detallar las funciones de los archivos que contienen.

Carpeta .vscode

La carpeta .vscode contiene un archivo de configuración **«settings.json»** para el IDE Visual Studio Code, que personaliza diversos aspectos del entorno de desarrollo, ver Apéndice D. El contenido del archivo incluye ajustes para la familia de fuentes, el tamaño de fuente, la altura de línea y el tamaño de tabulación del editor. También configura la inserción de espacios, el ajuste de palabras y el guardado automático de archivos con un retraso específico. Además, define el tema de color y el tema de iconos del área de trabajo, desactiva la vista previa del editor y ajusta el tamaño de fuente del terminal integrado. También se incluye una confirmación para la eliminación de archivos, el formato automático al guardar, la desactivación del minimapa y la configuración del análisis y autocompletado de Python.

Carpeta CameraFiles

La carpeta CameraFiles contiene dos archivos .txt:

- **«camera_calibration_matrix.txt»**
- **«camera_distortion_matrix.txt»**

El primero es la matriz de calibración de la cámara y el segundo es la matriz de distorsión de la cámara. Ambos archivos son necesarios para las funciones del módulo «`cv2.aruco`», que se encargan de la detección y procesamiento de los códigos ArUco.

Carpeta images

La carpeta images contiene los códigos ArUco empleados en el proyecto. Estos códigos están disponibles en diferentes formatos, como .pdf, .svg y .png.

Carpeta LogsTelemetria

La carpeta LogsTelemetria contiene archivos .txt que registran la telemetría de cada vuelo. Por cada vuelo se genera un archivo .txt único e independiente. Estos archivos incluyen datos como la velocidad sobre el suelo (*Ground Speed*, GS), la velocidad vertical (*Vertical Speed*, VS), la altitud, el rumbo (*Heading*), la inclinación (*Roll* y *Pitch*) y las distancias a los obstáculos detectados por el Phantom 4 Advanced+.

Carpeta modulos

La carpeta modulos contiene todos los *scripts* necesarios para el adecuado funcionamiento del proyecto, los cuales serán descritos convenientemente en secciones posteriores de esta memoria.

Archivo .env

El archivo .env contiene la línea mostrada en el Código 4.12. Este archivo se utiliza para establecer variables de entorno necesarias para el proyecto. En este caso, define el «PYTHONPATH», que especifica la ruta donde se encuentran los módulos Python necesarios para el proyecto, permitiendo que el entorno de desarrollo y los *scripts* Python los localicen y utilicen correctamente. Además, este archivo facilita la configuración del entorno de desarrollo en diferentes máquinas, asegurando que todos los desarrolladores trabajen con las mismas configuraciones y rutas, lo cual es crucial para mantener la consistencia y evitar problemas de dependencias durante el desarrollo y la ejecución del proyecto.

```
PYTHONPATH=/home/omar/Escritorio/TFM/ArUco/Codes/TFM_Proyecto
```

Código 4.12: Archivo .env.

4.4.3 Manual de usuario

En la presente sección se proporciona un manual de usuario para la utilización de la aplicación de procesamiento de imágenes desarrollada. El objetivo de este manual es explicar detalladamente cómo un usuario ajeno al proyecto puede utilizar la aplicación y sus funcionalidades asociadas. Se describen los pasos necesarios para operar la aplicación de manera efectiva, asegurando que cualquier usuario pueda comprender y emplear las herramientas desarrolladas sin necesidad de conocimientos previos del proyecto.

El presente manual de usuario se presenta en este punto de la memoria antes que la explicación de la filosofía de programación y los *scripts* de Python que dan vida al

proyecto. Esto se debe a la enorme complejidad subyacente en la aplicación; explicar primero el manual de usuario puede ayudar a los lectores a entender gradualmente lo que la aplicación hace y el porqué de muchas de las decisiones adoptadas durante su desarrollo. Comprender el manual de usuario es más sencillo que abordar directamente toda la lógica interna de la aplicación, lo que facilita una introducción progresiva al desarrollo y funcionamiento del proyecto.

Como se ha mencionado a lo largo de esta memoria, el desarrollo de la aplicación móvil se ha realizado en Python utilizando el IDE Visual Studio Code. Por lo tanto, el primer paso para ejecutar la aplicación es abrir Visual Studio Code. Una vez abierto, el siguiente paso es abrir la carpeta del proyecto. Tal y como se explicó anteriormente, esta carpeta ha sido diseñada para incluir todos los elementos, rutas, variables de entorno y archivos necesarios para que el proyecto funcione correctamente.

Una vez abierta la carpeta raíz del proyecto, el siguiente paso es localizar el archivo Python **«main.py»**. Este archivo es el único que debe ejecutarse para poner en marcha la aplicación de procesamiento de imágenes. **«main.py»** contiene toda la lógica de programación y se encarga de llamar al resto de *scripts* de Python necesarios para el correcto funcionamiento del programa. La ventaja de centralizar toda la lógica en un solo archivo es que simplifica la ejecución y comprensión de la aplicación para un usuario que no ha participado en el desarrollo del proyecto. Esto facilita el uso de las funcionalidades de la aplicación y permite una experiencia de usuario más intuitiva.

Una vez ejecutado el archivo **«main.py»**, pasados unos segundos, aparecerá un mensaje en pantalla solicitando al usuario que establezca el modo de funcionamiento de la aplicación. En este punto, el usuario debe presionar la tecla correspondiente al número 1 para seleccionar el modo de funcionamiento como simulador o la tecla correspondiente al número 2 para seleccionar el modo de funcionamiento como vuelo real. Una vez tomada esta decisión, y nuevamente después de unos segundos, se solicitará al usuario que elija si desea ejecutar la aplicación en modo local (utilizando un servidor local) o en el servidor del móvil Android en el que está instalada la aplicación desarrollada con el DJI SDK. Si el usuario desea el modo local, deberá presionar la tecla correspondiente al número 1 seguida de la tecla enter. Si desea el modo servidor móvil Android, deberá presionar la tecla correspondiente al número 2 seguida de la tecla enter.

En este punto conviene explicar más en profundidad el concepto de servidor local y servidor en el móvil Android. Para el presente TFM, como se ha explicado a lo largo de esta memoria, el esquema de comunicaciones entre el ordenador embarcado (re-Computer J3011) y la aplicación móvil desarrollada con el DJI SDK e instalada en un móvil Android, se basa en un sencillo esquema cliente-servidor. En este esquema, el re-Computer J3011 actúa como cliente y la aplicación móvil como servidor. La aplicación de procesamiento de imágenes, al detectar códigos ArUco, envía una serie de cadenas de texto al servidor (aplicación móvil), quien las recibe, interpreta y, en función de la cadena de texto recibida, ordena al dron ejecutar ciertas acciones.

El problema con este esquema de comunicaciones, especialmente durante la fase de desarrollo de la aplicación, es que es necesario tener constantemente encendidos el dron, la aplicación móvil y el mando que actúa como puente entre ambos. Este requerimiento retrasaba en ciertas ocasiones el avance y desarrollo de ciertas etapas de la aplicación. A consecuencia de esto, surgió el concepto de servidor local. El servidor local no es más que un *script* de Python que actúa como servidor, en este caso como *localhost*. El esquema de comunicaciones cliente-servidor se mantiene intacto, pero con la salvedad de que ahora el servidor no es la aplicación móvil sino el servidor local (*localhost*). Las cadenas que el cliente (reComputer J3011) envía al servidor local son las mismas que enviaría al servidor de la aplicación móvil. Sin embargo, lógicamente, estas cadenas de texto no se transforman en acciones que se envían al dron.

Aunque es cierto que con este enfoque de servidor local el dron no puede volar, resulta muy útil para un desarrollo más rápido y una depuración más eficiente de los códigos. Permite implementar cambios de manera rápida y observar el efecto de estos cambios de forma inmediata.

Manual de usuario del modo de funcionamiento como simulador

Anteriormente ya se introdujo la funcionalidad básica de este modo de funcionamiento. Recordemos que este modo de funcionamiento está diseñado para la operación del dron con el reComputer J3011 en tierra. La manera de utilizar este modo es la siguiente: el primer paso es ejecutar el *script* **«main.py»** como se indicó anteriormente. Luego, cuando se le solicite al usuario que seleccione el modo de funcionamiento, debe pulsar la tecla número 1 seguida de «*Enter*» para elegir el modo simulador. A continuación, se selecciona el tipo de servidor sobre el que se desea ejecutar la aplicación, ya sea un servidor local o el servidor del móvil Android. Para las explicaciones en este manual, se asumirá la selección del servidor móvil Android, ya que representa el caso de uso real de la aplicación.

Primero, el usuario debe mostrar a la cámara del reComputer J3011 el código ArUco de despegue, ver Figura 4.5a. Una vez detectado este código, la aplicación de procesamiento de imágenes lo reconocerá y enviará la cadena de texto de despegue al servidor (aplicación móvil), quien la recibirá, interpretará y enviará la orden de despegue al dron. Después del despegue, el usuario puede controlar el dron utilizando el código ArUco de control, ver Figura 4.5c, moviéndolo como si fuera un volante y un acelerador.

Es importante recordar que, tal y como está implementada la aplicación móvil, solo se pueden ejecutar los movimientos del dron de uno en uno. Los movimientos posibles son: subir-bajar (*Throttle*), avanzar-retroceder (*Pitch*), desplazarse a la derecha-izquierda (*Roll*) y girar sobre sí mismo (*Yaw*). Con el modo simulador, no se pueden combinar movimientos simultáneamente; por ejemplo, no se puede subir al mismo tiempo que el dron avanza.

Para efectuar estos movimientos, el usuario debe seguir las siguientes indicaciones:

- Para mantener el dron quieto, el código ArUco debe estar centrado en el fotograma capturado.
- Para controlar el *Yaw*, se debe girar la hoja sobre su eje como si fuera un volante. Cuando se gira la hoja, se envía automáticamente el comando de movimiento. Para cancelar este comando, se debe devolver la hoja a su posición inicial.
- Para avanzar (*Pitch*), se debe desplazar la hoja hacia arriba en el fotograma; para retroceder, desplazarla hacia abajo. Para anular estos movimientos, devolver la hoja a su posición inicial.
- Para girar a la derecha o izquierda (*Roll*), se debe desplazar la hoja hacia la derecha o izquierda respecto al centro del fotograma capturado. Para anular estos movimientos, devolver la hoja a su posición inicial.
- Para subir o bajar (*Throttle*), se debe acercar o alejar la hoja de la cámara del reComputer J3011. Para cancelar el movimiento, devolver la hoja a su posición inicial.

Todos estos movimientos se basan en el cálculo de los errores mencionados anteriormente. Los errores son cero cuando el centro del marcador coincide con el centro del fotograma capturado. Siempre que el centro del marcador no coincida con el centro del fotograma, los errores serán no nulos y, como consecuencia, se enviarán cadenas de texto al servidor para ordenar al dron que se mueva y corrija estos errores.

La lógica interna del programa es tal que la cadena de texto se envía al servidor cuando el error supera un determinado umbral. Por ejemplo, la cadena de texto que ordena al dron girar a la derecha se envía al servidor no cuando el `errorYaw` >0, sino cuando el `errorYaw` >0.10. Estos umbrales pueden ajustarse convenientemente para asegurar un control más fluido y seguro de la aeronave en vuelo, evitando movimientos erráticos o demasiado sensibles.

Manual de usuario del modo de funcionamiento como vuelo real

En primer lugar, es conveniente recordar que este modo de funcionamiento está diseñado para la operación y control del dron de manera automática, con el reComputer J3011 embarcado en el mismo. Este modo pretende aproximarse a las aplicaciones presentes en el estado del arte actual, donde el manejo automático del dron se realiza mediante el uso de códigos ArUco. No obstante, es importante tener en cuenta que el presente TFM es un trabajo académico, por lo que replicar este tipo de aplicaciones avanzadas excedería los límites tanto temporales como de dificultad del proyecto.

De esta reflexión, y buscando un equilibrio entre complejidad y funcionalidad, surge este modo de funcionamiento, cuya lógica interna es más avanzada que la del modo simulador y representa un punto intermedio entre dicho simulador y las aplicaciones

avanzadas del estado del arte actual. Este modo de funcionamiento permite la operación automática de despegue, vuelo entre *waypoints* y aterrizaje del dron.

Para entender mejor este modo de funcionamiento, es útil revisar cómo está programado. La implementación sigue una estructura basada en fases y utiliza un diccionario para simular la estructura *switch-case*, que no está disponible en Python. Las fases consideradas son: despegue, posicionamiento, *hold*, avance y aterrizaje. En cada fase, se ejecutan acciones específicas dependiendo del estado del dron.

Antes que nada, para utilizar este modo de funcionamiento, primero se debe ejecutar el *script* **«main.py»**, luego pulsar 1 y «*Enter*» para seleccionar este modo, y finalmente pulsar 2 y «*Enter*» para seleccionar el servidor en el móvil Android, que es el caso que se explicará en este manual de usuario.

El funcionamiento de este modo es el siguiente:

1. **Fase de despegue:** Cuando el sistema de visión del reComputer J3011 detecta el código ArUco de despegue, envía la cadena de texto de despegue al servidor (aplicación móvil), que la recibe, interpreta y envía la orden de despegue al dron.

2. **Fase de posicionamiento:** Una vez finalizada la fase de despegue, el dron entra en la fase de posicionamiento. El objetivo de esta fase es posicionar el dron en torno a un código ArUco de manera que los cuatro errores (*pitch*, *roll*, *yaw*, *throttle*) sean cero. Debido a la limitación de corregir los errores de uno en uno, esta fase se programa buscando el error más grande y minimizándolo en cada iteración. A través de un bucle de control, se minimiza consecutivamente el error más grande hasta que todos los errores son cero.

3. **Fase de avance:** Con todos los errores minimizados, el dron pasa a la fase de avance, donde se envía una cadena de texto al servidor para que el dron avance hacia delante. Esta fase termina cuando el dron encuentra un código ArUco de control o de aterrizaje. Es importante notar que si nunca encuentra uno de estos códigos, la fase no termina, lo cual es una debilidad del código.

4. **Nueva fase de posicionamiento:** Al detectar un nuevo código ArUco, el dron se detiene y entra nuevamente en la fase de posicionamiento para centrarse en el código detectado. Si el código detectado es de control, el dron vuelve a la fase de avance; si es de aterrizaje, el dron pasa a la fase de aterrizaje.

5. **Fase de aterrizaje:** En esta fase final, el dron envía una cadena de texto con la orden de aterrizaje al servidor, que la interpreta y envía la orden al dron, finalizando así este modo de funcionamiento.

Este modo de funcionamiento representa aplicaciones que hoy en día están presentes en el estado del arte actual. La programación de este modo de funcionamiento no ha sido trivial, y detrás del mismo hay muchos pequeños factores que han tenido que ser

considerados. Aunque se es consciente de que, como cualquier aplicación de *software*, el programa tiene ciertas debilidades y hay casos específicos que no se han contemplado, se cree firmemente que el programa cumple perfectamente con los objetivos planteados en este TFM. En pruebas locales, se ha verificado el funcionamiento adecuado de este modo de operación, y los resultados han sido satisfactorios. Sin embargo, debido a los enormes desafíos que implicó el embarcado del reComputer J3011, este modo de vuelo no ha sido suficientemente probado en condiciones de vuelo reales. A pesar de esto, la infraestructura, la lógica y las secuencias de control han demostrado ser efectivas en tierra, lo que sugiere que, aunque puede ser necesario refinar ciertos aspectos relacionados con el vuelo, estos ajustes serán menores.

4.4.4 Filosofía de programación

Debido a la complejidad que entraña la aplicación de procesamiento de imágenes, no solo por su lógica interna, sino también por la multitud de *scripts* interdependientes que intervienen en ella, es complicado, en una memoria como la presente, explicar paso a paso todo el trabajo realizado para desarrollar la aplicación. Al igual que en cualquier proyecto de programación, es difícil detallar todas y cada una de las soluciones adoptadas, los parches realizados y las razones que llevaron a estas decisiones. Para abordar este desafío, surge esta sección de la memoria.

En esta sección, se pretende explicar la filosofía general del funcionamiento de la aplicación, es decir, dar a conocer al lector los *scripts* de Python que representan los pilares de la aplicación y cómo están conectados entre sí. Además, esta sección busca proporcionar al lector una serie de ideas y conceptos generales sobre cómo se ha llevado a cabo la programación de la aplicación, permitiéndole entender mejor su funcionamiento interno y el porqué de muchas de las decisiones tomadas. En la sección posterior a esta, se describirán uno a uno los *scripts* de Python que intervienen en la aplicación. En esta sección, se mencionarán solo los *scripts* más importantes o aquellos cuyo rol es fundamental para el funcionamiento de la aplicación.

En primer lugar, uno de los principales desafíos en la programación de la aplicación fue diseñar una forma de soportar todas las funcionalidades necesarias. Entre estas se incluían la captura continua de *frames* de una cámara y la búsqueda constante de códigos ArUco en dichos *frames*. En caso de detectarse códigos ArUco, la aplicación debía enviar una serie de cadenas de texto al servidor mediante un esquema cliente-servidor implementado en Python. Además, la aplicación debía recibir y decodificar continuamente la telemetría proveniente del dron, que se enviaba a la aplicación móvil a través del mando actuando como puente. Posteriormente, la aplicación móvil, actuando como servidor, enviaba esta telemetría al reComputer J3011 (cliente), que la recibía, mostraba en pantalla y utilizaba para realizar acciones de control, como la monitorización constante de las distancias a los obstáculos para evitar colisiones.

Todas las acciones expuestas anteriormente, y muchas otras que no se han mencionado, indican que la aplicación debe realizar simultáneamente una multitud de operaciones que, aunque parecen diferentes, son interdependientes. La necesidad de dar soporte simultáneo a todas estas operaciones fue una de las razones que llevó a plantear el proyecto de forma modular, es decir, a través de diferentes *scripts*, cada uno orientado a una operación distinta. Hasta aquí, no parece haber ningún problema, ya que mediante el planteamiento modular y haciendo uso, por ejemplo, de hilos (*Threads*), se podría paralelizar las operaciones que la aplicación debía realizar, de manera que se ejecutasen de manera independiente y simultánea.

Sin embargo, el principal desafío que surgió fue cómo interconectar los distintos módulos del proyecto, ya que, para que un módulo ejecutase su función, necesitaba los resultados en tiempo real de otro módulo. Este desafío se resolvió mediante el uso de variables globales. Se desarrolló un módulo específico dedicado a albergar estas variables globales, lo que permitió que todos los módulos del proyecto pudieran acceder y actualizar sus valores. Este enfoque garantizó que todas las partes del proyecto estuvieran sincronizadas en tiempo real, asegurando que cada módulo tuviera acceso inmediato a la información actualizada necesaria para llevar a cabo sus operaciones de manera eficiente y coordinada.

Una vez aclaradas estas ideas, parece claro que el camino para diseñar la aplicación implica el uso de diferentes módulos, hilos (*threads*) y variables globales para comunicar todos los módulos. Esta filosofía de programación modular permite dividir las tareas en componentes más manejables y especializadas, donde cada módulo se encarga de una operación específica. Los hilos permiten la ejecución simultánea de estas operaciones, mientras que las variables globales facilitan la comunicación en tiempo real entre los módulos.

Sin embargo, durante el diseño de la aplicación surgió un gran problema relacionado con el uso de *Threads* en Python. A diferencia de otros lenguajes de programación, Python no permite el verdadero paralelismo en *Threads* debido al *Global Interpreter Lock* (GIL), que asegura que solo un hilo ejecute bytecode de Python a la vez. Este bloqueo limita la capacidad de los *Threads* para realizar procesamiento concurrente, afectando el rendimiento de la aplicación.

La alternativa para superar esta limitación es el uso del módulo *subprocessing*, que permite la creación de procesos independientes en lugar de hilos. Los procesos tienen su propio espacio de memoria y pueden ejecutarse en paralelo en diferentes núcleos del procesador, eliminando las restricciones impuestas por el GIL. Sin embargo, la comunicación entre procesos es mucho más compleja que entre *threads*. El enfoque de usar variables globales no es aplicable en este caso, ya que cada proceso tiene su propio espacio de memoria y no puede compartir variables de manera directa. Esto introdujo un desafío adicional en el diseño de la aplicación, ya que se necesitaba implementar mecanismos de comunicación interproceso (IPC) más avanzados, como colas (*queues*)

o tuberías (*pipes*), para asegurar que los procesos pudieran intercambiar información de manera eficiente y en tiempo real.

Para solventar los problemas y estructurar la aplicación de manera eficiente, se han seguido los siguientes enfoques y principios de diseño:

Script principal: «main.py»

El archivo **«main.py»** es el único archivo ejecutable del proyecto y desempeña un papel crucial en la coordinación de todas las funcionalidades de la aplicación. Este *script* se encarga de inicializar variables globales, seleccionar modos de ejecución, y gestionar la comunicación entre los distintos módulos. Además, crea y gestiona los hilos y procesos necesarios para el funcionamiento del sistema.

Thread del move

Uno de los componentes esenciales del *script* **«main.py»** es la creación del hilo (*Thread*) denominado «*Thread* del *move*». Este hilo se encarga de monitorizar constantemente los errores de control del dron. Si estos errores superan un determinado umbral, el hilo envía las cadenas de texto pertinentes al servidor para que este comunique las acciones correctivas al dron. En esencia, el «*Thread* del *move*» se ocupa de la supervisión de los errores y la emisión de comandos al servidor para mover el dron y corregir dichos errores.

Proceso para la Telemetría

A pesar de los problemas asociados con el uso de procesos en Python, como la complejidad de la comunicación entre ellos y la imposibilidad de utilizar variables globales compartidas, se decidió mover la monitorización de la telemetría a un proceso independiente en lugar de un hilo. Esto se debe a que el rendimiento de la aplicación era deficiente cuando se utilizaba un hilo para esta tarea. Por lo tanto, la telemetría se gestiona mediante un proceso que recibe, decodifica y muestra en pantalla la información del dron en tiempo real, así como la utiliza para la monitorización constante de las distancias a los obstáculos para evitar colisiones.

Bucle principal y captura de *Frames*

Después de crear el hilo para el *move* y el proceso para la telemetría, el *script* **«main.py»** entra en un bucle infinito cuyo objetivo es capturar *frames* de la cámara y pasarlos a la función «ImageProcessing». Esta función es otra pieza clave del proyecto, ya que se encarga de analizar cada *frame* en busca de posibles códigos ArUco. Dependiendo del tipo de código ArUco detectado (despegue, aterrizaje o control), la función actualiza las variables globales correspondientes. Por ejemplo, si se detecta un código ArUco de control, se calculan los errores de control y se actualizan las variables globales relativas a estos errores, asegurando que el resto de las partes del programa estén informadas en tiempo real de los cambios.

Conexión y coordinación La correcta conexión y coordinación entre estos componentes permiten que la aplicación funcione de manera eficiente y coordinada. Aunque hay muchos otros *scripts* que contribuyen al funcionamiento de la aplicación, realizando tareas de mantenimiento y soporte, los pilares descritos anteriormente representan los elementos fundamentales que dan vida a la aplicación. El enfoque modular y el uso de hilos y procesos aseguran que la aplicación pueda manejar múltiples tareas simultáneamente, manteniendo una comunicación efectiva y una operación fluida.

4.4.5 *Scripts* y códigos de la aplicación

En la introducción de la subsección anterior se argumentó que, en aplicaciones de *software* complejas como la desarrollada en el presente TFM, explicar paso a paso todos los *scripts* involucrados junto con la lógica interna de cada uno de ellos es una tarea ardua y compleja. Para abordar esta cuestión, surgieron las subsecciones anteriores y la presente. En la subsección anterior, se intentó introducir al lector en la filosofía empleada para el desarrollo de la aplicación, describiendo las partes que la componen y cómo están interconectadas. Asimismo, se comentaron ciertos problemas surgidos durante el desarrollo, cuyo conocimiento podría ayudar a futuros integrantes del proyecto a comprender algunas de las decisiones tomadas.

Este epígrafe tiene como objetivo completar la tarea iniciada en el epígrafe anterior. Ahora se procederá a revisar cada *script* del proyecto, ofreciendo una pequeña descripción de la lógica de cada uno y su utilidad dentro del proyecto. Las explicaciones intentarán ser lo más didácticas posibles, enfocándose más en el nivel conceptual que en el nivel de programación. No obstante, se adjuntarán en forma de apéndices todos los *scripts*, para que se puedan consultar libremente y comprender mejor su implementación en caso de que alguna explicación no sea suficientemente clara.

Script «main.py», Apéndice E

El archivo **«main.py»** es el núcleo central del proyecto y el único archivo ejecutable, encargado de coordinar y orquestar las diferentes funciones y módulos necesarios para el funcionamiento del sistema. En este *script* se lleva a cabo la inicialización de variables globales, la selección del modo de ejecución y el tipo de servidor, la creación de hilos y procesos, y la captura y procesamiento de imágenes.

El *script* comienza con la importación de varias librerías esenciales y módulos personalizados. Entre ellos se incluyen «`numpy`», «`cv2`» para procesamiento de imágenes, y varios módulos del proyecto como «`GlobalVariables`», «`moveSimulador`», «`moveVuelo`», «`ImageProcessingArUcoSimulador`», «`ImageProcessingArUcoVuelo`», entre otros. Luego, se procede a la inicialización de diversas variables globales que serán utilizadas a lo largo de la aplicación. Estas variables permiten mantener un estado compartido entre los diferentes componentes del sistema, garantizando la coherencia y sincronización de las operaciones.

Uno de los primeros pasos críticos es la selección del modo de ejecución, donde el usuario puede optar entre el modo simulador o vuelo real. Dependiendo de la elección, se asignan diferentes funciones de procesamiento de imágenes y control de movimiento. Esta flexibilidad permite adaptar la aplicación a diferentes entornos de prueba y uso.

A continuación, se ofrece al usuario la opción de ejecutar la aplicación en un servidor local o en un móvil Android. Dependiendo de la selección, se establece una conexión con el servidor correspondiente. En el caso del servidor local, se abre una nueva terminal para ejecutar el servidor, mientras que en el caso del servidor en el móvil Android, se establece una conexión directa a través de la red. Este esquema cliente-servidor es fundamental para el funcionamiento distribuido de la aplicación.

Uno de los componentes esenciales del *script* es la creación del hilo denominado "*Thread* del *move*". Este hilo se encarga de monitorear constantemente los errores y, en caso de que estos superen un determinado umbral, enviar las cadenas de texto pertinentes al servidor para que comunique estas acciones al dron. La capacidad de ejecutar esta función en segundo plano permite que el programa realice otras tareas simultáneamente, mejorando la capacidad de respuesta y eficiencia del sistema.

Además, se crea un proceso separado para la recepción y decodificación de la telemetría del dron. A pesar de los problemas relacionados con el uso de procesos y variables globales, este enfoque se adoptó debido a que un hilo no ofrecía el rendimiento necesario para la aplicación. Este proceso asegura la recepción y procesamiento eficiente de la telemetría, información crítica para la operación segura del dron.

Después de la creación del hilo y el proceso, el *script* entra en un bucle infinito donde captura continuamente *frames* de la cámara, los procesa mediante la función «`ImageProcessing`», y actualiza las variables globales en función de los códigos ArUco detectados y los errores calculados. Esta función es clave para el proyecto, ya que se encarga de analizar cada *frame* capturado, buscar códigos ArUco y calcular los errores necesarios para el control del dron. La lógica interna de esta función asegura que todas las partes del programa estén informadas en tiempo real de las actualizaciones, permitiendo una operación coordinada y eficiente.

Script «GlobalVariables.py», Apéndice F

El *script* **«GlobalVariables.py»** es crucial para la correcta operatividad del proyecto, ya que define y gestiona todas las variables globales que necesitan ser accesibles y compartidas entre los diferentes módulos y procesos de la aplicación. Este enfoque asegura que todos los componentes del sistema estén sincronizados y puedan acceder a la información más actualizada en tiempo real. A continuación, se describe en detalle la lógica y la funcionalidad de este script.

En primer lugar, se hace uso del módulo «`multiprocessing`» de Python para definir varias variables compartidas. Esto es fundamental para permitir que los procesos independientes puedan comunicarse y compartir datos de manera eficiente. Las variables

se definen utilizando «Value» y «Array», que son estructuras de datos seguras para el acceso concurrente.

Estas variables compartidas son esenciales para mantener y actualizar los datos de telemetría del dron, como la dirección (*heading*), la velocidad sobre el suelo (*Ground Speed*, GS), la velocidad vertical (*Vertical Speed*, VS), la altitud, la inclinación (*roll* y *pitch*) y las distancias a los obstáculos. Cada variable se inicializa con un tipo de dato específico, garantizando la precisión y consistencia de los datos compartidos.

Además de las variables compartidas mediante «multiprocessing», se definen otras variables globales que controlan el estado y las acciones del dron. Estas variables permiten controlar diversos aspectos del funcionamiento del dron, desde la detección de códigos ArUco específicos (despegue, aterrizaje y control) hasta la gestión de comandos de movimiento y el estado general del vuelo. Variables como «errorPitch», «errorRoll», «errorYaw» y «errorThrottle» son críticas para el sistema de control del dron, permitiendo calcular y corregir desviaciones en tiempo real.

La integración de estas variables globales asegura que todos los módulos y hilos del proyecto estén alineados y puedan acceder a la información necesaria para ejecutar sus respectivas funciones de manera eficiente. Por ejemplo, el hilo que maneja el movimiento del dron puede acceder y actualizar los errores de control en tiempo real, mientras que el proceso encargado de la telemetría puede proporcionar datos de sensores actualizados al sistema.

Script «ColoresEstilosFormatos.py», Apéndice G

El script **«ColoresEstilosFormatos.py»** define una serie de constantes que se utilizan para aplicar colores y estilos a los textos en la terminal, mejorando la legibilidad y la organización visual de los mensajes. Se definen códigos de escape para colores de texto (como negro, rojo, verde, azul, etc.), colores de fondo (como fondo negro, fondo rojo, fondo verde, etc.) y estilos (como negrita y subrayado). Además, incluye un código de reseteo para restaurar el formato por defecto de la terminal. Estas constantes son útiles para resaltar información importante, errores o advertencias, y para diferenciar visualmente diferentes tipos de mensajes durante la ejecución de la aplicación.

Script «isRotationMatrix.py», Apéndice H

El script **«isRotationMatrix.py»** incluye una función fundamental para la verificación de matrices de rotación en el espacio tridimensional. La función «isRotationMatrix» se encarga de determinar si una matriz dada cumple con las propiedades necesarias para ser considerada una matriz de rotación. Las matrices de rotación son matrices cuadradas que representan rotaciones en 3D y poseen características específicas: son ortogonales, tienen un determinante de $+1$ y conservan las normas de los vectores.

La lógica interna de la función implica calcular la transpuesta de la matriz proporcionada y multiplicarla por la matriz original. Luego, se compara este resultado con la

matriz identidad de 3x3. Si la norma de la diferencia entre estas matrices es menor que un umbral muy pequeño (1e-6), la matriz se considera una matriz de rotación y la función devuelve *True*. De lo contrario, devuelve *False*. Esta verificación es crucial para asegurar que las matrices utilizadas en el procesamiento de imágenes y en el cálculo de transformaciones espaciales sean válidas y precisas.

Script «rotationMatrixToEulerAngles.py», Apéndice I

El *script* **«rotationMatrixToEulerAngles.py»** incluye una función que es crucial para la conversión de matrices de rotación en ángulos de Euler.
La función «`rotationMatrixToEulerAngles`» toma una matriz de rotación 3x3 y la convierte en un conjunto de ángulos de Euler. Los ángulos de Euler son una forma de representar la orientación de un objeto en el espacio tridimensional mediante tres ángulos.

El proceso de conversión comienza verificando si la matriz proporcionada es una matriz de rotación válida utilizando la función «`isRotationMatrix`». Luego, se calcula la variable «`sy`», que es la raíz cuadrada de la suma de los cuadrados de dos elementos específicos de la matriz de rotación. Esta variable se utiliza para identificar posibles singularidades en la matriz. Si no se detecta ninguna singularidad, los ángulos de Euler se calculan utilizando funciones de «`atan2`» de la biblioteca «`math`». En caso de que haya una singularidad, se aplican cálculos especiales para obtener los ángulos de Euler. Finalmente, la función devuelve un arreglo de «`numpy`» con los ángulos de Euler en radianes. Esta conversión es esencial en aplicaciones de visión por computadora y robótica para interpretar y manipular orientaciones espaciales.

Script «socket_client_Proyecto.py», Apéndice J

El *script* **«socket_client_Proyecto.py»** es fundamental para establecer la comunicación entre el reComputer J3011 y el servidor, que puede estar ejecutándose localmente o en un dispositivo móvil Android. La función principal «`socket_client_Proyecto.py`» se encarga de crear y configurar un socket cliente utilizando el protocolo IPv4 y TCP para la transmisión de datos de manera confiable.

La función toma como parámetros la dirección IP y el puerto del servidor al que se desea conectar. Una vez configurados estos parámetros, se intenta establecer una conexión con el servidor. Si la conexión es exitosa, se informa al usuario y se retorna el objeto «`socket`», el cual se utilizará para enviar y recibir datos durante la ejecución del programa. En caso de que ocurra un error durante la conexión, se captura la excepción y se informa al usuario, retornando «`None`» para indicar que la conexión no se pudo establecer. Este mecanismo es crucial para la funcionalidad de la aplicación, ya que permite la comunicación continua y efectiva entre el cliente y el servidor.

Script «socket_server.py», Apéndice K

El *script* **«socket_server.py»** es crucial para gestionar la comunicación entre el servidor y múltiples clientes en el proyecto. La función principal «server_program» configura y ejecuta el servidor, utilizando «sockets» para escuchar conexiones entrantes en un puerto específico. Este servidor puede aceptar múltiples conexiones simultáneamente, utilizando hilos (*threads*) para manejar cada conexión de un cliente de manera independiente.

El *script* primero obtiene el nombre y la dirección IP del host, y luego crea y vincula un «socket» a esa dirección y puerto. El servidor se pone en modo de escucha, esperando conexiones entrantes. Cuando un cliente se conecta, se crea un hilo nuevo que ejecuta la función «gestor_clientes», la cual maneja la comunicación con el cliente específico. Esta función se encarga de recibir datos del cliente, procesarlos, y enviar respuestas apropiadas, mientras monitorea la actividad del cliente y maneja los tiempos de espera y desconexiones.

La estructura del servidor permite una gestión eficiente de múltiples clientes, asegurando que las conexiones activas se mantengan y se gestionen adecuadamente. La utilización de hilos asegura que cada cliente tenga un canal de comunicación independiente, lo que mejora la eficiencia y la capacidad de respuesta del sistema. Además, el *script* maneja adecuadamente las excepciones y proporciona mensajes informativos sobre el estado de las conexiones y el servidor.

Script «Telemetry.py», Apéndice L

El *script* **«Telemetry.py»** es crucial para la operación del proyecto, ya que se encarga de recibir y procesar la telemetría del dron en tiempo real. La función principal del *script* es «recibir_telemetria», que establece una conexión con el servidor de telemetría del dron y recibe datos continuamente, los cuales son luego procesados y almacenados en variables globales para su uso en otras partes del proyecto.

La función «recibir_bytes» asegura que se reciban la cantidad exacta de bytes necesarios para cada paquete de datos, manejando posibles interrupciones en la conexión. La función «conectar_server_telemetria» establece la conexión inicial con el servidor de telemetría, asegurando que la comunicación esté configurada correctamente.

Una vez establecida la conexión, el *script* entra en un bucle en el que recibe datos de telemetría, como la velocidad en tierra, la velocidad vertical, la altitud, el rumbo, la inclinación y las distancias a los obstáculos. Estos datos se desempaquetan y se almacenan en variables globales utilizando mecanismos de bloqueo para asegurar la consistencia de los datos.

Además, el *script* registra todos los mensajes de telemetría en un archivo de texto, proporcionando un registro detallado del estado del dron a lo largo del tiempo. En caso de pérdida de conexión, el *script* intenta reconectar automáticamente y continúa recibiendo datos una vez restablecida la conexión.

En resumen, **«Telemetry.py»** es fundamental para la monitorización y control del dron, asegurando que los datos de telemetría estén disponibles en tiempo real para la aplicación y permitiendo la realización de acciones basadas en estos datos.

Script «RasPi_gstreamer_pipeline.py», Apéndice M

El *script* **«RasPi_gstreamer_pipeline.py»** es fundamental para la configuración y operación de la cámara CSI en el proyecto, utilizando GStreamer para la captura de vídeo. Este *script* se compone de varias funciones clave que permiten la configuración detallada de la cámara, la visualización del vídeo en tiempo real y la obtención de las capacidades del sensor de la cámara.

La función principal del *script* es «`raspi_gstreamer_pipeline`», que crea un pipeline de GStreamer para capturar vídeo desde la cámara CSI. Esta función acepta numerosos parámetros que permiten ajustar la configuración de la cámara, incluyendo la resolución de captura y visualización, la tasa de *frames*, el método de volteo, la exposición, la ganancia, la saturación, el contraste, el brillo, el tono, el balance de blancos, la compensación de exposición, la nitidez y la reducción de ruido. El pipeline generado se utiliza para capturar y procesar el vídeo de la cámara en tiempo real.

La función «`show_camera`» utiliza el pipeline creado para abrir una ventana y mostrar el vídeo capturado en tiempo real. La función configura y abre la cámara, y entra en un bucle donde se capturan y muestran los *frames* hasta que se recibe una señal para cerrar la ventana.

Además, el *script* incluye la función «`get_camera_capabilities`», que ejecuta comandos del sistema para obtener y listar las capacidades del sensor de la cámara. Esta información se guarda en un archivo de texto para su consulta posterior.

Script «moveSimulador.py», Apéndice N

El *script* **«moveSimulador.py»** se encarga de la lógica de control para el modo simulador del dron. Este *script* crea un bucle de control que se ejecuta continuamente mientras la aplicación está en funcionamiento. Su propósito principal es supervisar los errores calculados en la orientación y posición del dron y enviar comandos correspondientes al servidor para corregir estos errores.

La función principal «`moveSimulador`» recibe como argumento un «`socket_cliente`», que es el canal de comunicación con el servidor. Dentro del bucle de control, se verifica si hay una nueva imagen procesada. Si es así, se realizan los siguientes pasos:

1. Copia de variables globales: Se copian localmente las variables globales relevantes como detecciones de códigos de despegue, aterrizaje y control, así como los errores de orientación (*Roll*, *Pitch*, Yaw) y altura (*Throttle*).

2. Detección y envío de comandos: Dependiendo de los errores detectados, se ajustan los valores de los errores en base a un umbral, de manera que se eliminan

los errores muy pequeños y se limitan los máximos. Basándose en estos errores ajustados, se envían comandos específicos al servidor para corregir la orientación y posición del dron. Estos comandos incluyen acciones para mover el dron hacia adelante, atrás, arriba, abajo, girar a la izquierda, a la derecha, y estabilizarlo.

3. Monitorización y ajuste: Se supervisan continuamente las variables globales de telemetría para registrar y ajustar el estado del dron en tiempo real. Esta información incluye la velocidad sobre el suelo, la velocidad vertical, la altitud, el rumbo, la inclinación (*Roll* y *Pitch*), y las distancias a los obstáculos.

El uso de pequeñas pausas «`time.sleep`» entre las iteraciones del bucle asegura que el ciclo de control no sobrecargue el sistema y permite un funcionamiento más eficiente. Al final del ciclo de control, se imprime en consola el estado de las variables de telemetría para monitorizar el rendimiento y las acciones del dron.

Script «moveVuelo.py», Apéndice Ñ

El *script* **«moveVuelo.py»** es fundamental para el control del dron durante un vuelo real, gestionando diferentes fases del vuelo como despegue, posicionamiento, avance y aterrizaje. El *script* emplea un bucle de control que monitoriza continuamente el estado de vuelo del dron, utilizando un diccionario de estados para ejecutar la función correspondiente según la fase del vuelo en la que se encuentra.

El diccionario de estados («`dicc_estados`») asocia cada estado de vuelo con su respectiva función: despegar, posicionarse, avanzar y aterrizar. La función «`switch`» se encarga de llamar a la función adecuada basándose en el estado actual de vuelo almacenado en la variable global «`estado_vuelo`».

Durante la fase de despegue, la función despegar envía un comando de despegue al dron y actualiza el estado de vuelo. En la fase de posicionamiento, se ajustan los errores de control (*roll*, *pitch*, *yaw* y *throttle*) para mantener el dron estabilizado. Esta fase utiliza umbrales para ignorar errores muy pequeños y limitar los máximos, asegurando movimientos precisos y seguros. La función posicionarse también determina el error más grande en valor absoluto y envía los comandos correspondientes para corregir dicho error.

La fase de avance se activa una vez que el dron está correctamente posicionado, enviando un comando para avanzar. Esta fase se mantiene hasta que el dron detecta un código de control o de aterrizaje. En la fase de aterrizaje, la función aterrizar envía el comando de aterrizaje y actualiza el estado del dron a «no_state».

En resumen, **«moveVuelo.py»** gestiona el control de vuelo del dron mediante un enfoque modular y basado en estados, asegurando que cada fase del vuelo se ejecute de manera ordenada y eficiente, adaptándose a las condiciones detectadas en tiempo real.

Script «ImageProcessingArUcoSimulador.py», Apéndice O

El *script* **«ImageProcessingArUcoSimulador.py»** está diseñado para procesar imágenes capturadas por una cámara y detectar marcadores ArUco en un entorno de simulación. Este *script* es fundamental para el funcionamiento del dron en modo simulador, ya que permite detectar y actuar sobre diferentes tipos de marcadores ArUco.

El *script* comienza con la importación de diversas bibliotecas necesarias para el procesamiento de imágenes, cálculos matemáticos y manejo de variables globales. La función principal, `«ImageProcessingSimulador»`, se encarga de analizar la imagen y detectar los marcadores ArUco.

Primero, la imagen se convierte a escala de grises para simplificar el procesamiento. Luego, se detectan los marcadores ArUco utilizando el diccionario y parámetros proporcionados. Si se detectan los marcadores deseados (despegue, control, aterrizaje), se estima su pose en el espacio tridimensional utilizando la matriz de la cámara y sus coeficientes de distorsión.

Para cada marcador detectado, se calculan las coordenadas de su centro y se estiman los vectores de rotación y traslación. Estos valores se utilizan para dibujar los ejes del marcador en la imagen, proporcionando una visualización de su orientación y posición relativa a la cámara.

Los errores de control (*roll*, *pitch*, *yaw*, *throttle*) se calculan basándose en la posición y orientación del marcador en la imagen. Se aplican umbrales para ignorar errores muy pequeños y limitar los errores máximos, asegurando movimientos precisos del dron. Estos errores se asignan a variables globales, que luego se utilizan en otras partes del programa para controlar el movimiento del dron.

Además, el *script* incluye funciones auxiliares para calcular la distancia de trabajo (`«workingDistance»`) y convertir el campo de visión diagonal en campo de visión horizontal (`«FOVdiag2FOVhori»`), las cuales son esenciales para la correcta interpretación de las imágenes y la geometría de la cámara.

Script «ImageProcessingArUcoVuelo.py», Apéndice P

El *script* **«ImageProcessingArUcoVuelo.py»** se encarga de procesar imágenes en tiempo real para detectar y gestionar los marcadores ArUco durante el vuelo del dron. Utiliza la biblioteca OpenCV para convertir las imágenes a escala de grises y detectar los marcadores ArUco. La detección de estos marcadores permite identificar diferentes estados y acciones del dron, como despegue, control y aterrizaje, mediante los identificadores específicos de cada marcador.

En el caso del marcador de despegue, si se detecta y el despegue no se ha ejecutado previamente, se cambia el estado del vuelo a «despegue». Para los marcadores de control, se calcula el centro del marcador y se estima la pose del marcador respecto a

la cámara, obteniendo los vectores de rotación y traslación. Esto permite calcular los errores de orientación (*Roll*, *Pitch*, *Yaw*) y distancia (*Throttle*) basados en la posición y orientación del marcador en la imagen. Estos errores se ajustan según unos umbrales definidos para eliminar errores muy pequeños y limitar los máximos, asegurando una respuesta controlada del dron.

En el caso del marcador de aterrizaje, se realiza un proceso similar al de despegue, detectando el marcador y calculando su pose. Si se encuentra en un estado adecuado, se cambia el estado del vuelo a «aterrizaje».

El script también incluye funciones auxiliares importantes para el procesamiento de imágenes y la geometría de la cámara. La función «`workingDistance`» calcula la distancia de trabajo del marcador a la cámara basada en el tamaño del marcador y el campo de visión horizontal de la cámara. La función «`FOVdiag2FOVhori`» convierte el campo de visión diagonal de la cámara en campo de visión horizontal, esencial para la correcta interpretación de las imágenes y la geometría de la cámara. Estas funciones permiten un procesamiento más preciso y una mejor interpretación de los datos visuales para el control del dron.

4.4.6 Consideraciones finales

Matriz de calibración de la cámara

Para la detección de los códigos ArUco, es esencial considerar que las funciones utilizadas, pertenecientes al módulo «`cv2.aruco`» de la biblioteca «`OpenCV`», requieren como argumento la matriz de calibración de la cámara. En el contexto de este proyecto, se ha empleado la cámara Raspberry Pi NoIR Camera V2, como se mencionó en secciones anteriores. Existen diversas metodologías para calcular la matriz de calibración de la cámara; en este caso, se ha optado por una calibración realizada con MATLAB.

La calibración en MATLAB para una cámara que captura imágenes con una resolución de 640x480 píxeles implica varios pasos. Inicialmente, se deben tomar múltiples imágenes de un patrón de referencia conocido, como un tablero de ajedrez, desde diferentes ángulos y posiciones. Estas imágenes se utilizan para identificar puntos de referencia en el patrón, que son posteriormente empleados en algoritmos de calibración que estiman los parámetros intrínsecos de la cámara, incluyendo la matriz de calibración y los coeficientes de distorsión.

Para la toma de las imágenes con la cámara Raspberry Pi NoIR Camera V2, se desarrolló un script en Python que permite capturar las imágenes necesarias para el proceso de calibración. Este script automatiza la captura de múltiples imágenes desde diferentes ángulos y posiciones, asegurando una cobertura adecuada del patrón de referencia. El código del script puede ser consultado en el Apéndice Q.

Alternativamente, el proceso de calibración también puede realizarse utilizando Python y la biblioteca «OpenCV», que ofrece herramientas robustas para este propósito. Otra opción sería emplear los propios códigos ArUco para la calibración. Además, existen métodos analíticos para determinar la matriz de calibración de la cámara, los cuales se describen a continuación.

Para calcular la matriz de calibración de la cámara, es esencial conocer los parámetros intrínsecos, los cuales pueden ser determinados utilizando el campo de visión angular (AFOV, por sus siglas en inglés). Específicamente, se requiere conocer los valores del AFOV horizontal (AFOVx) y vertical (AFOVy). En el caso de la Raspberry Pi NoIR Camera V2, estos valores son los siguientes [78]:

- AFOV horizontal (AFOVx): 62.2 grados
- AFOV vertical (AFOVy): 48.8 grados

Estos valores pueden ser convertidos a radianes para facilitar los cálculos:

- AFOVx = 62.2 grados = 1.085 radianes
- AFOVy = 48.8 grados = 0.851 radianes

El siguiente paso es el cálculo de las distancias focales fx y fy:

$$f_{\mathrm{x}} = \frac{W}{2 \cdot \tan\left(\frac{AFOV_{\mathrm{x}}}{2}\right)}$$

$$f_{\mathrm{y}} = \frac{H}{2 \cdot \tan\left(\frac{AFOV_{\mathrm{y}}}{2}\right)}$$

Donde:

- f_{x} es la distancia focal en píxeles en la dirección horizontal.
- f_{y} es la distancia focal en píxeles en la dirección vertical.
- W es el ancho de la imagen en píxeles, 640 píxeles.
- H es el alto de la imagen en píxeles, 480 píxeles.
- $AFOV_{\mathrm{x}}$ es el campo de visión angular horizontal en radianes.
- $AFOV_{\mathrm{y}}$ es el campo de visión angular vertical en radianes.

Calculo de f_{x}:

$$f_{\mathrm{x}} = \frac{640}{2 \cdot \tan\left(\frac{1{,}085}{2}\right)} = \frac{640}{2 \cdot \tan\left(0{,}5425\right)} = \frac{640}{2 \cdot 0{,}6082} \approx 526{,}12 \,\text{píxeles}$$

Cálculo de f_{y}:

$$f_{\mathrm{y}} = \frac{480}{2 \cdot \tan\left(\frac{0{,}851}{2}\right)} = \frac{480}{2 \cdot \tan\left(0{,}4255\right)} = \frac{480}{2 \cdot 0{,}4532} \approx 529{,}47 \,\text{píxeles}$$

Finalmente, el punto principal c_{x} y c_{y} se asume que está en el centro de la imagen:

$$c_{\mathrm{x}} = \frac{640}{2} = 320 \,\text{píxeles}$$

$$c_{\mathrm{y}} = \frac{480}{2} = 240 \,\text{píxeles}$$

El coeficiente de sesgo s generalmente es 0 para la mayoría de las cámaras:

$$s = 0$$

Así, la matriz de parámetros intrínsecos K es:

$$K = \begin{pmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 526{,}12 & 0 & 320 \\ 0 & 529{,}47 & 240 \\ 0 & 0 & 1 \end{pmatrix}$$

Puertos por los que opera el servidor

Para convertir la aplicación móvil en un servidor, se añaden dos *scripts* desarrollados en Java al código fuente de la misma. Los nombres de estos *scripts* y sus respectivas funcionalidades son las siguientes:

- **«TCPServer.java»**: Implementa un servidor en Java que opera a través del puerto 8000 y se encarga de recibir comandos provenientes del ordenador embarcado. La función principal de este *script* es habilitar un puerto en el servidor que permita la recepción de comandos. Una vez que los comandos llegan a la aplicación móvil, el esquema de comunicaciones con el dron sigue siendo el mismo descrito anteriormente, ver Figura 1.8.

- **«CockpitServer.java»**: Habilita el puerto 9007 para enviar información al ordenador embarcado. Su función es reenviar, a través del puerto 9007 del servidor implementado en la aplicación móvil, los datos de telemetría que el dron envía a la aplicación móvil a través del puente.

La comunicación entre el cliente (ordenador embarcado) y el servidor (aplicación móvil) se efectúa vía WiFi, es decir, a través de Internet. Para el correcto funcionamiento del sistema, tanto el cliente como el servidor deben estar conectados a la misma red local. Esto se puede lograr conectando la aplicación móvil y el ordenador embarcado al mismo *router*, o bien compartiendo un punto de acceso a Internet desde un dispositivo externo y conectando tanto el cliente como el servidor a este punto de acceso.

4.5 Embarcado del conjunto en el dron

En esta sección, se abordará el proceso de embarcado del reComputer J3011 en el dron. Es importante destacar que no solo se debe embarcar el reComputer J3011, sino también la cámara Raspberry Pi NoIR Camera V2, que está conectada a este, y la batería que proporciona energía al reComputer J3011 durante el vuelo. Cabe mencionar que, tal y como se explicó en capítulos precedentes de esta memoria, todo este equipo se embarcará en un dron DJI Phantom 4 Advanced+.

Los drones DJI son dispositivos privativos, diseñados con especificaciones estrictas y optimizados para operar con su propio sistema de estabilización y una distribución de peso predefinida. Estos drones están concebidos para funcionar eficientemente dentro de los parámetros y condiciones para los que fueron diseñados. Cualquier modificación, como la adición de instrumentos embarcados, altera las condiciones originales de operación, lo que puede desestabilizar el sistema y afectar negativamente a su rendimiento.

La incorporación de elementos adicionales, como el reComputer J3011, la cámara Raspberry Pi NoIR Camera V2 y la batería, introduce variables desconocidas que no fueron consideradas en el diseño original del dron. Estas adiciones pueden modificar el centro de gravedad y la aerodinámica del dispositivo, lo que incrementa el riesgo de fallos durante el vuelo. La estabilidad del dron podría verse comprometida, aumentando la probabilidad de accidentes y daños al equipo. Por lo tanto, es crucial evaluar cuidadosamente las implicaciones de añadir peso y componentes adicionales a drones que no están específicamente diseñados para tales modificaciones.

Considerando las limitaciones previamente mencionadas en cuanto a la modificación del peso y la aerodinámica de los drones DJI, es relevante destacar que tanto en los foros oficiales de DJI como el propio fabricante, afirman que el Phantom 4 Advanced+ está diseñado para portar hasta 1 kg de peso adicional sin comprometer la seguridad en vuelo ni poner en riesgo la integridad de la aeronave.

Para corroborar que la máxima carga útil es de 1 kg, la empresa DRONExpert, ver [79], llevó a cabo un estudio de carga utilizando un Phantom 3, tal como se detalla en la referencia [80]. En este estudio, se verificó que 1 kg es la máxima carga de pago que se puede embarcar sin comprometer significativamente el rendimiento del dron. Cargas superiores a 1 kg, aunque técnicamente manejables por el dron, resultaron en un considerable aumento del consumo energético, lo que redujo drásticamente el tiempo de vuelo. Además, estas cargas superiores afectaron negativamente la estabilidad y el control de la aeronave, comprometiendo su manejo seguro. A pesar de que el estudio se realizó con un Phantom 3, los resultados son extrapolables al Phantom 4 Advanced+, objeto del presente TFM, ya que ambos modelos poseen dimensiones y características estructurales muy similares. Por lo tanto, las conclusiones sobre la carga máxima de 1 kg son aplicables al Phantom 4 Advanced+.

Esta capacidad de soportar hasta 1 kg adicional es precisamente lo que hace viable este proyecto. Sin esta posibilidad, el Trabajo de Fin de Máster (TFM) perdería su sentido, ya que no sería viable embarcar los componentes necesarios a bordo del dron. Por lo tanto, la afirmación del fabricante de que el Phantom 4 Advanced+ puede manejar esta carga adicional es fundamental para la realización y justificación del presente TFM. Esta capacidad permite integrar los equipos y sistemas necesarios sin comprometer la operatividad y seguridad del dron, lo que otorga validez y viabilidad al proyecto.

El hecho de que el dron pueda embarcar hasta 1 kg adicional no implica que esta carga pueda ser incorporada de cualquier manera. Es esencial realizar un estudio exhaustivo para determinar la forma más eficiente de integrar todos los elementos necesarios, con el fin de evitar alterar significativamente el centro de gravedad original del dron.

Este análisis debe garantizar que la distribución del peso adicional no comprometa la estabilidad y la maniobrabilidad del dron. Aunque el Phantom 4 Advanced+ está diseñado para soportar esta carga adicional, una mala distribución del peso puede llevar a problemas de equilibrio y control durante el vuelo, incrementando el riesgo de accidentes y daños a la aeronave. Por lo tanto, es crucial que el proceso de embarque se realice de manera cuidadosa y meticulosa, asegurando que la operatividad y la seguridad del dron no se vean comprometidas. Esta consideración es fundamental para la viabilidad del proyecto, ya que asegura que la incorporación de los componentes necesarios se realice de manera segura y efectiva.

El objetivo de la presente sección es llevar a cabo estudios de viabilidad para embarcar todos los equipos necesarios sin alterar significativamente las variables y parámetros para los que el Phantom 4 Advanced+ fue diseñado. Estos estudios buscan demostrar que es posible añadir hasta 1 kg de peso adicional, como afirma DJI, sin comprometer la seguridad y el rendimiento del dron. Además, tienen como finalidad verificar la veracidad de las afirmaciones de DJI respecto a la capacidad de carga del dron. A menudo, las empresas del sector pueden exagerar o ajustar los datos sobre las modificaciones que los usuarios pueden realizar en sus productos, los cuales no siempre están diseñados para soportar tales cambios. Por lo tanto, es crucial realizar un estudio riguroso

que no solo garantice la seguridad del dron, sino que también confirme la precisión de las especificaciones proporcionadas por el fabricante.

4.5.1 Análisis del estado del arte

La primera actividad llevada a cabo fue una revisión detallada del estado del arte relacionado con el embarque de componentes. Se investigó en diversas fuentes en línea para comprender cómo otros usuarios han abordado el embarque de componentes en el dron DJI Phantom 4 Advanced+. Esta revisión incluyó la consulta de foros, artículos técnicos y publicaciones especializadas para identificar las soluciones y estrategias empleadas por la comunidad. El objetivo era recopilar información sobre las mejores prácticas y posibles desafíos asociados con la integración de equipos adicionales en este modelo específico de dron.

Para el modelo específico de dron DJI Phantom 4 Advanced+, se encontraron algunas piezas sueltas diseñadas e impresas en 3D, como soportes para cámaras, protectores para las hélices y otros componentes menores. Sin embargo, no se halló ninguna información ni diseño que proporcionara un soporte adecuado para el tipo de embarque que se pretende llevar a cabo en este TFM, es decir, la integración del reComputer J3011, la batería y la cámara Raspberry Pi NoIR Camera V2.

Aunque el estudio del estado del arte no arrojó soluciones completas para el embarque del reComputer J3011, la batería y la cámara Raspberry Pi NoIR Camera V2, sí se encontró el diseño en 3D de un soporte para integrar un gimbal en el Phantom 4 Advanced+, como se muestra en la Figura 4.11. Aunque este soporte específico no es adecuado para embarcar todos los componentes requeridos, sirvió como fuente de inspiración para desarrollar un diseño propio. Utilizando ciertas piezas del soporte para la cámara, se empleó este diseño como punto de partida para la creación de un soporte adecuado para el TFM que permitiese embarcar todos los componentes necesarios para este proyecto: el reComputer J3011, la batería y la cámara.

4.5.2 Diseños preliminares en FreeCAD

Después del análisis del estado del arte, cuyo objetivo fundamental era la búsqueda de ideas que permitieran abordar el tema del embarcado, se comenzó con el diseño en 3D de los primeros prototipos de soportes para embarcar los componentes necesarios. Para el presente TFM, todos los diseños en 3D se realizaron en FreeCAD. La elección de FreeCAD sobre otros programas privativos como CATIA, SolidWorks o Fusion 360 se debió a varias razones. En primer lugar, FreeCAD es un software libre y gratuito, lo que lo hace accesible para cualquier investigador sin necesidad de incurrir en costos adicionales. Además, FreeCAD es multiplataforma, permitiendo trabajar en ordenadores con diferentes sistemas operativos, lo cual es una ventaja significativa en entornos de investigación donde se utilizan diversos tipos de hardware y software. Otra ventaja

Figura 4.11: Soporte para cámara 360 para DJI Phantom 4 y Phantom 4 Pro [81].

de FreeCAD es su comunidad activa de usuarios y desarrolladores, que ofrece soporte continuo y actualizaciones frecuentes, asegurando que el software evolucione y mejore constantemente.

Los primeros prototipos diseñados adoptaron una filosofía ligeramente diferente al soporte para el gimbal mostrado en la Figura 4.11. El principal inconveniente de este soporte radicaba en que impedía la capacidad de despegue y aterrizaje del dron sobre una superficie horizontal. Esto se debe a que el soporte se extiende hacia abajo, más allá de las patas del dron, lo que provoca que choque contra el suelo. Por lo tanto, para despegar con este tipo de soporte incorporado, sería necesario elevar el dron sobre algún objeto que permita un espacio libre entre las patas y el suelo, evitando así el contacto del soporte con la superficie. Esta limitación planteaba desafíos significativos para la operatividad del dron en condiciones normales de vuelo y aterrizaje.

Para superar el desafío previamente mencionado, se concibió un diseño de soportes que se anclan a los laterales del dron. De esta manera, manteniendo los elementos necesarios embarcados, se permite que el dron despegue y aterrice de forma natural, resolviendo así el problema planteado por el soporte para el gimbal anterior. Este enfoque evita que los soportes choquen contra el suelo al despegar o aterrizar.

Este nuevo enfoque de diseño dio lugar al prototipo mostrado en la Figura 4.12. Este prototipo cuenta con dos compartimentos, cada uno ubicado a un lado del dron. En uno de los compartimentos se alojaría la batería, mientras que en el otro se integraría

el reComputer J3011 con la cámara. Cabe destacar que en la Figura 4.12 se muestra solo uno de los dos compartimentos, ya que el diseño es el mismo para ambos. La única diferencia radica en las dimensiones de los compartimentos, que deben ajustarse a las especificaciones de los elementos embarcados.

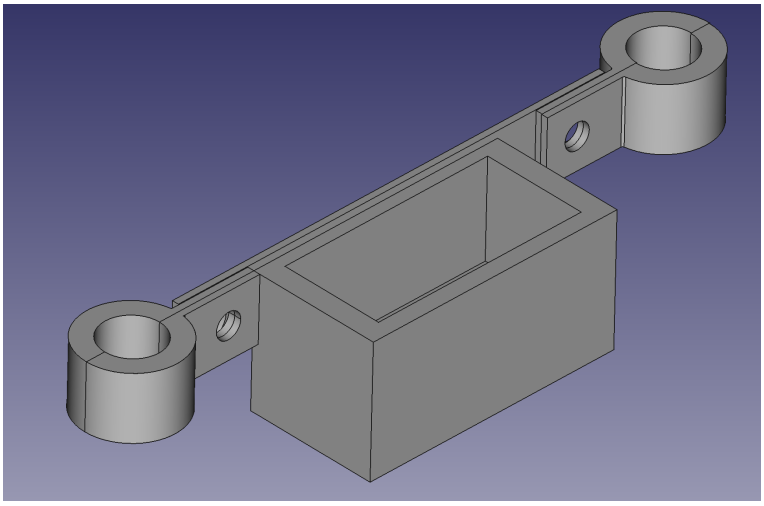


Figura 4.12: Primer prototipo para la tarea de embarque en el dron [Elaboración Propia].

No obstante, este prototipo fue finalmente descartado debido a la diferencia de pesos entre los elementos embarcados. El reComputer J3011 destaca por su ligereza, y esta se puede aumentar aún más al quitar la carcasa de aluminio, dejando solo el módulo Jetson Nano en su interior. Sin embargo, la batería utilizada en este proyecto es bastante pesada y no se puede aligerar de ninguna manera.

El problema radica en que el peso del compartimento de la batería es significativamente mayor que el del compartimento del Jetson Nano, lo que crea un desequilibrio en el dron y genera un momento que tiende a volcarlo. Se intentó solucionar este problema añadiendo más peso al compartimento del Jetson Nano, pero esto resultó en un peso total embarcado que superaba el límite de un kilogramo, establecido como máximo.

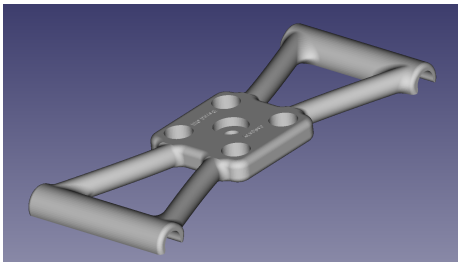
Debido a esta razón, y para evitar diseñar un soporte que provocara un desequilibrio en el dron, se decidió descartar la filosofía de diseño de los compartimentos laterales.

Por lo tanto, la única vía de diseño posible hasta ahora fue volver a la filosofía del soporte para el gimbal anterior, es decir, diseñar un soporte para los elementos embarcados que estuviera ubicado debajo del dron. Como se mencionó anteriormente, este tipo de diseño dificultaba el despegue y el aterrizaje, ya que estos deberían realizarse desde superficies lo suficientemente elevadas para evitar que los compartimentos diseñados tocaran el suelo. En otras palabras, el dron tendría que despegar y aterri-

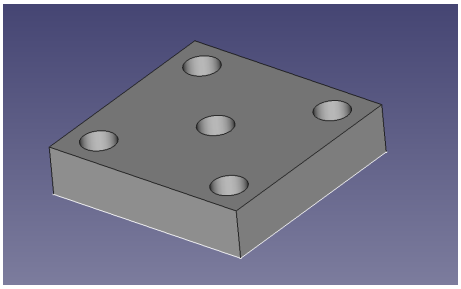
zar con sus patas apoyadas en una plataforma alta para prevenir el contacto de los compartimentos con el suelo.

Para solucionar el problema del despegue y aterrizaje desde una superficie elevada, que plantea el diseño anteriormente mencionado donde el soporte se acopla por debajo del dron, muchos usuarios dentro de la comunidad han buscado soluciones innovadoras mediante la creación de piezas en 3D, ver [82] y [83]. Estas piezas actúan como extensores de la superficie de aterrizaje del dron, proporcionando una mayor elevación. Estos extensores se adhieren a las patas del dron, incrementando la altura del mismo y permitiendo que los compartimentos adicionales no entren en contacto con el suelo durante el despegue y el aterrizaje. Esta solución no solo mantiene el dron elevado, sino que también asegura una mayor estabilidad al despegar y aterrizar, minimizando riesgos de daño a los componentes embarcados. Esta estrategia ha sido adoptada ampliamente debido a su efectividad y facilidad de implementación, ofreciendo una solución práctica para los desafíos presentados por la adición de componentes debajo del dron.

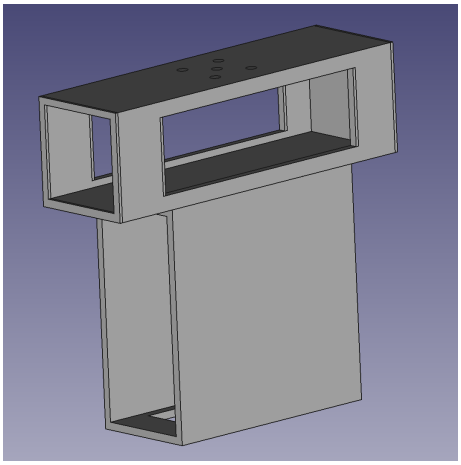
De esta nueva filosofía surge el prototipo mostrado en la Figura 4.13. Este prototipo utiliza la pieza puente del soporte para el gimbal de la cámara, ver Figura 4.13a, que une las dos patas del dron. Sobre esta pieza puente se diseñó un conector, ver Figura 4.13b, que une el puente con dos compartimentos, ver Figura 4.13c, situados debajo del dron, uno encima del otro. El compartimento superior está destinado a albergar la batería, mientras que el compartimento inferior aloja el reComputer J3011. Es importante señalar que este segundo prototipo permite mantener la configuración del centro de gravedad de la aeronave, evitando así el problema del momento de vuelco que se presentaba en el primer prototipo.



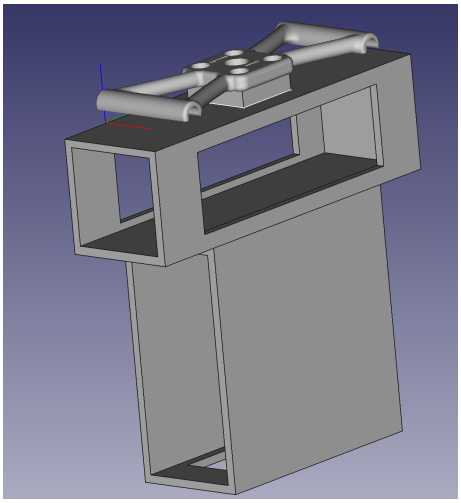
(a) Pieza puente del soporte [81].



(b) Conector [Elaboración Propia].



(c) Compartimentos [Elaboración Propia].



(d) Montaje final [Elaboración Propia].

Figura 4.13: Segundo prototipo para la tarea de embarque en el dron [Elaboración propia].

Una vez planteado el prototipo, el siguiente paso era el diseño en 3D de las piezas necesarias. Sin embargo, esto no se llevó a cabo por dos razones principales. La primera fue la dificultad para acceder a impresoras 3D adecuadas para realizar la impresión de las piezas. La segunda, y más importante, fue el tema del tiempo. Hasta el momento, la tarea de embarcado había consumido más recursos temporales de los inicialmente previstos. Demorar aún más el proyecto buscando y realizando las impresiones en 3D se volvió inviable para este TFM. Por lo tanto, se optó por buscar una alternativa que permitiera agilizar el proceso de embarcado en el dron.

4.5.3 Solución alternativa

Esta solución alternativa, ver Figura 4.14, surge de los problemas descritos anteriormente. Su objetivo es satisfacer la necesidad de embarcar el reComputer J3011 en el dron de una manera más rápida y eficiente, sin prolongar excesivamente la duración del TFM.

Para implementar esta solución, se adquirieron varios materiales: una caja de registro, una plancha de metal, un conjunto de tornillos y unas abrazaderas de aluminio. Mediante técnicas básicas de bricolaje, se logró montar la caja de registro en el dron utilizando las abrazaderas y los tornillos. Esta configuración inicial permitió asegurar de manera efectiva la caja al dron, proporcionando una estructura estable para las pruebas de vuelo.

En las primeras pruebas de vuelo, se utilizó la caja de registro, ver Figura 4.14a. Sin embargo, se observó que la plancha de metal ofrecía ventajas significativas en términos de facilidad y rapidez para la realización de pruebas. Por esta razón, la caja de registro fue posteriormente reemplazada por la plancha de metal, lo que permitió agilizar el proceso y mejorar la eficiencia en las pruebas subsecuentes, ver Figura 4.14b.

(a) Solución alternativa con la caja de registro [Elaboración propia].

(b) Solución alternativa con la plancha de metal [Elaboración propia].

Figura 4.14: Solución alternativa para el embarcado del reComputer J3011 en el dron [Elaboración Propia].

Las primeras pruebas de vuelo, utilizando la plancha de metal como soporte, se realizaron sin ningún peso adicional ni elementos embarcados. Dado que estas eran las primeras pruebas con una modificación estructural en el dron, el objetivo principal era evaluar la estabilidad y la respuesta del dron ante la incorporación de la nueva estructura. Solo se emplearon la plancha de aluminio y los tornillos de anclaje, evitando agregar peso extra. Esta decisión se tomó para proceder de manera progresiva, incrementando gradualmente la complejidad de las pruebas y evaluando el comportamiento del dron sin comprometer la seguridad de la operación.

La realización de estas primeras pruebas reveló los primeros problemas. Si bien el dron era capaz de despegar y ser pilotado manualmente sin inconvenientes, surgían dificultades durante los despegues automáticos. En estos casos, el dron no lograba mantener una altura constante respecto al suelo, sino que ascendía de manera indefinida sin detenerse.

Tras una exhaustiva investigación, se descubrió la causa del problema. El DJI Phantom 4 Advanced+ está equipado con sensores de posición en su parte inferior, que utiliza para calcular la distancia al suelo y mantener una altura estable. Sin embargo, al añadir la plancha de metal debajo del dron, el haz de estos sensores intersectaba con la plancha. Esto provocaba que el dron interpretara la plancha como el suelo. La distancia entre los sensores y la plancha era constante y muy pequeña. Debido a esto, el dron intentaba ascender continuamente para corregir lo que percibía como una altura inadecuada. Sin embargo, dado que la distancia entre los sensores y la plancha no cambiaba, el dron seguía subiendo indefinidamente en un intento de ajustarse a una altura que nunca alcanzaba.

Ante este problema, surgieron dos posibles vías de desarrollo. La primera implicaba el rediseño completo del soporte, mientras que la segunda consistía en averiguar el ángulo del haz de los sensores para, manteniendo el diseño actual, reposicionar la plancha de metal de manera que no interfiriera con estos. Esto podría lograrse desplazando la plancha hacia adelante o hacia atrás.

Se consultó el manual del usuario del dron, ver [35], y se descubrió que el haz de los sensores de posicionamiento tiene un ángulo de 60 grados. Con esta información, se decidió desplazar la plancha de metal hacia adelante para evitar la colisión entre el haz del sensor y la plancha.

A continuación, se llevaron a cabo nuevas pruebas en vuelo y se comprobó que el problema se solucionaba al desplazar la plancha de metal hacia adelante. Con este problema resuelto, se procedió a realizar pruebas en vuelo con carga. Sobre la plancha de aluminio se colocó un peso de aproximadamente 300 gramos para analizar el comportamiento del dron. Se eligió empezar con un peso reducido, de 300 gramos, comparado con el kilogramo máximo admitido, para evaluar progresivamente el comportamiento del dron.

Durante estas pruebas, se observó que el dron presentaba problemas de estabilidad. Aunque no se produjo ningún incidente, el dron tendía a desplazarse hacia adelante e inclinarse hacia el lado donde se había colocado la carga. Este comportamiento indicaba que la distribución del peso en la parte frontal de la plancha provocaba un desequilibrio, lo que hacía que el dron intentara volcarse en esa dirección. Esta observación resaltó la necesidad de ajustar la ubicación de la carga para mejorar la estabilidad del dron en futuras pruebas.

Todas estas pruebas y los problemas surgidos subrayan la complejidad inherente al diseño de un soporte adecuado para este dron. Primero, es crucial que la ubicación del soporte no interfiera con los sensores del dron. Cualquier intersección con la sensorización puede afectar gravemente la capacidad del dron para mantener una altura y orientación correctas. Segundo, es igualmente importante que, una vez que se evita la interferencia con los sensores, la distribución del peso en el soporte se configure de manera que se mantenga el centro de gravedad del dron lo más equilibrado posible. Como se ha demostrado, incluso pequeños cambios en el centro de gravedad pueden provocar inestabilidad, afectando el rendimiento y la seguridad del dron en vuelo. Estos dos desafíos resaltan la delicada tarea de diseñar un soporte que garantice tanto la funcionalidad de los sensores como la estabilidad del dron.

4.5.4 Pruebas de embarcado finales

Las pruebas anteriores y los problemas derivados de las mismas resultaron en una considerable demora en la realización del TFM. Abordar correctamente el problema del embarcado, con el objetivo de encontrar una solución robusta y eficiente, implicaría un replanteamiento exhaustivo de todo el enfoque del proyecto. Este proceso de reenfoque excede tanto los límites temporales como el nivel de dificultad previstos para el presente trabajo. Por lo tanto, es necesario reconocer que, aunque se han identificado y analizado los desafíos, resolverlos completamente requeriría un esfuerzo adicional significativo que va más allá del alcance de este proyecto.

A pesar de estas dificultades, se han realizado pruebas en vuelo con el reComputer J3011 embarcado, implementando el embarque del mismo de la siguiente manera. Se mantuvo la plancha de metal como soporte para el embarcado de los componentes. La plancha de metal se colocó en una posición que no interfiriese con los haces de los sensores del dron. Para abordar el problema del desequilibrio causado por la adición de elementos y evitar la alteración del centro de gravedad del dron, se decidió realizar las pruebas finales sin la batería, debido a su considerable peso.

En lugar de embarcar la batería, se optó por alimentar el reComputer J3011 mediante un cable de corriente lo suficientemente largo para permitir el vuelo del dron, manteniendo el dispositivo conectado a una toma de corriente. Este cable se fijó al dron con una brida para evitar interferencias. Además, se desmontó la carcasa de aluminio del

reComputer J3011, embarcando solo el módulo Jetson Nano, que se fijó a la plancha de metal con velcro de doble cara.

El módulo Jetson Nano se colocó lo más centrado y retrasado posible en la plancha, a pesar de que esta esté adelantada para evitar la interferencia con la sensorización del dron, con el fin de mantener el centro de gravedad del sistema lo más equilibrado posible. Toda esta configuración de montaje se ilustra en la Figura 4.15.

Antes de realizar las primeras pruebas en vuelo con el Jetson Nano ya embarcado en el dron, era fundamental establecer todo el sistema de comunicaciones que permitiría dar soporte al vuelo. A continuación, se recuerda todo este sistema de comunicaciones. Como se comentó en secciones previas de este trabajo, una vez embarcado el reComputer J3011 en el dron, la única forma de comandarlo es de forma remota a través de VNC. Por lo tanto, el primer paso fue configurar la conexión VNC entre el reComputer J3011 y un ordenador en tierra.

Durante la configuración de esta conexión surgió un problema inesperado: la configuración por defecto de VNC solo permite el control remoto desde un ordenador externo si el reComputer J3011 está conectado a una pantalla externa vía HDMI. Para solucionar este problema, se consideraron dos soluciones comunes en el estado del arte: modificar los archivos de configuración de Linux o utilizar un conector HDMI fantasma. Este conector es una clavija que se conecta al puerto HDMI del reComputer J3011 y engaña al sistema haciéndole creer que está conectado a una pantalla externa, cuando en realidad no es así. Esta última solución fue la que se implementó en el presente proyecto.

Una vez configurada la conexión VNC, que permite comandar el reComputer J3011 desde un ordenador externo, y con el dron ya en vuelo, se ejecutó la aplicación de procesamiento de imágenes desarrollada en Python, que corre dentro del reComputer J3011. Esta aplicación, como se mencionó en secciones previas, captura constantemente *frames* del entorno y, en función de la detección o no de códigos ArUco, envía una serie de cadenas de texto a un servidor. La aplicación móvil instalada en el mando del dron y desarrollada con el DJI SDK actúa como servidor, recibe y procesa estas cadenas de texto, y envía comandos al dron para que ejecute acciones de movimiento. Es importante destacar que estos comandos se envían al dron a través del mando, que actúa como intermediario y transmite los comandos al dron por radiofrecuencia.

Llegados a este punto, es crucial destacar la complejidad del sistema de conexiones implementado en este escenario. En primer lugar, existe una conexión vía WiFi entre dos ordenadores: el reComputer J3011 y un ordenador externo desde el cual se controla el reComputer J3011 a través de VNC. Además, se establecen dos conexiones adicionales: otra conexión WiFi entre el reComputer J3011 y el servidor (la aplicación móvil), y una conexión por radiofrecuencia entre el mando del dron y el propio dron.

Figura 4.15: Montaje final del reComputer J3011 en el dron Phantom 4 Advanced+ [Elaboración Propia].

Es importante señalar que el reComputer J3011 gestiona simultáneamente dos conexiones WiFi: una con el ordenador externo a través de VNC, que permite su control remoto, y otra con el servidor (aplicación móvil), que recibe y envía comandos al reComputer J3011.

Finalmente, tal como se mencionó anteriormente en esta memoria, para que todo el sistema de comunicaciones funcione correctamente, es fundamental que todas las conexiones WiFi se realicen con todos los dispositivos conectados al mismo router o a un mismo punto de acceso, a través del cual se comparte la conexión a Internet.

En la Figura 4.16 se presenta el complejo esquema de comunicaciones. En esta figura se puede observar cómo un ordenador externo recibe de forma remota la imagen del reComputer J3011 a través de VNC. Además, se muestra cómo el sistema de visión del propio dron envía la imagen capturada al mando del dron. Este esquema ilustra claramente que las conexiones entre todos los elementos del sistema se han establecido de manera satisfactoria, permitiendo una comunicación fluida y efectiva entre los distintos componentes involucrados.

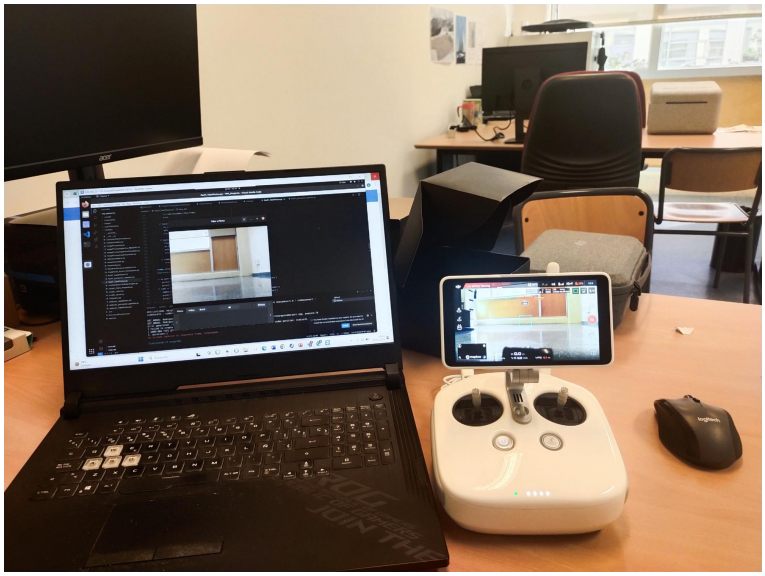


Figura 4.16: Verificación final de las conexiones durante las pruebas en vuelo [Elaboración Propia].

En Figura 4.17 se muestra una imagen de las pruebas en vuelo finales realizadas, ilustrando los resultados obtenidos y el contexto en el que se desarrollaron dichas pruebas.

Figura 4.17: Prueba en vuelo final [Elaboración Propia].

Capítulo 5

Conclusiones y trabajos futuros

El presente capítulo tiene como objetivo presentar, a modo de resumen final, las conclusiones derivadas de la realización de este TFM, así como el amplio abanico de posibilidades que se abren para futuros trabajos de investigación y desarrollo. Se pretende sintetizar los hallazgos más significativos obtenidos durante el proyecto y proponer diversas líneas de trabajo que podrían ser exploradas para continuar y expandir el alcance de los resultados aquí obtenidos.

5.1 Conclusiones

En primer lugar, se destaca que en la presente sección se recogen las conclusiones más significativas e importantes derivadas del proyecto. Uno de los objetivos principales al redactar esta memoria ha sido mantener un enfoque didáctico, facilitando al lector una comprensión adecuada de los conceptos tratados y de las decisiones adoptadas. Por esta razón, a lo largo del documento, se ha procurado incluir pequeñas conclusiones intermedias, cuyo propósito ha sido ayudar al lector a entender el contexto y el ecosistema de trabajo de manera progresiva y clara.

1. **Complejidad del ecosistema y entorno de desarrollo del TFM**

    - Una de las conclusiones más importantes de este proyecto es que su realización ha puesto de manifiesto la complejidad inherente a este tipo de iniciativas, donde intervienen múltiples elementos. Este trabajo ha demostrado lo desafiante que es operar en un ecosistema en el que convergen tanto componentes de *hardware* como de *software*. Como habrá podido apreciar el lector, el contexto del presente TFM se desarrolla en un entorno multidisciplinar, donde se integran diversos elementos. Entre ellos se

destacan los drones empleados (DJI Mavic Mini y DJI Phantom 4 Advanced+), el ordenador embarcado (reComputer J3011), la cámara (Raspberry Pi NoIR Camera V2), el soporte que acopla el ordenador al dron, el teléfono móvil con la aplicación desarrollada con el SDK de DJI, la propia aplicación móvil, el mando del dron, los diversos sistemas de comunicaciones entre estos componentes, y los dos lenguajes de programación utilizados (Java y Python). La realización de este proyecto ejemplifica la esencia de la ingeniería, que no es una rama unidisciplinar, sino multidisciplinar. El desarrollo de este proyecto ha requerido la integración y correcto funcionamiento de diversas disciplinas, lo cual añade un grado significativo de dificultad. No solo se trata de que cada componente funcione adecuadamente de manera individual, sino también de que todos estos componentes se integren y operen de manera conjunta y eficiente. El éxito del proyecto y la consecución de los objetivos planteados dependen de la sinergia entre estos múltiples elementos, que deben funcionar bien tanto de manera individual como colectiva.

2. **Demostración del adecuado funcionamiento de todas las comunicaciones presentes**

   - Una de las conclusiones más importantes de este proyecto es la demostración del correcto funcionamiento de todas las comunicaciones entre los distintos elementos del sistema. Como se mencionó anteriormente, una de las mayores dificultades del presente TFM ha sido su carácter multidisciplinar. Dentro de este contexto, lograr una resolución satisfactoria de las comunicaciones entre todos los componentes del sistema ha sido un logro significativo. Entre estas comunicaciones, se destacan las siguientes:

     - **Comunicación entre el dron y el mando a través de radiofrecuencia:** Este enlace permite la transmisión de datos de control y telemetría entre el dron y su mando, asegurando una operación precisa y segura del dron.
     - **Comunicación entre el mando del dron y el móvil Android:** La aplicación móvil desarrollada con el SDK de DJI está instalada en un dispositivo Android. La comunicación entre el mando y el móvil se realiza mediante un cable USB, transfiriendo datos cruciales de la operación del dron a la aplicación móvil.
     - **Comunicación entre el ordenador embarcado y la aplicación móvil:** Este enlace se resuelve mediante un esquema cliente-servidor, donde el ordenador embarcado (reComputer J3011) actúa como el cliente y la aplicación móvil como el servidor. Este esquema permite la transmisión y recepción de datos necesarios para el procesamiento de imágenes y otras funciones críticas.

- **Comunicación entre el ordenador embarcado reComputer J3011 y un ordenador externo:** A través de VNC vía WiFi, es posible controlar el reComputer J3011 remotamente durante las operaciones en vuelo. Esta comunicación es esencial para la supervisión y el control remoto del sistema embarcado.

- Para garantizar el correcto funcionamiento de todo el sistema de comunicaciones, es imprescindible que todos los elementos estén conectados a la misma red WiFi local. Esto asegura una coordinación fluida y eficiente entre todos los componentes del sistema.

- En conclusión, este TFM ha demostrado la efectividad de todas y cada una de las comunicaciones mencionadas, estableciendo una infraestructura robusta para las comunicaciones que puede ser utilizada y mejorada en futuros trabajos y desarrollos relacionados con este proyecto.

3. **Resolución del problema de la transferencia de la imagen**

   - La tercera conclusión del presente TFM se centra en la resolución del problema de la transferencia de imágenes. Una de las motivaciones clave de este proyecto era abordar un problema común en el estado del arte actual: las imágenes captadas por los sistemas de visión del dron y enviadas al mando vía radiofrecuencia, que luego se transfieren por cable USB al móvil con la aplicación Android, no permiten un procesamiento posterior debido a problemas de calidad, retardo y lag.

   - La solución más efectiva, adoptada en este TFM, consistió en embarcar un ordenador a bordo del dron (reComputer J3011) con su propio sistema de visión (Raspberry Pi NoIR Camera V2) y realizar el procesamiento de las imágenes directamente en el ordenador embarcado, sin necesidad de transferirlas a la aplicación del usuario para su procesamiento. Esto significa que no es necesario transmitir las imágenes, ya que se procesan directamente en el ordenador embarcado.

   - El presente TFM ha demostrado que esta vía es factible y funciona de manera precisa y robusta. Mediante el desarrollo de una aplicación de procesamiento de imágenes en Python, se ha demostrado que es posible realizar el procesamiento de las imágenes en el ordenador embarcado, permitiendo un procesamiento posterior efectivo. La tarea de procesamiento posterior explorada en este proyecto ha sido la detección de códigos ArUco y, en función de esta detección, el envío de una serie de cadenas de texto al servidor (aplicación móvil), que se encarga de transmitir, a través del mando, una serie de comandos de movimiento al dron.

- Además, se ha demostrado la utilidad y robustez de los códigos ArUco en este tipo de tareas de control de drones. Esta metodología no solo resuelve el problema de la transferencia de imágenes, sino que también demuestra que, utilizando el SDK de DJI, es posible explorar opciones más avanzadas y salirse del estricto ecosistema dictaminado por DJI, abriendo nuevas posibilidades para el desarrollo y control de drones más allá de las limitaciones iniciales.

- En resumen, la implementación de un ordenador a bordo con procesamiento de imágenes en tiempo real ha proporcionado una solución efectiva al problema de la transferencia de imágenes y ha demostrado ser una vía viable para futuros desarrollos en el control y procesamiento autónomo de drones.

4. **Embarcado en el dron y problemas derivados del mismo**

- El TFM carecería de sentido sin el embarcado del ordenador auxiliar (reComputer J3011) en el dron. Este embarque no solo incluye el ordenador auxiliar, sino también la cámara y, más importante, la batería que alimenta al ordenador durante el vuelo. Aunque a priori podría parecer una tarea sencilla, la naturaleza privativa de los drones DJI, que no están diseñados para este tipo de modificaciones estructurales, presentó muchos más problemas de los esperados, obligando a buscar soluciones alternativas para abordar este desafío.

- A pesar de que se desarrollaron varios prototipos para el embarcado, la falta de tiempo llevó a adoptar un diseño más tradicional que permitiera una implementación más rápida en el flujo de trabajo del proyecto. Esta solución alternativa permitió identificar problemas relacionados con la oclusión del sistema de sensorización del dron y la modificación del centro de gravedad, así como las implicaciones de ambos problemas en la estabilidad y control del vuelo del dron.

- Debido a estas dificultades y la excesiva demora que implicaron, las pruebas en vuelo finales se realizaron sin la batería embarcada. La batería, debido a su peso, modificaba en exceso la posición del centro de gravedad del dron, haciendo que el vuelo no fuera estable y dificultando su control. Al no disponer de recursos económicos para adquirir una batería más ligera, la única opción fue realizar las pruebas con el reComputer J3011 alimentado mediante un cable conectado a una toma de corriente.

- El considerable tiempo invertido en resolver estos problemas también impidió testar en vuelo el segundo modo de funcionamiento de la aplicación de procesamiento de imágenes, que permitiría el vuelo automático del dron entre *waypoints* usando códigos ArUco.

- A pesar de todas las dificultades, el embarque permitió una primera aproximación a este complejo mundo, revelando algunos de los múltiples problemas que pueden surgir. Además, se destaca la flexibilidad mantenida para abordar los desafíos, encontrando soluciones alternativas y creativas para resolver los problemas que iban apareciendo. Aunque el segundo modo de funcionamiento de la aplicación no pudo ser suficientemente testeado, el uso del dron DJI Mavic Mini demostró el correcto funcionamiento en vuelo del primer modo de funcionamiento de la aplicación.

- En conclusión, todo el trabajo realizado y las dificultades enfrentadas han preparado el terreno para futuros proyectos y exploraciones, creando una base sólida para continuar desarrollando soluciones innovadoras en este campo.

5.2 Trabajos futuros

Una vez expuestas las conclusiones del presente TFM, a continuación se citan algunas de las posibles vías de mejora y evolución del proyecto. Únicamente se consideran aquellas mejoras y trabajos futuros que tienen más sentido en el corto y medio plazo, ya sea porque no se han podido completar en el presente TFM o porque es necesario realizar estas acciones para subsanar problemas surgidos durante el desarrollo del proyecto.

1. **Búsqueda de una solución efectiva y robusta al problema del embarcado en el dron**

   - El trabajo más importante a realizar en el futuro es la búsqueda de una solución efectiva y robusta para el embarcado del reComputer J3011, la batería y la cámara en el dron. Para lograr esto, se debería llevar a cabo un estudio aún más profundo sobre el tema con el objetivo de desarrollar un soporte específico para este dron que resuelva los problemas de oclusión de la sensorización y la modificación del centro de gravedad.

   - Una posible vía de mejora es la exploración y adquisición de baterías más ligeras. Esto podría mitigar los problemas de peso y permitir un mejor balanceo del dron durante el vuelo. Además, otra opción es diseñar un adaptador o batería personalizada que permita alimentar tanto al dron como al reComputer J3011 con una única fuente de energía, optimizando así la distribución del peso y la eficiencia energética.

   - Asimismo, se podría investigar la posibilidad de utilizar materiales avanzados y técnicas de impresión 3D para crear un soporte más ligero y resistente. Este soporte debería ser capaz de acomodar todos los componentes necesarios sin comprometer la estabilidad ni la funcionalidad del dron. También

sería útil considerar el rediseño de la estructura del soporte para evitar la interferencia con los sensores del dron, manteniendo al mismo tiempo un centro de gravedad óptimo.

- También, se podrían desarrollar algoritmos de control que compensen dinámicamente los cambios en el centro de gravedad, ajustando los parámetros de vuelo en tiempo real para mantener la estabilidad y el control del dron.

2. **Testeo del modo de funcionamiento automático de la aplicación de procesamiento de imágenes**

   - Debido a los numerosos problemas derivados del proceso de embarque en el dron, en el presente TFM no se pudo testar en vuelo el modo de funcionamiento automático de la aplicación de procesamiento de imágenes, a pesar de haber sido desarrollado y probado en un entorno local. Este modo de funcionamiento permite que el dron realice vuelos automáticos entre *waypoints* utilizando códigos ArUco. Por lo tanto, una vez resuelto el problema del embarcado, el siguiente paso a realizar en el corto y medio plazo sería la ejecución de estas pruebas en vuelo. Estas pruebas son cruciales para demostrar el adecuado funcionamiento del modo automático y para obtener *feedback* que permita modificar, optimizar y depurar el código de la aplicación de procesamiento de imágenes. Los resultados obtenidos de estas pruebas en vuelo proporcionarán la información necesaria para realizar ajustes precisos y mejorar la eficiencia y la robustez del sistema en condiciones reales de operación.

3. **Desarrollo de una estación de control en tierra que permita el manejo del reComputer J3011 vía SSH**

   - Otro trabajo futuro importante es el desarrollo de una estación de control en tierra que permita manejar el reComputer J3011 a través de SSH. En el presente TFM, la única manera de controlar el reComputer J3011 cuando este se encuentra en vuelo ha sido a través de VNC. Aunque VNC ha sido funcional, SSH representa una solución más eficiente y robusta para este propósito. SSH (*Secure Shell*) proporciona una conexión segura y encriptada, permitiendo un control remoto más fiable y con menor latencia en comparación con VNC. Además, SSH consume menos recursos del sistema, lo cual es crucial para mantener el rendimiento óptimo del reComputer J3011 durante las operaciones de vuelo. El desarrollo de una estación de control en tierra podría llevarse a cabo mediante el uso de Java, creando una aplicación que permita conectar vía SSH al reComputer J3011. Esta estación de control no solo facilitaría el comando remoto del ordenador embarcado, sino que también permitiría monitorear su estado y ejecutar scripts o comandos necesarios para el procesamiento de imágenes

y otras tareas críticas. Implementar una solución de este tipo mejoraría significativamente la operatividad y flexibilidad del sistema, asegurando que el reComputer J3011 pueda ser controlado de manera eficiente y segura durante las misiones de vuelo.

4. **Integración del proyecto en ROS**

   - A largo plazo, una de las mejoras más significativas para el presente proyecto sería su integración en un ecosistema de ROS (*Robot Operating System*). ROS es un *framework* flexible para escribir software de robots que ofrece una serie de herramientas, librerías y convenciones que simplifican la creación de comportamientos complejos en sistemas robóticos tanto grandes como pequeños.

   - Integrar este proyecto en ROS permitiría modularizar el sistema en nodos independientes, cada uno encargado de una tarea específica, como el control del dron, el procesamiento de imágenes y la comunicación. Esta modularidad facilita la gestión y el desarrollo del proyecto, permitiendo añadir o modificar componentes sin afectar al sistema global. Además, ROS proporciona una amplia gama de paquetes preexistentes para tareas comunes en robótica, como la navegación, la percepción y la manipulación. La reutilización de estos paquetes puede acelerar el desarrollo y reducir la necesidad de escribir código desde cero.

   - ROS está diseñado para ser altamente escalable, permitiendo que el sistema crezca y se adapte a nuevas necesidades y funcionalidades. Esto es esencial para un proyecto de largo plazo, donde las necesidades pueden evolucionar significativamente con el tiempo. Los pasos para integrar el proyecto en ROS incluyen la instalación de ROS en el reComputer J3011 y en cualquier otro ordenador que forme parte del sistema de control, el desarrollo de nodos ROS específicos para cada tarea, la configuración de temas y servicios de ROS para manejar la comunicación entre nodos, y la integración del proyecto con herramientas de ROS como RViz para la visualización de datos y Gazebo para la simulación del dron en entornos virtuales.

5. **Exploración de aplicaciones de IA para aprovechar todo el rendimiento del Jetson Nano**

   - Otro trabajo futuro significativo es la exploración y desarrollo de aplicaciones de inteligencia artificial (IA) para aprovechar todo el rendimiento del Jetson Nano de NVIDIA. Los dispositivos Jetson están específicamente diseñados para el desarrollo de aplicaciones de IA, ofreciendo capacidades avanzadas de procesamiento de datos y aprendizaje automático que pue-

den potenciar significativamente las funcionalidades de proyectos basados en robótica y procesamiento de imágenes.

- Aunque en el presente proyecto no se han empleado aplicaciones de IA, a medio y largo plazo resultaría muy interesante desarrollar aplicaciones de procesamiento de imágenes que utilicen técnicas de IA. Estas aplicaciones podrían incluir la clasificación y reconocimiento de objetos, el seguimiento de múltiples objetivos en tiempo real, la detección de anomalías, y la navegación autónoma basada en la visión.

- La implementación de IA en el Jetson Nano no solo permitiría maximizar su rendimiento, sino que también abriría nuevas posibilidades para mejorar la eficiencia y precisión del sistema de procesamiento de imágenes. Por ejemplo, mediante el uso de redes neuronales convolucionales (CNNs) y otras arquitecturas de aprendizaje profundo, se podría mejorar significativamente la capacidad del sistema para interpretar y reaccionar ante el entorno en tiempo real. Además, la IA puede aportar mejoras sustanciales en la toma de decisiones del dron, permitiendo una mayor autonomía y capacidad de adaptación a entornos dinámicos y complejos. La incorporación de algoritmos de IA para el análisis de datos y la toma de decisiones podría hacer que el dron sea más robusto y eficiente, reduciendo la necesidad de intervención humana y aumentando su capacidad para realizar tareas complejas de manera autónoma.

- En resumen, explorar y desarrollar aplicaciones de IA para el Jetson Nano no solo aprovecharía al máximo el *hardware* disponible, sino que también proporcionaría avances significativos en las capacidades del dron y del sistema en general, preparando el camino para futuras innovaciones y mejoras en el campo de la robótica y el procesamiento de imágenes.

Apéndice A

Presupuesto

En este apéndice se recogen los diferentes costes derivados de la realización del presente TFM, incluyendo aquellos relacionados con mano de obra, materiales de oficina, instalaciones y equipos. Tras una primera clasificación de los recursos necesarios para el correcto desempeño de este proyecto, se detallarán los costes unitarios asociados a cada uno de ellos, presentando, para concluir, el presupuesto final del proyecto.

Para ello, los precios aplicados a los diferentes elementos se han basado en tarifas legales vigentes o, en su defecto, en estimaciones coherentes acordes con los precios de mercado. Este análisis exhaustivo permite obtener una visión clara y precisa de los recursos económicos requeridos y facilita la planificación y gestión financiera del proyecto.

En primer lugar, deben definirse dos variables que serán ampliamente utilizadas en la posterior determinación de los costes unitarios en la Sección A.2. Dichos parámetros son el coste de amortización de los bienes utilizados y la tasa horaria, calculados según se expresa en las Ecuaciones A.1 y A.2.

$$c_{\mathrm{a}} = \frac{VC - VR}{n_{\mathrm{a}}} \tag{A.1}$$

$$t_{\mathrm{h}} = \frac{c_{\mathrm{a}}}{h} \tag{A.2}$$

Siendo:

c_{a}: amortización [€/año]

VC: valor de compra [€]

VR: valor residual al cabo del período de amortización [€]

n_{a}: período de amortización [años]

t_{h}: tasa horaria [€/h]

h: horas trabajadas al año [h]

Para el cálculo de la tasa horaria, se han considerado 40 horas de trabajo semanales durante las semanas no festivas ni de vacaciones de cada año. Contemplando 46 semanas laborables, se llega a un total de:

$$40 \cdot 46 = 1.840 \text{ horas trabajadas al año} \tag{A.3}$$

A.1 Clasificación de los recursos a tener en cuenta

A continuación, se definen los recursos, agrupándolos según su naturaleza, que han sido requeridos para la correcta realización del presente proyecto.

- Mano de obra

Código	Descripción	Medido en
MO01	Ingeniero Técnico Aeronáutico	h
MO02	Profesor/a Titular de Universidad	h

Tabla A.1: Mano de obra implicada en el desarrollo del proyecto.

- *Hardware*

Código	Descripción	Medido en
HW01	Ordenador embarcado reComputer J3011	h
HW02	Raspberry Pi NoIR Camera V2	ud
HW03	Ordenador portátil ROG Zephyrus M16	h
HW04	Dron DJI Mavic Mini	ud
HW05	Dron DJI Phantom 4 Advanced+	ud
HW06	Teléfono móvil HUAWEI P30 Lite New Edition	ud
HW07	Ratón y teclado inalámbricos Logitech MK270	ud
HW08	Disco duro SSD externo 1TB - Lacie Rugged Mini	ud
HW09	Monitor externo ASUS VZ24EHE	ud

Tabla A.2: *Hardware* requerido para el desarrollo del proyecto.

- *Software*

Código	Descripción	Medido en
SW02	Licencia de Autodesk AutoCAD 2024	h
SW03	Licencia de Overleaf (editor online LaTeX)	h
SW04	Licencia de Microsoft Office 2021	h
SW05	Licencia de Adobe Acrobat Reader DC	h

Tabla A.3: *Software* utilizado para el desarrollo del proyecto.

- Lugar de trabajo

Código	Descripción	Medido en
-	Alquiler	%
-	Suministros	%

Tabla A.4: Aspectos relacionados con el lugar de trabajo necesario para el desarrollo del proyecto.

- Material de oficina

Código	Descripción	Medido en
MF01	Bolígrafo Pilot Super Grip <M>	ud
MF02	Paquete de folios Fabriano Copy 2 A4	ud
MF03	Consulta de material bibliográfico	h

Tabla A.5: Material de oficina empleado para el desarrollo del proyecto.

A.2 Desglose de costes unitarios

Como se ha mencionado previamente, en esta sección se determinará el coste unitario (o tasa horaria) relativo a cada uno de los recursos que han sido empleados para la realización del presente proyecto.

A.2.1 Coste de mano de obra unitario

- Ingeniero Técnico Aeronáutico [84]

$$\text{Salario bruto anual} = 21.600{,}00\ € \tag{A.4}$$

$$\text{Coste horario} = \frac{21.600{,}00}{1.840} = 11{,}74\ €/\text{h} \tag{A.5}$$

- Profesor/a Titular de Universidad [85]

$$\text{Salario bruto anual} = 37.608{,}52\ € \tag{A.6}$$

$$\text{Coste horario} = \frac{37.608{,}52}{1.840} = 20{,}44\ €/\text{h} \tag{A.7}$$

A.2.2 Coste de *hardware* unitario

- Ordenador embarcado reComputer J3011

  El ordenador embarcado utilizado en la realización de este proyecto presenta las siguientes características:

  - Procesador: NVIDIA(R) Jetson Orin(TM) Nano SoM 8G

  El precio de compra se encuentra alrededor de 600 €, y se ha estimado un valor residual del 20 % y un período de amortización de 5 años.

$$c_{\text{a}} = \frac{600 - 260}{5} = 68\ €/\text{año} \tag{A.8}$$

$$t_{\text{h}} = \frac{68}{1.840} = 0{,}0370\ €/\text{h} \tag{A.9}$$

- Ordenador portátil ROG Zephyrus M16

  El equipo personal utilizado en la realización de este proyecto presenta las siguientes características:

  - Procesador: Intel(R) Core(TM) i9-12900H, 14 núcleos a 2,5 GHz
  - Memoria RAM: 32 GB

El precio de compra se encuentra alrededor de 2.700 €, y se ha estimado un valor residual del 20 % y un período de amortización de 5 años.

$$c_{\mathrm{a}} = \frac{2.700 - 540}{5} = 432 \text{ €/año} \tag{A.10}$$

$$t_{\mathrm{h}} = \frac{432}{1.840} = 0{,}23 \text{ €/h} \tag{A.11}$$

- Raspberry Pi NoIR Camera V2

$$\text{Coste de compra} = 25{,}00 \text{ €} \tag{A.12}$$

- Dron DJI Mavic Mini

$$\text{Coste de compra} = 360{,}00 \text{ €} \tag{A.13}$$

- Dron DJI Phantom 4 Advanced+

$$\text{Coste de compra} = 1200{,}00 \text{ €} \tag{A.14}$$

- Teléfono móvil HUAWEI P30 Lite New Edition

$$\text{Coste de compra} = 142{,}00 \text{ €} \tag{A.15}$$

- Ratón y teclado inalámbricos Logitech MK270

$$\text{Coste de compra} = 25{,}99 \text{ €} \tag{A.16}$$

- Disco duro SSD externo 1TB - Lacie Rugged Mini

$$\text{Coste de compra} = 179{,}99 \text{ €} \tag{A.17}$$

- Monitor externo ASUS VZ24EHE

$$\text{Coste de compra} = 115{,}99 \text{ €} \tag{A.18}$$

A.2.3 Coste de *software* unitario

- Autodesk AutoCAD 2024

  El valor anual de una licencia estándar de Autodesk AutoCAD 2024 asciende a 2.342 € [86], por lo que la tasa horaria derivada de su uso es de:

$$t_{\mathrm{h}} = \frac{2.342}{1.840} = 1{,}27\ €/\mathrm{h} \tag{A.19}$$

- Overleaf (editor online LaTeX)

  El valor anual de una licencia estándar para estudiantes de Overleaf es de 79 € [87], por lo que la tasa horaria derivada de su uso es de:

$$t_{\mathrm{h}} = \frac{79}{1.840} = 0{,}04\ €/\mathrm{h} \tag{A.20}$$

- Microsoft Office 2021

  El valor de una licencia anual de Microsoft Office[11] es de 140,40 € [88], por lo que la tasa horaria derivada de uso es:

$$t_{\mathrm{h}} = \frac{140{,}40}{1.840} = 0{,}08\ €/\mathrm{h} \tag{A.21}$$

- Adobe Acrobat Reader DC

  Se trata de un *software* de licencia gratuita, por lo que su tasa horaria es de $t_{\mathrm{h}} = 0{,}00\ €/\mathrm{h}$.

A.2.4 Coste asociado al lugar de trabajo

Dado que el proyecto se ha llevado a cabo tanto en las instalaciones de la universidad como en el propio lugar de residencia del autor, resulta complejo discernir los gastos asociados al uso cotidiano de la vivienda de aquellos derivados de la realización del proyecto. Para abordar esta dificultad, se ha realizado una estimación de dichos gastos generales. Concretamente, se ha considerado que estos gastos generales suponen un 18 % del coste total de ejecución del trabajo. Este porcentaje engloba diversos conceptos, tales como el alquiler, los consumos de agua, electricidad y calefacción, las labores de mantenimiento y la conexión a internet. Esta estimación permite incluir de manera justa y proporcionada los costes indirectos asociados al proyecto, asegurando una visión integral de los recursos económicos empleados.

[11] Se ha considerado la tarifa Microsoft 365 Empresa Estándar.

A.2.5 Coste de material de oficina unitario

- Bolígrafo Pilot Super Grip <M>

$$\text{Coste de compra} = 1{,}65\ € \tag{A.22}$$

- Paquete de folios Navigator Universal A4

$$\text{Coste de compra} = 3{,}65\ € \tag{A.23}$$

- Consulta de material bibliográfico

  Dado que todas las referencias bibliográficas que han sido consultadas son de libre acceso, la tasa horaria derivada de su uso es de $t_{\text{h}} = 0{,}00\ €/\text{h}$.

A.3 Presupuesto total

A continuación se adjuntan dos tablas que se han estimado oportunas para determinar el presupuesto definitivo del presente proyecto. Dicho presupuesto presenta una apariencia reducida, dadas las características del proyecto llevado a cabo. Las partes que se presentan son el presupuesto desglosado de los recursos según su naturaleza y el presupuesto final.

Código	Cantidad	Descripción del recurso	Precio	Importe
		Mano de obra		
MO01	337,50	Ingeniero Técnico Aeronáutico	11,74	3.962,25
MO02	22,50	Profesor/a Titular de Universidad	20,44	459,89
			Total	**4.422,14 €**
		Hardware		
HW01	150	Ordenador embarcado reComputer J3011	0,037	5,55
HW02	1	Raspberry Pi NoIR Camera V2	25,00	25,00
HW03	150	Ordenador portátil ROG Zephyrus M16	0,23	34,50
HW04	1,00	Dron DJI Mavic Mini	360,00	360,00
HW05	1,00	Dron DJI Phantom 4 Advanced+	1.200,00	1.200,00
HW06	1,00	Teléfono móvil HUAWEI P30 Lite New Edition	142,00	142,00
HW07	1,00	Ratón y teclado inalámbricos Logitech MK270	25,99	25,99
HW08	1,00	Disco duro SSD externo 1TB - Lacie Rugged Mini	179,99	179,99
HW09	1,00	Monitor externo ASUS VZ24EHE	115,99	115,99
			Total	**2.089,02 €**
		Software		
SW02	10,00	Licencia de Autodesk AutoCAD 2024	1,27	12,73
SW03	40,00	Licencia de Overleaf (editor online LaTeX)	0,04	1,72
SW04	1,00	Licencia de Microsoft Office 2021	0,08	0,08
SW05	1,00	Licencia de Adobe Acrobat Reader DC	0,00	0,00
			Total	**14,53 €**
		Material de oficina		
MF01	3,00	Bolígrafo Pilot Super Grip <M>	1,65	4,95
MF03	1,00	Paquete de folios Navigator Universal A4	3,65	3,65
MF04	10,00	Consulta de material bibliográfico	0,00	0,00
			Total	**8,60 €**
			Total recursos	**6.534,29 €**

Tabla A.6: Presupuesto desglosado de los recursos según su naturaleza.

TOTAL EJECUCIÓN MATERIAL	**6.534,29 €**
18 % Gastos Generales	1.176,17
6 % Beneficio Industrial	1.372,20
TOTAL EJECUCIÓN POR CONTRATA	**9.082,66 €**
21 % I.V.A.	1.907,35
TOTAL PRESUPUESTO C/I.V.A.	**10.990,02 €**

Tabla A.7: Presupuesto total.

Asciende el presupuesto proyectado, a la expresada cantidad de:

DIEZ MIL NOVECIENTOS NOVENTA EUROS CON CERO DOS CÉNTIMOS

Apéndice B

Pliego de condiciones

El presente Pliego de Condiciones define los requisitos técnicos y las especificaciones necesarias para la realización del Trabajo Fin de Máster (TFM) titulado «`Control de un dron DJI a través del computador Jetson Nano`». La finalidad de este pliego es servir como una guía completa para la ejecución del proyecto, asegurando que todos los participantes comprendan y sigan los estándares y objetivos fijados. El TFM realizado tiene aplicaciones prácticas en áreas como la robótica, la realidad aumentada y la navegación autónoma de drones

B.1 Objetivo

En un proyecto eminentemente de *software*, como el presente TFM, un pliego de condiciones tiene como objetivo definir la relación entre el desarrollador y el usuario final. Es esencial para especificar claramente qué puede aportar el *software* y qué se espera del usuario final para que este pueda aprovechar al máximo todas las funcionalidades del *software* desarrollado. Este documento sirve como una guía para asegurar que el usuario final comprenda cómo interactuar con el *software* de manera efectiva y eficiente.

El presente TFM no solo se enfoca en el *software*, sino que también incluye componentes de *hardware*. Por lo tanto, en los casos necesarios, se detallarán las especificaciones y normativas relativas a la parte del *hardware*. Cabe destacar que este TFM está redactado conforme a la Norma UNE-EN ISO 12100:2012, que regula los conceptos básicos de seguridad en máquinas y los principios generales de diseño. Esta normativa proporciona un marco de referencia para garantizar que tanto el hardware como el software se desarrollen siguiendo los estándares de seguridad y eficiencia, asegurando así un producto final robusto y confiable. La inclusión de estas especificaciones busca

ofrecer una visión completa y detallada del proyecto, abarcando todos los aspectos técnicos y normativos necesarios para su correcta implementación y uso.

B.2 Condiciones de uso del proyecto

B.2.1 Conocimientos previos

El presente TFM es un proyecto multidisciplinar, por lo que se requiere una serie de conocimientos previos en diversas áreas de la ingeniería. En primer lugar, es esencial tener ciertos conocimientos de programación, especialmente en los lenguajes Python y Java, que son los utilizados en este proyecto. Estos conocimientos son necesarios para realizar pequeñas mejoras y modificaciones en los códigos, aunque no se requiere un nivel avanzado.

Además, se necesita familiaridad con el sistema operativo Linux, ya que el reComputer J3011 funciona bajo este sistema. También es importante tener conocimientos básicos en telecomunicaciones para entender el esquema de comunicaciones planteado. Adicionalmente, se deben poseer habilidades en el pilotaje de drones, ya que este TFM implica operaciones de vuelo. Por último, es crucial estar al tanto de la normativa vigente que regula el vuelo de drones.

Por último, es importante señalar que los conocimientos teóricos subyacentes al presente TFM deberían ser adquiribles mediante la lectura de esta memoria. En caso de ser necesario, también se pueden consultar todas las referencias y recursos bibliográficos adjuntados a lo largo del documento. Estos recursos proporcionan una base sólida para entender los conceptos y fundamentos del proyecto, garantizando así que cualquier lector interesado pueda profundizar en los temas tratados y aplicar los conocimientos adquiridos de manera efectiva.

B.2.2 *Software* necesario

El usuario final no necesita instalar ningún *software* adicional para utilizar la aplicación de procesamiento de imágenes y explorar sus funcionalidades, ya que el reComputer J3011 ya tiene todos los programas y códigos necesarios preinstalados. Sin embargo, se requiere un ordenador auxiliar con el *software* VNC instalado para controlar el reComputer J3011 durante el vuelo. Además, es necesario que el mando del dron tenga instalada la aplicación desarrollada con el SDK de DJI, lo cual ya ha sido realizado durante el transcurso de este proyecto, por lo que el usuario final no debe preocuparse por este aspecto.

B.2.3 *Hardware* necesario

El *hardware* necesario para el proyecto incluye el dron Phantom 4 Advanced+, el reComputer J3011, la cámara Raspberry Pi NoIR V2, la batería para alimentar el sistema, el soporte para embarcar todos los componentes, un ordenador portátil personal, una pantalla externa, un teclado y un ratón para usar el reComputer J3011 en tierra.

B.2.4 Consideraciones finales

El usuario final debe ser consciente de que la realización de operaciones con un dron está sujeta a una normativa específica. La normativa vigente en territorio nacional es el Real Decreto 517/2024, de 4 de junio. Es importante destacar que todas las operaciones no sujetas a esta normativa están penadas, por lo que el usuario debe tener conocimientos adecuados de dichas regulaciones.

B.3 Condiciones de distribución

Este proyecto no está sujeto a ninguna licencia restrictiva y su uso, modificación e implementación de mejoras es completamente libre. En consecuencia, cualquier persona interesada puede consultar todos los códigos derivados de la realización del presente trabajo. Estos códigos se adjuntan en forma de anexos en la presente memoria.

Se permite a cualquier individuo modificar, reproducir y evolucionar el proyecto, siempre y cuando estos cambios no impliquen un beneficio económico. Es decir, el uso del proyecto debe alinearse con los principios del *software* libre, asegurando que nadie se lucre a costa del trabajo realizado. Este enfoque fomenta la colaboración abierta y el desarrollo continuo, permitiendo que la comunidad de usuarios y desarrolladores contribuya al avance y perfeccionamiento del proyecto sin restricciones comerciales.

B.4 Mantenimiento y soporte

El proyecto no está sujeto a ningún tipo de mantenimiento ni soporte actual, ya que se trata de un trabajo de índole académica. No existe un compromiso de mantenimiento y soporte en el tiempo por parte del autor. Esta es una de las razones por las que los códigos y el proyecto han sido hechos públicos, permitiendo que futuros usuarios se encarguen de continuar con la vida del proyecto y proporcionen el mantenimiento, soporte y evolución necesarios.

En este sentido, se deja en manos de futuros integrantes o usuarios del proyecto la responsabilidad de realizar los pasos futuros. La continuidad y mejora del proyecto dependerán de la contribución de la comunidad, que podrá adaptar y expandir el proyecto según las necesidades y avances tecnológicos que surjan.

B.5 Responsabilidades

Todas las responsabilidades derivadas del uso de los materiales desarrollados en el presente proyecto recaen exclusivamente en el usuario final. El autor no se hace responsable de ninguna de estas responsabilidades. Es importante destacar que el vuelo de un dron puede conllevar ciertos riesgos si no se realiza bajo la normativa y los criterios de uso necesarios. El autor no asume ninguna responsabilidad por accidentes o daños que puedan surgir relacionados con este tema. Por lo tanto, se insta a los usuarios a cumplir con todas las regulaciones pertinentes y a operar el dron de manera segura y responsable.

B.6 Documentación del proyecto

La presente memoria constituye la única documentación del proyecto. Con esta consideración en mente, se ha tratado de que su redacción sea lo más didáctica posible, facilitando así la comprensión de los conceptos y procedimientos descritos.

Para complementar las explicaciones proporcionadas, se han adjuntado múltiples fuentes bibliográficas a lo largo de la memoria. Estas referencias permiten al lector profundizar en aquellos temas que puedan resultar más complejos, asegurando que disponga de todos los recursos necesarios para entender y aplicar los conocimientos presentados en este trabajo. Esta documentación integral tiene como objetivo proporcionar una guía clara y exhaustiva para cualquier usuario o investigador que desee continuar con el desarrollo del proyecto o utilizar sus componentes en futuros trabajos.

Apéndice C

Objetivos de desarrollo sostenible

Los Objetivos de Desarrollo Sostenible (ODS) son una iniciativa global impulsada por las Naciones Unidas para abordar los principales desafíos a los que se enfrenta la humanidad. Estos objetivos fueron establecidos en 2015 y forman parte de la Agenda 2030 para el Desarrollo Sostenible, una hoja de ruta para lograr un futuro mejor y más sostenible para todos [89].

Los ODS comprenden 17 objetivos interconectados que abordan una amplia gama de problemas sociales, económicos y ambientales, tales como la erradicación de la pobreza, la protección del planeta y el aseguramiento de la prosperidad para todos. Algunos de estos objetivos incluyen la educación de calidad, la igualdad de género, el acceso a agua limpia y saneamiento, y la acción por el clima [90].

La Agenda 2030 es un plan de acción adoptado por los 193 Estados miembros de la ONU para alcanzar estos objetivos para el año 2030. Esta agenda enfatiza la necesidad de una colaboración mundial entre los gobiernos, el sector privado y la sociedad civil para enfrentar estos desafíos globales [91].

En el presente Trabajo Fin de Máster, se han identificado varios Objetivos de Desarrollo Sostenible con los que el proyecto está alineado. Estos son algunos de los objetivos que el TFM apoya directamente:

Objetivos de Desarrollo Sostenible	Alto	Medio	Bajo	Nulo
1. Fin de la pobreza.				X
2. Hambre cero.				X
3. Salud y bienestar.				X
4. Educación de calidad.	X			
5. Igualdad de género.				X
6. Agua limpia y saneamiento.				X
7. Energía asequible y no contaminante.			X	
8. Trabajo decente y crecimiento económico.		X		
9. Industria, innovación e infraestructuras.	X			
10. Reducción de las desigualdades.				X
11. Ciudades y comunidades sostenibles.	X			
12. Producción y consumo responsables.		X		
13. Acción por el clima.			X	
14. Vida submarina.				X
15. Vida de ecosistemas terrestres.				X
16. Paz, justicia e instituciones sólidas.				X
17. Alianzas para lograr objetivos.	X			

Tabla C.1: Evaluación de los Objetivos de Desarrollo Sostenible [Elaboración propia].

C.1 Justificación de la alineación del proyecto con los ODS

C.1.1 ODS4 - Educación de calidad

Este proyecto está perfectamente alineado con el Objetivo de Desarrollo Sostenible (ODS) de educación de calidad. La memoria ha sido redactada con un enfoque didáctico, asegurando que los temas tratados se expliquen de manera clara y accesible. Además, se han incluido numerosas referencias y recursos bibliográficos para que los lectores puedan profundizar en los conceptos presentados. Al proporcionar una documentación exhaustiva y accesible, el proyecto contribuye a la educación continua y al fortalecimiento de habilidades técnicas y teóricas en áreas multidisciplinares como la robótica, la programación y la navegación autónoma de drones.

C.1.2 ODS7 - Energía asequible y no contaminante

Aunque de manera indirecta, el proyecto también contribuye al ODS de energía asequible y no contaminante. La navegación autónoma de drones, tal como se plantea en este TFM, puede ser utilizada para la monitorización y mantenimiento de aerogeneradores e instalaciones fotovoltaicas. Esta aplicación práctica facilita la supervisión eficiente y la gestión de infraestructuras de energía renovable, promoviendo así la generación de energía limpia y no contaminante. La capacidad de los drones para acceder a áreas de difícil acceso y proporcionar datos precisos y en tiempo real es fundamental para la optimización y sostenibilidad de las energías renovables.

C.1.3 ODS8 - Trabajo decente y crecimiento económico

El proyecto también se alinea con el ODS de trabajo decente y crecimiento económico. La implementación de sistemas de navegación autónoma de drones abre nuevas oportunidades en diversas industrias, desde la agricultura de precisión hasta la vigilancia de infraestructuras y la logística. Al fomentar la innovación y el desarrollo tecnológico, se generan nuevos empleos y se impulsa el crecimiento económico. La capacidad de los drones para mejorar la eficiencia operativa y reducir costos en diferentes sectores industriales es un motor clave para el desarrollo económico sostenible.

C.1.4 ODS9 - Industria, innovación e infraestructuras

El proyecto está fuertemente alineado con el ODS de industria, innovación e infraestructuras. La integración de tecnologías avanzadas como el Jetson Nano y la programación en Python y Java para el control de drones representa un avance significativo en la innovación tecnológica. Este proyecto no solo demuestra una aplicación innovadora de la tecnología existente, sino que también establece una base sólida para futuras mejoras y desarrollos en el campo de la robótica y las infraestructuras inteligentes.

C.1.5 ODS11 - Ciudades y comunidades sostenibles

El proyecto también contribuye al ODS de ciudades y comunidades sostenibles. La navegación autónoma de drones puede ser utilizada para una variedad de aplicaciones urbanas, como la gestión del tráfico, la vigilancia de infraestructuras, la monitorización ambiental y la entrega de servicios esenciales. Estas aplicaciones mejoran la sostenibilidad y la calidad de vida en las ciudades, promoviendo comunidades más resilientes y eficientes. La capacidad de los drones para recopilar y analizar datos en tiempo real es crucial para el desarrollo de soluciones urbanas inteligentes y sostenibles.

C.1.6 ODS13 - Acción por el clima

El proyecto apoya el ODS de acción por el clima mediante el uso de drones para el monitoreo ambiental y la recopilación de datos climáticos. Los drones pueden acceder a áreas remotas y difíciles de alcanzar, proporcionando información valiosa sobre los cambios en el medio ambiente y ayudando a identificar patrones y tendencias relacionadas con el cambio climático. Esta capacidad de monitoreo es esencial para desarrollar e implementar estrategias efectivas de mitigación y adaptación al cambio climático.

C.1.7 ODS17 - Alianzas para lograr los objetivos

Finalmente, el proyecto está fuertemente orientado hacia el ODS de alianzas para lograr los objetivos. Todos los recursos y códigos desarrollados en el presente TFM son de acceso público y libre distribución, fomentando la colaboración y las sinergias entre diferentes comunidades y sectores. Al hacer que el proyecto sea accesible para todos, se promueve la cooperación y el intercambio de conocimientos, esenciales para el avance colectivo hacia los objetivos de desarrollo sostenible. Esta apertura y transparencia son fundamentales para crear alianzas sólidas y efectivas que impulsen el progreso hacia un futuro sostenible.

Apéndice D

Archivo de configuración settings.json para VSCode

```
{
    "editor.fontFamily": "Fira Code, Consolas, 'Courier New',
        monospace",
    "editor.fontSize": 12,
    "editor.lineHeight": 22,
    "editor.tabSize": 4,
    "editor.insertSpaces": true,
    "editor.wordWrap": "on",
    "files.autoSave": "afterDelay",
    "files.autoSaveDelay": 1000,
    "workbench.colorTheme": "Default Dark Modern",
    "workbench.iconTheme": "material-icon-theme",
    "workbench.editor.enablePreview": false,
    "terminal.integrated.fontSize": 16,
    "explorer.confirmDelete": true,
    "editor.formatOnSave": true,
    "editor.minimap.enabled": false,
    "python.analysis.typeCheckingMode": "off",
    "python.analysis.autoImportCompletions": true,
    "editor.rulers": [
        75
    ],
    "python.analysis.extraPaths": [
        "/home/omar/Escritorio/TFM/ArUco"
    ],
    "python.defaultInterpreterPath": "/home/omar/Escritorio/TFM/ArUco
        /VirtualEnvironments/EntVirt_Py310/bin/python",
    "python.envFile": "${workspaceFolder}/.env"
}
```

Código D.1: Archivo de configuración settings.json para el IDE Visual Studio Code.

Apéndice E

Archivo main.py

```
# ---------------------- #
# OMAR RABANAL GUTIÉRREZ #
# 19 de Junio del 2024   #
# ---------------------- #


# ------------------------------------------------------------------ #
import numpy as np
import math
import cv2
from typing import Tuple, Any
import time
import threading
import sys
import subprocess
import os
import multiprocessing
# ------------------------------------------------------------------ #


# ------------------------------------------------------------------ #
import modulos.GlobalVariables as gv
from modulos.webcam_gstreamer_pipeline import
    webcam_gstreamer_pipeline
from modulos.RasPi_gstreamer_pipeline import raspi_gstreamer_pipeline
from modulos.moveSimulador import moveSimulador
from modulos.moveVuelo import moveVuelo
from modulos.ImageProcessingArUcoSimulador import
    ImageProcessingSimulador
from modulos.ImageProcessingArUcoVuelo import ImageProcessingVuelo
from modulos.isRotationMatrix import isRotationMatrix
from modulos.rotationMatrixToEulerAngles import
    rotationMatrixToEulerAngles
from modulos.socket_client_Proyecto import socket_cliente_ArUco
import modulos.ColoresEstilosFormatos as est
```

```
from modulos.Telemetry import recibir_telemetria
# ------------------------------------------------------------------ #


# ------------------------------------------------------------------ #
                                # ---- #
                                # MAIN #
                                # ---- #

                ########################################
                # INICIALIZACIÓN DE VARIABLES GLOBALES #
                ########################################
gv.getImage = False

lower_color = np.array([140, 114, 0]) #   15, 0, 0
upper_color = np.array([255, 255, 255]) # 38, 255, 255

gv.KpRP = 0.4
gv.KpY  = 1
gv.KpT  = 0.6

gv.errorRoll     = 0.0
gv.errorPitch    = 0.0
gv.errorYaw      = 0.0
gv.errorThrottle = 0.0

gv.take_off_id_detected = False
gv.landing_id_detected  = False
gv.control_id_detected  = False

gv.yaw_left_command_send    = False
gv.yaw_right_command_send   = False
gv.go_down_command_send     = False
gv.go_up_command_send       = False
gv.roll_left_command_send   = False
gv.roll_right_command_send  = False
gv.go_Fordward_command_send = False
gv.go_Backward_command_send = False

gv.estado_vuelo = "no_state"

gv.take_off_executed = False
gv.landing_executed  = False
gv.posicionamiento_terminado = False

subprocess.run(['clear'])

time.sleep(1)

################################################
#SELECCIÓN DEL MODO DE EJECUCCIÓN DEL PROGRAMA
#SIMULADOR / VUELO REAL
################################################
modo_ejecucion = int(input(f"{est.cyan}Seleccione "
```

```
                            f"el modo de ejecucción del programa,
                                simulador (1) "
                            f"o vuelo real (2) --> {est.reset}"))

if modo_ejecucion == 1:
    move = moveSimulador
    ImageProcessing = ImageProcessingSimulador

if modo_ejecucion == 2:
    move = moveVuelo
    ImageProcessing = ImageProcessingVuelo

subprocess.run(['clear'])

time.sleep(1)

################################################
#SELECCIÓN DEL TIPO DE SERVIDOR (LOCAL/ANDROID)
#Y CONEXIÓN CON EL MISMO
################################################
while True:
    tipo_servidor = int(input(f"{est.cyan}Seleccione "
                              f"donde desea ejecutar la aplicación,
                                  en local (1) "
                              f"o en el móvil Android (2) --> {est.
                                  reset}"))

    if tipo_servidor == 1 or tipo_servidor == 2:
        break
    else:
        print(f"{est.rojo}Introduzca una opción válida...{est.reset}"
            )
        time.sleep(3)
        subprocess.run(['clear'])
        continue

# Caso del servidor local
if tipo_servidor == 1:

    # ------------------------------------------------- #
    # APERTURA DEL SERVIDOR EN UNA TERMINAL INDEPENDIENTE #
    # ------------------------------------------------- #

    # Ruta del directorio del archivo main
    ruta_abs_actual = os.path.dirname(os.path.abspath(__file__))
    # Ruta absoluta del script del servidor
    ruta_server = os.path.join(ruta_abs_actual, 'modulos/
        socket_server.py')

    # subprocess --> permite ejecutar comandos del sistema operativo
        desde
    # un script de Python (archivo main.py).
    subprocess.Popen(['gnome-terminal', '--', 'python3', ruta_server
        ])

```

```
135     time.sleep(5) # Para dar un tiempo a que el servidor
136                   # se inicie correctamente
137 
138     # ------------------------------ #
139     # Intentar conectarse al servidor #
140     # ------------------------------ #
141     # Datos del servidor
142     ip_server = "127.0.1.1"
143     puerto_servidor = 5000
144 
145     socket_cliente = socket_cliente_ArUco(ip_server,puerto_servidor)
146 
147     if socket_cliente == None:
148         print(f"{est.rojo}Al no poder establecer comunicación"
149               " con el servidor no se puede"
150               " continuar con la ejecución del programa, por ello, se
                    "
151               f" procede a finalizar el mismo{est.reset}\n")
152         print(f"{est.cyan}Fin del programa{est.reset}\n")
153         sys.exit()
154 
155 if tipo_servidor == 2:
156     # ------------------------------ #
157     # Intentar conectarse al servidor #
158     # ------------------------------ #
159     # Datos del servidor
160     ip_server = "192.168.1.58"
161     puerto_servidor = 8000
162 
163     socket_cliente = socket_cliente_ArUco(ip_server,puerto_servidor)
164 
165     if socket_cliente == None:
166         print(f"{est.rojo}Al no poder establecer comunicación"
167               " con el servidor no se puede"
168               " continuar con la ejecución del programa, por ello, se
                    "
169               f" procede a finalizar el mismo{est.reset}\n")
170         print(f"{est.cyan}Fin del programa{est.reset}\n")
171         sys.exit()
172 
173 ######################################################################
174 ######################################################################
175                     ##################################
176                     # CREACIÓN DEL THREAD DEL MOVE #
177                     ##################################
178 # Creación de un nuevo hilo para ejecutar la función 'move' en
        segundo plano.
179 # Esto permite que la función 'move' opere de manera independiente
        del
180 # hilo principal, permitiendo que el programa realice otras
181 # tareas simultáneamente.
182 
183 th = threading.Thread(target = move, args = (socket_cliente,), daemon
         = True)
```

```
184 #th =  multiprocessing.Process(target = move, args = (socket_cliente
        ,), daemon = True)
185
186 # Establece el hilo como un hilo daemon.
187 # Un hilo daemon significa que si el programa principal termina,
188 # el hilo daemon se cierra automáticamente sin esperar a que
189 # finalice su ejecución. No se necesita hacer un join para esperar
        que
190 # el hilo termine. Esto es útil cuando no se necesita que el hilo
        complete
191 # su trabajo antes de cerrar el programa.
192 # En resumen, si el programa principal termina, los hilos daemon se
193 # detienen inmediatamente, incluso si están en medio de una tarea.
194
195 # Inicia la ejecución del hilo. Una vez llamado 'start', el hilo
        comenzará a
196 # ejecutar la función 'move' en segundo plano. Esto permite la
        ejecución
197 # concurrente sin bloquear el hilo principal del programa, mejorando
        la
198 # capacidad de respuesta y eficiencia del programa.
199 th.start()
200 print(f"{est.cyan}El Thread que controla la función"
201       f" move ha comenzado...{est.reset}\n")
202
203                 ########################################
204                 # CREACIÓN DEL THREAD DE LA TELEMETRÍA #
205                 ########################################
206 process_telemetria = multiprocessing.Process(target =
        recibir_telemetria, args = (socket_cliente, gv.heading_global, gv
        .GS_global, gv.VS_global, gv.alt_global, gv.roll_global, gv.
        pitch_global, gv.d1_global, gv.d2_global, gv.d3_global, gv.
        d4_global), daemon=True)
207
208 #thread_telemetria = threading.Thread(target = recibir_telemetria,
        args = (socket_cliente,),daemon = True)
209
210 print(f"{est.cyan}El Thread que controla la función"
211       f" de recepción de la telemetría ha comenzado...{est.reset}\n")
212
213 process_telemetria.start()
214 ########################################################################
215 ########################################################################
216
217 # Obtener un diccionario predefinido de marcadores ArUco.
218 # DICT_6X6_250 significa que cada marcador en el diccionario es de 6
        x6 bits
219 # y hay 250 marcadores únicos disponibles en el diccionario.
220 # Este diccionario es utilizado para identificar marcadores ArUco en
        imágenes.
221 aruco_dict = cv2.aruco.getPredefinedDictionary(cv2.aruco.DICT_6X6_250
        )
222
223 # Crear una instancia de DetectorParameters, que contiene los pará
        metros
```

```
224 # de configuración utilizados por el detector de marcadores ArUco.
225 # Estos parámetros pueden ser ajustados para modificar la
        sensibilidad
226 # y precisión de la detección de marcadores.
227 parameters = cv2.aruco.DetectorParameters()
228
229 # Establece la ruta al directorio donde se almacenan los archivos de
230 # calibración de la cámara.
231 # La calibración de la cámara es necesaria para ajustar la precisión
        de la
232 # detección de objetos y la localización en aplicaciones de
233 # visión por computadora.
234
235 # Ruta del directorio del archivo main
236 ruta_abs_actual = os.path.dirname(os.path.abspath(__file__))
237
238 # Ruta de la carpeta de los archivos de la cámara
239 calib_path = os.path.join(ruta_abs_actual, 'CameraFiles/')
240
241 # Carga la matriz de calibración de la cámara desde un archivo de
        texto.
242 # Esta matriz es esencial para transformar las coordenadas de imagen
        en coordenadas del mundo real
243 # y para la correcta proyección de objetos 3D en 2D en imágenes
        capturadas por la cámara.
244 camera_matrix = np.loadtxt(calib_path+'cameraMatrix.txt', delimiter =
         ',')
245
246 # Carga los coeficientes de distorsión de la cámara desde un archivo
        de texto.
247 # Estos coeficientes se utilizan para corregir distorsiones ópticas
        en la
248 # imagen, tales como la distorsión radial y tangencial, mejorando la
249 # precisión de las operaciones de visión por computadora.
250 camera_distortion = np.loadtxt(calib_path+'cameraDistortion.txt',
        delimiter = ',')
251
252 # Define una matriz de rotación de 180 grados alrededor del eje X.
253 # Inicializa una matriz 3x3 con ceros, que será usada para la rotació
        n.
254 R_flip = np.zeros((3,3), dtype=float)
255
256 # Establece los elementos diagonales de la matriz de rotación.
257 # El valor 1.0 en la primera fila y columna significa que el eje X no
         cambia.
258 # Los valores -1.0 en la segunda y tercera filas y columnas indican
        una
259 # inversión en los ejes Y y Z, lo que resulta en una rotación de 180
260 # grados alrededor del eje X.
261 R_flip[0,0] = 1.0  # No cambia el eje X
262 R_flip[1,1] = -1.0 # Invierte el eje Y
263 R_flip[2,2] = -1.0 # Invierte el eje Z
264
265 # En la llamada a cv2.VideoCapture(create_gstreamer_pipeline(), cv2.
        CAP_GSTREAMER),
```

```
# se utilizan dos argumentos importantes para inicializar una captura de
# video con OpenCV a través de una pipeline de GStreamer personalizada.

    # El primer argumento, create_gstreamer_pipeline(), es una función que
    # retorna una cadena de configuración de GStreamer, especificando cómo
    # capturar y procesar el video. Este argumento le dice a VideoCapture
    # la fuente del video y cómo debe ser manipulado antes de que
    # OpenCV lo procese.

    # El segundo argumento, cv2.CAP_GSTREAMER, es una constante que
    # especifica que OpenCV debe usar el backend de GStreamer para manejar
    # la captura de video. Al usar este backend, OpenCV puede interactuar
    # directamente con GStreamer, aprovechando la flexibilidad y el poder
    # de GStreamer para la manipulación y captura de video en tiempo real.

cam = cv2.VideoCapture(webcam_gstreamer_pipeline(), cv2.CAP_GSTREAMER)

if cam.isOpened() == False:
    print(f"{est.rojo}No se ha podido abrir la cámara{est.reset}\n")
    sys.exit()

# Coordenadas de los vértices del cuadrado de errores cero para una
# distancia de trabajo de 30 cm y un marcador ArUco de 7.4 cm
square_vertices = [(234, 154), (406, 154), (406, 326), (234, 326)]

try:
    while True:

        ret, frame = cam.read()

        if ret == False:
            print(f"{est.rojo}No se puedo capturar el"
                  f" siguiente frame. Terminando...{est.reset}\n")
            break

        # La función ImageProcessing modifica directamente el 'frame' que
        # recibe como argumento.

        # En Python, los objetos como los frames de imágenes se pasan por
        # referencia a las funciones, lo que significa que cualquier
        # modificación dentro de ImageProcessing afecta al frame original.

```

```
308         # Por esto, cuando se detectan y marcan elementos como
                marcadores
309         # ArUco en el frame, estos cambios son visibles
                inmediatamente
310         # después en la misma instancia del frame al usar cv2.imshow.
311
312         # No es necesario retornar el frame modificado desde la funci
                ón,
313         # ya que las alteraciones se aplican directamente al objeto
                de
314         # imagen que continúa existiendo en el contexto global del
                programa.
315
316         ImageProcessing(frame, aruco_dict, parameters, camera_matrix,
                 camera_distortion, R_flip)
317
318         # Dibujar el cuadrado rojo
319         cv2.line(frame, square_vertices[0], square_vertices[1], (0,
                0, 255), 2)
320         cv2.line(frame, square_vertices[1], square_vertices[2], (0,
                0, 255), 2)
321         cv2.line(frame, square_vertices[2], square_vertices[3], (0,
                0, 255), 2)
322         cv2.line(frame, square_vertices[3], square_vertices[0], (0,
                0, 255), 2)
323
324         cv2.imshow("frame", frame)
325
326         # Cerrar la ventana si se presiona la tecla 'q'
327         if cv2.waitKey(1) & 0xFF == ord('q'):
328                 print(" ")
329                 print(f"{est.cyan}Cerrando la ventana gráfica...{est.
                        reset}\n")
330                 gv.running = False
331                 break
332
333     time.sleep(10)
334     print(f"{est.cyan}Fin Main{est.reset}\n")
335
336 finally:
337
338     # Liberar la cámara y destruir todas las ventanas
339     cam.release()
340     cv2.destroyAllWindows()
341
342     # Cerrar conexión del cliente con el servidor
343     print(f"{est.cyan}La conexión con el servidor {ip_server}:{
            puerto_servidor} "
344           f"se ha cerrado satisfactoriamente{est.reset}\n")
345     socket_cliente.send("desco_cliente".encode())
346     socket_cliente.close()
347     print(f"{est.cyan}Fin del programa{est.reset}\n")
348 # ------------------------------------------------------------------ #
```

Código E.1: Archivo main.py.

Apéndice F

Archivo GlobalVariables.py

```
from multiprocessing import Value, Array

# Definición de las variables compartidas
heading_global = Value('d', 0.0)
GS_global      = Value('f', 0.0)
VS_global      = Value('f', 0.0)
alt_global     = Value('f', 0.0)
roll_global    = Value('d', 0.0)
pitch_global   = Value('d', 0.0)
d1_global      = Value('f', 0.0)
d2_global      = Value('f', 0.0)
d3_global      = Value('f', 0.0)
d4_global      = Value('f', 0.0)

getImage:bool
running:bool

errorPitch:float
errorRoll:float
errorYaw:float
errorThrottle:float

KpRP:float
KpY:float
KpT:float

take_off_id_detected:bool
landing_id_detected:bool
control_id_detected:bool

yaw_left_command_send:bool
yaw_right_command_send:bool

go_down_command_send:bool
go_up_command_send:bool
```

```
36
37 roll_right_command_send:bool
38 roll_left_command_send:bool
39
40 go_Fordward_command_send:bool
41 go_Backward_command_send:bool
42
43 estado_vuelo:str
44
45 take_off_executed:bool
46 landing_executed:bool
47 posicionamiento_terminado:bool
```

Código F.1: GlobalVariables.py.

Apéndice G

Archivo ColoresEstilosFormatos.py

```
# Colores de texto
negro = "\033[30m"
rojo = "\033[31m"
verde = "\033[32m"
amarillo = "\033[33m"
azul = "\033[34m"
magenta = "\033[35m"
cyan = "\033[36m"
blanco = "\033[37m"

# Colores de fondo
fondo_negro = "\033[40m"
fondo_rojo = "\033[41m"
fondo_verde = "\033[42m"
fondo_amarillo = "\033[43m"
fondo_azul = "\033[44m"
fondo_magenta = "\033[45m"
fondo_cyan = "\033[46m"
fondo_blanco = "\033[47m"

# Estilos
reset = "\033[0m"
negrita = "\033[1m"
subrayado = "\033[4m"
```

Código G.1: Archivo ColoresEstilosFormatos.py.

Apéndice H

Archivo isRotationMatrix.py

```
# ---------------------- #
# OMAR RABANAL GUTIÉRREZ #
# 16 de Junio del 2024   #
# ---------------------- #


# ------------------------------------------------------------------ #
import numpy as np
import math
import cv2
import cv2.aruco as aruco
from typing import Tuple, Any
import time
import threading
import sys
# ------------------------------------------------------------------ #


# ------------------------------------------------------------------ #
"""

isRotationMatrix(R)

Verifica si una matriz dada es una matriz de rotación.

Las matrices de rotación son matrices cuadradas que representan una
rotación en el espacio tridimensional.

Estas matrices tienen varias propiedades importantes:
    1. Son ortogonales: El producto de una matriz de rotación y su
    transpuesta da como resultado la matriz identidad.

    2. Tienen un determinante de +1.

    3. Conservan las normas de los vectores.
```

```
36
37  Argumentos de entrada:
38  ----------------------
39      R (numpy.ndarray): Una matriz cuadrada de 3x3 que se desea
            verificar si
40      es una matriz de rotación.
41
42  Argumentos de salida:
43  ---------------------
44      bool: Retorna True si la matriz R es una matriz de rotación, de
            lo
45      contrario retorna False.
46
47  Propósito de la función:
48  ------------------------
49  La función isRotationMatrix se utiliza para determinar si una matriz
        dada
50  cumple con las propiedades necesarias para ser considerada una
51  matriz de rotación.
52
53  Lógica interna:
54  ---------------
55      1. Se calcula la transpuesta de la matriz R.
56      2. Se multiplica la transpuesta por la matriz original.
57      3. Se compara el resultado con la matriz identidad de 3x3.
58      4. Se calcula la norma de la diferencia entre el resultado del
            paso
59      3 y la matriz identidad.
60      5. Si la norma es menor que un pequeño umbral (1e-6), se
            considera que
61      R es una matriz de rotación y la función retorna True.
62      De lo contrario, retorna False.
63
64  """
65
66  def isRotationMatrix(R: np.ndarray) -> bool:
67      # Transponer la matriz R
68      Rt = np.transpose(R)
69
70      # Multiplicar la transpuesta de R por R misma
71      shouldBeIdentity = np.dot(Rt, R)
72
73      # Crear una matriz identidad del mismo tipo y dimensiones que R
74      I = np.identity(3, dtype=R.dtype)
75
76      # Calcular la norma de la diferencia entre shouldBeIdentity y la
            matriz identidad
77      n = np.linalg.norm(I - shouldBeIdentity,'fro')
78
79      # Verificar si la norma es menor que un umbral pequeño (1e-6)
80      if n < 1e-6:
81          return True
82
83      else:
84          return False
```

```
85
86  # ---------------------------------------------------------------- #
```

Código H.1: isRotationMatrix.py.

Apéndice I

Archivo rotationMatrixToEulerAngles.py

```
# ---------------------- #
# OMAR RABANAL GUTIÉRREZ #
# 16 de Junio del 2024   #
# ---------------------- #


# ------------------------------------------------------------------ #
import numpy as np
import math
import cv2
import cv2.aruco as aruco
from typing import Tuple, Any
import time
import threading
import sys
# ------------------------------------------------------------------ #


# ------------------------------------------------------------------ #
from modulos.isRotationMatrix import isRotationMatrix
# ------------------------------------------------------------------ #




# ------------------------------------------------------------------ #
"""

rotationMatrixToEulerAngles(R)

Convierte una matriz de rotación en ángulos de Euler.

```

```
33 Los ángulos de Euler son una forma de representar la orientación de
       un objeto
34 en el espacio tridimensional utilizando tres ángulos.
35
36 Esta función toma una matriz de rotación 3x3 y la convierte en los
37 correspondientes ángulos de Euler.
38
39 Argumentos de entrada:
40 ----------------------
41     R (numpy.ndarray): Una matriz de rotación cuadrada de 3x3 que se
42     desea convertir en ángulos de Euler.
43
44 Argumentos de salida:
45 ---------------------
46     numpy.ndarray: Un arreglo de numpy de tamaño 3 que contiene los
47     ángulos de Euler (x, y, z) en radianes.
48
49 Propósito de la función:
50 ------------------------
51 La función rotationMatrixToEulerAngles se utiliza para convertir una
52 matriz de rotación a su representación en ángulos de Euler.
53
54 Lógica interna:
55 ---------------
56     1. Se verifica si la matriz R es una matriz de rotación válida.
57
58     2. Se calcula la variable 'sy' que representa la raíz cuadrada de
           la
59     suma de los cuadrados de los elementos R[0, 0] y R[1, 0].
60
61     3. Se determina si 'sy' es menor que un pequeño umbral (1e-6)
           para
62     identificar una posible singularidad.
63
64     4. Si no hay singularidad, se calculan los ángulos de Euler (x, y
           , z)
65     utilizando funciones de atan2 de la librería math.
66
67     5. Si hay una singularidad, se calculan los ángulos de Euler (x,
           y, z)
68     con un caso especial.
69
70     6. Se retorna un arreglo de numpy con los ángulos de Euler.
71
72 """
73
74 def rotationMatrixToEulerAngles(R: np.ndarray) -> np.ndarray:
75
76     # Verificar si R es una matriz de rotación
77     assert (isRotationMatrix(R) == True)
78
79     # Calcular sy, la raíz cuadrada de la suma de los
80     #  cuadrados de R[0, 0] y R[1, 0]
81     sy = math.sqrt(R[0, 0] * R[0, 0] + R[1, 0] * R[1, 0])
82
```

```
    # Verificar si sy es menor que un pequeño umbral (1e-6)
    # para identificar una singularidad

    if sy < 1e-6:
        singular = True

    else:
        singular = False

    # Si no hay singularidad, calcular los ángulos de Euler
        utilizando atan2
    if singular == False:
        x = math.atan2(R[2, 1], R[2, 2])
        y = math.atan2(-R[2, 0], sy)
        z = math.atan2(R[1, 0], R[0, 0])
    # Si hay singularidad, calcular los ángulos de Euler con un caso
        especial
    else:
        x = math.atan2(-R[1, 2], R[1, 1])
        y = math.atan2(-R[2, 0], sy)
        z = 0

    # Retornar un arreglo de numpy con los ángulos de Euler (x, y, z)
         en radianes
    return np.array([x, y, z])

# ------------------------------------------------------------------ #
```

Código I.1: rotationMatrixToEulerAngles.py.

Apéndice J

Archivo socket_client_Proyecto.py

```
# ---------------------- #
# OMAR RABANAL GUTIÉRREZ #
# 29 de Junio del 2024   #
# ---------------------- #


# ------------------------------------------------------------------ #
import socket
from typing import Optional
import modulos.ColoresEstilosFormatos as est
# ------------------------------------------------------------------ #


# ------------------------------------------------------------------ #
def socket_cliente_ArUco(ip_server:str = "127.0.1.1",
                         puerto_servidor:int = 5000) -> Optional[
                             socket.socket]:

    ip_server = ip_server
    puerto_servidor = puerto_servidor

    print(" ")
    print("-----------------------------------------------")
    print("|             DATOS DEL SERVIDOR              |")
    print("-----------------------------------------------")
    print(f"| Dirección IP del host     --> {ip_server}    |")
    print(f"| Puerto de escucha activo  --> {puerto_servidor}
              |")
    print("-----------------------------------------------")
    print(" ")

    socket_cliente = socket.socket(socket.AF_INET, socket.SOCK_STREAM
        )

    try:
```

```
33          socket_cliente.connect((ip_server,puerto_servidor))
34          print(f"{est.cyan}Se ha establecido satisfactoriamente "
35                f"la conexión con el servidor {ip_server} en el "
36                f"puerto {puerto_servidor}{est.reset}\n")
37          return socket_cliente
38
39      except socket.error as exception:
40          print(f"{est.rojo}Error al conectar con el "
41                f"servidor {ip_server}:{puerto_servidor}: {exception}{
                  est.reset}\n")
42          return None
43  # ------------------------------------------------------------------ #
```

Código J.1: Archivo socket_client_Proyecto.py.

Apéndice K

Archivo socket_server.py

```
# ---------------------- #
# OMAR RABANAL GUTIÉRREZ #
# 21 de Junio del 2024   #
# 22 de Junio del 2024   #
# 23 de Junio del 2024   #
# ---------------------- #


# ---------------------------------------------------------------- #
import socket
import datetime
import threading
import time
import random
import string
from typing import Tuple
# ---------------------------------------------------------------- #


# ---------------------------------------------------------------- #
def gestor_clientes(conn: socket.socket,
                    address: Tuple[str, int],
                    verbose:bool = True) -> None:

    global server_on

    print("-----------------------------------------")
    print(f"Se ha establecido comunicación con un cliente")
    print(f"    - Dirección IP del cliente --> {address[0]}")
    print(f"    - Puerto del cliente       --> {address[1]}")
    print("-----------------------------------------")
    print(" ")

    conn.settimeout(30)

```

```
36      while True:
37
38          try:
39              # Recibe datos del cliente (conn.recv(1024).decode()),
40              # decodificando hasta 1024 bytes.
41              datos_recibidos = conn.recv(1024).decode()
42
43              if server_on == False:
44                  break
45
46              # Si el cliente cierra la conexión este enviará una
47              # cadena vacía
48              if datos_recibidos == "desco_cliente":
49                  print(f"El cliente {address[0]}-{address[1]} se ha
                        desconectado")
50                  print(" ")
51                  conn.close()
52                  return
53
54              tiempo = datetime.datetime.now()
55              tiempo_formateado = tiempo.strftime("%Y-%m-%d %H:%M:%S")
56
57              str_rd_enviar = []
58              for i in range(10):
59                  str_rd_enviar.append(random.choice(random.choice(
                        string.ascii_letters)))
60              code2send = ''.join(str_rd_enviar)
61              conn.send(code2send.encode())
62
63              if verbose == True:
64                  print("-----------------------------------------------
                        ")
65                  print(f"Cliente                --> {address[0]}\n"
66                        f"Puerto cliente         --> {address[1]}\n"
67                        f"Timestamp              --> {tiempo_formateado
                              }\n"
68                        f"Mensaje del cliente    --> {datos_recibidos}\
                              n"
69                        f"Respuesta del servidor --> {code2send}")
70                  print("-----------------------------------------------
                        ")
71                  print(" ")
72
73          except socket.timeout:
74              if server_on == True:
75                  print(f"Timeout: El cliente {address[0]}-{address[1]}
                         no ha"
76                        " realizado peticiones durante los últimos 30
                              segundos.")
77                  print(" ")
78                  continue
79
80              if server_on == False:
81                  break
82
```

```
83          except Exception as e:
84              print(f"Error: {e}")
85              print(" ")
86              return
87
88      msg_error = "desco_server"
89      conn.send(msg_error.encode())
90      conn.close()
91  # ------------------------------------------------------------------ #
92
93
94  # ------------------------------------------------------------------ #
95  def server_program() -> None:
96
97      global server_on
98
99      # socket.gethostname() se utiliza para obtener el nombre del host
            en el
100     # que se está ejecutando el servidor. Este nombre se usará para
            vincular
101     # el socket.
102     nombre_host_server = socket.gethostname()
103
104     # Obtención de la dirección ip del host (servidor)
105     ip_server = socket.gethostbyname(nombre_host_server)
106
107     # Es importante elegir un número de puerto por encima de 1024, ya
            que
108     # los puertos por debajo de este número suelen estar reservados
            para
109     # servicios estándar.
110     puerto_servidor = 5000
111
112     print(" ")
113     print("-----------------------------------------------")
114     print("|             DATOS DEL SERVIDOR              |")
115     print("-----------------------------------------------")
116     print(f"| Nombre del servidor       --> {nombre_host_server}
                |")
117     print(f"| Dirección IP del host     --> {ip_server}    |")
118     print(f"| Puerto de escucha activo  --> {puerto_servidor}
                |")
119     print("-----------------------------------------------")
120     print(" ")
121
122     # Crea un objeto de socket utilizando socket.socket().
123     # Este es el socket de escucha que espera conexiones entrantes.
124     server_socket = socket.socket()
125
126     # Vinculación del socket al host y al puerto especificados.
127     # La función bind() toma una tupla que contiene el host y el
            puerto.
128     server_socket.bind((ip_server,puerto_servidor))
129
```

```
130     # Configura el socket para que pueda escuchar conexiones
            entrantes.
131     # El argumento 2 indica que el servidor puede manejar hasta dos
132     # conexiones en cola antes de aceptar.
133     server_socket.listen(5)
134 
135     timeout_server = 10
136     server_socket.settimeout(timeout_server)
137 
138     print(f"Esperando la recepción de conexiones por el puerto {
            puerto_servidor}...")
139     print(" ")
140     time.sleep(2)
141 
142     server_on = True
143 
144     clientes = []
145     clientes_nuevos = []
146 
147     while True:
148 
149         try:
150 
151             # Acepta una conexión entrante.
152             # accept() devuelve un nuevo objeto de socket conn que se
                    utiliza para
153             # comunicarse con el cliente conectado, y una tupla
                    address que
154             # contiene la dirección IP y el puerto del cliente.
155             conn, address = server_socket.accept()
156 
157             # Crear un nombre único para el hilo basado en la IP y
                    puerto
158             # del cliente
159             thread_name = f"Thread-{address[0]}:{address[1]}"
160 
161             thread_cliente = threading.Thread(name = thread_name,
162                                               target=gestor_clientes,
163                                               args=(conn, address,
                                                      server_on))
164 
165             clientes.append(thread_cliente)
166             clientes_nuevos.append(thread_cliente)
167 
168             print(f"{thread_name} ha comenzado...")
169             print(" ")
170             thread_cliente.start()
171 
172         except socket.timeout:
173             estados_threads = []
174             for thread in clientes:
175                 estados_threads.append(thread.is_alive())
176             threads_vivos = estados_threads.count(True)
177 
178             print("| ======================================== |")
```

```
179             print(f"| Conexiones activas --> {threads_vivos}
                              |")
180             print(f"| Clientes nuevos (últimos 15 sec) --> {len(
                    clientes_nuevos)}   |")
181             print("| ======================================== |")
182             print(" ")
183 
184             clientes_nuevos = []
185             continue
186 
187         except Exception as e:
188             print(f"Error {e}. Cerrando el servidor...")
189             print(" ")
190             server_on = False
191             break
192 
193         except KeyboardInterrupt:
194             print(" Cerrando el servidor... KeyboardInterrupt")
195             print(" ")
196             print("Espere 30 segundos para cerrar las conexiones
                    activas "
197                   "pendientes y proceder al cierre definitivo del
                        servidor.")
198             print(" ")
199             server_on = False
200             break
201 
202     for thread in clientes:
203         thread.join()
204 
205     print("Fin del programa socket_server.")
206     print(" ")
207     time.sleep(5)
208     server_socket.close()
209 # ------------------------------------------------------------------- #
210 
211 
212 # ------------------------------------------------------------------- #
213 if __name__ == "__main__":
214     server_program()
215 # ------------------------------------------------------------------- #
```

Código K.1: Archivo socket_server.py.

Apéndice L

Archivo Telemetry.py

```
# ------------------------------------------------------------------ #
import socket
import struct
import os
from datetime import datetime
import time
import threading
from multiprocessing import Value
import multiprocessing
# ------------------------------------------------------------------ #


# ------------------------------------------------------------------ #
import modulos.ColoresEstilosFormatos as est
import modulos.GlobalVariables as gv
# ------------------------------------------------------------------ #


# ------------------------------------------------------------------ #
def escritura_txt(mensaje:str) -> None:

    # Ruta directorio modulos
    ruta_abs_actual = os.path.dirname(os.path.abspath(__file__))
    # Ruta relativa carpeta LogsTelemetria
    ruta_rel_LogsTelemetria = os.path.join(ruta_abs_actual, '..', '
        LogsTelemetria')
    # Ruta absoluta carpeta images
    ruta_abs_LogsTelemetria = os.path.abspath(ruta_rel_LogsTelemetria
        )

    #timestamp_nombre = datetime.now().strftime("%Y-%m-%d_%H-%M-%S")
    archivo_txt = ruta_abs_LogsTelemetria + "/" + "telemetria.txt"

    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

```

```
34      with open(archivo_txt, "a") as log_file:
35          log_file.write(f"{timestamp} - {mensaje}\n")
36  # ------------------------------------------------------------------ #
37
38
39  # ------------------------------------------------------------------ #
40  def recibir_bytes(sock:socket.socket, num_bytes_objetivo:int) ->
        bytes:
41
42      # Se inicializa una variable buffer
43      buffer = bytearray()
44
45      # Bucle infinito que asegura que se sigan recibiendo datos hasta
46      # acumular el número de bytes esperado
47      while len(buffer) < num_bytes_objetivo:
48
49          paquete = sock.recv(num_bytes_objetivo - len(buffer))
50
51          if not paquete:
52              raise ConnectionError(f"{est.rojo}Conexión cerrada
                    inesperadamente...{est.reset}")
53
54          # Añade el contenido de paquete al final de buffer
55          buffer.extend(paquete)
56
57      return bytes(buffer)
58  # ------------------------------------------------------------------ #
59
60
61  # ------------------------------------------------------------------ #
62  def conectar_server_telemetria(socket_cliente:socket.socket) -> None:
63
64      global socket_cliente_telemetria, connection_msg
65
66      msg_inicio = "COCKPIT\r"
67      socket_cliente.sendall(msg_inicio.encode())
68      time.sleep(4)
69
70      # Datos del servidor
71      ip_server       = "192.168.1.58"
72      puerto_servidor = 9007
73
74      # Establecer la conexión al servidor
75      socket_cliente_telemetria = socket.socket(socket.AF_INET, socket.
            SOCK_STREAM)
76      socket_cliente_telemetria.connect((ip_server, puerto_servidor))
77
78      # Mensaje de conexión
79      connection_msg = socket_cliente_telemetria.recv(1024).decode('utf
            -8')
80      print(f"{est.cyan}Mensaje del servidor: {connection_msg}{est.
            reset}")
81      #escritura_txt(f"Mensaje del servidor: {connection_msg}")
82  # ------------------------------------------------------------------ #
83
```

```
84
85  # ------------------------------------------------------------------ #
86  def timer():
87      global wait_time
88      while True:
89          if wait_time == False:
90              time.sleep(1)
91              wait_time = True
92  # ------------------------------------------------------------------ #
93
94
95  # ------------------------------------------------------------------ #
96  def recibir_telemetria(socket_cliente: socket.socket, heading_global,
        GS_global, VS_global, alt_global, roll_global, pitch_global,
    d1_global, d2_global, d3_global, d4_global) -> None:
97
98      global socket_cliente_telemetria, connection_msg, wait_time
99
100     wait_time = False
101
102     th_timer = threading.Thread(target = timer, daemon = True)
103     th_timer.start()
104
105     try:
106         while True:
107
108             conectar_server_telemetria(socket_cliente)
109
110             if "Aircraft connected" in connection_msg:
111                 print(f"{est.cyan}Esperando datos de telemetría...{
                        est.reset}")
112                 escritura_txt("Esperando datos de telemetría...")
113
114                 # Sincronización inicial
115                 while True:
116
117                     datos = recibir_bytes(socket_cliente_telemetria,
                            52)
118
119                     # Desempaquetar VS y GS para ver si son cero
120                     VS  = struct.unpack_from('>f', datos, 0)[0]
121                     GS  = struct.unpack_from('>f', datos, 4)[0]
122                     alt = struct.unpack_from('>f', datos, 8)[0]
123
124                     if VS == 0.0 and GS == 0.0 and alt == 0.0:
125                         print(f"{est.cyan}Sincronización inicial
                                completada{est.reset}")
126                         escritura_txt("Sincronización inicial
                                completada")
127                         break
128                     else:
129                         print(f"{est.amarillo}Buscando sincronización
                                inicial...{est.reset}")
130                         escritura_txt("Buscando sincronización
                                inicial...")
```

```
131
132                 # Ya estamos sincronizados
133                 while True:
134
135                     try:
136                         datos = recibir_bytes(
                                socket_cliente_telemetria, 52)
137
138                         GS      = struct.unpack_from('>f', datos, 0)
                            [0]
139                         VS      = struct.unpack_from('>f', datos, 4)
                            [0]
140                         alt     = struct.unpack_from('>f', datos, 8)
                            [0]
141                         heading = struct.unpack_from('>d', datos, 12)
                            [0]
142                         roll    = struct.unpack_from('>d', datos, 20)
                            [0]
143                         pitch   = struct.unpack_from('>d', datos, 28)
                            [0]
144                         d1 = struct.unpack_from('>f', datos, 36)[0]
145                         d2 = struct.unpack_from('>f', datos, 40)[0]
146                         d3 = struct.unpack_from('>f', datos, 44)[0]
147                         d4 = struct.unpack_from('>f', datos, 48)[0]
148
149                         with heading_global.get_lock():
150                             heading_global.value = heading
151
152                         with GS_global.get_lock():
153                             GS_global.value = GS
154
155                         with VS_global.get_lock():
156                             VS_global.value = VS
157
158                         with alt_global.get_lock():
159                             alt_global.value = alt
160
161                         with roll_global.get_lock():
162                             roll_global.value = roll
163
164                         with pitch_global.get_lock():
165                             pitch_global.value = pitch
166
167                         with d1_global.get_lock():
168                             d1_global.value = d1
169
170                         with d2_global.get_lock():
171                             d2_global.value = d2
172
173                         with d3_global.get_lock():
174                             d3_global.value = d3
175
176                         with d4_global.get_lock():
177                             d4_global.value = d4
178
```

```
179                         message = f"GS: {GS}, VS: {VS}, Altitude: {
                                alt}, Heading: {heading}, Roll: {roll},
                                Pitch: {pitch}, Distances: [{d1}, {d2}, {
                                d3}, {d4}]"
180                         if wait_time == True:
181                             #print(message)
182                             escritura_txt(message)
183                             wait_time = False
184
185                     except struct.error as e:
186                         print(f"{est.rojo}Error al desempaquetar los
                                datos: {e}{est.reset}")
187                         escritura_txt(f"Error al desempaquetar los
                                datos: {e}")
188                         continue
189
190                     except ConnectionError:
191                         print(f"{est.rojo}Conexión perdida.
                                Reintentando...{est.reset}")
192                         escritura_txt(f"Conexión perdida.
                                Reintentando...")
193                         socket_cliente_telemetria.close()
194                         break
195
196             else:
197                 print(f"{est.rojo}Error al conectarse al servidor
                        para recibir la telemetría: {e}{est.reset}")
198                 escritura_txt(f"Error al conectarse al servidor para
                        recibir la telemetría: {e}")
199
200     except Exception as e:
201         print(f"{est.rojo}Error recibiendo datos: {e}{est.reset}")
202         escritura_txt(f"Error recibiendo datos: {e}")
203
204     finally:
205         print(f"{est.cyan}Cerrando Telemetría...{est.reset}\n")
206         escritura_txt(f"Cerrando Telemetría...")
207         socket_cliente_telemetria.close()
208 # ------------------------------------------------------------------ #
```

Código L.1: Telemetry.py.

Apéndice M

Archivo RasPi_gstreamer_pipeline.py

```
# ---------------------- #
# OMAR RABANAL GUTIÉRREZ #
# 28 de Junio del 2024   #
# ---------------------- #


# ------------------------------------------------------------------ #
import cv2
import os
import subprocess
# ------------------------------------------------------------------ #


# ------------------------------------------------------------------ #
import modulos.ColoresEstilosFormatos as est
# ------------------------------------------------------------------ #


# ------------------------------------------------------------------ #
def raspi_gstreamer_pipeline(sensor_id = 0,
                capture_width = 640, capture_height = 480,
                display_width = 640, display_height = 480,
                framerate = 29,
                flip_method = 0,
                exposure_time = None, exposure_time_min = 13000,
                    exposure_time_max = 683709000,
                gain = None, gain_min = 1.0, gain_max = 10.625,
                saturation = 0.55,
                contrast = 1.0,
                brightness = 0.0,
                hue = 0.0,
                wbmode = 1,
```

```
32                  exposurecompensation = 0,
33                  sharpness = 1.0,
34                  noise_reduction = 1.0):
35
36      """
37      Crea un pipeline de GStreamer para capturar video desde una cá
            mara CSI.
38
39      -----------
40      Parámetros:
41      -----------
42          - sensor_id (int): El ID del sensor de la cámara a utilizar.
43            Predeterminado es 0.
44            Rango: Típicamente 0 hasta el número de sensores conectados
                  - 1.
45
46          - capture_width (int): El ancho de la resolución de captura
                de video.
47            Predeterminado es 1920.
48            Rango: Depende de las capacidades de la cámara.
49
50          - capture_height (int): La altura de la resolución de captura
                 de video.
51            Predeterminado es 1080.
52            Rango: Depende de las capacidades de la cámara.
53
54          - display_width (int): El ancho de la resolución de
                visualización de video.
55            Predeterminado es 640.
56            Rango: Cualquier número entero positivo.
57
58          - display_height (int): La altura de la resolución de
                visualización de video.
59            Predeterminado es 480.
60            Rango: Cualquier número entero positivo.
61
62          - framerate (int): La tasa de cuadros por segundo del video.
63            Predeterminado es 29.
64            Rango: Depende de las capacidades de la cámara y la
                  resolución.
65
66          - flip_method (int): El método para voltear la imagen.
67            Predeterminado es 0.
68              Valores:
69                  - 0: Sin rotación
70                  - 1: Rotar 90 grados en sentido antihorario
71                  - 2: Rotar 180 grados
72                  - 3: Rotar 90 grados en sentido horario
73                  - 4: Volteo horizontal
74                  - 5: Volteo diagonal superior derecha
75                  - 6: Volteo vertical
76                  - 7: Volteo diagonal superior izquierda
77
78          - exposure_time (int, opcional): El tiempo de exposición
                específico en nanosegundos.
```

```
            Predeterminado es None.
            Si se especifica, sobrescribe exposure_time_min y
                exposure_time_max.

        - exposure_time_min (int): El tiempo mínimo de exposición en
            nanosegundos.
            Predeterminado es 13000.
            Rango: Dependiente de la cámara.

        - exposure_time_max (int): El tiempo máximo de exposición en
            nanosegundos.
            Predeterminado es 683709000.
            Rango: Dependiente de la cámara.

        - gain (float, opcional): El valor específico de ganancia.
            Predeterminado es None.
            Si se especifica, sobrescribe gain_min y gain_max.

        - gain_min (float): El valor mínimo de ganancia.
            Predeterminado es 1.0.
            Rango: Dependiente de la cámara.

        - gain_max (float): El valor máximo de ganancia.
            Predeterminado es 10.625.
            Rango: Dependiente de la cámara.

        - saturation (float): Ajusta la saturación del video.
            Predeterminado es 1.0.
            Rango: 0.0 (escala de grises) a valores mayores de 1.0 (más
                saturado).

        - contrast (float): Ajusta el contraste del video.
            Predeterminado es 1.0.
            Rango: 0.0 a 2.0.

        - brightness (float): Ajusta el brillo del video.
            Predeterminado es 0.0.
            Rango: -1.0 a 1.0.

        - hue (float): Ajusta el tono del video.
            Predeterminado es 0.0.
            Rango: -1.0 a 1.0.

        - wbmode (int): Ajusta el modo de balance de blancos.
            Predeterminado es 1.
            Valores:
                - 0: Off
                - 1: Auto
                - 2: Incandescent
                - 3: Fluorescent
                - 4: Daylight
                - 5: Cloudy

        - exposurecompensation (int): Compensación de exposición.
            Predeterminado es 0.
```

```
130             Rango: Dependiente de la cámara.
131
132         - sharpness (float): Ajusta la nitidez del video.
133           Predeterminado es 1.0.
134           Rango: 0.0 a 2.0.
135
136         - noise_reduction (float): Reducción de ruido.
137           Predeterminado es 1.0.
138           Rango: 0.0 a 1.0.
139
140     Retorna:
141     --------
142     - str: Una cadena de texto del pipeline de GStreamer.
143     """
144
145     if exposure_time == None:
146         exposure_param = f"exposuretimerange='{exposure_time_min} {
                exposure_time_max}'"
147     else:
148         exposure_param = f"exposuretimerange='{exposure_time} {
                exposure_time}'"
149
150     if gain == None:
151         gain_param = f"gainrange='{gain_min} {gain_max}'"
152     else:
153         gain_param = f"gainrange='{gain} {gain}'"
154
155     pipeline = (
156         f"nvarguscamerasrc sensor-id={sensor_id} {exposure_param} {
                gain_param} wbmode={wbmode} exposurecompensation={
                exposurecompensation} ! "
157         f"video/x-raw(memory:NVMM), width=(int){capture_width},
                height=(int){capture_height}, framerate=(fraction){
                framerate}/1 ! "
158         f"nvvidconv flip-method={flip_method} ! "
159         f"video/x-raw, width=(int){display_width}, height=(int){
                display_height} ! "
160         f"videobalance saturation={saturation} contrast={contrast}
                brightness={brightness} hue={hue} sharpness={sharpness *
                (1 - noise_reduction)} ! "
161         f"videoconvert ! videoscale ! "
162         f"video/x-raw, format=(string)BGR ! appsink"
163     )
164
165     print(" ")
166     print(f"{est.cyan}Pipeline:{est.reset}\n{pipeline}")
167     print(" ")
168     return pipeline
169 # ------------------------------------------------------------------- #
170
171
172 # ------------------------------------------------------------------- #
173 def show_camera():
174
175     window_title = "CSI Camera"
```

```
176
177     cam = cv2.VideoCapture(raspi_gstreamer_pipeline(sensor_id = 0,
178                     capture_width = 640, capture_height = 480,
179                     display_width = 640, display_height = 480,
180                     framerate = 29,
181                     flip_method = 0,
182                     exposure_time = 683709000/2, exposure_time_min =
                            13000, exposure_time_max = 683709000,
183                     gain = 1, gain_min = 1.0, gain_max = 10.625,
184                     saturation = 0.6,
185                     contrast = 1.0,
186                     brightness = 0.0,
187                     hue = 0,
188                     wbmode = 1,
189                     exposurecompensation = 1,
190                     sharpness = 2,
191                     noise_reduction = 0.5),
192                     cv2.CAP_GSTREAMER)
193
194     if cam.isOpened() == True:
195
196         try:
197
198             # Crea una ventana con propiedades específicas
199             window_handle = cv2.namedWindow(window_title, cv2.
                    WINDOW_AUTOSIZE)
200
201             while True:
202
203                 ret, frame = cam.read()
204
205                 if ret == False:
206                     print("No se pudo capturar el siguiente frame.
                            Terminando...")
207                     break
208
209                 # Verifica si la ventana aún está abierta
210                 if cv2.getWindowProperty(window_title, cv2.
                        WND_PROP_AUTOSIZE) >= 0:
211                     cv2.imshow(window_title, frame)
212                 else:
213                     break
214
215                 keyCode = cv2.waitKey(1) & 0xFF
216                 if keyCode == 27 or keyCode == ord('q'):
217                     break
218
219         finally:
220             cam.release()
221             cv2.destroyAllWindows()
222
223     else:
224         print("Error: Unable to open camera")
225 # ------------------------------------------------------------------ #
226
```

```
227
228 # ------------------------------------------------------------------ #
229 def get_camera_capabilities(device: str = '/dev/video0',
230                             archivo_txt: str = "RasPi_Capabilities.
                                    txt") -> None:
231
232     # Definir los comandos para obtener las capacidades de la cámara
            V4L2
233     commands = [
234         "v4l2-ctl --list-devices",
235         f"v4l2-ctl -d {device} --all",
236         f"v4l2-ctl -d {device} --list-formats-ext"
237     ]
238
239     # Ruta del directorio actual del script
240     ruta_actual = os.path.dirname(os.path.abspath(__file__))
241
242     # Ruta completa para el archivo de texto
243     fichero_ruta = os.path.join(ruta_actual, archivo_txt)
244
245     print(" ")
246     # Apertura del archivo para escribir los resultados de los
            comandos
247     with open(fichero_ruta, 'w') as file:
248         for command in commands:
249             # Ejecutar cada comando en un nuevo proceso y captura la
                    salida estándar y de error
250             process = subprocess.Popen(command, shell = True, stdout
                    = subprocess.PIPE, stderr = subprocess.PIPE)
251             stdout, stderr = process.communicate()
252
253             # Si hay algún error, se imprime y se sigue con el
                    siguiente comando
254             if stderr == True:
255                 print(f"{est.rojo}Error al obtener las capacidades
                        del sensor con el comando '{command}': {est.reset
                        }{stderr.decode()}\n")
256                 file.write(f"Error: {stderr.decode()}\n")
257                 continue
258
259             # Decodificar la salida estándar y escribirla en el
                    archivo
260             output = stdout.decode()
261             print(f"{est.cyan}Resultado del comando '{command}':{est.
                    reset}\n{output}\n")
262             file.write(f"Resultado del comando '{command}':\n{output
                    }\n")
263
264     # Mensaje de que el archivo de resultados se ha generado
            correctamente
265     print(f"{est.verde}Capacidades del sensor {device} guardadas en {
            fichero_ruta}\n{est.reset}")
266 # ------------------------------------------------------------------ #
267
268
```

```
269 # ------------------------------------------------------------------- #
270 if __name__ == "__main__":
271     show_camera()
272     get_camera_capabilities()
273 # ------------------------------------------------------------------- #
```

Código M.1: Archivo RasPi_gstreamer_pipeline.py.

Apéndice N

Archivo moveSimulador.py

```python
# ---------------------- #
# OMAR RABANAL GUTIÉRREZ #
# 19 de Junio del 2024   #
# ---------------------- #


# ---------------------------------------------------------------- #
import numpy as np
import math
import cv2
from typing import Tuple, Any
import time
import threading
import sys
import socket
# ---------------------------------------------------------------- #


# ---------------------------------------------------------------- #
import modulos.GlobalVariables as gv
from modulos.isRotationMatrix import isRotationMatrix
from modulos.rotationMatrixToEulerAngles import
    rotationMatrixToEulerAngles
import modulos.ColoresEstilosFormatos as est
# ---------------------------------------------------------------- #


# ---------------------------------------------------------------- #
def moveSimulador(socket_cliente: socket.socket) -> None:

    # Inicio del bucle de control
    gv.running = True

    while gv.running == True:

```

```
35          if gv.getImage == True: # Comprueba si hay una nueva imagen
                procesada
36
37              gv.getImage = False # Restablece la bandera de control de
                    imágenes
38
39              ######################################
40              # COPIA LOCAL DE VARIABLES GLOBALES #
41              ######################################
42              take_off_id_detected = gv.take_off_id_detected
43              landing_id_detected = gv.landing_id_detected
44              control_id_detected = gv.control_id_detected
45
46              errorRoll = gv.errorRoll
47              errorPitch = gv.errorPitch
48              errorYaw = gv.errorYaw
49              errorThrottle = gv.errorThrottle
50
51              #print('ERoll = %.5f  EPitch = %.5f  EYaw  = %.5f
                    EThrottle = %.5f' % (errorRoll, errorPitch, errorYaw,
                     errorThrottle))
52
53              if take_off_id_detected == True:
54                  msg_takeoff = "TAKE OFF ACTION\r"
55                  socket_cliente.sendall(msg_takeoff.encode())
56                  print(f"{est.verde}Función move: Comando de despegue
                        enviado al puente...{est.reset}")
57                  time.sleep(2)
58
59              if landing_id_detected == True:
60                  msg_landing = "LAND ACTION\r"
61                  msg_cero    = "0\r"
62                  socket_cliente.sendall(msg_cero.encode())
63                  socket_cliente.sendall(msg_landing.encode())
64                  print(f"{est.verde}Función move: Comando de
                        aterrizaje enviado al puente...{est.reset}")
65                  time.sleep(2)
66
67              if control_id_detected == True:
68                  ###################################
69                  # Ajuste de los errores de pitch #
70                  ###################################
71                  if abs(errorPitch) < 0.1:
72                      errorPitch = 0 # Si el error es muy pequeño, se
                            ignora
73                  if abs(errorPitch) > 0.7:
74                      errorPitch = 0.7 * (errorPitch / abs(errorPitch))
                              # Limita el error de pitch a +/-0.7
75
76                  ###################################
77                  # Ajuste de los errores de roll #
78                  ###################################
79                  if abs(errorRoll) < 0.1:
80                      errorRoll = 0  # Ignora pequeños errores de roll
81                  if abs(errorRoll) > 0.7:
```

```
82                      errorRoll = 0.7 * (errorRoll / abs(errorRoll))  #
                            Limita el error de roll a +/-0.7
83
84                  ################################
85                  # Ajuste de los errores de yaw #
86                  ################################
87                  if abs(errorYaw) < 0.1:
88                      errorYaw = 0  # Ignora pequeños errores de yaw
89                  if abs(errorYaw) > 0.8:
90                      errorYaw = 0.8 * (errorYaw / abs(errorYaw))  #
                            Limita el error de yaw a +/-0.8
91
92                  ######################################
93                  # Ajuste de los errores de throttle #
94                  ######################################
95                  if abs(errorThrottle) < 0.1:
96                      errorThrottle = 0  # Ignora pequeños errores de
                            throttle
97                  if abs(errorThrottle) > 0.6:
98                      errorThrottle = 0.6 * (errorThrottle / abs(
                            errorThrottle))  # Limita el error de
                            throttle a +/-0.6
99
100                 ####################################
101                 # PRINTEO DE LOS ERRORES AJUSTADOS #
102                 ####################################
103                 print(f"{est.magenta}ERoll = {errorRoll:.5f} "
104                                 f"EPitch = {errorPitch:.5f} "
105                                 f"EYaw = {errorYaw:.5f} "
106                                 f"EThrottle = {errorThrottle:.5f}{
                                        est.reset}")
107
108                 ################################################
109                 # ------------------------------------------ #
110                 # ENVÍO DE COMANDOS PARA LA CORRECIÓN DEL YAW #
111                 # ------------------------------------------ #
112                 msg_yawleft  = "YAW LEFT\r"
113                 msg_yawright = "YAW RIGHT\r"
114                 msg_cero     = "0\r"
115
116                 if errorYaw > 0.2:
117                     if gv.yaw_right_command_send == False:
118                         socket_cliente.sendall(msg_yawright.encode())
119                         print(f"{est.verde}Comando YAW RIGHT enviado
                                    al puente...{est.reset}")
120                         gv.yaw_right_command_send = True
121
122                 if errorYaw < -0.2:
123                     if gv.yaw_left_command_send == False:
124                         socket_cliente.sendall(msg_yawleft.encode())
125                         print(f"{est.verde}Comando YAW LEFT enviado
                                    al puente...{est.reset}")
126                         gv.yaw_left_command_send = True
127
128                 if -0.1 < errorYaw < 0.1:
```

```
129                     if gv.yaw_right_command_send == True or gv.
                            yaw_left_command_send == True:
130                         socket_cliente.sendall(msg_cero.encode())
131                         print(f"{est.verde}Comando 0 enviado al
                                puente...{est.reset}")
132                         gv.yaw_right_command_send = False
133                         gv.yaw_left_command_send = False
134                 ######################################################
135
136                 ######################################################
137                 # ------------------------------------------------ #
138                 # ENVÍO DE COMANDOS PARA LA CORRECIÓN DEL THROTLE #
139                 # ------------------------------------------------ #
140                 msg_goDown = "GO DOWN\r"
141                 msg_goUp   = "GO UP\r"
142                 msg_cero   = "0\r"
143
144                 if errorThrottle < -0.20:
145                     if gv.go_down_command_send == False:
146                         socket_cliente.sendall(msg_goDown.encode())
147                         print(f"{est.verde}Comando GO DOWN enviado al
                                puente...{est.reset}")
148                         gv.go_down_command_send = True
149
150                 if errorThrottle > 0.20:
151                     if gv.go_up_command_send == False:
152                         socket_cliente.sendall(msg_goUp.encode())
153                         print(f"{est.verde}Comando GO UP enviado al
                                puente...{est.reset}")
154                         gv.go_up_command_send = True
155
156                 if -0.10 < errorThrottle < 0.10:
157                     if gv.go_down_command_send == True or gv.
                            go_up_command_send == True:
158                         socket_cliente.sendall(msg_cero.encode())
159                         print(f"{est.verde}Comando 0 enviado al
                                puente...{est.reset}")
160                         gv.go_down_command_send = False
161                         gv.go_up_command_send = False
162                 ######################################################
163
164                 ######################################################
165                 # ------------------------------------------------ #
166                 # ENVÍO DE COMANDOS PARA LA CORRECIÓN DEL ROLL #
167                 # ------------------------------------------------ #
168                 msg_rollRight = "ROLL RIGHT\r"
169                 msg_rollLeft  = "ROLL LEFT\r"
170                 msg_cero      = "0\r"
171
172                 if errorRoll > 0.20:
173                     if gv.roll_right_command_send == False:
174                         socket_cliente.sendall(msg_rollRight.encode()
                                )
175                         print(f"{est.verde}Comando ROLL RIGHT enviado
                                al puente...{est.reset}")
```

```
176                         gv.roll_right_command_send = True
177
178                 if errorRoll < -0.20:
179                     if gv.roll_left_command_send == False:
180                         socket_cliente.sendall(msg_rollLeft.encode())
181                         print(f"{est.verde}Comando ROLL LEFT enviado
                                al puente...{est.reset}")
182                         gv.roll_left_command_send = True
183
184                 if -0.10 < errorRoll < 0.10:
185                     if gv.roll_right_command_send == True or gv.
                            roll_left_command_send == True:
186                         socket_cliente.sendall(msg_cero.encode())
187                         print(f"{est.verde}Comando 0 enviado al
                                puente...{est.reset}")
188                         gv.roll_right_command_send = False
189                         gv.roll_left_command_send = False
190                 ##################################################
191
192                 ##################################################
193                 # ------------------------------------------- #
194                 # ENVÍO DE COMANDOS PARA LA CORRECIÓN DEL PITCH #
195                 # ------------------------------------------- #
196                 msg_goFordward = "GO FORWARD\r"
197                 msg_goBackward = "GO BACKWARD\r"
198                 msg_cero       = "0\r"
199
200                 if errorPitch > 0.20:
201                     if gv.go_Fordward_command_send == False:
202                         socket_cliente.sendall(msg_goFordward.encode
                                ())
203                         print(f"{est.verde}Comando GO FORWARD enviado
                                 al puente...{est.reset}")
204                         gv.go_Fordward_command_send = True
205
206                 if errorPitch < -0.20:
207                     if gv.go_Backward_command_send == False:
208                         socket_cliente.sendall(msg_goBackward.encode
                                ())
209                         print(f"{est.verde}Comando GO BACKWARD
                                enviado al puente...{est.reset}")
210                         gv.go_Backward_command_send = True
211
212                 if -0.10 < errorPitch < 0.10:
213                     if gv.go_Fordward_command_send == True or gv.
                            go_Backward_command_send == True:
214                         socket_cliente.sendall(msg_cero.encode())
215                         print(f"{est.verde}Comando 0 enviado al
                                puente...{est.reset}")
216                         gv.go_Fordward_command_send = False
217                         gv.go_Backward_command_send = False
218                 ##################################################
219
220             time.sleep(0.3) # Pequeña pausa para no sobrecargar el
                    ciclo de control
```

```
221
222         else:
223
224             time.sleep(0.01) # Pausa muy breve si no hay imagen nueva
                    para procesar
225
226 ######################################################################
227         with gv.heading_global.get_lock():
228             heading_value = gv.heading_global.value
229         with gv.GS_global.get_lock():
230             GS_value = gv.GS_global.value
231         with gv.VS_global.get_lock():
232             VS_value = gv.VS_global.value
233         with gv.alt_global.get_lock():
234             alt_value = gv.alt_global.value
235         with gv.roll_global.get_lock():
236             roll_value = gv.roll_global.value
237         with gv.pitch_global.get_lock():
238             pitch_value = gv.pitch_global.value
239         with gv.d1_global.get_lock():
240             d1_value = gv.d1_global.value
241         with gv.d2_global.get_lock():
242             d2_value = gv.d2_global.value
243         with gv.d3_global.get_lock():
244             d3_value = gv.d3_global.value
245         with gv.d4_global.get_lock():
246             d4_value = gv.d4_global.value
247         print(f"GS: {GS_value}, VS: {VS_value}, Alt: {alt_value},
                Heading: {heading_value}, Roll: {roll_value}, Pitch: {
                pitch_value}, Distances: [{d1_value}, {d2_value}, {
                d3_value}, {d4_value}]")
248 ######################################################################
249
250     print(f"{est.cyan}Fin move{est.reset}\n") # Indica en la consola
            que el bucle de control ha terminado
251 # ---------------------------------------------------------------- #
```

Código N.1: moveSimulador.py.

Apéndice Ñ

Archivo moveVuelo.py

```
# ------------------------------------------------------------------ #
import socket
import time
# ------------------------------------------------------------------ #


# ------------------------------------------------------------------ #
import modulos.GlobalVariables as gv
import modulos.ColoresEstilosFormatos as est
# ------------------------------------------------------------------ #


# ------------------------------------------------------------------ #
def moveVuelo(socket_cliente: socket.socket) -> None:

    # Inicio del bucle de control
    gv.running = True

    global socket_cliente_move
    socket_cliente_move = socket_cliente

    # Creacción del diccionario de estados de vuelo
    global dicc_estados
    dicc_estados = {"no_state": no_state,
                    "despegue": despegar,
                    "posicionamiento": posicionarse,
                    "avance": avanzar,
                    "aterrizaje": aterrizar}

    while gv.running == True:

        if gv.getImage == True: # Comprueba si hay una nueva imagen
            procesada

```

```
34              gv.getImage = False # Restablece la bandera de control de
                    imágenes
35
36              switch(gv.estado_vuelo)
37
38          else:
39
40              time.sleep(0.01) # Pausa muy breve si no hay imagen nueva
                    para procesar
41
42      print(f"{est.cyan}Fin move{est.reset}\n") # Indica en la consola
            que el bucle de control ha terminado
43  # ------------------------------------------------------------------- #
44
45
46  # ------------------------------------------------------------------- #
47  def switch(estado_vuelo:str) -> None:
48      global dicc_estados
49      accion = dicc_estados.get(estado_vuelo)
50      accion()
51  # ------------------------------------------------------------------- #
52
53
54  # ------------------------------------------------------------------- #
55  def no_state():
56      pass
57  # ------------------------------------------------------------------- #
58
59
60  # ------------------------------------------------------------------- #
61  def despegar():
62      global socket_cliente_move
63      msg_takeoff = "TAKE OFF ACTION\r"
64      socket_cliente_move.sendall(msg_takeoff.encode())
65      print(f"{est.verde}Función move: Comando de despegue enviado al
            puente...{est.reset}")
66      gv.take_off_executed = True
67      gv.landing_executed  = False
68      time.sleep(5)
69      print(f"{est.amarillo}Fase de despegue terminada{est.reset}")
70  # ------------------------------------------------------------------- #
71
72
73  # ------------------------------------------------------------------- #
74  def posicionarse():
75
76      global socket_cliente_move
77      while True:
78
79          if gv.control_id_detected == False and gv.
                take_off_id_detected == False and gv.landing_id_detected
                == False:
80              msg_cero = "0\r"
81              socket_cliente_move.sendall(msg_cero.encode())
```

```
82              print(f"{est.rojo}Fase de posicionamiento interrumpida{
                    est.reset}")
83              gv.go_Fordward_command_send = False
84              gv.go_Backward_command_send = False
85              gv.yaw_right_command_send   = False
86              gv.yaw_left_command_send    = False
87              gv.go_down_command_send     = False
88              gv.go_up_command_send       = False
89              gv.roll_right_command_send  = False
90              gv.roll_left_command_send   = False
91              break
92
93          ############################################################
94          # COPIA LOCAL DE LAS VARIABLES GLOBALES DE LOS ERRORES #
95          ############################################################
96          errorRoll     = gv.errorRoll
97          errorPitch    = gv.errorPitch
98          errorYaw      = gv.errorYaw
99          errorThrottle = gv.errorThrottle
100
101         ###################################
102         # Ajuste de los errores de pitch #
103         ###################################
104         if abs(errorPitch) < 0.1:
105             errorPitch = 0 # Si el error es muy pequeño, se ignora
106         if abs(errorPitch) > 0.7:
107             errorPitch = 0.7 * (errorPitch / abs(errorPitch))  #
                    Limita el error de pitch a +/-0.7
108
109         ###################################
110         # Ajuste de los errores de roll #
111         ###################################
112         if abs(errorRoll) < 0.1:
113             errorRoll = 0  # Ignora pequeños errores de roll
114         if abs(errorRoll) > 0.7:
115             errorRoll = 0.7 * (errorRoll / abs(errorRoll))  # Limita
                    el error de roll a +/-0.7
116
117         ###################################
118         # Ajuste de los errores de yaw #
119         ###################################
120         if abs(errorYaw) < 0.1:
121             errorYaw = 0  # Ignora pequeños errores de yaw
122         if abs(errorYaw) > 0.8:
123             errorYaw = 0.8 * (errorYaw / abs(errorYaw))  # Limita el
                    error de yaw a +/-0.8
124
125         ######################################
126         # Ajuste de los errores de throttle #
127         ######################################
128         if abs(errorThrottle) < 0.1:
129             errorThrottle = 0  # Ignora pequeños errores de throttle
130         if abs(errorThrottle) > 0.6:
131             errorThrottle = 0.6 * (errorThrottle / abs(errorThrottle)
                    )  # Limita el error de throttle a +/-0.6
```

```
132
133             ####################################
134             # PRINTEO DE LOS ERRORES AJUSTADOS #
135             ####################################
136             print(f"{est.magenta}ERoll = {errorRoll:.5f} "
137                   f"EPitch = {errorPitch:.5f} "
138                   f"EYaw = {errorYaw:.5f} "
139                   f"EThrottle = {errorThrottle:.5f}{est.reset}")
140
141             ######################################################
142             # BÚSQUEDA DEL ERROR MÁS GRANDE EN VALOR ABSOLUTO #
143             ######################################################
144             errores_abs = [abs(errorRoll),abs(errorPitch),abs(errorYaw),
                    abs(errorThrottle)]
145             max_error_abs = max(errores_abs)
146             max_error_abs_index = errores_abs.index(max_error_abs)
147
148             #print(f"{est.amarillo}El error más grande en valor absoluto
                    es: {max_error_abs}{est.reset}")
149             #print(f"{est.amarillo}El índice del error más grande en
                    valor absoluto es: {max_error_abs_index}{est.reset}")
150
151             if max_error_abs == 0:
152                 msg_cero = "0\r"
153                 socket_cliente_move.sendall(msg_cero.encode())
154                 print(f"{est.amarillo}Fase de posicionamiento terminada{
                        est.reset}")
155                 gv.posicionamiento_terminado = True
156                 break
157
158             ############################
159             # CORRECIÓN DE LOS ERRORES #
160             ############################
161
162             # CORRECIÓN DE ROLL
163             if max_error_abs_index == 0:
164
165                 gv.go_Fordward_command_send = False
166                 gv.go_Backward_command_send = False
167
168                 gv.yaw_right_command_send = False
169                 gv.yaw_left_command_send  = False
170
171                 gv.go_down_command_send = False
172                 gv.go_up_command_send   = False
173
174                 msg_rollRight = "ROLL RIGHT\r"
175                 msg_rollLeft  = "ROLL LEFT\r"
176
177                 if gv.roll_right_command_send == False:
178                     if errorRoll > 0.10:
179                         socket_cliente_move.sendall(msg_rollRight.encode
                                ())
180                         print(f"{est.verde}Comando ROLL RIGHT enviado al
                                puente...{est.reset}")
```

```
                    gv.roll_right_command_send = True

            if gv.roll_left_command_send == False:
                if errorRoll < -0.10:
                    socket_cliente_move.sendall(msg_rollLeft.encode()
                        )
                    print(f"{est.verde}Comando ROLL LEFT enviado al
                        puente...{est.reset}")
                    gv.roll_left_command_send = True

        # CORRECIÓN DE PITCH
        if max_error_abs_index == 1:

            gv.roll_right_command_send = False
            gv.roll_left_command_send  = False

            gv.yaw_right_command_send = False
            gv.yaw_left_command_send  = False

            gv.go_down_command_send = False
            gv.go_up_command_send   = False

            msg_goFordward = "GO FORWARD\r"
            msg_goBackward = "GO BACKWARD\r"

            if gv.go_Fordward_command_send == False:
                if errorPitch > 0.10:
                    socket_cliente_move.sendall(msg_goFordward.encode
                        ())
                    print(f"{est.verde}Comando GO FORWARD enviado al
                        puente...{est.reset}")
                    gv.go_Fordward_command_send = True

            if gv.go_Backward_command_send == False:
                if errorPitch < -0.10:
                    socket_cliente_move.sendall(msg_goBackward.encode
                        ())
                    print(f"{est.verde}Comando GO BACKWARD enviado al
                         puente...{est.reset}")
                    gv.go_Backward_command_send = True

        # CORRECIÓN DE YAW
        if max_error_abs_index == 2:

            gv.go_Fordward_command_send = False
            gv.go_Backward_command_send = False

            gv.roll_right_command_send = False
            gv.roll_left_command_send  = False

            gv.go_down_command_send = False
            gv.go_up_command_send   = False

            msg_yawleft  = "YAW LEFT\r"
            msg_yawright = "YAW RIGHT\r"
```

```
230
231             if gv.yaw_right_command_send == False:
232                 if errorYaw > 0.2:
233                     socket_cliente_move.sendall(msg_yawright.encode()
                            )
234                     print(f"{est.verde}Comando YAW RIGHT enviado al
                            puente...{est.reset}")
235                     gv.yaw_right_command_send = True
236
237             if gv.yaw_left_command_send == False:
238                 if errorYaw < -0.2:
239                     socket_cliente_move.sendall(msg_yawleft.encode())
240                     print(f"{est.verde}Comando YAW LEFT enviado al
                            puente...{est.reset}")
241                     gv.yaw_left_command_send = True
242
243         # CORRECIÓN DE THROTTLE
244         if max_error_abs_index == 3:
245
246             gv.go_Fordward_command_send = False
247             gv.go_Backward_command_send = False
248
249             gv.roll_right_command_send = False
250             gv.roll_left_command_send  = False
251
252             gv.yaw_right_command_send = False
253             gv.yaw_left_command_send  = False
254
255             msg_goDown = "GO DOWN\r"
256             msg_goUp   = "GO UP\r"
257
258             if gv.go_down_command_send == False:
259                 if errorThrottle < -0.20:
260                     socket_cliente_move.sendall(msg_goDown.encode())
261                     print(f"{est.verde}Comando GO DOWN enviado al
                            puente...{est.reset}")
262                     gv.go_down_command_send = True
263
264             if gv.go_up_command_send == False:
265                 if errorThrottle > 0.20:
266                     socket_cliente_move.sendall(msg_goUp.encode())
267                     print(f"{est.verde}Comando GO UP enviado al
                            puente...{est.reset}")
268                     gv.go_up_command_send = True
269
270         time.sleep(0.3)
271
272     if gv.posicionamiento_terminado == True:
273         if gv.take_off_id_detected == True or gv.control_id_detected
                == True:
274             time.sleep(3)
275             gv.estado_vuelo = "avance"
276         if gv.landing_id_detected == True:
277             time.sleep(3)
278             gv.estado_vuelo = "aterrizaje"
```

```
279     else:
280         gv.estado_vuelo = "no_state"
281 # ------------------------------------------------------------------ #
282 
283 
284 # ------------------------------------------------------------------ #
285 def avanzar():
286     global socket_cliente_move
287 
288     print(f"{est.amarillo}Comenzando fase de avance{est.reset}")
289 
290     msg_goFordward = "GO FORWARD\r"
291     socket_cliente_move.sendall(msg_goFordward.encode())
292     print(f"{est.verde}Comando GO FORWARD enviado al puente...{est.
        reset}")
293 
294     gv.estado_vuelo = "no_state"
295     time.sleep(5)
296 
297     while True:
298         if gv.control_id_detected == True or gv.landing_id_detected
            == True:
299             gv.posicionamiento_terminado = False
300             break
301         time.sleep(0.3)
302 # ------------------------------------------------------------------ #
303 
304 
305 # ------------------------------------------------------------------ #
306 def aterrizar():
307     global socket_cliente_move
308     print(f"{est.amarillo}Comenzando fase de aterrizaje{est.reset}")
309     msg_landing = "LAND ACTION\r"
310     socket_cliente_move.sendall(msg_landing.encode())
311     print(f"{est.verde}Función move: Comando de aterrizaje enviado al
          puente...{est.reset}")
312     gv.landing_executed  = True
313     gv.take_off_executed = False
314     gv.posicionamiento_terminado = False
315     gv.estado_vuelo = "no_state"
316     time.sleep(2)
317 # ------------------------------------------------------------------ #
```

Código Ñ.1: moveVuelo.py.

Apéndice O

Archivo ImageProcessingArUcoSimulador.py

```python
# ---------------------- #
# OMAR RABANAL GUTIÉRREZ #
# 16 de Junio del 2024   #
# ---------------------- #


# ------------------------------------------------------------------ #
import numpy as np
import math
import cv2
import cv2.aruco as aruco
from typing import Tuple, Any
import time
import threading
import sys
import random as rd
# ------------------------------------------------------------------ #


# ------------------------------------------------------------------ #
import modulos.GlobalVariables as gv
from modulos.isRotationMatrix import isRotationMatrix
from modulos.rotationMatrixToEulerAngles import
    rotationMatrixToEulerAngles
import modulos.ColoresEstilosFormatos as est
# ------------------------------------------------------------------ #


# ------------------------------------------------------------------ #
def ImageProcessingSimulador(image: np.ndarray,
                             aruco_dict: Any,
```

```
31                              parameters: cv2.aruco.DetectorParameters
                                    ,
32                              camera_matrix: np.ndarray,
33                              camera_distortion: np.ndarray,
34                              R_flip: np.ndarray) -> None:
35
36      # Definición del id de los códigos ArUco a detectar
37      id_take_off = 2
38      id_control  = 10
39      id_landing  = 7
40
41      marker_size = 7.4  #cm
42      distanciaZ  = 30   #cm
43
44      AFOV_diag_deg_webcam = 55                                       #[
            deg]
45      AFOV_hori_deg_webcam = FOVdiag2FOVhori(AFOV_diag_deg_webcam) #[
            deg]
46
47      # Paso 1: Convertir la imagen de entrada a escala de grises
48          # Convertir a escala de grises simplifica el procesamiento de
                la
49          # imagen al reducir la complejidad de la imagen, lo que es
50          # beneficioso para la detección de marcadores ArUco, ya que
                estos
51          # solo necesitan contraste entre blanco y negro.
52      gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
53
54      # Paso 2: Detectar marcadores ArUco
55          # 'corners' son las coordenadas de las esquinas de los
                marcadores detectados.
56          # 'ids' son los identificadores de los marcadores detectados
                que
57          # corresponden a los definidos en 'aruco_dict'.
58          # 'rejectedImgPoints' contiene los candidatos de marcadores
                que
59          # fueron considerados pero no reconocidos como válidos.
60      corners, ids, rejectedImgPoints = cv2.aruco.detectMarkers(gray,
            aruco_dict, parameters=parameters)
61
62      # Paso 3: Verificar si el marcador deseado está presente
63
64      ###################
65      # ArUco Take-off #
66      ###################
67      if len(corners) > 0 and ids[0] == id_take_off:
68          take_off_id_detected = True
69
70          #print("Función ArUco_Control: Código ArUco de despegue
                detectado...")
71
72          for corner in corners:
73              centerX = (corner[0][0][0] + corner[0][1][0] + corner
                    [0][2][0] + corner[0][3][0]) / 4
```

```
74              centerY = (corner[0][0][1] + corner[0][1][1] + corner
                    [0][2][1] + corner[0][3][1]) / 4
75
76          ret = cv2.aruco.estimatePoseSingleMarkers(corners,
                marker_size, camera_matrix, camera_distortion)
77          rvec, tvec = ret[0][0,0,:], ret[1][0,0,:]
78
79          cv2.aruco.drawDetectedMarkers(image, corners)
80          cv2.drawFrameAxes(image, camera_matrix, camera_distortion,
                rvec, tvec, 5)
81
82      else:
83          take_off_id_detected = False
84
85      ##################
86      # ArUco Landing #
87      ##################
88      if len(corners) > 0 and ids[0] == id_landing:
89          landing_id_detected = True
90
91          #print("Código ArUco de aterrizaje detectado, procediendo al
                aterrizaje de la aeronave...")
92
93          for corner in corners:
94              centerX = (corner[0][0][0] + corner[0][1][0] + corner
                    [0][2][0] + corner[0][3][0]) / 4
95              centerY = (corner[0][0][1] + corner[0][1][1] + corner
                    [0][2][1] + corner[0][3][1]) / 4
96
97          ret = cv2.aruco.estimatePoseSingleMarkers(corners,
                marker_size, camera_matrix, camera_distortion)
98          rvec, tvec = ret[0][0,0,:], ret[1][0,0,:]
99
100         cv2.aruco.drawDetectedMarkers(image, corners)
101         cv2.drawFrameAxes(image, camera_matrix, camera_distortion,
                rvec, tvec, 5)
102
103     else:
104         landing_id_detected = False
105
106     ##################
107     # ArUco Control #
108     ##################
109     if len(corners) > 0 and ids[0] == id_control:
110         control_id_detected = True
111
112         # Calcular el centro del marcador detectado
113             # Las 'corners' son arrays con las coordenadas de las
                    esquinas de
114             # cada marcador detectado en la imagen.
115
116             # centerX y centerY calculan el promedio de las esquinas
                    para
117             # encontrar el centro del marcador en la imagen.
118
```

```
119             # Esto se hace en el sistema de coordenadas de la imagen,
120             # donde x es horizontal y y es vertical desde la esquina
121             # superior izquierda.
122         for corner in corners:
123             centerX = (corner[0][0][0] + corner[0][1][0] + corner
                    [0][2][0] + corner[0][3][0]) / 4
124             centerY = (corner[0][0][1] + corner[0][1][1] + corner
                    [0][2][1] + corner[0][3][1]) / 4
125 
126         for corner in corners:
127             tam_aruco_pix = abs(corner[0][1][0] - corner[0][0][0])
                        # [pix]
128             if tam_aruco_pix < 10:
129                 tam_aruco_pix = abs(corner[0][2][0] - corner
                        [0][1][0]) # [pix]
130         #print(f"{est.magenta}ArUco en Píxeles: {tam_aruco_pix}{est.
            reset}")
131 
132         # Paso 4: Estimar la pose del marcador
133             # 'estimatePoseSingleMarkers' calcula la posición y
                orientación
134             # del marcador en el espacio tridimensional respecto a la
                 cámara.
135 
136             # rvec es el vector de rotación del marcador medido en el
                 sistema de
137             # coordenadas de la cámara.
138 
139             # tvec es el vector de traslación que indica la posición
                del marcador
140             # en el espacio, medido en el sistema de coordenadas de
                la cámara.
141 
142         ret = cv2.aruco.estimatePoseSingleMarkers(corners,
            marker_size, camera_matrix, camera_distortion)
143         rvec, tvec = ret[0][0,0,:], ret[1][0,0,:]
144 
145         # Paso 5: Visualizar marcadores y ejes
146             # 'drawDetectedMarkers' dibuja los marcadores detectados
147             # en la imagen.
148 
149             # 'drawFrameAxes' añade un sistema de coordenadas visual
                en el
150             # marcador para ilustrar la orientación del marcador.
151 
152         cv2.aruco.drawDetectedMarkers(image, corners)
153         cv2.drawFrameAxes(image, camera_matrix, camera_distortion,
            rvec, tvec, 5)
154 
155         # Paso 6 y 7: Calcular la matriz de rotación y la posición
156             # R_ct transforma coordenadas desde el sistema del
                marcador
157             # al sistema de la cámara.
158 
```

```
159             # R_tc es la transpuesta de R_ct, transformando
                    coordenadas
160             # desde el sistema de la cámara al del marcador.
161
162         R_ct = np.matrix(cv2.Rodrigues(rvec)[0])
163         R_tc = R_ct.T
164
165         # Paso 8: Calcular ángulos de Euler para el marcador y la cá
                mara
166             # 'rotationMatrixToEulerAngles' convierte la matriz de
                    rotación
167             # R_tc en ángulos de Euler (pitch, roll, yaw).
168
169         pitch_marker, roll_marker, yaw_marker =
                rotationMatrixToEulerAngles(R_tc)
170
171         # Calcula la posición de la cámara desde el marcador en el
                sistema del marcador.
172         pos_camera = -R_tc * np.matrix(tvec).T
173
174         pitch_camera, roll_camera, yaw_camera =
                rotationMatrixToEulerAngles(R_flip * R_tc)
175
176         # Paso 9: Calcular errores basados en la posición del centro
                y la orientación
177         # Calcular el error de Roll
178             # El error de Roll se calcula como la desviación
                    proporcional
179             # del centro del marcador detectado
180             # respecto al centro horizontal de la imagen (320 píxeles
                     es el
181             # centro de una imagen de 640 de ancho).
182
183             # KpRP es un factor proporcional que escala el error para
184             # adecuar la respuesta del sistema de control.
185         errorRoll = gv.KpRP * (centerX - 320) / 320
186
187         # Calcular el error de Pitch
188             # El error de Pitch se calcula de manera similar al error
189             #  de Roll pero en la dirección vertical.
190
191             # 240 píxeles es el centro vertical para una imagen de
                    480 de alto.
192
193             # Este error indica cuánto debe ajustarse la inclinación
                    de la
194             # cámara para centrar el marcador verticalmente.
195         errorPitch = gv.KpRP * (240 - centerY) / 240
196
197         # Calcular el error de Yaw
198             # El error de Yaw se basa en la orientación de la cámara
199             # respecto al marcador, específicamente el ángulo yaw.
200
201             # El ángulo se normaliza dividiendo por pi para obtener
                    un
```

```
202             # valor entre -1 y 1, y se multiplica por -1 para
                    corregir la
203             # dirección de ajuste.
204
205             # KpY es el factor proporcional para el error de Yaw,
                    similar
206             # a KpRP, ajustando la magnitud de la respuesta del
                    control.
207         errorYaw = gv.KpY * - yaw_camera / np.pi
208
209         # Calcular el error de Throttle
210             # El error de Throttle se utiliza para ajustar la altura
                    o la
211             # distancia al marcador.
212
213             # Se calcula como la desviación de la posición Z de la cá
                    mara
214             # respecto a una distancia deseada (distanciaZ).
215
216             # KpT escala el error para controlar la respuesta de
217             # acercamiento o alejamiento del sistema.
218
219         wd = workingDistance(AFOV_hori_deg_webcam,image,corners,
                marker_size)
220         print(f"{est.amarillo}Distancia del marcador a la cámara: {wd
                }{est.reset}")
221
222         errorThrottle = gv.KpT * (distanciaZ - wd) / distanciaZ
223         #print(f"{est.amarillo}errorThrottle: {errorThrottle}{est.
                reset}")
224
225         #errorThrottle = (gv.KpT * (distanciaZ - pos_camera[2]) /
                distanciaZ).item()
226
227     else:
228         # En caso de no detectar el marcador, se establecen los
                errores a cero
229         errorRoll = 0
230         errorPitch = 0
231         errorYaw = 0
232         errorThrottle = 0
233
234         control_id_detected = False
235
236     ####################################
237     # ASIGNACIÓN A VARIABLES GLOBALES #
238     ####################################
239     gv.take_off_id_detected = take_off_id_detected
240     gv.landing_id_detected = landing_id_detected
241     gv.control_id_detected = control_id_detected
242
243     gv.errorRoll = errorRoll
244     gv.errorPitch = errorPitch
245     gv.errorYaw = errorYaw
246     gv.errorThrottle = errorThrottle
```

```
247
248     ####################################
249     # ERRORES VERDADEROS EN TIEMPO REAL #
250     ####################################
251     #print(f"{est.amarillo}Función ImageProcessing: Imagen procesada
            --> Variables Globales actualizadas{est.reset}")
252     #print(f"{est.amarillo}ERoll = {gv.errorRoll:.5f} "
253     #                      f"EPitch = {gv.errorPitch:.5f} "
254     #                      f"EYaw   = {gv.errorYaw:.5f} "
255     #                      f"EThrottle = {gv.errorThrottle:.5f}{est.
            reset}")
256
257     gv.getImage = True
258 # ------------------------------------------------------------------ #
259
260
261 # ------------------------------------------------------------------ #
262 def workingDistance(AFOV_H_deg_arg: float,
263                     image: np.ndarray,
264                     corners: np.ndarray,
265                     marker_size: float) -> float:
266
267     AFOV_H_deg = AFOV_H_deg_arg             # [deg]
268     AFOV_H_rad = math.radians(AFOV_H_deg)   # [rad]
269     height, width, channels = image.shape   # [pix,pix,-]
270     reso_hori_pix = width                   # [pix]
271
272     for corner in corners:
273         eta = abs(corner[0][1][0] - corner[0][0][0])   # [pix]
274         chi = abs(corner[0][0][1] - corner[0][1][1])   # [pix]
275     aruco_pix = math.sqrt(eta**2 + chi**2)             # [pix]
276     aruco_cm  = marker_size                            # [cm]
277
278     FOV_H_cm = reso_hori_pix * (aruco_cm / aruco_pix) # [cm]
279
280     wd = (FOV_H_cm/2) / (math.tan(AFOV_H_rad/2))      # [cm]
281
282     return wd
283 # ------------------------------------------------------------------ #
284
285
286 # ------------------------------------------------------------------ #
287 def FOVdiag2FOVhori(FOV_diag_arg:float) -> float:
288
289     # FoV diagonal en grados
290     fov_diagonal = FOV_diag_arg
291
292     # Relación de aspecto de 720p es 16:9
293     aspect_ratio_width = 16
294     aspect_ratio_height = 9
295
296     # Convertir FoV diagonal a radianes
297     fov_diagonal_rad = math.radians(fov_diagonal)
298
299     # Calcular el FoV horizontal
```

```
300     fov_horizontal_rad = 2 * math.atan(math.tan(fov_diagonal_rad / 2)
            * (aspect_ratio_width / math.sqrt(aspect_ratio_width**2 +
            aspect_ratio_height**2)))
301
302     # Convertir de radianes a grados
303     fov_horizontal_deg = math.degrees(fov_horizontal_rad)
304
305     return fov_horizontal_deg
306 # ------------------------------------------------------------------ #
```

Código O.1: ImageProcessingArUcoSimulador.py.

Apéndice P

Archivo ImageProcessingArUcoVuelo.py

```
# ---------------------- #
# OMAR RABANAL GUTIÉRREZ #
# 16 de Junio del 2024   #
# ---------------------- #


# ------------------------------------------------------------------ #
import numpy as np
import math
import cv2
import cv2.aruco as aruco
from typing import Tuple, Any
import time
import threading
import sys
import random as rd
# ------------------------------------------------------------------ #


# ------------------------------------------------------------------ #
import modulos.GlobalVariables as gv
from modulos.isRotationMatrix import isRotationMatrix
from modulos.rotationMatrixToEulerAngles import
    rotationMatrixToEulerAngles
import modulos.ColoresEstilosFormatos as est
# ------------------------------------------------------------------ #


# ------------------------------------------------------------------ #
def ImageProcessingVuelo(image: np.ndarray,
                         aruco_dict: Any,
                         parameters: cv2.aruco.DetectorParameters,
```

```
32                          camera_matrix: np.ndarray,
33                          camera_distortion: np.ndarray,
34                          R_flip: np.ndarray) -> None:
35
36      # Definición del id de los códigos ArUco a detectar
37      id_take_off = 2
38      id_control  = 10
39      id_landing  = 7
40
41      marker_size = 7.4  #cm
42      distanciaZ  = 30   #cm
43
44      AFOV_diag_deg_webcam = 55                                    #[
            deg]
45      AFOV_hori_deg_webcam = FOVdiag2FOVhori(AFOV_diag_deg_webcam) #[
            deg]
46
47      # Paso 1: Convertir la imagen de entrada a escala de grises
48          # Convertir a escala de grises simplifica el procesamiento de
                la
49          # imagen al reducir la complejidad de la imagen, lo que es
50          # beneficioso para la detección de marcadores ArUco, ya que
                estos
51          # solo necesitan contraste entre blanco y negro.
52      gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
53
54      # Paso 2: Detectar marcadores ArUco
55          # 'corners' son las coordenadas de las esquinas de los
                marcadores detectados.
56          # 'ids' son los identificadores de los marcadores detectados
                que
57          # corresponden a los definidos en 'aruco_dict'.
58          # 'rejectedImgPoints' contiene los candidatos de marcadores
                que
59          # fueron considerados pero no reconocidos como válidos.
60      corners, ids, rejectedImgPoints = cv2.aruco.detectMarkers(gray,
            aruco_dict, parameters=parameters)
61
62      # Paso 3: Verificar si el marcador deseado está presente
63
64      ###################
65      # ArUco Take-off #
66      ###################
67      if len(corners) > 0 and ids[0] == id_take_off:
68
69          if gv.take_off_executed == False:
70              gv.estado_vuelo = "despegue"
71
72          take_off_id_detected = True
73
74          #print("Función ArUco_Control: Código ArUco de despegue
                detectado...")
75
76          for corner in corners:
```

```
            centerX = (corner[0][0][0] + corner[0][1][0] + corner
                [0][2][0] + corner[0][3][0]) / 4
            centerY = (corner[0][0][1] + corner[0][1][1] + corner
                [0][2][1] + corner[0][3][1]) / 4

        ret = cv2.aruco.estimatePoseSingleMarkers(corners,
            marker_size, camera_matrix, camera_distortion)
        rvec, tvec = ret[0][0,0,:], ret[1][0,0,:]

        cv2.aruco.drawDetectedMarkers(image, corners)
        cv2.drawFrameAxes(image, camera_matrix, camera_distortion,
            rvec, tvec, 5)

    else:
        take_off_id_detected = False

    ##################
    # ArUco Control #
    ##################
    if len(corners) > 0 and (ids[0] == id_control or ids[0] ==
        id_take_off or ids[0] == id_landing):

        if gv.posicionamiento_terminado == False and gv.
            take_off_executed == True:
            gv.estado_vuelo = "posicionamiento"

        if ids[0] == id_control:
            control_id_detected = True
        else:
            control_id_detected = False

        # Calcular el centro del marcador detectado
            # Las 'corners' son arrays con las coordenadas de las
                esquinas de
            # cada marcador detectado en la imagen.

            # centerX y centerY calculan el promedio de las esquinas
                para
            # encontrar el centro del marcador en la imagen.

            # Esto se hace en el sistema de coordenadas de la imagen,
            # donde x es horizontal y y es vertical desde la esquina
            # superior izquierda.
        for corner in corners:
            centerX = (corner[0][0][0] + corner[0][1][0] + corner
                [0][2][0] + corner[0][3][0]) / 4
            centerY = (corner[0][0][1] + corner[0][1][1] + corner
                [0][2][1] + corner[0][3][1]) / 4

        for corner in corners:
            tam_aruco_pix = abs(corner[0][1][0] - corner[0][0][0])
                  # [pix]
        #print(f"{est.magenta}ArUco en Píxeles: {tam_aruco_pix}{est.
            reset}")

```

```
120         # Paso 4: Estimar la pose del marcador
121             # 'estimatePoseSingleMarkers' calcula la posición y
                    orientación
122             # del marcador en el espacio tridimensional respecto a la
                    cámara.
123
124             # rvec es el vector de rotación del marcador medido en el
                    sistema de
125             # coordenadas de la cámara.
126
127             # tvec es el vector de traslación que indica la posición
                    del marcador
128             # en el espacio, medido en el sistema de coordenadas de
                    la cámara.
129
130         ret = cv2.aruco.estimatePoseSingleMarkers(corners,
                marker_size, camera_matrix, camera_distortion)
131         rvec, tvec = ret[0][0,0,:], ret[1][0,0,:]
132
133         # Paso 5: Visualizar marcadores y ejes
134             # 'drawDetectedMarkers' dibuja los marcadores detectados
135             # en la imagen.
136
137             # 'drawFrameAxes' añade un sistema de coordenadas visual
                    en el
138             # marcador para ilustrar la orientación del marcador.
139
140         cv2.aruco.drawDetectedMarkers(image, corners)
141         cv2.drawFrameAxes(image, camera_matrix, camera_distortion,
                rvec, tvec, 5)
142
143         # Paso 6 y 7: Calcular la matriz de rotación y la posición
144             # R_ct transforma coordenadas desde el sistema del
                    marcador
145             # al sistema de la cámara.
146
147             # R_tc es la transpuesta de R_ct, transformando
                    coordenadas
148             # desde el sistema de la cámara al del marcador.
149
150         R_ct = np.matrix(cv2.Rodrigues(rvec)[0])
151         R_tc = R_ct.T
152
153         # Paso 8: Calcular ángulos de Euler para el marcador y la cá
                mara
154             # 'rotationMatrixToEulerAngles' convierte la matriz de
                    rotación
155             # R_tc en ángulos de Euler (pitch, roll, yaw).
156
157         pitch_marker, roll_marker, yaw_marker =
                rotationMatrixToEulerAngles(R_tc)
158
159         # Calcula la posición de la cámara desde el marcador en el
                sistema del marcador.
160         pos_camera = -R_tc * np.matrix(tvec).T
```

```
161
162         pitch_camera, roll_camera, yaw_camera =
                rotationMatrixToEulerAngles(R_flip * R_tc)
163
164         # Paso 9: Calcular errores basados en la posición del centro
                y la orientación
165         # Calcular el error de Roll
166             # El error de Roll se calcula como la desviación
                    proporcional
167             # del centro del marcador detectado
168             # respecto al centro horizontal de la imagen (320 píxeles
                    es el
169             # centro de una imagen de 640 de ancho).
170
171             # KpRP es un factor proporcional que escala el error para
172             # adecuar la respuesta del sistema de control.
173         errorRoll = gv.KpRP * (centerX - 320) / 320
174
175         # Calcular el error de Pitch
176             # El error de Pitch se calcula de manera similar al error
177             #  de Roll pero en la dirección vertical.
178
179             # 240 píxeles es el centro vertical para una imagen de
                    480 de alto.
180
181             # Este error indica cuánto debe ajustarse la inclinación
                    de la
182             # cámara para centrar el marcador verticalmente.
183         errorPitch = gv.KpRP * (240 - centerY) / 240
184
185         # Calcular el error de Yaw
186             # El error de Yaw se basa en la orientación de la cámara
187             # respecto al marcador, específicamente el ángulo yaw.
188
189             # El ángulo se normaliza dividiendo por pi para obtener
                    un
190             # valor entre -1 y 1, y se multiplica por -1 para
                    corregir la
191             # dirección de ajuste.
192
193             # KpY es el factor proporcional para el error de Yaw,
                    similar
194             # a KpRP, ajustando la magnitud de la respuesta del
                    control.
195         errorYaw = gv.KpY * - yaw_camera / np.pi
196
197         # Calcular el error de Throttle
198             # El error de Throttle se utiliza para ajustar la altura
                    o la
199             # distancia al marcador.
200
201             # Se calcula como la desviación de la posición Z de la cá
                    mara
202             # respecto a una distancia deseada (distanciaZ).
203
```

```
204             # KpT escala el error para controlar la respuesta de
205             # acercamiento o alejamiento del sistema.
206 
207         wd = workingDistance(AFOV_hori_deg_webcam,image,corners,
                marker_size)
208         #print(f"{est.amarillo}Distancia del marcador a la cámara: {
                wd}{est.reset}")
209 
210         errorThrottle = gv.KpT * (distanciaZ - wd) / distanciaZ
211         #print(f"{est.amarillo}errorThrottle: {errorThrottle}{est.
                reset}")
212 
213         #errorThrottle = (gv.KpT * (distanciaZ - pos_camera[2]) /
                distanciaZ).item()
214 
215     else:
216         # En caso de no detectar el marcador, se establecen los
                errores a cero
217         errorRoll     = 0
218         errorPitch    = 0
219         errorYaw      = 0
220         errorThrottle = 0
221 
222         control_id_detected = False
223 
224     ##################
225     # ArUco Landing #
226     ##################
227     if len(corners) > 0 and ids[0] == id_landing:
228 
229         #if gv.posicionamiento_terminado == True:
230             #if gv.landing_executed == False:
231                 #gv.estado_vuelo = "aterrizaje"
232 
233         landing_id_detected = True
234 
235         #print("Código ArUco de aterrizaje detectado, procediendo al
                aterrizaje de la aeronave...")
236 
237         for corner in corners:
238             centerX = (corner[0][0][0] + corner[0][1][0] + corner
                    [0][2][0] + corner[0][3][0]) / 4
239             centerY = (corner[0][0][1] + corner[0][1][1] + corner
                    [0][2][1] + corner[0][3][1]) / 4
240 
241         ret = cv2.aruco.estimatePoseSingleMarkers(corners,
                marker_size, camera_matrix, camera_distortion)
242         rvec, tvec = ret[0][0,0,:], ret[1][0,0,:]
243 
244         cv2.aruco.drawDetectedMarkers(image, corners)
245         cv2.drawFrameAxes(image, camera_matrix, camera_distortion,
                rvec, tvec, 5)
246 
247     else:
248         landing_id_detected = False
```

```
249
250     ###################################
251     # ASIGNACIÓN A VARIABLES GLOBALES #
252     ###################################
253     gv.take_off_id_detected = take_off_id_detected
254     gv.landing_id_detected = landing_id_detected
255     gv.control_id_detected = control_id_detected
256
257     gv.errorRoll = errorRoll
258     gv.errorPitch = errorPitch
259     gv.errorYaw = errorYaw
260     gv.errorThrottle = errorThrottle
261
262     ###################################
263     # ERRORES VERDADEROS EN TIEMPO REAL #
264     ###################################
265     #print(f"{est.amarillo}Función ImageProcessing: Imagen procesada
            --> Variables Globales actualizadas{est.reset}")
266     #print(f"{est.amarillo}ERoll = {gv.errorRoll:.5f} "
267     #                    f"EPitch = {gv.errorPitch:.5f} "
268     #                    f"EYaw   = {gv.errorYaw:.5f} "
269     #                    f"EThrottle = {gv.errorThrottle:.5f}{est.
            reset}")
270
271     gv.getImage = True
272 # ------------------------------------------------------------------ #
273
274
275 # ------------------------------------------------------------------ #
276 def workingDistance(AFOV_H_deg_arg: float,
277                     image: np.ndarray,
278                     corners: np.ndarray,
279                     marker_size: float) -> float:
280
281     AFOV_H_deg = AFOV_H_deg_arg             # [deg]
282     AFOV_H_rad = math.radians(AFOV_H_deg)   # [rad]
283     height, width, channels = image.shape   # [pix,pix,-]
284     reso_hori_pix = width                   # [pix]
285
286     for corner in corners:
287         eta = abs(corner[0][1][0] - corner[0][0][0])  # [pix]
288         chi = abs(corner[0][0][1] - corner[0][1][1])  # [pix]
289     aruco_pix = math.sqrt(eta**2 + chi**2)            # [pix]
290     aruco_cm  = marker_size                           # [cm]
291
292     FOV_H_cm = reso_hori_pix * (aruco_cm / aruco_pix) # [cm]
293
294     wd = (FOV_H_cm/2) / (math.tan(AFOV_H_rad/2))      # [cm]
295
296     return wd
297 # ------------------------------------------------------------------ #
298
299
300 # ------------------------------------------------------------------ #
301 def FOVdiag2FOVhori(FOV_diag_arg:float) -> float:
```

```
302
303     # FoV diagonal en grados
304     fov_diagonal = FOV_diag_arg
305
306     # Relación de aspecto de 720p es 16:9
307     aspect_ratio_width = 16
308     aspect_ratio_height = 9
309
310     # Convertir FoV diagonal a radianes
311     fov_diagonal_rad = math.radians(fov_diagonal)
312
313     # Calcular el FoV horizontal
314     fov_horizontal_rad = 2 * math.atan(math.tan(fov_diagonal_rad / 2)
            * (aspect_ratio_width / math.sqrt(aspect_ratio_width**2 +
           aspect_ratio_height**2)))
315
316     # Convertir de radianes a grados
317     fov_horizontal_deg = math.degrees(fov_horizontal_rad)
318
319     return fov_horizontal_deg
320 # ------------------------------------------------------------------ #
```

Código P.1: ImageProcessingArUcoVuelo.py.

Apéndice Q

Archivo RasPi_TakePhotos.py

```
# ---------------------- #
# OMAR RABANAL GUTIÉRREZ #
# 05 de Julio del 2024   #
# ---------------------- #


# ------------------------------------------------------------------ #
import numpy as np
import math
import cv2
from typing import Tuple, Any
import time
import threading
import sys
import subprocess
from datetime import datetime
import os
# ------------------------------------------------------------------ #


# ------------------------------------------------------------------ #
import modulos.ColoresEstilosFormatos as est
# ------------------------------------------------------------------ #


# ------------------------------------------------------------------ #
def raspi_gstreamer_pipeline(sensor_id = 0,
        capture_width = 640, capture_height = 480,
        display_width = 640, display_height = 480,
        framerate = 29,
        flip_method = 0,
        exposure_time = None, exposure_time_min = 13000,
            exposure_time_max = 683709000,
        gain = None, gain_min = 1.0, gain_max = 10.625,
        saturation = 0.55,
```

```
35          contrast = 1.0,
36          brightness = 0.0,
37          hue = 0.0,
38          wbmode = 1,
39          exposurecompensation = 0,
40          sharpness = 1.0,
41          noise_reduction = 1.0):
42
43      """
44      Crea un pipeline de GStreamer para capturar video desde una cá
            mara CSI.
45
46      -----------
47      Parámetros:
48      -----------
49          - sensor_id (int): El ID del sensor de la cámara a utilizar.
50            Predeterminado es 0.
51            Rango: Típicamente 0 hasta el número de sensores conectados
                  - 1.
52
53          - capture_width (int): El ancho de la resolución de captura
                de video.
54            Predeterminado es 1920.
55            Rango: Depende de las capacidades de la cámara.
56
57          - capture_height (int): La altura de la resolución de captura
                 de video.
58            Predeterminado es 1080.
59            Rango: Depende de las capacidades de la cámara.
60
61          - display_width (int): El ancho de la resolución de
                visualización de video.
62            Predeterminado es 640.
63            Rango: Cualquier número entero positivo.
64
65          - display_height (int): La altura de la resolución de
                visualización de video.
66            Predeterminado es 480.
67            Rango: Cualquier número entero positivo.
68
69          - framerate (int): La tasa de cuadros por segundo del video.
70            Predeterminado es 29.
71            Rango: Depende de las capacidades de la cámara y la
                  resolución.
72
73          - flip_method (int): El método para voltear la imagen.
74            Predeterminado es 0.
75              Valores:
76                  - 0: Sin rotación
77                  - 1: Rotar 90 grados en sentido antihorario
78                  - 2: Rotar 180 grados
79                  - 3: Rotar 90 grados en sentido horario
80                  - 4: Volteo horizontal
81                  - 5: Volteo diagonal superior derecha
82                  - 6: Volteo vertical
```

```
                - 7: Volteo diagonal superior izquierda

        - exposure_time (int, opcional): El tiempo de exposición
          específico en nanosegundos.
          Predeterminado es None.
          Si se especifica, sobrescribe exposure_time_min y
              exposure_time_max.

        - exposure_time_min (int): El tiempo mínimo de exposición en
          nanosegundos.
          Predeterminado es 13000.
          Rango: Dependiente de la cámara.

        - exposure_time_max (int): El tiempo máximo de exposición en
          nanosegundos.
          Predeterminado es 683709000.
          Rango: Dependiente de la cámara.

        - gain (float, opcional): El valor específico de ganancia.
          Predeterminado es None.
          Si se especifica, sobrescribe gain_min y gain_max.

        - gain_min (float): El valor mínimo de ganancia.
          Predeterminado es 1.0.
          Rango: Dependiente de la cámara.

        - gain_max (float): El valor máximo de ganancia.
          Predeterminado es 10.625.
          Rango: Dependiente de la cámara.

        - saturation (float): Ajusta la saturación del video.
          Predeterminado es 1.0.
          Rango: 0.0 (escala de grises) a valores mayores de 1.0 (más
              saturado).

        - contrast (float): Ajusta el contraste del video.
          Predeterminado es 1.0.
          Rango: 0.0 a 2.0.

        - brightness (float): Ajusta el brillo del video.
          Predeterminado es 0.0.
          Rango: -1.0 a 1.0.

        - hue (float): Ajusta el tono del video.
          Predeterminado es 0.0.
          Rango: -1.0 a 1.0.

        - wbmode (int): Ajusta el modo de balance de blancos.
          Predeterminado es 1.
          Valores:
              - 0: Off
              - 1: Auto
              - 2: Incandescent
              - 3: Fluorescent
              - 4: Daylight
```

```
133                 - 5: Cloudy
134 
135         - exposurecompensation (int): Compensación de exposición.
136           Predeterminado es 0.
137           Rango: Dependiente de la cámara.
138 
139         - sharpness (float): Ajusta la nitidez del video.
140           Predeterminado es 1.0.
141           Rango: 0.0 a 2.0.
142 
143         - noise_reduction (float): Reducción de ruido.
144           Predeterminado es 1.0.
145           Rango: 0.0 a 1.0.
146 
147     Retorna:
148     --------
149     - str: Una cadena de texto del pipeline de GStreamer.
150     """
151 
152     if exposure_time == None:
153         exposure_param = f"exposuretimerange='{exposure_time_min} {
                exposure_time_max}'"
154     else:
155         exposure_param = f"exposuretimerange='{exposure_time} {
                exposure_time}'"
156 
157     if gain == None:
158         gain_param = f"gainrange='{gain_min} {gain_max}'"
159     else:
160         gain_param = f"gainrange='{gain} {gain}'"
161 
162     pipeline = (
163         f"nvarguscamerasrc sensor-id={sensor_id} {exposure_param} {
                gain_param} wbmode={wbmode} exposurecompensation={
                exposurecompensation} ! "
164         f"video/x-raw(memory:NVMM), width=(int){capture_width},
                height=(int){capture_height}, framerate=(fraction){
                framerate}/1 ! "
165         f"nvvidconv flip-method={flip_method} ! "
166         f"video/x-raw, width=(int){display_width}, height=(int){
                display_height} ! "
167         f"videobalance saturation={saturation} contrast={contrast}
                brightness={brightness} hue={hue} sharpness={sharpness *
                (1 - noise_reduction)} ! "
168         f"videoconvert ! videoscale ! "
169         f"video/x-raw, format=(string)BGR ! appsink"
170     )
171 
172     print(" ")
173     print(f"{est.cyan}Pipeline:{est.reset}\n{pipeline}")
174     print(" ")
175     return pipeline
176 # ------------------------------------------------------------------- #
177 
178 
```

```
179
180
181 # ------------------------------------------------------------------ #
182 def caso_uno():
183     global window_title, frame, tiempo_captura_fotos,
        tiempo_espera_fotos, tiempo_ultima_foto, finalizar_programa,
        no_safe, safe
184
185     tiempo_espera = time.time()
186     subprocess.run(['clear'])
187     print(f"{est.cyan}Pulse:\n [g] para guardar\n [c] para continuar\
        n [q] para salir{est.reset}")
188
189     while True:
190         tiempo_espera_guardar = time.time()
191
192         if cv2.getWindowProperty(window_title, cv2.WND_PROP_AUTOSIZE)
                >= 0:
193             cv2.imshow(window_title, frame)
194         else:
195             break
196
197         keyCode = cv2.waitKey(1) & 0xFF
198         if keyCode == ord('g'):
199             no_safe = False
200             safe = True
201             ruta_fotos = "/home/omar/Escritorio/Fotos/"
202             if os.path.exists(ruta_fotos) == False:
203                 print(" ")
204                 print(f"{est.rojo}La ruta de almacenamiento de imá
                    genes introducida no es válida... Finalizando{est
                    .reset}")
205                 sys.exit()
206             now = datetime.now()
207             current_time = now.strftime("%Y-%m-%d_%H:%M:%S")
208             nombre_foto = ruta_fotos + "Captura_" + current_time + ".
                    png"
209             cv2.imwrite(nombre_foto,frame)
210             break
211
212         if keyCode == ord('c'):
213             no_safe = True
214             safe = False
215             break
216
217         if keyCode == ord('q'):
218             finalizar_programa = True
219             no_safe = True
220             safe = False
221             break
222
223         if tiempo_espera_guardar - tiempo_espera >=
                tiempo_espera_fotos:
224             break
225
```

```
226     tiempo_ultima_foto = time.time()
227     subprocess.run(['clear'])
228 
229     if no_safe == True and finalizar_programa == False:
230         print(f"{est.amarillo}Captura no guardada continuando el
                programa...{est.reset}")
231         print(f"{est.cyan}Espere {tiempo_captura_fotos} segundos para
                 poder guardar una nueva imagen{est.reset}")
232     if safe == True:
233         nombre_foto_sin_ruta = "Captura_" + current_time + ".png"
234         print(f"{est.verde}La foto {nombre_foto_sin_ruta} se ha
                guardado correctamente en {ruta_fotos}{est.reset}")
235         print(f"{est.cyan}Espere {tiempo_captura_fotos} segundos para
                 poder guardar una nueva imagen{est.reset}")
236 # ------------------------------------------------------------------ #
237 
238 
239 
240 
241 # ------------------------------------------------------------------ #
242 def caso_dos():
243     global window_title, frame, tiempo_captura_fotos,
            tiempo_espera_fotos, tiempo_ultima_foto, finalizar_programa,
            no_safe, safe
244 
245     tiempo_espera = time.time()
246     subprocess.run(['clear'])
247     print(f"{est.cyan}Se procede a guardar el frame capturado \nEn 5
            segundos el programa se reanudará \nSi desea salir del
            programa pulse [q]{est.reset}")
248 
249     while True:
250         tiempo_espera_guardar = time.time()
251 
252         if cv2.getWindowProperty(window_title, cv2.WND_PROP_AUTOSIZE)
                >= 0:
253             cv2.imshow(window_title, frame)
254         else:
255             break
256 
257         keyCode = cv2.waitKey(1) & 0xFF
258         if keyCode == ord('q'):
259             finalizar_programa = True
260             break
261 
262         if tiempo_espera_guardar - tiempo_espera >= 5:
263             break
264 
265     no_safe = False
266     safe = True
267     ruta_fotos = "/home/omar/Escritorio/Fotos/"
268     if os.path.exists(ruta_fotos) == False:
269         print(" ")
270         print(f"{est.rojo}La ruta de almacenamiento de imágenes
                introducida no es válida... Finalizando{est.reset}")
```

```
271             sys.exit()
272         now = datetime.now()
273         current_time = now.strftime("%Y-%m-%d_%H:%M:%S")
274         nombre_foto = ruta_fotos + "Captura_" + current_time + ".png"
275         cv2.imwrite(nombre_foto,frame)
276 
277         tiempo_ultima_foto = time.time()
278         subprocess.run(['clear'])
279 
280         if safe == True:
281             nombre_foto_sin_ruta = "Captura_" + current_time + ".png"
282             print(f"{est.verde}La foto {nombre_foto_sin_ruta} se ha
                    guardado correctamente en {ruta_fotos}{est.reset}")
283             print(f"{est.cyan}Espere {tiempo_captura_fotos} segundos para
                     poder guardar una nueva imagen{est.reset}")
284 # ------------------------------------------------------------------- #
285 
286 
287 
288 
289 # ------------------------------------------------------------------- #
290 def switch(clave_caso:int) -> None:
291     global dicc_casos
292     ref_func_caso = dicc_casos.get(clave_caso)
293     ref_func_caso()
294 # ------------------------------------------------------------------- #
295 
296 
297 
298 
299 # ------------------------------------------------------------------- #
300 def webcam_TakePhotos(caso:str = "caso1") -> None:
301 
302     global dicc_casos, window_title, frame, tiempo_captura_fotos,
            tiempo_espera_fotos, tiempo_ultima_foto, finalizar_programa,
            no_safe, safe
303 
304     dicc_casos = {"caso1": caso_uno,
305                   "caso2": caso_dos
306                   }
307 
308     window_title = "CSI Camera"
309 
310     cam = cv2.VideoCapture(raspi_gstreamer_pipeline(sensor_id = 0,
311                 capture_width = 640, capture_height = 480,
312                 display_width = 640, display_height = 480,
313                 framerate = 29,
314                 flip_method = 0,
315                 exposure_time = 683709000/2, exposure_time_min =
                        13000, exposure_time_max = 683709000,
316                 gain = 1, gain_min = 1.0, gain_max = 10.625,
317                 saturation = 0.6,
318                 contrast = 1.0,
319                 brightness = 0.0,
320                 hue = 0,
```

```
321                 wbmode = 1,
322                 exposurecompensation = 1,
323                 sharpness = 2,
324                 noise_reduction = 0.5),
325                 cv2.CAP_GSTREAMER)
326
327     if cam.isOpened() == True:
328
329         try:
330
331             # Crea una ventana con propiedades específicas
332             window_handle = cv2.namedWindow(window_title, cv2.
                    WINDOW_AUTOSIZE)
333
334             # Inicialización de la variable tiempo_ultima_foto
335             tiempo_ultima_foto = time.time()
336
337             while True:
338
339                 finalizar_programa = False
340                 no_safe = False
341                 safe = False
342
343                 tiempo_actual = time.time()
344
345                 ret, frame = cam.read()
346
347                 if ret == False:
348                     print(f"{est.rojo}No se pudo capturar el
                            siguiente frame. Terminando...{est.reset}")
349                     break
350
351                 if tiempo_actual - tiempo_ultima_foto >=
                        tiempo_captura_fotos:
352                     switch(caso)
353                     if finalizar_programa == False:
354                         print(f"{est.cyan}Espere {
                                tiempo_captura_fotos} segundos para poder
                                 guardar una nueva imagen{est.reset}")
355
356                 if finalizar_programa == True:
357                     break
358
359                 if no_safe == True:
360                     continue
361
362                 if safe == True:
363                     continue
364
365                 # Verifica si la ventana aún está abierta
366                 if cv2.getWindowProperty(window_title, cv2.
                        WND_PROP_AUTOSIZE) >= 0:
367                     cv2.imshow(window_title, frame)
368                 else:
369                     break
```

```
                keyCode = cv2.waitKey(1) & 0xFF
                if keyCode == 27 or keyCode == ord('q'):
                    break

        finally:
            print(" ")
            print(f"{est.cyan}Finalizando el programa{est.reset}\n")
            cam.release()
            cv2.destroyAllWindows()

    else:
        print(" ")
        print(f"{est.rojo}Error: Unable to open camera{est.reset}\n")
# ------------------------------------------------------------------- #




# ------------------------------------------------------------------- #
if __name__ == "__main__":

    global tiempo_captura_fotos, tiempo_espera_fotos

    tiempo_captura_fotos = 5
    tiempo_espera_fotos = 3

    print(" ")
    caso = input(f"{est.cyan}Introduzca el modo de captura de imá
        genes: \n  Modo 1 --> Captura continua (cada 10 sec) con opci
        ón de guardado \n  Modo 2 --> Captura continua (cada 10 sec)
        con guardado automático \n{est.reset}")

    webcam_TakePhotos(caso)
# ------------------------------------------------------------------- #
```

Código Q.1: Archivo RasPi_TakePhotos.py.

Bibliografía

[1] Fortune Business Insights. *Commercial Drone Market Size, Share | Global Forecast, 2030*. 2024. URL: https://www.fortunebusinessinsights.com/commercial-drone-market-102171 (visitado 15-07-2024) (vid. pág. 1).

[2] Jenny Beechener. *Drone Industry Insights report predicts 7,7% CAGR drone market growth to reach USD54,6 million by 2030*. 2023. URL: https://www.unmannedairspace.info/uncategorized/drone-industry-insights-market-report-predicts-7-7-cagr-to-reach-usd54-6m-by-2030/ (visitado 15-07-2024) (vid. págs. 1, 2).

[3] Easy Drones. *Empresa de Drones en Agricultura*. 2024. URL: https://easydrones.es/agricultura/ (visitado 15-07-2024) (vid. pág. 4).

[4] Aerones. *Aerones lanza el servicio de inspección de palas de aerogeneradores con drones autónomos*. 2024. URL: https://aerones.com/es/aerones-lanza-el-servicio-de-inspeccion-de-palas-de-aerogeneradores-con-drones-autonomos/ (visitado 15-07-2024) (vid. pág. 4).

[5] Correos. *Correos exhibe los drones desarrollados en el marco del proyecto Delorean*. 2024. URL: https://www.correos.com/sala-prensa/correos-exhibe-los-drones-desarrollados-en-el-marco-del-proyecto-delorean/# (visitado 15-07-2024) (vid. pág. 4).

[6] UMILES Group. *Drones para Cine ¡La revolución cinematográfica!* 2022. URL: https://umilesgroup.com/drones-para-cine/ (visitado 15-07-2024) (vid. pág. 4).

[7] Teller Report. *DJI's journey to dominate the global drone market*. 2023. URL: https://www.tellerreport.com/news/2023-04-30-dji-s-journey-to-dominate-the-global-drone-market.BkmqL_M2Xh.html (visitado 15-07-2024) (vid. pág. 3).

[8] Bestployov. *Product Details*. 2024. URL: https://bestployov.best/product_details/55415205.html (visitado 15-07-2024) (vid. pág. 5).

[9] DJI. *DJI Developer Technologies*. 2024. URL: https://developer.dji.com/ (visitado 15-07-2024) (vid. pág. 5).

[10] Leya San José Moralejo. *Implementación de una aplicación Android para el control del dron DJI Mavic Mini*. Trabajo Fin de Máster. Curso académico 2020/2021, Tutor: Ángel Rodas Jordá. Valencia, 2021 (vid. págs. 15, 16, 19, 66-68).

[11] Juan Martos Bianqui. *Control automático con Simulink de un mini dron. Aplicación al seguimiento de trayectorias mediante realimentación visual*. Trabajo Fin de Máster. Tutor: Ángel Rodas Jordá. Valencia, 2020 (vid. pág. 18).

[12] Javier Luque Benítez. *Diseño e implementación de un UAS compacto para trabajos indoor*. Trabajo Fin de Máster. Tutor: Ángel Rodas Jordá, Director: Pablo Antonio Morcillo Pallarés. Valencia, 2020 (vid. pág. 18).

[13] Jorge Martín Pariente. *Automatización del dron Parrot Bebop 2 mediante procesamiento de imágenes por ordenador*. Trabajo Fin de Máster. Tutor: Ángel Rodas Jordá. Valencia, 2020 (vid. pág. 18).

[14] Miguel Marco Estevens. *Diseño y configuración de un sistema aéreo no tripulado con capacidad de procesamiento de imagen*. Trabajo Fin de Grado. Tutor: Ángel Rodas Jordá. Valencia, 2018 (vid. pág. 18).

[15] Vicent Barrera Gaspar. *Diseño y construcción de un UAS mediante Raspberry Pi y código abierto*. Trabajo Fin de Grado. Tutor: Ángel Rodas Jordá, Director: Pablo Antonio Morcillo Pallarés. Valencia, 2017 (vid. pág. 18).

[16] ResearchGate. *General architecture of an unmanned aerial system*. 2024. URL: https://www.researchgate.net/figure/General-architecture-of-an-unmanned-aerial-system_fig2_335397900 (visitado 15-07-2024) (vid. pág. 25).

[17] Agencia Estatal de Seguridad Aérea (AESA). *Curso de Formación A1/A3 Completo*. 2024. URL: https://www.seguridadaerea.gob.es/sites/default/files/Curso.Formacion.A1.A3.Completo.v6.pdf (visitado 15-07-2024) (vid. págs. 25, 30).

[18] Beyond Vision. *VTOne: AI-Powered VTOL Drone*. 2024. URL: https://beyond-vision.com/vtone-ai-powered-vtol-drone/ (visitado 15-07-2024) (vid. pág. 27).

[19] Richard DePaso. *Learning to Fly a Drone*. 2014. URL: https://aardvarkvideolasvegas.com/learning-to-fly-a-drone/ (visitado 15-07-2024) (vid. pág. 27).

[20] Motor Authority. *China's Ehang unveils autonomous passenger drone*. 2023. URL: https://www.motorauthority.com/news/1101747_china-s-ehang-unveils-autonomous-passenger-drone (visitado 15-07-2024) (vid. pág. 28).

[21] SAlex24. *Optical Flow oscillating position*. 2023. URL: https://discuss.px4.io/t/optical-flow-oscillating-position/34470/7 (visitado 15-07-2024) (vid. pág. 28).

[22] EASA. *European Union Aviation Safety Agency*. 2024. URL: https://www.easa.europa.eu/en (visitado 15-07-2024) (vid. pág. 28).

[23] Agencia Estatal de Seguridad Aérea (AESA). *Agencia Estatal de Seguridad Aérea*. 2024. URL: https://www.seguridadaerea.gob.es/ (visitado 15-07-2024) (vid. pág. 29).

[24] Movilidad y Agenda Urbana Ministerio de Transportes. *Dirección General de Aviación Civil - Secretaría General de Transporte y Movilidad*. 2024. URL: https://www.transportes.gob.es/aviacion-civil/organizacion-y-funciones/secretaria-general-de-transporte-y-movilidad/direccion-general-aviacion-civil (visitado 15-07-2024) (vid. pág. 29).

[25] Boletín Oficial del Estado (BOE). *Reglamento de Ejecución (UE) 2019/947 de la Comisión de 24 de mayo de 2019*. 2019. URL: https://www.boe.es/doue/2019/152/L00045-00071.pdf (visitado 15-07-2024) (vid. pág. 29).

[26] Boletín Oficial del Estado (BOE). *Reglamento Delegado (UE) 2019/945 de la Comisión de 12 de marzo de 2019*. 2019. URL: https://www.boe.es/doue/2019/152/L00001-00040.pdf (visitado 15-07-2024) (vid. pág. 30).

[27] Boletín Oficial del Estado (BOE). *Real Decreto 1036/2017, de 15 de diciembre*. 2017. URL: https://www.boe.es/boe/dias/2017/12/29/pdfs/BOE-A-2017-15721.pdf (visitado 15-07-2024) (vid. pág. 30).

[28] Boletín Oficial del Estado (BOE). *Real Decreto 517/2024, de 4 de junio*. 2024. URL: https://www.boe.es/boe/dias/2024/06/05/pdfs/BOE-A-2024-11377.pdf (visitado 15-07-2024) (vid. pág. 30).

[29] RosettaDrone. *RosettaDrone GitHub Repository*. 2024. URL: https://github.com/RosettaDrone/rosettadrone (visitado 15-07-2024) (vid. págs. 31, 33).

[30] ArduPilot. *MAVLink Message Flow*. 2024. URL: https://ardupilot.org/dev/_images/mavlink-message-flow.png (visitado 19-06-2024) (vid. pág. 32).

[31] QGroundControl Documentation. *QGroundControl Quick Start*. 2024. URL: https://docs.qgroundcontrol.com/Stable_V4.3/en/qgc-user-guide/getting_started/quick_start.html (visitado 17-05-2024) (vid. pág. 32).

[32] Mark D. Jacobsen. *Rosetta Drone*. 2024. URL: https://markdjacobsen.com/portfolio/rosetta-drone/ (visitado 15-07-2024) (vid. pág. 32).

[33] RosettaDrone. *RosettaDrone Wiki*. 2024. URL: https://github-wiki-see.page/m/RosettaDrone/rosettadrone/wiki (visitado 15-07-2024) (vid. pág. 33).

[34] PaoloOne. *Rosetta Drone + QGroundControl = ArduPilot on Mavic*. 2019. URL: https://mavicpilots.com/threads/rosetta-drone-qgroundcontrol-ardupilot-on-mavic.57361/ (visitado 15-07-2024) (vid. pág. 33).

[35] DJI. *DJI Phantom 4 Advanced/Advanced+ User Manual*. 2017. URL: https://dl.djicdn.com/downloads/Phantom_4_Advanced/20171010/Phantom_4_Adv_and_Adv_Plus_User_Manual_EN.pdf (visitado 18-07-2024) (vid. págs. 34, 56, 122).

[36] Sophie111. *Third party app on Phantom 4 Pro+*. 2018. URL: https://forum.dji.com/forum.php?mod=viewthread&tid=162235 (visitado 06-06-2024) (vid. pág. 34).

[37] Clearpath Robotics. *Intro to the Robot Operating System*. 2024. URL: https://www.clearpathrobotics.com/assets/guides/melodic/ros/Intro%20to%20the%20Robot%20Operating%20System.html (visitado 15-07-2024) (vid. pág. 36).

[38] Oleg Kalachev. *Online ArUco Markers Generator*. 2024. URL: https://chev.me/arucogen/ (visitado 15-07-2024) (vid. pág. 38).

[39] Universidad de Córdoba. *ArUco - Aplicaciones de la Visión Artificial*. 2024. URL: https://www.uco.es/investiga/grupos/ava/portfolio/aruco/ (visitado 15-07-2024) (vid. págs. 39, 40).

[40] Sergio Garrido y Alexander Panov. *Detection of ArUco Markers*. 2024. URL: https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html (visitado 15-07-2024) (vid. pág. 40).

[41] Tentone. *ArUco*. 2024. URL: https://github.com/tentone/aruco (visitado 15-07-2024) (vid. pág. 40).

[42] The Drone Dojo. *Precision Landing a Drone on an Aruco Array*. 2019. URL: https://www.youtube.com/watch?v=tJMMEgprBn4 (visitado 23-04-2024) (vid. pág. 40).

[43] Newegg Insider. *Nvidia Jetson Nano: What is it, and what can it do?* 2020. URL: https://www.newegg.com/insider/nvidia-jetson-nano-dev-kit-makers-overview-what-can-it-do/ (visitado 15-07-2024) (vid. pág. 41).

[44] NVIDIA Developer. *Jetson Modules, Support, Ecosystem, and Lineup*. 2024. URL: https://developer.nvidia.com/embedded/jetson-modules (visitado 15-07-2024) (vid. pág. 41).

[45] NVIDIA Developer Blog. *Jetson Nano: AI Computing*. 2019. URL: https://developer.nvidia.com/blog/jetson-nano-ai-computing/ (visitado 15-07-2024) (vid. pág. 41).

[46] Viso.ai. *NVIDIA Jetson: Edge AI Platform*. 2024. URL: https://viso.ai/edge-ai/nvidia-jetson/ (visitado 15-07-2024) (vid. pág. 41).

[47] Seeed Studio. *reComputer J3011-Edge AI Device with Jetson Orin™ Nano 8GB module*. 2024. URL: https://www.seeedstudio.com/reComputer-J3011-p-5590.html (visitado 11-04-2024) (vid. págs. 48-50).

[48] Seeed Studio. *Getting Started with reComputer Industrial*. 2023. URL: https://wiki.seeedstudio.com/reComputer_Industrial_Getting_Started/ (visitado 19-06-2024) (vid. pág. 50).

[49] Alex Alderson. *DJI revela el nuevo dron Mini 4K como probable alternativa más barata al Mini 4 Pro*. 2024. URL: https://www.notebookcheck.org/DJI-revela-el-nuevo-dron-Mini-4K-como-probable-alternativa-mas-barata-al-Mini-4-Pro.831313.0.html (visitado 28-05-2024) (vid. pág. 52).

[50] Fintan Corrigan. *DJI Mavic Mini Review Of Features, Specs Along With FAQs*. 2020. URL: https://www.dronezon.com/drone-reviews/dji-mavic-mini-review-of-features-specs-and-faqs/ (visitado 25-05-2024) (vid. pág. 53).

[51] Basil Kronfli. *DJI Mavic Mini review*. 2022. URL: https://www.techradar.com/reviews/dji-mavic-mini (visitado 30-04-2024) (vid. pág. 53).

[52] DC Rainmaker. *DJI Mavic Mini In-Depth Review*. 2019. URL: https://www.dcrainmaker.com/2019/12/dji-mavic-mini-in-depth-review.html (visitado 11-07-2024) (vid. pág. 53).

[53] TechGearLab. *DJI Mavic Mini Review*. 2020. URL: https://www.techgearlab.com/reviews/cool-gadgets/drones/dji-mavic-mini (visitado 03-07-2024) (vid. pág. 53).

[54] Drone Dreams Perú. *Phantom 4 Pro V 2.0 Advanced Drone DJI*. 2024. URL: https://www.dronedreams.com.pe/product/phantom-4-pro-v-2-0-advanced-drone-dji/ (visitado 15-07-2024) (vid. pág. 54).

[55] Christopher Bryan-Smith. *What is Rolling Shutter Effect? (And How to Avoid it)*. 2024. URL: https://expertphotography.com/rolling-shutter-effect/ (visitado 15-07-2024) (vid. pág. 55).

[56] Adobe Creative Cloud. *Rolling Shutter Effect*. 2024. URL: https://www.adobe.com/au/creativecloud/video/discover/rolling-shutter-effect.html (visitado 15-07-2024) (vid. pág. 55).

[57] StudioBinder. *What is Rolling Shutter?* 2024. URL: https://www.studiobinder.com/blog/what-is-rolling-stutter/ (visitado 15-07-2024) (vid. pág. 55).

[58] Lance Ulanoff. *DJI Phantom 4 review*. 2016. URL: https://www.techradar.com/reviews/cameras-and-camcorders/dji-phantom-4-1322207/review (visitado 21-06-2024) (vid. pág. 56).

[59] Billy Kyle. *Review: The DJI Phantom 4 Pro is the best drone that you can buy*. 2018. URL: https://dronedj.com/2018/08/20/review-phantom-4-pro-best-drone/ (visitado 05-07-2024) (vid. pág. 56).

[60] Matthew DeBord. *DJI Phantom 4 Advanced announced*. 2017. URL: https://www.digitaltrends.com/photography/dji-phantom-4-advanced-announced/ (visitado 19-04-2024) (vid. pág. 56).

[61] Elektor. *Raspberry Pi NoIR Camera Module V2*. 2024. URL: https://www.elektor.com/products/raspberry-pi-noir-camera-module-v2 (visitado 15-07-2024) (vid. pág. 58).

[62] Pimoroni. *Raspberry Pi Camera v2 - NoIR*. 2024. URL: https://shop.pimoroni.com/products/raspberry-pi-camera-module-v2?variant=19833929799 (visitado 15-07-2024) (vid. pág. 58).

[63] Rui Santos. *Guide to Raspberry Pi Camera V2 Module*. 2017. URL: https://randomnerdtutorials.com/guide-to-raspberry-pi-camera-v2-module/ (visitado 15-07-2024) (vid. pág. 58).

[64] Raspberry Pi. *Raspberry Pi NoIR Camera V2*. 2024. URL: https://www.raspberrypi.com/products/pi-noir-camera-v2/ (visitado 15-07-2024) (vid. pág. 58).

[65] Raspberry Pi. *Camera Board Comparisons: Pi NoIR V1 vs Pi NoIR V2*. 2017. URL: https://www.raspberrypi.com/news/camera-board-comparisons-pi-noir-v1-vs-pi-noir-v2/ (visitado 15-07-2024) (vid. pág. 58).

[66] Adafruit Industries. *Raspberry Pi NoIR Camera Board v2 - 8 Megapixels*. 2024. URL: https://www.adafruit.com/product/3100 (visitado 20-07-2024) (vid. pág. 58).

[67] RC Innovations. *Batería LiPo 4S 14.8V 1550mAh 95C U-Tech Pro*. 2024. URL: https://rc-innovations.es/shop/bateria-lipo-4s-14-8v-1550mah-95c-utech-pro?category=16#attr=3132,436,2373 (visitado 15-07-2024) (vid. pág. 60).

[68] FreeCAD. *FreeCAD Documentation*. 2024. URL: https://wiki.freecad.org/Main_Page (visitado 07-06-2024) (vid. pág. 63).

[69] NVIDIA. *Jetson Nano: Desarrollo de Producto*. 2024. URL: https://www.nvidia.com/es-la/autonomous-machines/embedded-systems/jetson-nano/product-development/ (visitado 15-07-2024) (vid. pág. 71).

[70] jasonhua. *How to use the supplied antennas with reComputer J3011*. 2022. URL: https://forum.seeedstudio.com/t/how-to-use-the-supplied-antennas-with-recomputer-j3011/273093 (visitado 15-07-2024) (vid. pág. 71).

[71] Amazon. *AKYGA AK-NB-08A Cable de alimentación para portátil*. 2024. URL: https://www.amazon.es/AKYGA-AK-NB-08A-Cable-alimentaci%C3%B3n-port%C3%A1til/dp/B07TYN6953 (visitado 15-07-2024) (vid. pág. 72).

[72] NVIDIA. *Jetson Nano Developer Kit User Guide*. 2024. URL: https://developer.download.nvidia.com/assets/embedded/secure/jetson/Nano/docs/NV_Jetson_Nano_Developer_Kit_User_Guide.pdf (visitado 15-07-2024) (vid. pág. 72).

[73] NVIDIA. *Get Started With Jetson Nano Developer Kit*. 2024. URL: https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-devkit (visitado 15-07-2024) (vid. pág. 72).

[74] NVIDIA. *Jetson Software Architecture*. 2024. URL: https://docs.nvidia.com/jetson/archives/r35.4.1/DeveloperGuide/text/AR/JetsonSoftwareArchitecture.html (visitado 15-07-2024) (vid. pág. 73).

[75] JetsonHacks. *JetsonHacksNano GitHub Repository*. 2024. URL: https://github.com/JetsonHacksNano (visitado 15-07-2024) (vid. págs. 76, 77).

[76] JetsonHacks. *NVIDIA Jetson Development - JetsonHacks*. 2024. URL: https://jetsonhacks.com/ (visitado 15-07-2024) (vid. pág. 77).

[77] JetsonHacks. *JetsonHacks YouTube Channel*. 2024. URL: https://www.youtube.com/@JetsonHacks/videos (visitado 15-07-2024) (vid. pág. 77).

[78] Raspberry Pi. *Raspberry Pi Documentation - Camera*. 2024. URL: https://www.raspberrypi.com/documentation/accessories/camera.html (visitado 15-07-2024) (vid. pág. 112).

[79] DRONExpert. *DRONExpert Netherlands - Specialist in thermal UAV solutions*. 2024. URL: https://dronexpert.nl/ (visitado 15-07-2024) (vid. pág. 115).

[80] DRONExpert. *DJI Phantom 3 payload test part2*. 2024. URL: https://www.youtube.com/watch?v=NbMKw5dxkFU (visitado 15-07-2024) (vid. pág. 115).

[81] Thingiverse. *DJI Mavic - Lipo saver*. 2024. URL: https://www.thingiverse.com/thing:2019114 (visitado 15-07-2024) (vid. págs. 117, 120).

[82] Thingiverse. *DJI Phantom 135mm Leg Extension*. 2024. URL: https://www.thingiverse.com/thing:2735825 (visitado 15-07-2024) (vid. pág. 119).

[83] Thingiverse. *Extra long DJI Phantom 3 - 4 landing gear*. 2024. URL: https://www.thingiverse.com/thing:2818197 (visitado 15-07-2024) (vid. pág. 119).

[84] Cesur. *¿Cuánto gana un ingeniero aeronáutico?* 2021. URL: https://www.cesurformacion.com/blog/cuanto-gana-un-ingeniero-aeronautico-en-2021-y-un-tecnico (visitado 27-01-2024) (vid. pág. 140).

[85] Universitat Politècnica de València. *Retribuciones profesorado funcionario en base a la ley de presupuestos para el ejercicio 2023*. 2023. URL: https://www.upv.es/entidades/SRH/retribuciones/U0950454.pdf (visitado 27-01-2024) (vid. pág. 140).

[86] Autodesk Inc. *Autodesk AutoCAD*. 2024. URL: https://www.autodesk.es/products/autocad/overview?term=1-YEAR&tab=subscription (visitado 27-01-2024) (vid. pág. 142).

[87] Overleaf. *Overleaf - Student Plans*. 2024. URL: https://es.overleaf.com/user/subscription/plans (visitado 27-01-2024) (vid. pág. 142).

[88] Microsoft. *Microsoft 365 Empresa Estándar - Planes con Teams*. 2024. URL: https://www.microsoft.com/es-es/microsoft-365/business/compare-all-microsoft-365-business-products (visitado 27-01-2024) (vid. pág. 142).

[89] Sostenibilidad.com. *¿Qué son los Objetivos de Desarrollo Sostenible?* 2024. URL: https://www.sostenibilidad.com/desarrollo-sostenible/que-son-los-objetivos-de-desarrollo-sostenible/?_adin=132415900 (visitado 15-07-2024) (vid. pág. 151).

[90] Wikipedia. *Objetivos de Desarrollo Sostenible*. 2024. URL: https://es.wikipedia.org/wiki/Objetivos_de_Desarrollo_Sostenible (visitado 15-07-2024) (vid. pág. 151).

[91] Programa de las Naciones Unidas para el Desarrollo (PNUD). *Objetivos de Desarrollo Sostenible*. 2024. URL: https://www.undp.org/es/sustainable-development-goals (visitado 15-07-2024) (vid. pág. 151).