UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Detección y localización de objetos remotos mediante visión por computador desde vehículos aéreos no tripulados

Trabajo Fin de Grado

Grado en Ingeniería Electrónica Industrial y Automática

AUTOR/A: Ballester Sanchís, Miquel

Tutor/a: González Sorribes, Antonio

CURSO ACADÉMICO: 2022/2023

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Detección y localización de objetos remotos mediante visión por computador desde vehículos aéreos no tripulados.

Trabajo Fin de Grado

Grado en Ingeniería Electrónica Industrial y Automática

AUTOR/A: Ballester Sanchis, Miquel

Tutor/a: González Sorribes, Antonio

CURSO: 2022/2023

AGRADECIMIENTOS

Me gustaría aprovechar este pequeño espacio para agradecer a todas aquellas personas que me han hecho posible llegar hasta aquí.

En primer lugar, me gustaría agradecerle a mi tutor Antonio, por confiar en mí y en este trabajo. Gracias por brindarme de la ayuda y los ánimos necesarios para llevarlo a cabo.

Gracias a todos aquellos compañeros, a los que ahora considero amigos, por los buenos ratos y por hacer que las horas y horas de estudio se hiciesen un poco más amenas.

Cómo no, muchísimas gracias a mi familia y mis buenos amigos. En definitiva, gracias a mis seres más queridos por todos los momentos de alegría y por ayudarme a superar los momentos de pena. Gracias por todas las experiencias que hemos vivido en compañía y por las que nos quedan por vivir.

Por último, tengo que agradecer a todos aquellos profesores y monitores que me han enseñado a lo largo de la vida, no solo conocimientos sino también a mejorar como persona, y sin los que actualmente no podría estar aquí. En definitiva, gracias a todas aquellas personas del sector de la educación, tanto formal como no formal. Sois vosotros y vosotras los que tenéis la verdadera capacidad de hacer de este mundo un lugar mejor.

RESUMEN

La finalidad del proyecto es diseñar y programar un algoritmo que mediante procesamiento de imagen sea capaz de, en primer lugar, diferenciar un objeto de interés dentro de un cierto espacio de búsqueda desde un vehículo aéreo dotado de una cámara, y en segundo lugar resolver la localización del objeto respecto a un sistema de referencia global.

Con esta finalidad se propone un algoritmo basado en el modelo Pin-Hole y matrices de transformación. El algoritmo propuesto se implementará en un dron DJI Tello el cual dispone de una cámara para la captación de imágenes, una IMU interna para resolver la posición y orientación (pose) del dron y un sistema de posicionamiento visual compuesto por una cámara y un módulo infrarrojo situados en la parte inferior de la aeronave, facilitando así el vuelo estacionario.

PALABRAS CLAVE: Vehículo aéreo no tripulado; Localización; Visión por Computador; Programación.

INDICE GENERAL

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Trabajo Fin de Grado

MEMORIA Y ANEXOS

ÍNDICE DE LA MEMORIA

1. Introducción

A continuación, se presenta una breve introducción a la historia de la robótica y se repasa la relevancia que tiene en la actualidad la robótica aérea.

En la historia, el primer momento en el que se hace presente la palabra *Robot* es en 1920, año en el que Karel Capek (1890-1938), destacado escritor de ciencia ficción checo, publica la novela *Rossum´s Universal Robot,* siendo pues esta palabra de origen checo y que significa "esclavo".

Sin embargo, el termino no se hubiese hecho tan famoso si no fuese por la película *Metrópolis* de 1926, en la que aparece un robot antropomórfico que fue muy popularizado, y posteriormente en 1950 gracias a Isaac Asimov (1920-1992), reconocido escritor y divulgador científico, quien postuló las conocidas como **3 leyes de la robótica** en el libro *Yo Robot* (Armesto Ángel, 2023).

Los primeros robots móviles que se crearon surgieron en 1948 gracias a William Grey Walter (1910-1977), destacado neurofisiólogo y pionero de la robótica, quien creó las denominadas **tortugas de Bristol.**

Más adelante, a principios de la década de 1960 se creó el primer robot industrial programable, el **Unimate.** Los encargados de su creación fueron George Devol (1912-2011) y Joseph Engelberger (1925-2015), quienes también fundarían la que sería la primera empresa dedicada a la robótica industrial, **Unimation.**

A principio del siglo XXI, en el año 2000 se crearon los primeros drones con capacidad de vuelo autónomo, los primeros robots aéreos (Armesto Ángel, s. f.).

Así pues, en las últimas décadas la robótica aérea ha experimentado un gran auge gracias al amplio rango de aplicación que esta tiene: vigilancia, seguridad, agricultura, exploración, arqueología… Los robots aéreos permiten la realización de trabajos que anteriormente eran muy difíciles, si no imposibles de realizar. Es por esto por lo que ha habido un gran aumento en la demanda de estos y en la investigación de nuevas tecnologías que mejoren su eficiencia y autonomía.

Una de las áreas de investigación de gran relevancia en la actualidad es el reconocimiento del entorno mediante **visión por computador**. Es así debido a la necesidad de los robots aéreos de reconocer su entorno y diferenciar los objetivos clave mediante la utilización de cámaras y algoritmos de procesamiento de imágenes para poder tener el **funcionamiento autónomo** que se busca. Esta área de investigación

tiene grandes aplicaciones en tareas como la asistencia en operaciones de búsqueda y rescate, la perimetración de incendios y la supervisión de cultivos, entre muchas otras.

En este contexto, este TFG se enfocará en la localización de objetivos mediante visión por computador en la **robótica aérea**, presentando así una metodología que implementará los métodos y técnicas necesarios. Además, se discutirán los resultados obtenidos y se discutirá su relevancia en el ámbito ya mencionado, buscando en todo momento contribuir al avance de la visión por computador en esta rama de la robótica.

1.1. Objetivos

Para organizar de una forma estructurada el proceso de la realización de este trabajo se han buscado una serie de objetivos o hitos que cumplir marcando así un camino a seguir hacia el objetivo final del proyecto.

A continuación, se muestran los más relevantes:

- **Consolidar los conocimientos** sobre los modelos y técnicas aprendidos en el grado y adquirir nuevos conocimientos necesarios para su aplicación en el proyecto.
- Elegir el **software y hardware** necesario y familiarizarse con sus características, particularidades y especificaciones técnicas.
- Establecer una **conexión** inalámbrica entre el computador encargado del procesamiento y la aeronave no tripulada elegida.
- **Calibrar la cámara** elegida a partir de sus especificaciones técnicas y del conocimiento de métodos de calibración.
- Aplicar la calibración para **corregir distorsiones** de las lentes de la cámara y obtener la matriz intrínseca de la misma.
- Desarrollar un algoritmo que mediante la extracción de características del objetivo sea capaz de **diferenciar** este del fondo en una imagen dada y calcular su centro de gravedad.
- **Calcular las ecuaciones** necesarias que a partir del modelo Pin Hole sean capaces de estimar la posición del centro de gravedad del objetivo con respecto de la cámara.
- Aplicar matrices de transformación para poder **obtener la posición** del objetivo con respecto de un sistema de referencia global, ya sea coincidente o no con el de la cámara del dron.

1.2. Motivación

La principal motivación para la realización de este proyecto radica en poder brindar de una herramienta mejorada al ámbito de la robótica aérea. La posibilidad de poder utilizar la visión por computador para localizar incendios o personas extraviadas en entornos naturales como el mar o la montaña es una realidad con mucho potencial para influenciar positivamente en la vida de las personas, así como en la fauna y la flora. Es así como este proyecto aspira a crear una mejora tangible en la sociedad, ayudando a los profesionales del rescate a evitar posibles riesgos y a poder ofrecer una respuesta más rápida ante situaciones de emergencia.

2. Estado del arte

Actualmente, dentro del ámbito de la robótica se pueden encontrar una gran variedad de diseños con distintas finalidades. Es muy difícil nombrar todas las clasificaciones posibles dentro de la robótica en este documento debido a la gran variedad dentro de estos, sin embargo, acto seguido se ofrece una pequeña clasificación de robots según distintos criterios.

2.1. Clasificación de robots según su aplicación

Esta clasificación diferencia los robots según la función para la que están diseñados. Entre ellos se pueden diferenciar:

- Robots **industriales**: Diseñados para la realización de tareas por lo general muy repetitivas dentro de un entorno industrial. Estos robots son los que más han crecido en número desde su creación, especialmente en países como China, Japón o Estados Unidos.
- Robots **médicos**: Destinados a ser usados con un fin enfocado a la medicina como la rehabilitación o la cirugía asistida. Dentro de esta clasificación destacan los renombrados nanobots, robots de escala diminuta que se inyectan en el cuerpo del paciente para combatir enfermedades muy concretas. Es importante destacar que la mayoría se encuentran en una fase de investigación y desarrollo.
- Robots de **servicio**: Destinados a prestar asistencia a las personas en entornos no controlados, sobre todo vistos con fines del ámbito doméstico. Estos son principalmente la cara visible de la robótica puesto que son los más presentes en la vida de la gente de a pie. También

existen dentro de esta clasificación los robots utilizados por profesionales para dar servicios, como los robots de exploración.

2.2. Clasificación de robots según su autonomía

Esta clasificación se centra en la necesidad de la intervención humana dentro del trabajo del robot para que este sea funcional. Aquí se puede ver:

- Robots **teleoperados**: Controlados remotamente por un piloto humano en tiempo real. Son los más comunes debido a su simplicidad en comparación con los otros de la lista.
- Robots **semiautónomos**: Son capaces de realizar tareas preprogramadas, pero con la con la condición de mantenerse bajo la supervisión y el control humano.
- Robots **autónomos**: Estos son los robots más independientes puesto que no requieren de ningún tipo de supervisión humana. Son capaces de tomar decisiones a partir de la información recibida de su entorno y de sus sensores.

2.3. Clasificación de robots según su capacidad de aprendizaje

En esta clasificación se dividen los robots según la capacidad de aprender y crear nuevas pautas de comportamiento propias:

- Robots **programables**: Estos robots se limitan a seguir unas tareas definidas según un programa fijo.
- Robots con **Inteligencia Artificial** (IA): Capaces de adaptarse y mejorar con el tiempo, aprendiendo a través de una serie de algoritmos y redes neuronales.

2.4. Clasificación de robots según su configuración

Aquí se dividen los robots según la estructura física con la que se han diseñado. En esta clasificación se puede observar:

- Robots manipuladores: Destinados a labores de ensamblaje, fabricación y manipulación, estos robots son característicos por sus diseños centrados en sus brazos y pinzas. Hay que destacar que actualmente se está realizando una gran labor en las conocidas como

soft gripper, pinzas blandas que permiten adaptarse mejor al objeto que se desea manipular permitiendo mover o coger objetos más delicados o de formas irregulares o amorfas.

- Robots humanoides: La forma de estos robots están basados en la anatomía humana, con un cuerpo y extremidades similares. La mayoría de los robots sociales (robots destinados a comunicarse con las personas) suelen ser de este tipo puesto que la similitud entre el robot y la persona con la que trata pretende formar un vínculo de confianza y comodidad. Sin embargo, en muchas ocasiones produce el efecto contrario ya que los humanos al ver una réplica antropomórfica que se acerca en exceso a su apariencia entran en un estado de rechazo y desconfianza conocido como valle inquietante (**uncanny valley**), aunque esta es una teoría que tiene muchos detractores y no está del todo demostrada.
- Robots móviles: Estos robots son diseñados para tener la capacidad de moverse a través de su entorno. A continuación, se hablará en mayor profundidad de estos.

2.4.1. Robots móviles

Como bien se ha dicho, los robots móviles son aquellos diseñados para moverse a través de su entorno, utilizando ruedas, patas, alas… A su vez se pueden diferenciar tres tipos de robots móviles, los robots móviles terrestres, los robots móviles acuáticos y los robots móviles aéreos.

A continuación, en la **Figura 1** se muestra una imagen ilustrativa de las distintas clasificaciones generales en robots móviles.

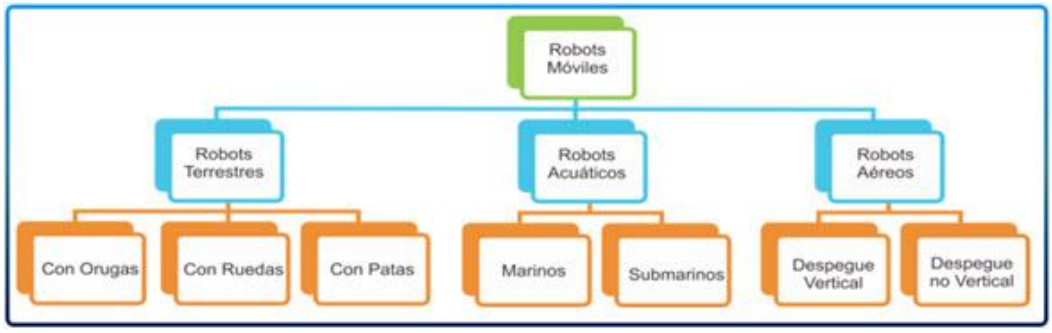


Figura 1. Clasificaciones generales de los robots móviles.

FUENTE: HTTPS://GO.GALE.COM/PS/I.DO?P=IFME&U=GOOGLESCHOLAR&ID=GALE|A302114813&V=2.1&IT=R&SID=GOOGLESCHOLAR&ASID=0FEBB240

2.4.1.1. Robots móviles terrestres

Los robots móviles terrestres, o vehículos terrestres no tripulados (UGV de sus siglas en inglés) como bien dice su nombre son aquellos destinados a moverse a través de la superficie terrestre sólida, por el suelo. Estos suelen tener actuadores (motores) que mueven sus patas o ruedas produciendo así el desplazamiento. Según la disposición y el tipo de su actuador se pueden dividir en diversos tipos.

Si el robot tiene patas se pueden clasificar según el número de patas (bípedos, cuadrúpedos, hexápodos…). En la **Figura 2**, **Figura 3** y **Figura 4** se muestran algunos ejemplos.

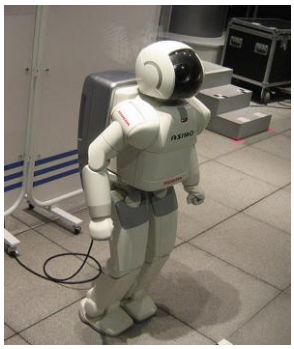


Figura 2.Robot Bípedo.

Figura 3. Robot Cuadrúpedo.

Figura 4. Robot hexápodo

FUENTE: HTTPS://HMONG.ES/WIKI/LEGGED_ROBOT

Por otro lado, los robots móviles terrestres con ruedas tienen una mayor diferenciación según las características de estas y su disposición. En la **Figura 5**, se muestran 6 de las configuraciones más comunes en robots móviles con ruedas.

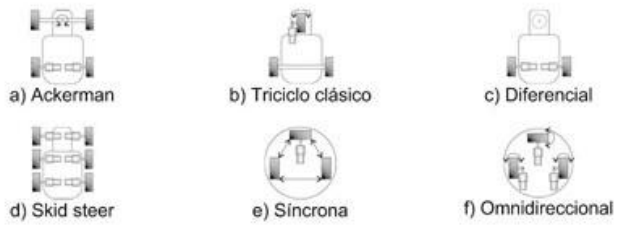


Figura 5. Configuraciones cinemáticas de los robots móviles con ruedas.

FUENTE:
HTTPS://GO.GALE.COM/PS/I.DO?P=IFME&U=GOOGLESCHOLAR&ID=GALE|A302114813&V=2.1&IT=R&SID=GOOGLESCHOLAR&ASID=0FEBB240

La configuración Ackerman es muy conocida puesto que es la utilizada generalmente en los vehículos móviles comerciales como los turismos, en las que las ruedas de delante son las encargadas de producir el giro en el vehículo. El caso de la configuración en triciclo clásico es muy similar, pero se dispone solamente de una rueda delantera que marca el giro del robot. En ambos casos es importante tener en cuenta que las ruedas no deben estar unidas entre sí por un eje fijo, en algunas situaciones pueden estar separados sus ejes y que sean ruedas que no producen tracción o estar unidas mediante un diferencial. Esto es así puesto que en las curvas ambas ruedas deben poder girar a distintas velocidades para evitar que alguna de las dos pierda tracción y derrape.

El caso de la configuración diferencial es muy utilizado en la robótica puesto que permite una mayor movilidad que los dos anteriores, en esta configuración se tienen dos ruedas con tracción independiente, es decir, ambas pueden actuar de manera independiente, siendo capaces de producir un giro de radios mucho más cerrados e incluso de hacer rotar el robot sobre sí mismo. El caso de la configuración *Skid Steer* es muy similar, también se basa en crear una diferencia de velocidades en las ruedas de ambos lados del robot para producir las maniobras. Esta configuración dispone de 4 o más ruedas con actuadores independientes, permitiendo además una gran fuerza de tracción y creando una configuración muy útil para terrenos irregulares.

En el caso de los robots omnidireccionales, estos se pueden conseguir mediante un control sobre todos los ejes verticales de las ruedas o con la utilización de ruedas especiales conocidas como ruedas omnidireccionales. Una de las ruedas omnidireccionales más conocidas son las ruedas suecas o ruedas *mecanum*. Estas permiten realizar movimientos en direcciones distintas a su dirección de giro (Barrientos Sotelo, García Sánchez, & Silva Ortigoza, 2007).

En la **Figura 6**, se presenta una imagen con algunos ejemplos de rudas que se pueden encontrar en la robótica terrestre.

Figura 6. Tipos de ruedas.

FUENTE: HTTPS://GO.GALE.COM/PS/I.DO?P=IFME&U=GOOGLESCHOLAR&ID=GALE|A302114813&V=2.1&IT=R&SID=GOOGLESCHOLAR&ASID=0FEBB240

Además de estas, otras formas de desplazamiento muy conocidas son las orugas, muy útiles para producir desplazamiento en terrenos especialmente irregulares, o con poco agarre.

2.4.1.2. Robots móviles acuáticos

En el caso de los robots móviles acuáticos, aquellos destinados a través de masa de agua, se pueden encontrar dos tipos de robots, aquellos destinados a moverse por la superficie o USV (del inglés *Unmanned Surface Vehicle*) y aquellos destinados a moverse por el interior del agua o UUV (del inglés *Unmanned Underwater Vehicle*), como el mostrado en la **Figura 7**.

Además, dentro de los USV se pueden encontrar operados de manera remota, semiautónomos o totalmente autónomos, siendo estos últimos conocidos como ASV por sus siglas en inglés. La misma diferenciación se puede realizar en los UUV siendo aquellos controlados remotamente como ROV o ROUV y aquellos autónomos como AUV (Rodríguez, 2023).

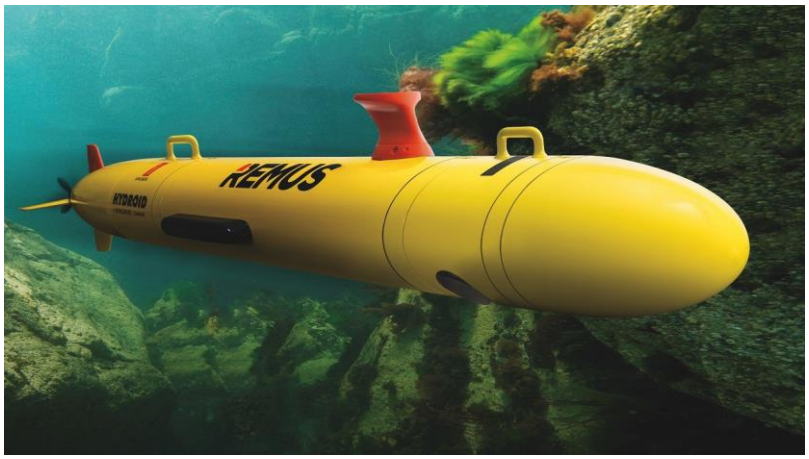


Figura 7. Remus 300, ejemplo de UUV.

FUENTE: HTTPS://WWW.NAVAL-TECHNOLOGY.COM/PROJECTS/REMUS-300-UNMANNED-UNDERWATER-VEHICLE-UUV/

2.4.1.3. Robots móviles aéreos

Por último, se pueden encontrar los robots móviles aéreos o UAV de sus siglas en inglés (Unmanned Air Vehicle). Esta clase de robots se desplaza a través del aire centrando una gran parte de su programación en mantener su sustentación aérea. Estos además se pueden dividir principalmente en dos grandes grupos: los robots que mantienen su sustentación aérea gracias a alas fijas (como los aviones) y los que lo hacen gracias a una o más alas rotatorias (como los helicópteros).

Los UAV de ala fija suelen utilizar la velocidad horizontal de la aeronave para generar una diferencia de presión en el ala creando así un empuje vertical hacia arriba. Este sistema permite a la aeronave volar por un periodo mucho más prolongado que en el caso de las alas rotatorias a costa de la necesidad de mantener una velocidad horizontal en todo momento, limitando su movilidad. Estas aeronaves suelen tener un sistema de despegue horizontal o HTOL (del inglés Horizontal Take-off and Landing) aunque también pueden servirse de sistemas de sustentación alternativos como motores a reacción para conseguir una fuerza de sustentación manteniendo la aeronave estática, de forma que también existen aeronaves de ala fija con un sistema de despegue vertical o VTOL (Vertical Take-off and Landing). Un ejemplo de aeronave de ala fija es el de la **Figura 8.**

Figura 8. UAV de ala fija.

FUENTE: HTTPS://TOPESDEGAMA.COM/LISTAS/GADGETS/DRONES-ALA-FIJA

En el caso de los UAV de ala rotatoria, estos deben su sustentación aérea a una o más hélices que, al hacer girar sus aspas, producen una fuerza de sentido ascendente que se opone a la gravedad. Estas aeronaves suelen ser VTOL puesto que no necesitan de ninguna velocidad horizontal para poder alzarse. La maniobrabilidad de estas aeronaves las hace ideales para ser pilotadas en espacios reducidos o con obstáculos, y la falta de necesidad de la velocidad horizontal, de la que ya se ha hablado, la hace especialmente útil para trabajos que requieran de un posicionamiento estático de la aeronave, todo ello a costa de una menor autonomía puesto que se le debe administrar de manera constante energía para hacer rotar sus hélices.

Cabe destacar que los UAVs de ala rotatoria se pueden clasificar a su vez según el número de hélices que posee, siendo el mínimo una hélice encargada de la sustentación, como lo es el caso de los helicópteros convencionales. En el caso de estos últimos necesitan una segunda hélice en posición horizontal para corregir la rotación generada por el factor P proveniente de la rotación de la hélice principal (la orientada en vertical). En el caso de las aeronaves de dos motores son especialmente conocidos los helicópteros con disposición de motores en tándem (**Figura 9**).

Figura 9. Helicóptero con doble motor en tándem.

FUENTE: HTTPS://AERONAVESMILITARESESPANOLAS.COM/HELICOPTERO-CHINOOK-FAMET/

En el caso del tándem nos es necesario ninguna hélice orientada en horizontal puesto que la rotación en sentido opuesto de sus dos hélices corrige el factor P producido por cada una de ellas. Por otro lado, existe otro tipo de aeronave de dos motores menos conocido, las aeronaves de hélices coaxiales (**Figura 10**).

Figura 10. Dron con hélices coaxiales.

FUENTE: HTTPS://AVIATIONWEEK.COM/DEFENSE-SPACE/AIRCRAFT-PROPULSION/DYNETICS-RECEIVES-CYLINDRICAL-COAXIAL-ROTOR-DRONES

Las aeronaves coaxiales se caracterizan por tener ambas hélices rotando sobre el mismo eje, una sobre la otra y en sentido opuesto. Al igual que en el tándem, esta rotación en sentido contrario de sus dos

hélices principales es suficiente para cancelar el factor P producido por cada una mutuamente.

Por último, las aeronaves de ala rotatoria más conocidas en la robótica aérea son las conocidas como aeronaves de multirrotor. Estas tienen más de dos hélices y pueden estar dispuestas en multitud de diseños. Aumentar el número de motores en un UAV lo hace más estable y le añade potencia vertical, pero implica aumentar todavía más el consumo de energía. Además, muchas de estas aeronaves tienen motores redundantes, es decir, son capaces de redistribuir la energía aportada a cada motor para mantener la sustentación aérea en caso de que alguno de sus motores fallara, lo que las hace más fiables y robustas. Un ejemplo de un multirrotor es el de la **Figura 11** (González Sorribes, 2023).

Figura 11. Multirrotor de cuatro hélices.

FUENTE: HTTPS://JAMESAPHOTO.CO.UK/AUTEL-EVO-NANO-REVIEW/

3. Nociones Teóricas

En este apartado se repasarán la gran mayoría de los conocimientos que han sido necesarios para la realización del proyecto.

3.1. Nociones básicas sobre aviónica

La aviónica, de la palabra inglesa *avionics,* procedente de *aviation* (aviación) y *electronics* (electrónica), se refiere a la "electrónica aplicada a la aviación" (Real Academia Española, s. f.).

Puesto que este proyecto no pretende profundizar en el funcionamiento de la aeronave en sí, solo se centrará en aquellos componentes de la aviónica verdaderamente importantes para el trabajo (González Sorribes, 2023).

3.1.1. Control de vuelo

El control de vuelo son aquellos algoritmos y directrices que la aeronave debe seguir para poder llevar a cabo su misión. Dentro de la programación de las aeronaves se debe diferenciar entre dos escalones de control. El control de bajo nivel y el de alto nivel.

El control de bajo nivel es aquel que encargado de que la aeronave se mantenga en todo momento en el aire. Para ello se desarrollan algoritmos que a partir de sensores son capaces de desarrollar un control sobre los actuadores de la aeronave para evitar que esta entre en pérdida. Esta programación debe de ser muy rápida para hacer frente a los problemas lo más pronto posible de forma que no afecten a la sustentación aérea o lo hagan en el menor nivel posible.

Por otro lado, el control de alto nivel se encarga de tareas más elaboradas como la comunicación entre distintas aeronaves (en el caso de un enjambre), la localización de objetivos o el seguimiento de trayectorias. Puesto que esta tarea es menos prioritaria para la aeronave, no es necesario que sea tan rápida como en la de bajo nivel. Es este tipo de control en el que se centra este trabajo.

3.1.2. Actuadores

Los actuadores son aquellos dispositivos que permiten transformar la energía en este caso eléctrica, aunque puede ser de combustión o ambas (dron hibrido), en un movimiento físico (eliteradiocontrol, s. f.).

En el caso de un cuadricóptero, los actuadores serán los motores que harán girar las hélices y se encargarán de generar el empuje necesario para

garantizar la sustentación aérea y qué la aeronave produzca los movimientos que se le ordenen. A la hora de calcular el empuje de los motores se debe tener en cuenta que el empuje total máximo de estos debe ser como mínimo el doble del peso de la aeronave. Es decir, con una aeronave de peso total igual a 1 kilogramo, el empuje de la aeronave debe poder llegar como mínimo a 2 kilogramos.

En la clasificación de los motores eléctricos se deben poder diferenciar entre los motores con escobillas o *brushed*, y los motores sin escobillas o *brushless*, los cuales son mucho más eficientes por lo que son los más utilizados en RPAS.

Dentro de estos se puede hacer además una clasificación entre *inrunners* y *outrunners*. La diferencia principal entre estos es que el primero es capaz de ofrecer una velocidad angular (rpm) mayor a costa de un par (potencia) más bajo, mientras que los segundos tienen mucha potencia, pero una velocidad angular menor.

3.1.3. Sensores

En el ámbito de la robótica, los sensores son uno de los componentes más importantes puesto que son los encargados de relacionar el procesamiento del robot con el estado de variables internas del mismo (sensores propioceptivos) o con variables externas pertenecientes al entorno (sensores exteroceptivos).

En el ámbito de la robótica aérea los sensores más importantes son:

- Altímetros. Dedicados a medir la altura a la que se encuentra la aeronave. Los más comunes son los altímetros barométricos, los cuales se aprovechan de la variación de presión que existe a distintas alturas de la atmósfera para obtener una estimación de su altura en cada instante.
- IMUs. Las IMU (Inertial Measurement Unit) son unos dispositivos encargados de la medición de parámetros inerciales de la aeronave como pueden ser la velocidad y aceleración tanto lineal como angular. Se hablará de estos dispositivos más adelante.

- Telémetros. Los telémetros láser empiten pulsos laser con los cuales, a partir del tiempo en el que tardan en detectar el reflejo de estos, pueden obtener una estimación de la distancia entre el telémetro y la superficie que ha producido la reflexión. También existen telémetros de ultrasonidos los cuales se basan en el mismo sistema de medición del tiempo de vuelo, pero en este caso no se utiliza un pulso laser sino una onda ultrasónica. Funciona de manera similar a como lo haría un radar.
- Sistemas LIDAR. La palabra LIDAR (Light Detection and Ranging) se refiere a un sistema capaz de obtener una serie de puntos de una superficie y calcular su distancia con respecto de la aeronave que lo transporta a partir de telemetría laser. Estos sistemas son muy utilizados para estimar la topografía de un determinado terreno. Dentro de los sistemas hay diferentes tipos de escaneado según el patrón que utiliza el LIDAR para cubrir el terreno con el haz laser.
- Cámaras. La utilización de las cámaras en la robótica aérea permite al robot obtener imágenes y videos que se pueden procesar en tiempo real o *a posteriori*. Estas imágenes procesadas pueden servir para multitud de objetivos: vigilancia, obtención de relieves, monitorización, salvamento… Las cámaras pueden ir desde cámaras sencillas a color o cámaras mucho más enfocadas, como las cámaras térmicas. Se hablará en más detenimiento sobre las cámaras y la óptica más adelante.

De entre todos estos sensores, los más importantes en este proyecto en concreto son las cámaras, la IMU y el altímetro.

3.1.4. Otros componentes

Otros componentes importantes a tener en cuenta antes de elegir la aeronave pueden ser:

- La batería. La batería es unos de los componentes más importantes a tener en cuenta puesto que será la encargada de

dar energía a toda la aeronave mientras está en el aire. Esta ha de cumplir unas características que tengan en cuenta el peso y el tamaño de esta además de su capacidad (mAh), su tensión nominal (V) o su tasa de carga y descarga (c).

- La estación tierra. La estación tierra es aquel sistema equipado con todo aquello necesario para que el o la operadora pueda poner en marcha la aeronave y comunicarse con esta. Debe de estar equipada con un buen sistema de comunicaciones y compone la interfaz entre el o la piloto y la aeronave.
- El sistema de comunicaciones a utilizar. La importancia del sistema de comunicaciones reside en que es el encargado de transmitir información entre la aeronave y la estación tierra. De este depende, en gran medida, la distancia a la que puede alejarse la aeronave de la estación y la robustez de la comunicación. Sin un buen sistema de comunicaciones y un sistema *failsafe* (el cual permite a la aeronave regresar a su punto de despegue en el caso de perder la conexión), la aeronave se podría quedar en "Stand-By" en el aire, es decir, completamente quieta sin recibir órdenes hasta que se quede sin energía, o directamente caer al suelo (Dronecasero, s. f.).

3.2. Matrices de transformación

3.2.1. Necesidades del posicionamiento de la aeronave

Para poder llevar a cabo la ubicación del objetivo en unas coordenadas mundo a partir de una aeronave se debe poder obtener la **posición y orientación** de esta previamente. A continuación, se muestra un pequeño esquema explicativo (**Figura 12**).

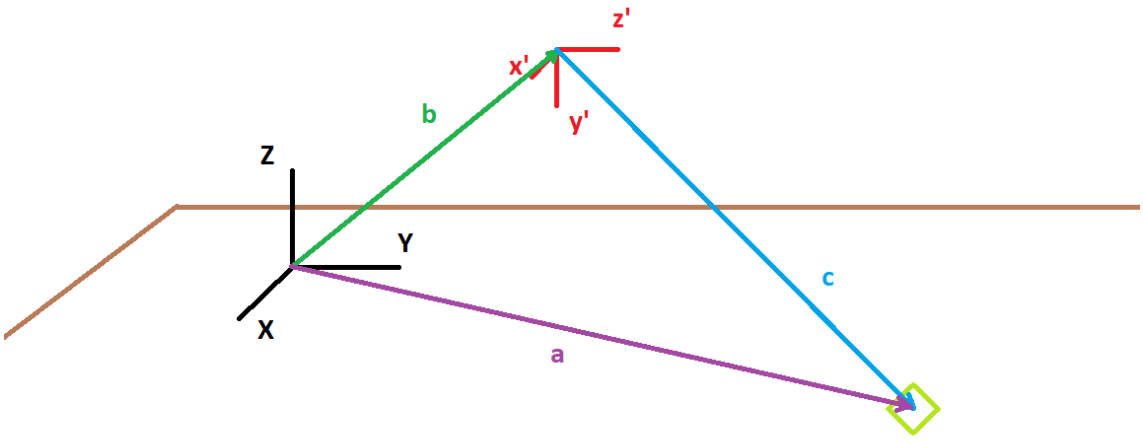


Figura 12. Esquema de localización.

FUENTE: ELABORACIÓN PROPIA

Como se puede observar, para poder obtener el vector a, vector de localización del objetivo (cuadrado verde) con respecto de unos ejes de coordenadas fijos (ejes negros), a partir del vector c, vector de localización del objetivo con respecto de la aeronave (ejes rojos), ha de saberse el vector b, vector de localización de la aeronave con respecto de los ejes fijos.

3.2.2. Coordenadas Cartesianas

Como es bien conocido, las coordenadas cartesianas son un sistema de coordenadas que se basa en un eje de referencia **OXYZ** sobre el cual se definen las posiciones a partir de un vector **p**, el cual está definido por su proyección en los **ejes XYZ** del sistema de referencia. Las tres coordenadas del vector se definen pues a partir de unidades de longitud únicamente. Los vectores unitarios que definen cada uno de los tres ejes de referencia anteriormente mencionados son $\vec{\imath}$ para el eje X, $\vec{\jmath}$ para el eje Y y $\vec{k}$ para el eje Z. En la **Figura 13** se encuentra una representación de este tipo de coordenadas (Cut the knot, s. f.).

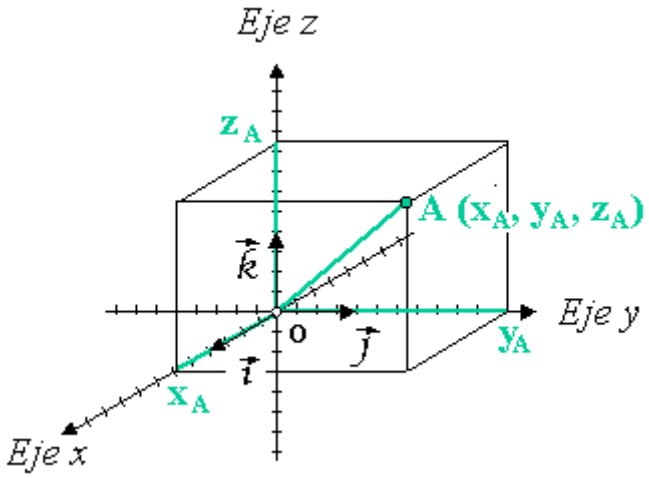


Figura 13. Sistema de coordenadas cartesianas.

FUENTE: HTTPS://ES.WIKIPEDIA.ORG/WIKI/COORDENADAS_CARTESIANAS

3.2.3. Orientación

Puesto que la ubicación del objetivo con respecto de la aeronave se obtiene a partir de un sistema de referencia propio de esta, el sistema de referencia rojo del que se ha hablado en el punto 3.2.1, es de vital importancia conocer también la orientación de este sistema de referencia con respecto del sistema de referencia fijo, es decir, la relación entre los ejes de color negro y los de color rojo. Para representar esta rotación de manera matemática a partir de las matrices de rotación.

Una **matriz de rotación** es una matriz de senos y cosenos que permite teniendo la posición de un punto representado por un vector *p'* respecto de unos ejes de referencia rotados *O'*, obtener el vector de posición *p* del mismo punto respecto de unos ejes de referencia con el mismo centro, pero sin rotar *O*. Es decir $p(X, Y, Z) = R \cdot p'(X', Y', Z')$.

En tres dimensiones esta matriz tiene 3 filas y 3 columnas y se puede descomponer en **rotaciones puras** sobre cada uno de los ejes de referencia (X, Y, Z), siendo estas:

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & Cos(\theta) & -Sin(\theta) \\ 0 & Sin(\theta) & Cos(\theta) \end{bmatrix}$$

$$R_y(\theta) = \begin{bmatrix} Cos(\theta) & 0 & Sin(\theta) \\ 0 & 1 & 0 \\ -Sin(\theta) & 0 & Cos(\theta) \end{bmatrix}$$

$$R_z(\theta) = \begin{bmatrix} Cos(\theta) & -Sin(\theta) & 0 \\ Sin(\theta) & Cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Para representar una rotación compuesta, una rotación que no se produce solo sobre un eje, se suele trabajar con los conocidos como **Ángulos de Euler.**

Existen diversas formas de definir una rotación compuesta a partir de los ángulos de Euler (ZXZ, ZYZ, …), en este trabajo de tratará con los conocidos como ángulos de **Euler tipo 3**. Este modelo de rotación se compone por los ángulos roll (ϕ), pitch (θ) y yaw(ψ), o en la aeronáutica conocidos como **alabeo, cabeceo y guiñada.**

En el caso de los ángulos de Euler tipo 3, las rotaciones se producen de la siguiente forma:

- Primeramente, se produce una rotación del ángulo *roll* sobre el eje x original.
- Seguidamente, se produce una rotación de pitch sobre el eje y original.
- Por último, se produce una rotación del ángulo yaw sobre el eje z original.

De esta forma una rotación completa definida por matriz de rotación a partir de los ángulos de Euler tipo 3 quedaría de la siguiente forma:

$$R(\phi, \theta, \psi) = R_z(\psi) * R_y(\theta) * R_x(\phi)$$

$$= \begin{bmatrix} C\psi S\theta & C\psi S\theta S\phi - S\psi C\phi & C\psi S\theta C\phi + S\psi S\phi \\ S\psi S\theta & S\psi S\theta S\phi + C\psi C\phi & S\psi S\theta C\phi - S\psi S\phi \\ -S\theta & C\theta S\phi & C\theta C\phi \end{bmatrix}$$

$$Siendo\ C\ coseno\ y\ S\ seno.$$

Es importante mencionar otras herramientas matemáticas para la representación de rotaciones en el espacio, tales como los pares de rotación o los cuaterniones. Los cuaterniones son herramientas matemáticas muy utilizadas puesto que permiten expresar la rotación de manera muy compacta lo que a nivel computacional es muy útil.

3.2.4. Matrices de transformación como herramienta matemática

Para simplificar una translación, la posición de un punto respecto de otro, una herramienta muy útil son las conocidas como coordenadas homogéneas. Si un punto esta expresado como $p = (x, y, z)^T$, su expresión en coordenadas homogéneas es $p' = (\eta x, \eta y, \eta z, \eta)^T$, siendo $\eta = 1$.

Así pues, conociendo esta herramienta, se puede definir la posición y orientación de unos ejes de coordenadas A sobre unos ejes de coordenadas B a partir de las matrices de transformación homogénea. Estas matrices son capaces de definir la translación y rotación de la **Figura 14** de la siguiente forma:

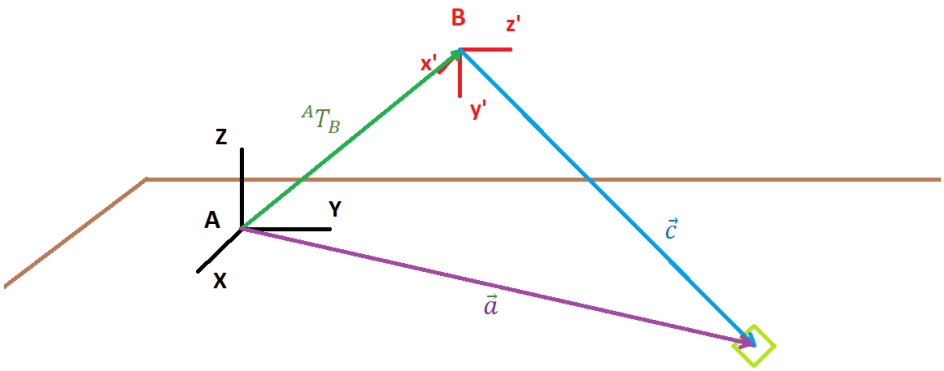


Figura 14. Ejemplo de matriz de transformación.

FUENTE: ELABORACIÓN PROPIA

Es así como los ejes A se podrán definir como $\boldsymbol{A} = {}^{\boldsymbol{A}}\boldsymbol{T}_{\boldsymbol{B}} * \boldsymbol{B}$.

Las matrices de transformación están compuestas por una matriz de rotación y unas coordenadas homogéneas que definen la rotación y la translación de un sistema de referencia con respecto del otro de la siguiente forma: $T = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix}$ siendo R la matriz de rotación (explicada anteriormente) y t las coordenadas homogéneas explicadas al inicio de este apartado (González Sorribes, 2023).

$$ {}^{A}T_{B} = \begin{bmatrix} {}^{A}R_{B\,3x3} & {}^{A}t_{B\,3x1} \\ 0_{1x3} & 1_{1x1} \end{bmatrix} = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Definiendo ${}^{A}R_{B}$ la rotación del sistema B con respecto de A y ${}^{A}t_{B\,3x1}$ la translación

De esta forma la matriz de transformación ${}^{A}T_{B}$ será:

Así pues, el vector $\vec{a}$ del dibujo podrá ser definido como:

$$ \vec{a} = {}^{A}T_{B} * \vec{c} = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} c_x \\ c_y \\ c_z \\ 1 \end{bmatrix} = \begin{bmatrix} a_x \\ a_y \\ a_z \\ 1 \end{bmatrix} $$

Como se puede observar tanto el vector $\vec{c}$ como el vector $\vec{a}$ están expresados mediante coordenadas homogéneas.

3.3. Unidad de medición inercial (IMU)

La IMU es, como se ha hablado anteriormente, un sensor vital de la aeronave puesto que esta mide las fuerzas inerciales del vehículo tales como su velocidad o la aceleración, tanto lineal como angular, parámetros clave para conocer la **pose** (posición y orientación) lo que permite realizar el control de bajo nivel y mantener la sustentación aérea además de corregir desviaciones o servir como información para ciertos algoritmos del control de alto nivel.

Las IMU convencionales suelen estar compuestas por un **acelerómetro** y un **giróscopo** triaxiales. El primero sirve para obtener las mediciones lineales en los tres ejes (x, y, z) mientras que el segundo se encarga de las mediciones angulares.

3.3.1. Grados de libertad

Una forma de diferenciar las distintas IMU del mercado son los grados de libertad de estas. Los **grados de libertad (DoF)** se define como el número de variables independientes entre sí que puede medir el sensor. En el caso de una IMU convencional como la que se ha mencionado anteriormente, esta es de 6 grados de libertad (acelerómetro en los tres ejes + giróscopo en los tres ejes).

Sin embargo, existen otras IMUS, por lo general de mayor precio, que son capaces de medir más variables. Estas pueden contar con un **magnetómetro** triaxial dedicado a medir el campo magnético a través de los tres ejes, muy útil para orientar la aeronave de manera absoluta gracias al campo magnético terrestre, lo que la convertiría en una IMU de 9 grados de libertad. Además de esto, pueden contar con un **barómetro** para estimar la altura de la aeronave, que la convertiría en una IMU de 10 grados de libertad.

La elección de la IMU debe de ser un compromiso entre lo que esta ofrece, teniendo en cuenta su precio, y lo que es necesario para su aplicación. Por ejemplo, no sería necesario una IMU de 10 grados de libertad si la aplicación no necesita conocer la altura, o no es necesario una IMU de 9 grados de libertad si se pueden estimar los parámetros de orientación absolutos a partir de la integración de otras variables que si se pueden conocer con una IMU de 6 DoF. Hay que tener en cuenta que este último caso también puede llevar a errores acumulados si el objetivo para el que se quiere utilizar requiere medidas muy precisas (Wayback Machine, s. f.).

3.4. Nociones básicas sobre óptica

La óptica según la Real Academia Española es la “parte de la física que estudia las leyes y los fenómenos de la luz” (Real Academia Española, s. f.). En este proyecto es importante entender ciertos conceptos sobre esta puesto que uno de sus objetivos es la visión por computador y con ello la utilización de las cámaras, las cuales se basan en el aprovechamiento de ciertas características de la óptica para poder obtener las imágenes.

Este apartado se centrará en los conceptos más aplicados en las cámaras, mientras que en los apartados siguientes se hará más hincapié en las teorías y modelos utilizados. Así pues, en la utilización de las cámaras existen tres conceptos muy necesarios de entender: las lentes, la distancia focal y el tamaño del sensor.

3.4.1. Lentes

Una de las formas más sencillas de cámara es la conocida como **cámara estenopeica** (**Figura 15**). En este se describe la cámara como una caja opaca en la que se le ha perforado un agujero a través del cual solo puede pasar teóricamente un rayo de luz. Para ello el agujero ha de ser realmente pequeño, del orden de 0.4 mm lo que hace que la imagen sea muy oscura y requiera de mucho **tiempo de exposición**, lo que la hace inútil para objetivos móviles, además de producir algunos desperfectos en la imagen como el **ennegrecimiento de los bordes** (De Blois, s. f.).

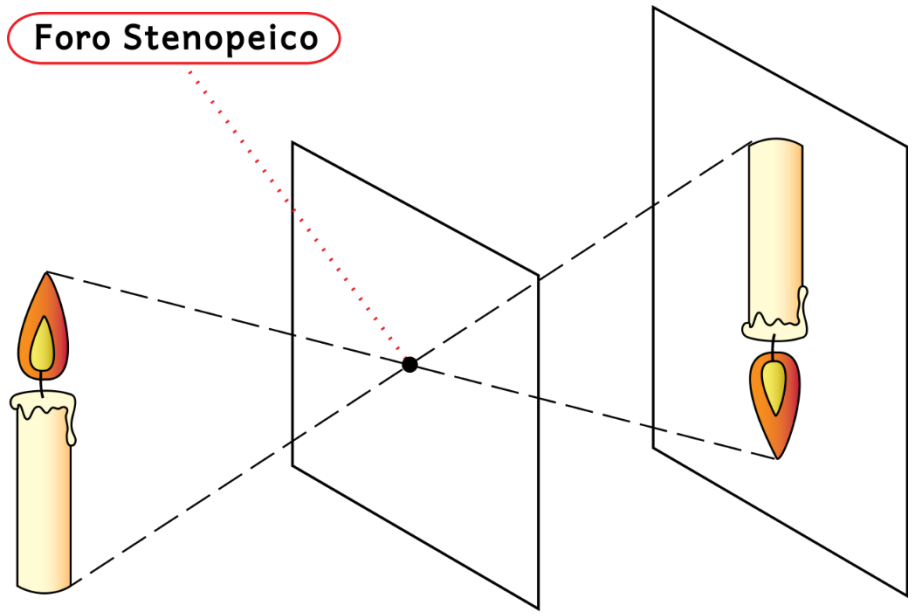


Figura 15. Ejemplo esquemático del funcionamiento de una cámara estenopeica.

FUENTE: HTTPS://COMMONS.WIKIMEDIA.ORG/WIKI/CATEGORY:PINHOLE_PHOTOGRAPHY

Para mejorar la adquisición de imágenes se crearon las cámaras con lentes. Estas cámaras disponen de unas lentes entre el objeto a fotografiar y el sensor de la cámara, las cuales concentran la luz permitiendo una apertura focal mayor, lo que produce una fotografía más luminosa, y por lo

tanto un tiempo de exposición considerablemente menor. En la **Figura 16** se muestra el efecto de una lente convexa simple.

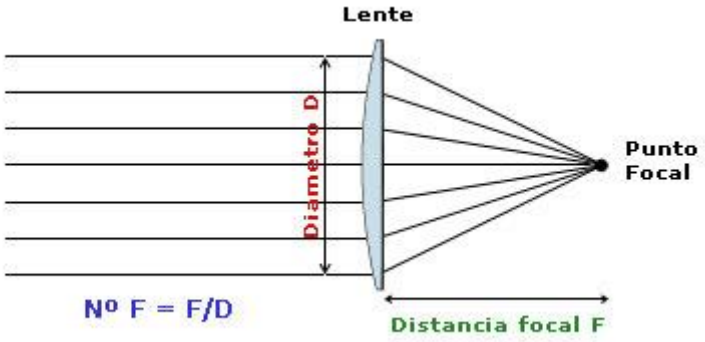


Figura 16. Esquema de una lente simple.

FUENTE: HTTPS://WWW.DECAMARAS.COM/CMS/CONTENT/VIEW/861/61-TIPOS-DE-OBJETIVOS-FOTOGRAFICOS-GUIA-TEORICA-Y-PRACTICA

Se profundizará más en las cámaras estenopeicas y sobre el efecto de las lentes en las imágenes más adelante.

3.4.2. Distancia focal

La distancia focal (**Figura 17**) es una característica fundamental de los objetivos de las cámaras, es decir, de las lentes. Esta representa la distancia en **milímetros** desde la lente hasta el **plano focal**, donde convergen los rayos de luz que traspasan la lente.

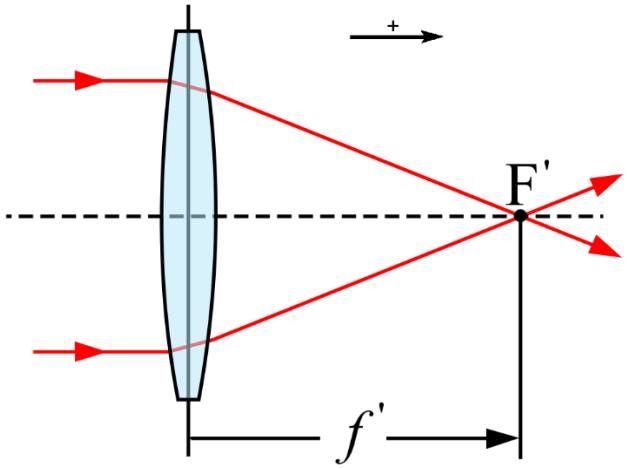


Figura 17. Representación esquemática de la distancia focal.

FUENTE: HTTPS://BYNAU.COM/QUE-ES-LA-DISTANCIA-FOCAL/

Así pues, este parámetro es uno de los más importantes a la hora de elegir un objetivo. La distancia focal es la encargada, junto con el tamaño del sensor, de asignar un ángulo de visión a la fotografía, es decir, la que cambia el *zum* en la cámara. Cuanto menor es la distancia focal de la lente, mayor será el ángulo de visión y el objeto se vea más pequeño (Bynau, 2021).

A continuación, en la **Figura 18**, se muestran unos ejemplos de una misma fotografía con distintas distancias focales, mientras que en la **Figura 19** se muestra el efecto que tiene la distancia focal en el ángulo de visión de una cámara con un determinado sensor.

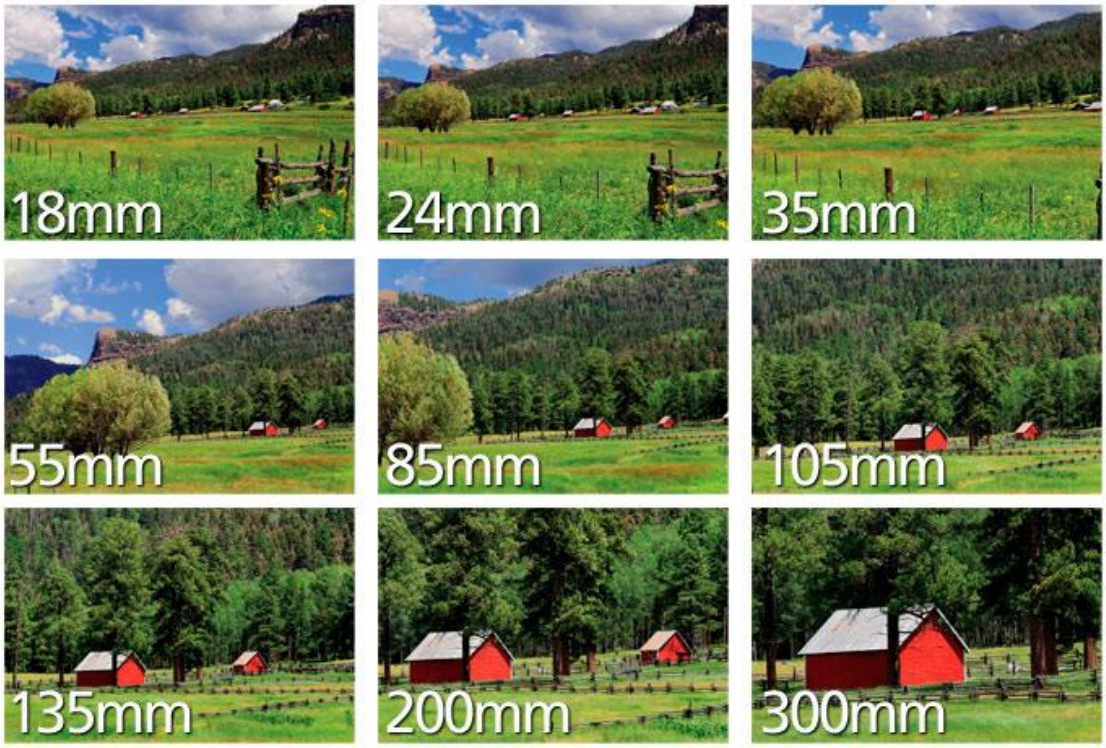


Figura 18. Diferencias entre distintas distancias focales.

FUENTE: HTTPS://WWW.NIKON.COM.MX/LEARN-AND-EXPLORE/A/TIPS-AND-TECHNIQUES/ENTENDIENDO-LA-DISTANCIA-FOCAL.HTML

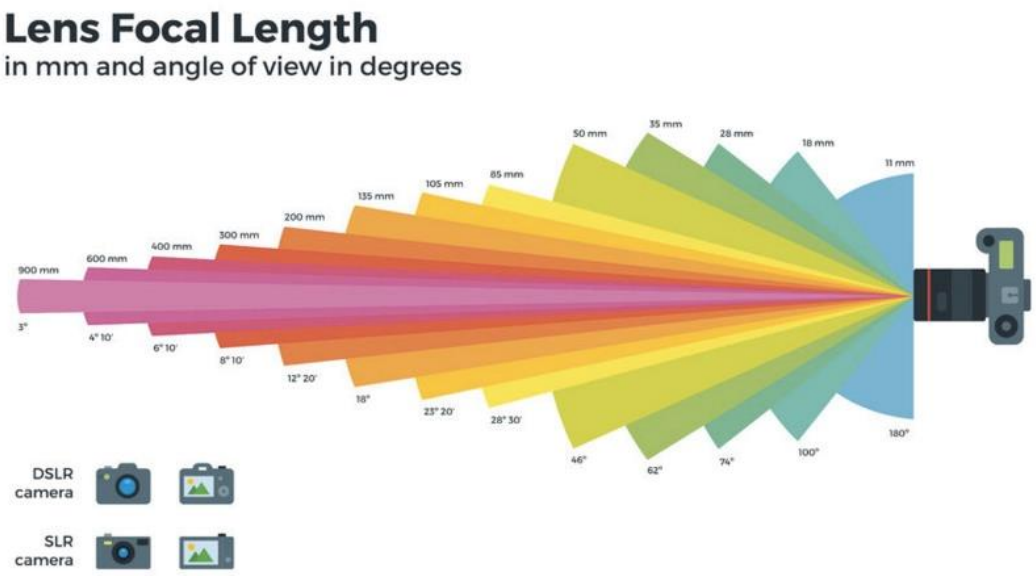


Figura 19. Ángulo de visión según la distancia focal.

FUENTE: ELBLOGDELAFOTOGRAFIA.COM/DISTANCIA-FOCAL/

3.4.3. Tamaño del sensor

El tamaño del sensor también es un parámetro de gran importancia en la fotografía puesto que este también afecta al ángulo de visión captado y altera lo que se conoce como *crop factor* o **factor de recorte.** Este no es más que un número que indica la relación entre el sensor utilizado y un sensor *full frame* (sensor de 35mm) (El blog de la fotografía, s. f.). Este número afecta al ángulo de visión puesto que multiplica la distancia focal de la lente, es decir, un sensor con factor de recorte de 1,8 utilizando una lente de focal 50 mm dará un resultado igual al de un sensor *full frame* de $50 * 1.8 = 90\ mm$.

En la **Figura 20**, se muestra una imagen representativa de algunos de los distintos tamaños de sensores que se pueden encontrar y en la **Figura 21** se muestra el efecto del factor de recorte en una imagen concreta.

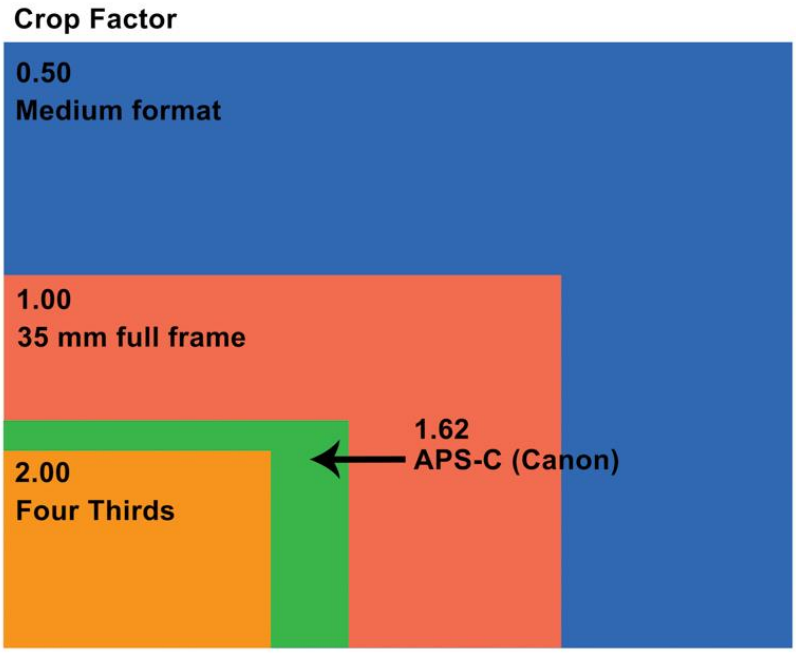


Figura 20. Tamaños de sensores y factor de recorte.

FUENTE: HTTPS://WWW.LIFEPIXEL.COM/PHOTO-TUTORIALS/FULL-FRAME-VS-CROP-SENSORS-WHAT-ARE-THEY-AND-WHICH-SHOULD-YOU-USE

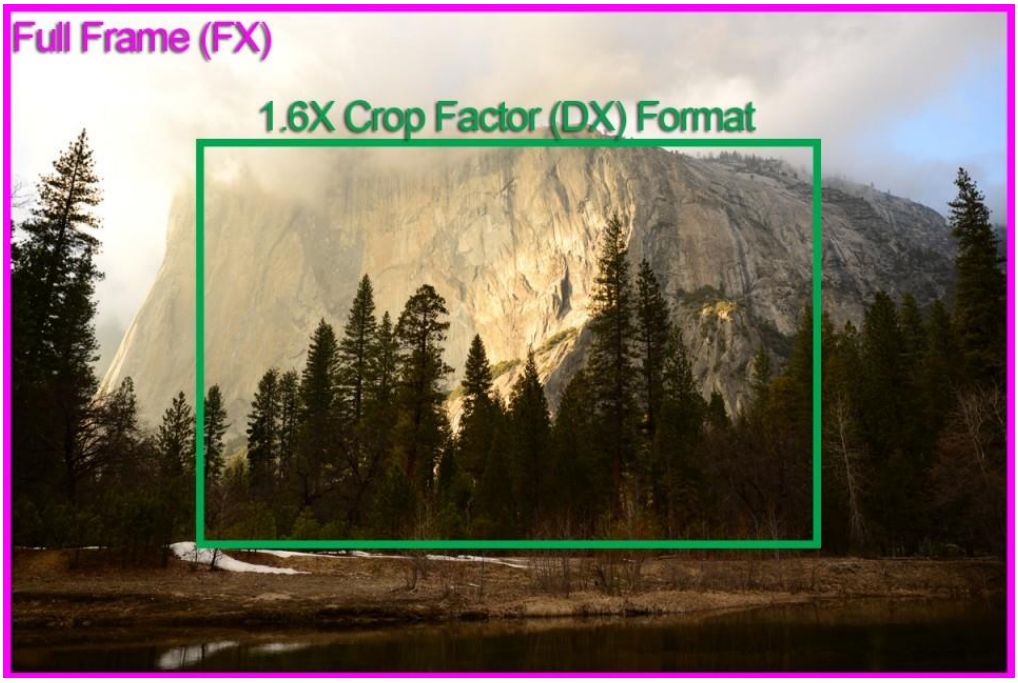


Figura 21. Efecto del factor de recorte en una imagen.

FUENTE: HTTPS://WWW.THE-PHOTOGRAPHY-BLOGGER.COM/FULL-FRAME-VS-CROP-SENSOR-CAMERA-ALL-YOU-NEED-TO-KNOW/

3.5. Modelo Pin-Hole

Como se ha mencionado antes, una cámara *Pin Hole* o cámara estenopeica es una cámara sin lentes y por lo tanto una de las cámaras más fáciles de modelar.

En este proyecto se ha utilizado el modelo Pin-Hole a pesar de utilizar una cámara con lentes puesto que se ha aplicado sobre ella una calibración de la que se hablará más tarde. Esto es así puesto que las lentes en una cámara pueden producir ciertas deformaciones en la imagen.

Con el modelo Pin-Hole, los parámetros que relacionan la cámara con el mundo 3D se ven representados por una matriz conocida como *camera matrix* la cual proporciona información para transformar los puntos 3D del mundo a puntos 2D de la imagen. Esta matriz está compuesta a su vez por dos matrices, la **matriz extrínseca** y la **matriz intrínseca**. La matriz extrínseca representa la localización de la cámara con respecto del objeto en el mundo 3D mientras que la matriz intrínseca representa los parámetros intrínsecos de la cámara (centro óptico y distancia focal) necesarios para la representación de la imagen dentro de esta, básicamente permite establecer la relación entre las coordenadas en pixeles de un punto en la imagen y sus coordenadas en el **plano imagen** (Mathworks, s. f.).

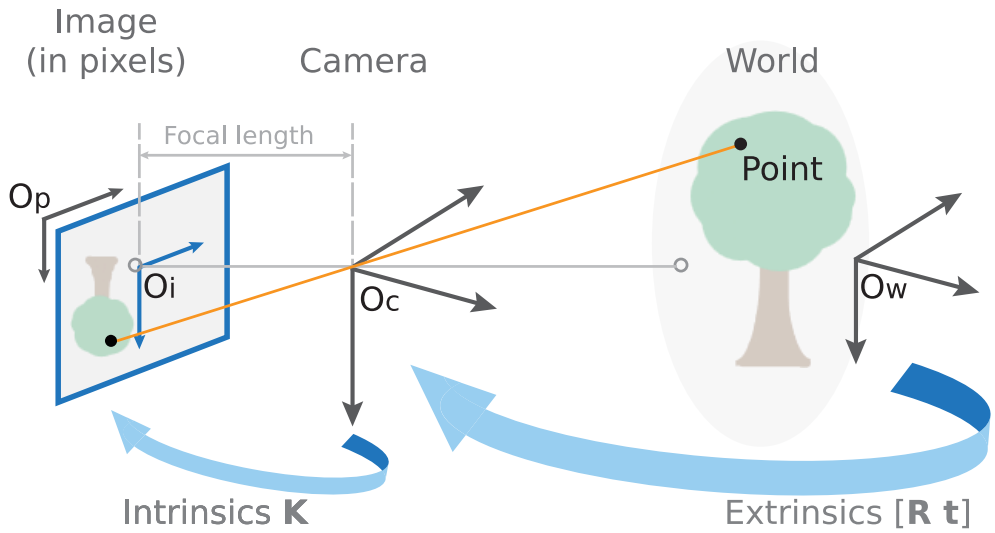


Figura 22. Representación de la matriz extrínseca e intrínseca.

FUENTE: HTTPS://ES.MATHWORKS.COM/HELP/VISION/UG/CAMERA-CALIBRATION.HTML

En la **Figura 22** se muestra la función de la matriz intrínseca “K” y la extrínseca “[R t]”. Así pues, la matriz de la cámara será $P = K \cdot [R\ t]$.

Cabe destacar que, así como la matriz extrínseca dependerá de la posición de la cámara en la escena (es una matriz de transformación), esta cambiará según la fotografía que se tome; mientras que la matriz intrínseca solo depende de la constitución de la cámara y, por lo tanto, una vez se obtenga, esta será siempre la misma.

3.6. Calibración de cámaras

Para poder utilizar el modelo Pin-Hole para traducir las coordenadas entre puntos del mundo real y el plano imagen es necesario realizar una calibración previa puesto que es necesario obtener la matriz intrínseca y conocer las distorsiones producidas por las lentes de las cámaras.

3.6.1. Obtención de la matriz de la cámara y sus vectores de distorsión

Para calibrar la cámara se ha optado por utilizar el **método de Zhang** el cual se basa en utilizar algún tipo de **patrón de calibración** plano (**Figura 24**), fácil de reconocer para obtener la pose de la cámara en varias fotografías en la que está ha sido alterada (**Figura 23**). Se deben de utilizar como mínimo tres imágenes con una posición y orientación diferente para considerar la calibración completa. El método busca en las distintas imágenes el patrón de calibración y a partir de ciertos puntos característicos obtiene la pose de la cámara con respecto del patrón en la imagen que esté tratando imagen en el momento, es decir obtiene la relación entre los puntos en el mundo (3D) y los puntos en la imagen (2D). Más tarde a partir de la relación entre los puntos de todas las imágenes se obtiene la matriz intrínseca y el vector de distorsión, el cual define las distintas distorsiones que tiene la imagen (Zhang, 2008).

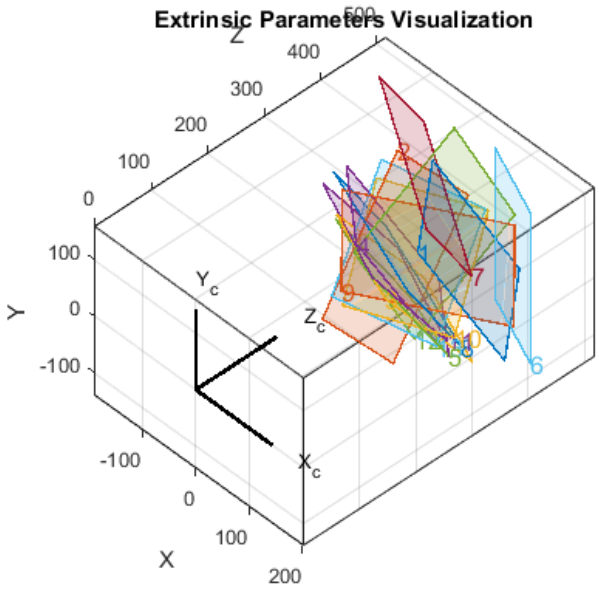


Figura 23. Representación de la pose de la cámara con respecto del patrón de calibración en distintas imágenes.

FUENTE: HTTPS://AISHACK.IN/TUTORIALS/CALIBRATING-UNDISTORTING-OPENCV-OH-YEAH/

Figura 24. Ejemplo de patrón de calibración y la detección de sus puntos característicos (esquinas).

FUENTE: HTTPS://STACKOVERFLOW.COM/QUESTIONS/39530110/CAMERA-CALIBRATION-FOR-STRUCTURE-FROM-MOTION-WITH-OPENCV-PYTHON

3.6.2. Distorsiones

Como se ha dicho anteriormente, las cámaras con lentes pueden presentar algunas distorsiones. Dentro de estas distorsiones se pueden diferenciar dos tipos: la **distorsión radial** y la **distorsión tangencial**.

La distorsión radial (**Figura 25**) es aquella producida por las lentes y es muy visible en las cámaras conocidas como “ojo de pez”. Esta distorsión puede ser negativa (**distorsión de barril**) o positiva (**distorsión pincushion**), y es más notable cuanto más lejos esté del centro de la imagen.

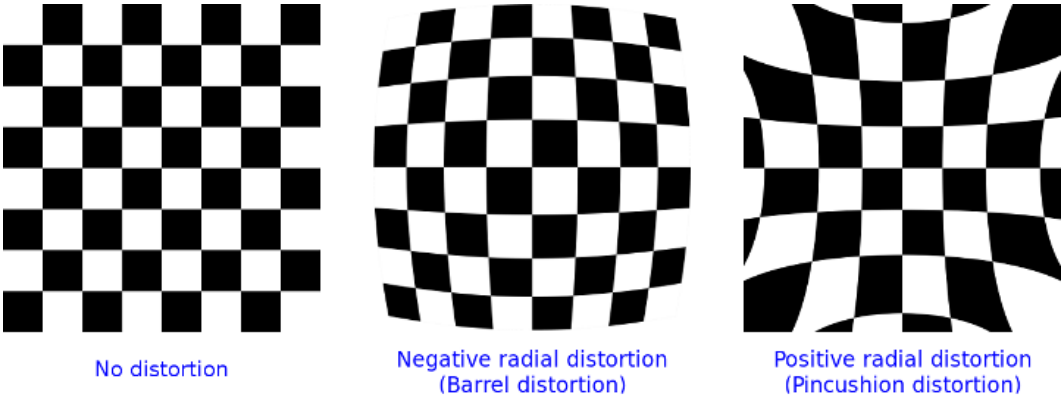


Figura 25. Efectos de la distorsión radial.

FUENTE: HTTPS://DOCS.OPENCV.ORG/4.X/D9/D0C/GROUP__CALIB3D.HTML

Por otro lado, la distorsión tangencial (**Figura 26**) es producida por la inclinación del sensor de la cámara con respecto a las lentes de la cámara.

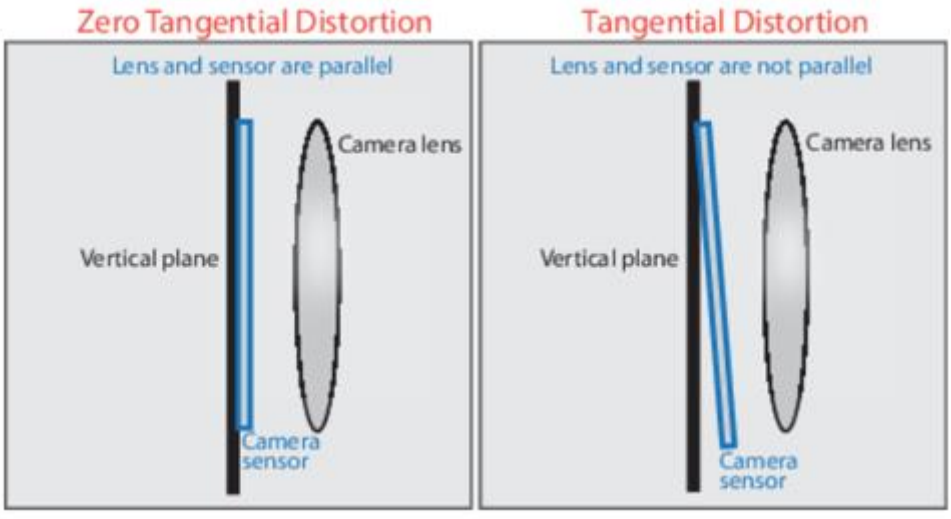


Figura 26. Distorsión tangencial.

FUENTE: HTTPS://ES.MATHWORKS.COM/HELP/VISIONHDL/UG/IMAGE-UNDISTORT.HTML

Estas distorsiones se pueden representar con unos coeficientes (OpenCV, s. f.).

En el caso de la distorsión radial:

$$x_{distorted} = x(1 + k_1 * r^2 + k_2 * r^4 + k_3 * r^6)$$

$$y_{distorted} = y(1 + k_1 * r^2 + k_2 * r^4 + k_3 * r^6)$$

x, y: pixeles sin distorsión

k_1, k_2, k_3: Coeficientes de distorsión

$r^2 = x^2 + y^2$

Y en el caso de la distorsión tangencial:

$$x_{distorted} = x + [2 * p_1 * x * y + p_2 * (r^2 + 2 * x^2)]$$

$$y_{distorted} = y + [p_1 * (r^2 + 2 * y^2) + 2 * p_2 * x * y]$$

x, y: pixeles sin distorsión

p_1, p_2: Coeficientes de distorsión

$r^2 = x^2 + y^2$

3.7. Optical Flow

El Optical Flow o **Flujo Óptico** es una técnica que mediante la comparativa de dos frames (fotogramas) consecutivos en el tiempo es capaz de distinguir ciertos puntos característicos de la imagen que se hayan visto **desplazados en el tiempo**, lo que significa un movimiento del objeto en la imagen o un movimiento de la cámara (OpenCV, s. f.). Una ejemplificación de esta herramienta se muestra en la **Figura 27**.

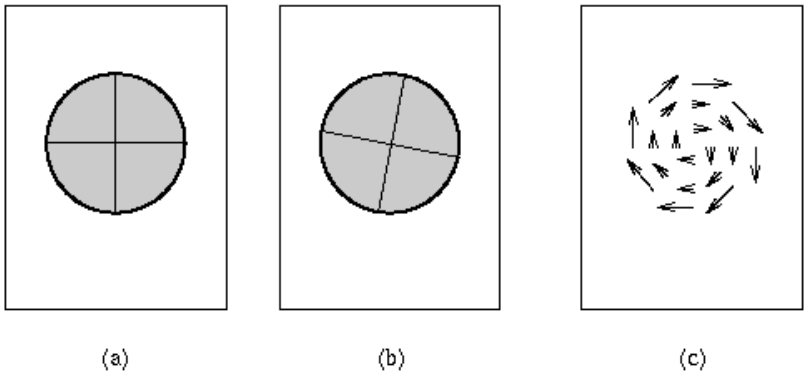


Figura 27. Flujo óptico de una rotación. a) t1; b) t2; c) Optical Flow (t1, t2).

FUENTE: HTTPS://USER.ENGINEERING.UIOWA.EDU/~DIP/LECTURE/MOTION2.HTML

A partir del análisis del flujo óptico, en la robótica se pueden realizar aplicaciones de gran interés como detección de movimiento o la **estimación del movimiento del robot**.

4. Hardware

En este punto se pretende exponer las diferentes herramientas de hardware que se han utilizado en la realización del trabajo. Para la elección de estos componentes se han pensado los diferentes dispositivos que se necesitan para esta aplicación. Para completar el proyecto es necesario un **robot aéreo**, o un dispositivo que se pueda incorporar a este, que transmita las imágenes y los datos de pose de la aeronave, y la **estación tierra**, compuesta por un dispositivo de **control de la aeronave** y uno de **procesamiento de las imágenes** captadas (aunque pueden ser el mismo dispositivo). Además, aunque no sea hardware propiamente dicho, es necesario un objetivo con alguna característica fácilmente diferenciable para la cámara.

4.1. Dron DJI Tello

Para la aeronave y el sistema de detección se ha decidido apostar por la utilización del dron comercial de las marcas DJI y Ryze conocido como Tello (**Figura 28**). Este es un dron de software abierto, fácil de programar y que tiene su propia librería para facilitar su programación. Además, es un dron barato (poco más de 100€) pero fiable puesto que dispone también de sistemas de estabilización bastante fiables, aunque limitados. El dron utiliza una cámara ubicada en la parte inferior de la aeronave junto con un sensor de altura identificar su desplazamiento con un sistema de posicionamiento visual (Optical Flow) y así poder corregir los movimientos indeseados (turbulencias). Esta estabilización se ve limitada por el peso ligero del dron (87 g) y la velocidad máxima no muy elevada que tiene (14,4 km/h en modo lento y 28,8km/h en modo rápido), haciéndolo fácilmente alterable ante ráfagas de viento de aproximadamente 15 km/h o más. Su autonomía con una batería totalmente cargada es de aproximadamente 13 minutos de duración, más que suficiente para realizar las pruebas necesarias para este proyecto. En definitiva, el DJI Tello es un dron barato, sencillo de manipular y perfecto para esta aplicación, aunque se queda muy corto para una aplicación profesional.

Figura 28. Dron Tello volando.

FUENTE: HTTPS://ELDUENDEMALL.COM/BLOG/REVIEW-DE-DRONE-DJI-TELLO/

4.1.1. Comunicación

El Tello basa su comunicación a partir de una **señal WiFi** (Wi-Fi 2.4) que crea a la cual se puede conectar fácilmente la estación tierra para empezar la comunicación. La señal es capaz de extenderse hasta los 100 metros de distancia (30 metros de altura), aunque no es recomendable a partir de 15 metros aproximadamente puesto que al alejarse de esta distancia la transmisión de imágenes a la estación tierra puede verse mermada en calidad y fotogramas por segundo (fps). Aunque el dron no tenga un sistema de seguridad ante perdida de la conexión con el control remoto (si pierde la conexión se queda volando hasta que se quede sin batería), sí lo tiene ante valores críticos de batería, es decir, cuando detecta una batería crítica, aterriza de manera automática suavemente.

Puesto que la conexión WiFi que genera la aeronave no restringe el número de dispositivos conectados a uno, se puede realizar una conexión con un dispositivo de control y un dispositivo de procesamiento de imágenes de manera separada, lo que facilita el control de la aeronave al permitir utilizar la aplicación Tello de la que se hablará más adelante exclusivamente para el control. En la **Figura 29** se muestra un esquema de las conexiones entre la aeronave y los dispositivos de la estación tierra.

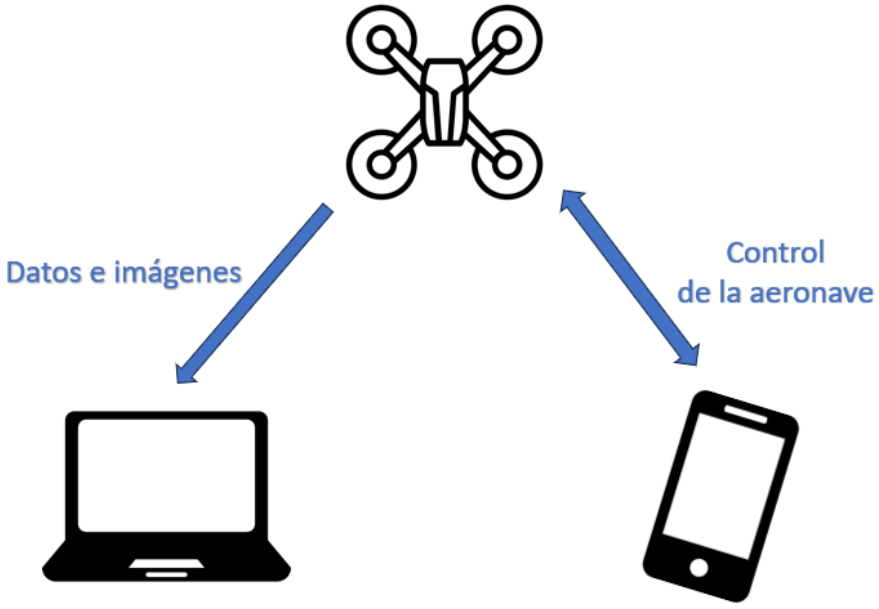


Figura 29. Esquema de comunicaciones.

FUENTE: ELABORACIÓN PROPIA

4.1.2. Sensores

El dron de DJI tiene una serie de sensores que servirán de gran ayuda para la realización del experimento. A continuación, se muestra una lista de estos:

- IMU con acelerómetro y giróscopo triaxial además de un sensor barométrico.
- Telémetro de infrarrojos en la parte inferior de la aeronave para ayudar en la obtención de la altura.
- Cámara en el inferior de la aeronave para el posicionamiento visual. Este se ve comprometido en situaciones de poca luz o sobre superficies reflectantes o cambiantes como superficies de agua.
- Cámara principal fija de 5 Megapíxeles (2592x1936) con una calidad de video máxima de 1280x720p y un ángulo de visión de 82.6 grados.

4.2. Estación tierra

La estación tierra, como se ha visto anteriormente estará compuesta por dos dispositivos: el sistema de procesamiento de datos y el sistema de control de la aeronave.

4.2.1. Sistema de procesamiento de datos

El sistema de procesamiento de datos estará compuesto por un ordenador con capacidad de conexión WiFi, un sistema operativo y un entorno visual. En el caso de este proyecto se ha utilizado un ordenador de la marca HP de las series 15s-fq1xxx. Este se encargará de recibir las imágenes y la información sobre la pose de la aeronave y realizar los cálculos necesarios para estimar la posición del objetivo, es decir, el procesamiento de imágenes se realizará fuera de la aeronave (off board) en tiempo real lo que permitirá que la aeronave se encargue del control de bajo nivel sin tener que sobrecargar su memoria, pero el buen procesamiento de imágenes dependerá mucho de la estabilidad de la conexión entre aeronave y estación tierra.

4.2.2. Sistema de control de la aeronave

El sistema de control de la aeronave lo formara un teléfono móvil, en este caso un Redmi Note 10S de la marca Xiaomi, con la aplicación “Tello” instalada, aplicación diseñada específicamente para esta aeronave. Este dispositivo se encargará únicamente de dar las ordenes de control a la aeronave (despegue, aterrizaje, posicionamiento, etc).

Los dispositivos de estación tierra son importantes puesto que dispositivos con un receptor Wi-Fi anticuado o con poca memoria pueden afectar al rendimiento del vuelo. Además, la versión de los sistemas operativo de ambos dispositivos debe ser compatible con el software utilizado.

4.3. Objetivo a localizar

El objetivo a localizar deberá ser un objeto de unas dimensiones lo suficientemente grandes como para ser localizado con facilidad por la cámara y con alguna característica lo suficientemente diferenciable. En este caso la

separación entre objeto y fondo se realizará a partir de un **filtrado por color** y área, es decir, se considerará objetivo aquello de un color determinado y de mayor tamaño en la imagen.

5. Software

El software es un componente **intangible** que forma parte de los dispositivos informáticos y que está compuesto por programas y aplicaciones diseñados para cumplir objetivos específicos. En este apartado se pretende dar una explicación de los distintos programas y otros componentes de software que se han utilizado en la realización del proyecto.

5.1. Aplicación Tello

Para poder realizar el control de la aeronave se ha utilizado la aplicación para teléfonos móviles “Tello”, una aplicación creada por los mismos desarrolladores del dron (Shenzhen RYZE Tech) que cuenta con todo aquello necesario para controlar la aeronave de manera muy fiable. Para utilizar la aplicación debe encenderse la aeronave y esperar hasta que el LED incorporado en la parte frontal parpadee en amarillo. Entonces se debe acceder al menú WiFi del teléfono y buscar la red llamada TELLO-XXXXXX. Una vez realizados estos pasos la aplicación puede ser ejecutada y ya se podrá acceder al control pleno de la aeronave. Es importante que cuando se esté conectado a la zona WiFi del Tello se tengan desconectadas las demás redes móviles del teléfono, sino es muy probable que la aplicación no funcione.

El menú de la aplicación (**Figura 30**) es muy sencillo e intuitivo, si se tienen dudas al respecto se puede consultar el manual de usuario de la aeronave.

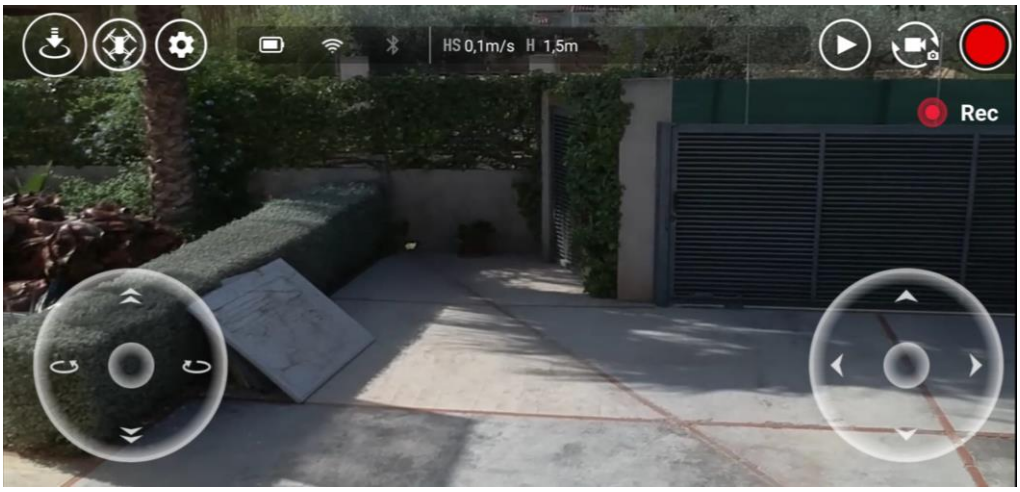


Figura 30. Menú principal de la aplicación.

FUENTE: ELABORACIÓN PROPIA

5.2. Python

El lenguaje de programación elegido para desarrollar el proyecto es **Python,** un lenguaje de programación de alto nivel diseñado por Guido van Rossum. Se trata de un lenguaje **interpretado** que basa su filosofía en buscar la mayor claridad, legibilidad y sencillez posible. Es un lenguaje muy práctico, con muchas herramientas y muy utilizado en la visión por computador, aunque es un lenguaje **multiparadigma**, es decir, que permite ser utilizado para multitud de aplicaciones y estilos de programación (Python, s. f.). Para instalar Python en Windows se debe acceder a la página oficial de descargas y descargar la última versión: https://www.python.org/downloads/. En el momento en el que se realiza este proyecto, la última versión es la 3.11 (**Figura 31**).

Figura 31. Apartado de descarga en la web de Pyhton.

FUENTE: ELABORACIÓN PROPIA

Al abrir el archivo instalado se ejecutará la guía de instalación de Python. Es importante que se habilite la opción *Add Python X.X to PATH* situado en la parte inferior de la ventana. Además, es importante acceder al apartado *Customize installation* para comprobar que se está instalando también la herramienta **Pip** (**Figura 32**).

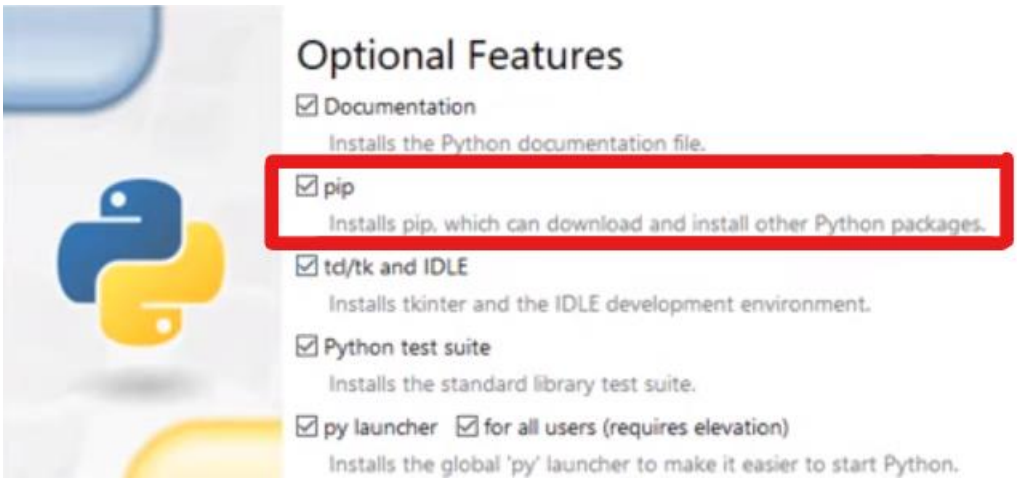


Figura 32. Apartado Customize installation.

FUENTE: ELABORACIÓN PROPIA

Pip es una herramienta muy útil para instalar algunos paquetes externos de Python. Más adelante se verán algunos ejemplos de su uso.

Una vez se haga clic en el botón de siguiente se pasará a la ventana de **opciones avanzadas**. Estas opciones se dejarán por defecto y se procederá a la instalación, pulsando el botón *install.*

Para comprobar la correcta instalación en Windows se deberá acceder al símbolo del sistema (cmd) y teclear *python –versión.* Si devuelve la versión de Python que se ha instalado significa que se ha completado correctamente la instalación (**Figura 33**). El sistema operativo utilizado en el proyecto es **Windows 11**, en el caso de utilizar un sistema operativo es posible que sea necesario reiniciar para poder realizar este último paso.

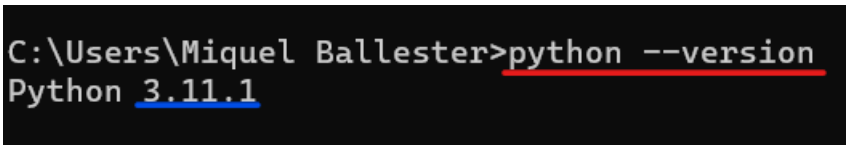


Figura 33. Comprobación de la versión de Python instalada (3.11.1).

FUENTE: ELABORACIÓN PROPIA

Para comprobar el correcto funcionamiento se puede realizar una prueba sencilla. Para ello, primero se debe entrar en el entorno de Python escribiendo en el símbolo del sistema *python.* Para saber si se ha entrado en el entorno de Python solo hay que fijarse que hay los tres símbolos de "mayor que" delante del cursor característicos de Python (>>>). Luego se puede utilizar la función print("Hola Mundo") y comprobar el retorno. Esta prueba se puede observar en la **Figura 34**.

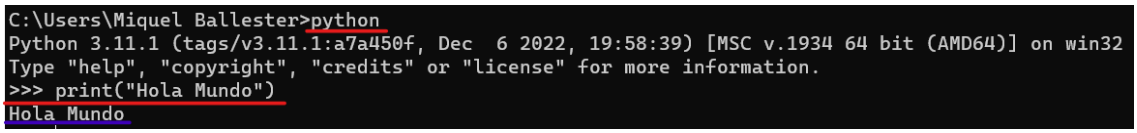


Figura 34. Comprobación del funcionamiento de Python.

FUENTE: ELABORACIÓN PROPIA

Para salir del entorno de Python se debe usar el comando exit().

5.2.1. Biblioteca estándar de Python

Dentro de las librerías utilizadas en el proyecto se encuentran algunas que están incluidas con la instalación de Python (Python, s. f.). Estas son:

- **Glob**. Una librería muy útil para acceder a algunos archivos a través de rutas al estilo **Unix**. En el proyecto servirá para acceder a una serie de imágenes obtenidas previamente para poder llevar a cabo la calibración.
- **Pickle.** Esta es una librería para **serializar y deserializar** estructuras en Python. En este proyecto será útil para guardar y recuperar datos entre programas.

- **Math.** Este módulo contiene las **funciones matemáticas** estándar de C. En el proyector servirá para utilizar funciones trigonométricas y definiciones de manera sencilla.
- **Time.** Esta librería incluye funciones relacionadas con el tiempo. En el proyecto será de utilidad para realizar **temporalizaciones** de bucles.

5.2.2. Librerías a instalar

Además de las librerías preinstaladas mencionadas, el proyecto va a requerir de otras librerías que requieren de una instalación. Para instalar estas librerías se va a hacer uso de la herramienta Pip anteriormente mencionada.

5.2.2.1. Librería Numpy

Numpy (**Figura 35**) es una librería dedicada a la utilización de arrays de n dimensiones y a la implementación de herramientas numéricas (Numpy, s. f.). En este proyecto se utilizará sobre todo para la implementación de matrices y vectores en el código.

Figura 35. Logotipo de la librería numpy.

FUENTE: HTTPS://NUMPY.ORG/

Para proceder a la instalación de la librería a través de pip tan solo se debe acceder al símbolo del sistema y escribir la siguiente línea: `pip install numpy`. La librería podrá ser importada en cualquier programa de Python inmediatamente después de que termine la instalación.

5.2.2.2. Librería Sympy

Sympy (**Figura 36**) es una librería para Python destinada al cálculo y las **matemáticas simbólicas** (SymPy, s. f.). Esta es de gran utilidad para obtener ecuaciones simbólicas que posteriormente se resolverán. En el proyecto se utiliza esta librería para desarrollar las ecuaciones de Pin-Hole junto con las matrices de transformación pertinentes para más tarde resolver la posición del objetivo.

Figura 36. Logotipo de la librería Sympy.

FUENTE: HTTPS://WWW.SYMPY.ORG/EN/INDEX.HTML

La instalación se realizará igual que la anterior librería, mediante pip escribiendo la siguiente línea en el cmd: `pip install -U sympy`.

5.2.2.3. Librería Matplotlib

La librería Matplotlib (**Figura 37**) está destinada a crear **visualizaciones y graficas** para Python (Matplotlib, s. f.). En el proyecto servirá para hacer un gráfico que recree las coordenadas globales y la detección del objeto en estas.

Figura 37. Logotipo Matplotlib.

FUENTE: HTTPS://MATPLOTLIB.ORG/2.1.0/API/_AS_GEN/MATPLOTLIB.PYPLOT.FIGURE.HTML

Para la instalación de la librería se debe usar la siguiente línea de código en el cmd: `python -m pip install -U matplotlib`.

5.2.2.4. Librería OpenCV

OpenCV (**Figura 38**) es una de las librerías más conocidas y probablemente la más utilizada para el análisis de imágenes. La librería se encarga de brindar herramientas optimizadas relacionadas con la **visión por computador** en tiempo real (OpenCv, s. f.). En el proyecto esta librería está enfocada a la identificación del objetivo, de su área y de sus coordenadas en pixeles. Estas coordenadas serán fundamentales para la obtención de las coordenadas reales en 3D.

Figura 38. Logotipo OpenCV.

FUENTE: HTTPS://FORUM.OPENCV.ORG/

Para proceder a la instalación de la librería con Pip se debe introducir esta línea en el símbolo del sistema: `pip install opencv-python`.

5.2.2.5. Librería DjiTelloPy

La librería DjiTelloPy es la librería oficial creada específicamente para la **programación y el control** del dron Tello, del que se ha hablado anteriormente, con Python. En el proyecto esta servirá para recibir las imágenes y los parámetros de pose del dron que posteriormente serán procesados. En la **Figura 39,** se puede observar el logotipo de DJI.

Figura 39. Logotipo de Dji.

FUENTE: HTTPS://WWW.DJI.COM/ES

Para la instalación de la librería se debe escribir la siguiente línea en el cmd: `pip install djitellopy`.

5.3. Visual Studio Code

En este proyecto se ha realizado la programación con Python utilizando el editor conocido como **Visual Studio Code** (**Figura 40**). Este es un editor gratuito y de código abierto compatible con varios lenguajes de programación y que dispone de múltiples herramientas para facilitar la visualización y la escritura del código (Visual Studio Code, s. f.).

Figura 40. Logotipo Visual Studio Code.

FUENTE: HTTPS://CODE.VISUALSTUDIO.COM/BRAND

Para descargar este programa solo hay que ir a su página oficial, https://code.visualstudio.com/, y darle al botón que pone *Download for Windows* en el caso de tener este sistema operativo instalado. Una vez instalado solo hay que ejecutar el archivo instalado y seguir los pasos.

Una vez dentro del programa, se recomienda instalar la extensión de Python la cual da soporte a este lenguaje de programación, mostrada en la **Figura 41**.

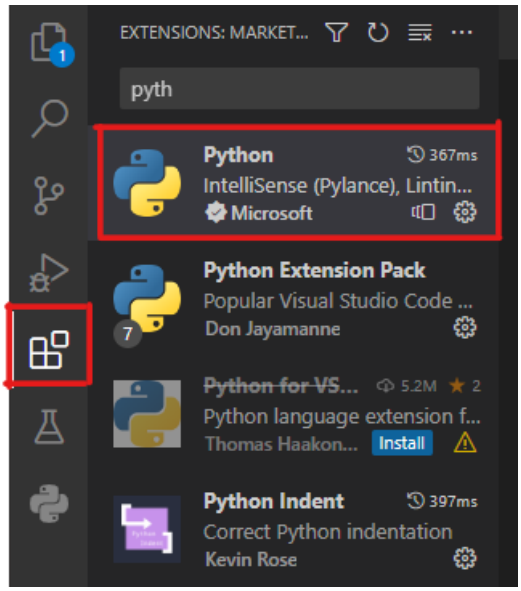


Figura 41. Extensión de Python en el menú de extensiones de Visual Studio Code.

FUENTE: ELABORACIÓN PROPIA

6. Detalles de la solución adoptada

En este punto se realizará una explicación detallada de las decisiones, cálculos y pasos realizados para diseñar el algoritmo y obtener los resultados que se discutirán más adelante. Además, se mencionarán diferentes alternativas que han sido contempladas en la realización del proyecto.

6.1. Descripción de la solución adoptada

El algoritmo ha sido diseñado y aplicado únicamente en el ordenador de la estación tierra encargado del procesamiento de datos e imágenes de la aeronave.

Así pues, el proceso completo desde que se obtiene la aeronave por primera vez hasta que se puede utilizar el algoritmo de localización pasa por dos códigos más breves previamente. Esto se ha hecho para que sea más fácil y organizado seguir el proyecto además de para **eliminar carga** del código principal. De esta forma, el proyecto al completo queda dividido en tres códigos: **GetImage.py, Calibration.py y Final.py**.

6.1.1. GetImage.py

El programa GetImage.py es un programa sencillo encargado de realizar y guardar fotografías a través de la cámara conectada. Este programa se ha utilizado para guardar las fotografías realizadas al patrón de calibración que posteriormente servirán para calibrar la cámara.

El código empieza con una pequeña inicialización de variables y se realiza la conexión con el dron:

```
5   from djitellopy import Tello
6   import cv2
7
8   tello = Tello()
9
10  # Conexión con el dron e inicialización de transmisión de imagenes
11  tello.connect()
12  tello.streamon()
13  num = 0 # Número de la primera foto
```

Código 1. Inicialización GetImage.py.

Como se puede observar en el **Código 1** las librerías utilizadas en este código son **djitellopy,** para establecer la conexión con la cámara del Tello, y **opencv** (cv2), para mostrar y guardar imágenes además de otras herramientas.

En este fragmento del código se inicializa la librería djitellopy mediante la línea `tello = Tello()`. Posteriormente se establece la conexión con la aeronave y se habilita la conexión con la cámara. Además, se inicializa la variable *num* a 0, variable la cual servirá para enumerar las imágenes guardadas.

```python
while True:
    frame = tello.get_frame_read().frame
    k = cv2.waitKey(5)  # Se le asigna a la variable k la tecla que se pulse

    # Si esta tecla es "esc" se romperá el bucle
    if k == 27:
        break

    # Si es "s" se guardará una imagen png del frame en una carpeta llamada "Imagenes"
    elif k == ord("s"):
        cv2.imwrite("Imagenes/img" + str(num) + ".png", frame)
        print("image saved!")
        num += 1

    cv2.imshow(
        "Img",
        frame,  # Muestra en la pantalla la imagen que está obteniendo en tiempo real.
    )

# Se liberán y destruyen todas las ventanas antes de terminar el código.
cv2.destroyAllWindows()
```

Código 2. Bucle principal del código GetImage.py.

En el bucle de **Código 2** la primera instrucción que se realiza es guardar en la variable *frame* el fotograma actual que capta cámara del dron mediante la instrucción `frame = tello.get_frame_read().frame` y posteriormente se lee si se está pulsando alguna tecla con `k = cv2.waitKey(5)` de forma que el valor de la tecla queda almacenado en la variable *k.*

A continuación, se comprueba el valor de la tecla de forma que si es el valor correspondiente con la tecla *escape* (27) se romperá el bucle y terminará el programa. Sin embargo, si no lo es y se pulsa la tecla *s,* se guardará la imagen en una carpeta llamada *Imágenes* (ubicada dentro del mismo directorio que el programa), con el nombre imgX.png, siendo X el número de la imagen que le corresponde, estando este almacenado en la variable *num*. Este guardado se realiza mediante la instrucción `cv2.imwrite(…)`. Además, se escribirá por pantalla *"image saved!"* utilizando la instrucción `print("image saved!")`.

Luego se muestra el fotograma en pantalla mediante la instrucción `cv2.imshow("Img",frame)` obteniendo una visualización en tiempo real de la imagen obtenida por la cámara.

Por último, al salir del bucle, se cierran todas las ventanas creadas con `cv2.destroyAllWindows()`.

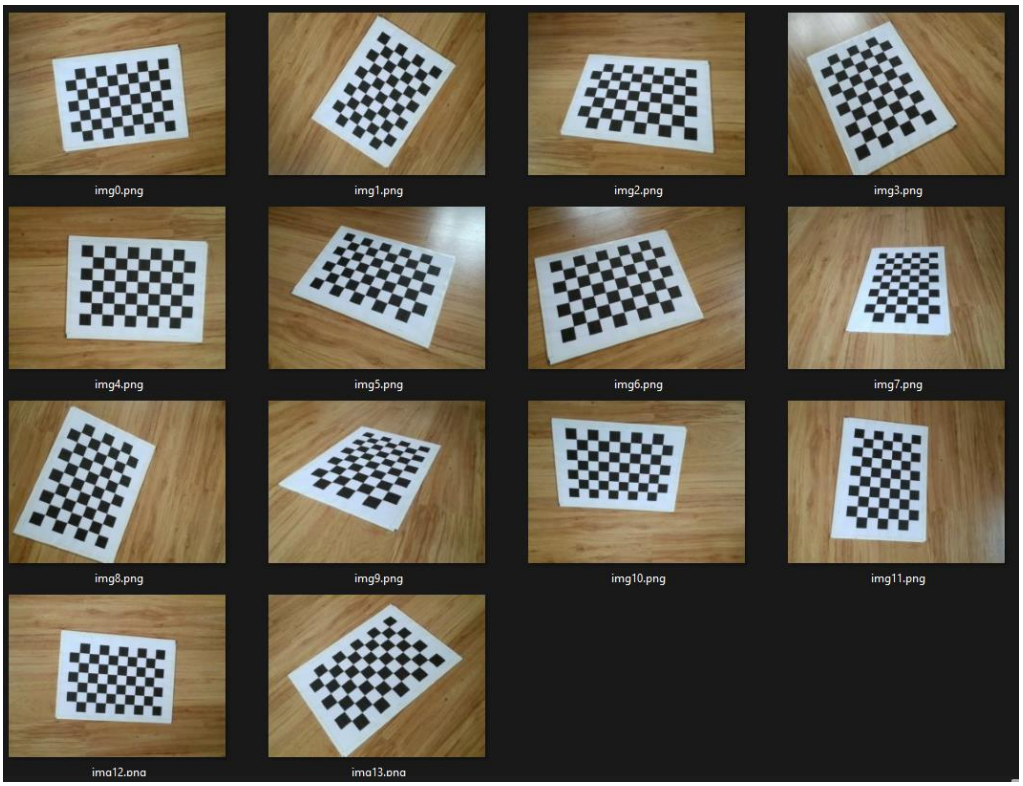


Figura 42. Imágenes obtenidas mediante GetImage.py.

FUENTE: ELABORACIÓN PROPIA

En la **Figura 42** se puede observar una captura del directorio *Imagenes* en el que se guardan las fotografías obtenidas del patrón de calibración desde distintos ángulos.

6.1.2. Calibration.py

El código Calibration.py se trata de un programa que recopila las imágenes anteriormente obtenidas y obtiene a partir de ellas la **matriz de la cámara** mediante la calibración explicada anteriormente. Este algoritmo de calibración basado en el método de Zhang ha sido adaptado a partir del código del video titulado *"Learn Camera Calibration in Python with OpenCV: Complete Step-by-Step Guide with Python Script"* (Nielsen, 2021).

En este código se han utilizado las siguientes librerías: **Numpy**, **OpenCV**, **Glob** y **Pickle**.

La primera parte del código, mostrado en **Código 3,** se muestra la inicialización de variables y vectores del código además de la lectura de las imágenes guardadas.

```
# OBTENCIÓN DE OBJECT POINTS Y IMAGE POINTS A PARTIR DEL TABLERO DE AJEDREZ

chessboardSize = (
    9,
    6,
)  # Tamaño del tablero de ajedrez, contando las esquinas comunes entre cuatro cuadrados
frameSize = (960, 720)  # Tamaño del frame

# termination criteria
criteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, 30, 0.001)

# Prepara la matriz de object points
objp = np.zeros((chessboardSize[0] * chessboardSize[1], 3), np.float32)
objp[:, :2] = np.mgrid[0 : chessboardSize[0], 0 : chessboardSize[1]].T.reshape(-1, 2)

size_of_chessboard_squares_mm = 25
objp = objp * size_of_chessboard_squares_mm

# Vector para almacenar los object points y los image points de todas las imagenes
objpoints = []  # Puntos 3D en el mundo real
imgpoints = []  # Puntos 2D en el plano imagen

# Almacena la ruta a todas las imagenes guardadas previamente en la carpeta Imagenes
images = glob.glob("Imagenes/*.png")
```

Código 3, Inicialización del código Calibration.py.

Las variables *chessboardSize y size_of_chessboard_squares_mm* contienen propiedades del patrón de calibración, siendo el primero el número de **puntos característicos** (cruces entre cuadrados) y el segundo el **tamaño de cada cuadrado**. La variable *frameSize* almacena el tamaño de los fotogramas en pixeles, en este caso 960x720 pixeles.

Por otro lado, *criteria* almacena el criterio que utilizará la función de refinamiento de esquinas, la cual se verá más adelante, para terminar la **iteración de refinamiento**.

En cuanto a la variable *objp*, esta almacena las **coordenadas** de los puntos característicos de la imagen en el **mundo real** respecto de un eje de coordenadas ubicado en el plano que contiene el patrón de calibración.

Por último, los vectores *objpoints* y *imgpoints* almacenarán las coordenadas en el mundo real y en la imagen de todas las fotografías, mientras que en *images* se recuperarán las fotografías guardadas en la carpeta *Imagenes.* Para ello se ha utilizado la función de la librería Glob `images = glob.glob("Imagenes/*.png")`, la cual recorre la carpeta cargando todos los archivos de extensión *.png*.

```
35  for (
36      image
37  ) in (
38      images
39  ):  # Analiza las imagenes contenidas en las rutas de la variable "images" una a una den
40      img = cv.imread(image)
41      gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)  # Convierte la imagen a escala de grises
42
43      # Busca las esquinas del tablero
44      ret, corners = cv.findChessboardCorners(gray, chessboardSize, None)
45
46      # Si las encuentra añade los object points correspondientes y los image points tras
47      if ret == True:
48          objpoints.append(objp)
49          corners2 = cv.cornerSubPix(gray, corners, (11, 11), (-1, -1), criteria)
50          imgpoints.append(corners2)
51
52          # Muestra las esquinas en una imagen
53          cv.drawChessboardCorners(img, chessboardSize, corners2, ret)
54          cv.imshow("img", img)
55          cv.waitKey(
56              1000
57          )  # Espera un segundo, esto solo sirve para dar tiempo a ver la imagen
58
59
60  cv.destroyAllWindows()
```

Código 4. Bucle para guardar los "object points" y los "image points" de cada imagen

En el bucle de **Código 4** se **analizarán** cada una de las imágenes cargadas. Para ello primeramente se convertirá el espacio de color de la imagen a **escala de grises** y se guardará en la variable *gray* mediante el comando de OpenCV `gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)`.

A continuación, se analizará el patrón de calibración buscando las esquinas entre cuadrados (puntos característicos) utilizando la función `cv.findChessboardCorners(…)`, de forma que se guardaran las coordenadas de las esquinas en la variable *corners* mientras que *ret* es una variable booleana que indica si se han encontrado, o no, esquinas en la imagen.

En el caso de haber encontrado esquinas se añadirán los puntos objeto al array *objpoints*. Cabe destacar que las coordenadas de los puntos objeto serán las mismas en todas las imágenes puesto que los ejes de coordenadas están contenidos y orientados según el patrón de calibración tal y como se ve en la **Figura 43**.

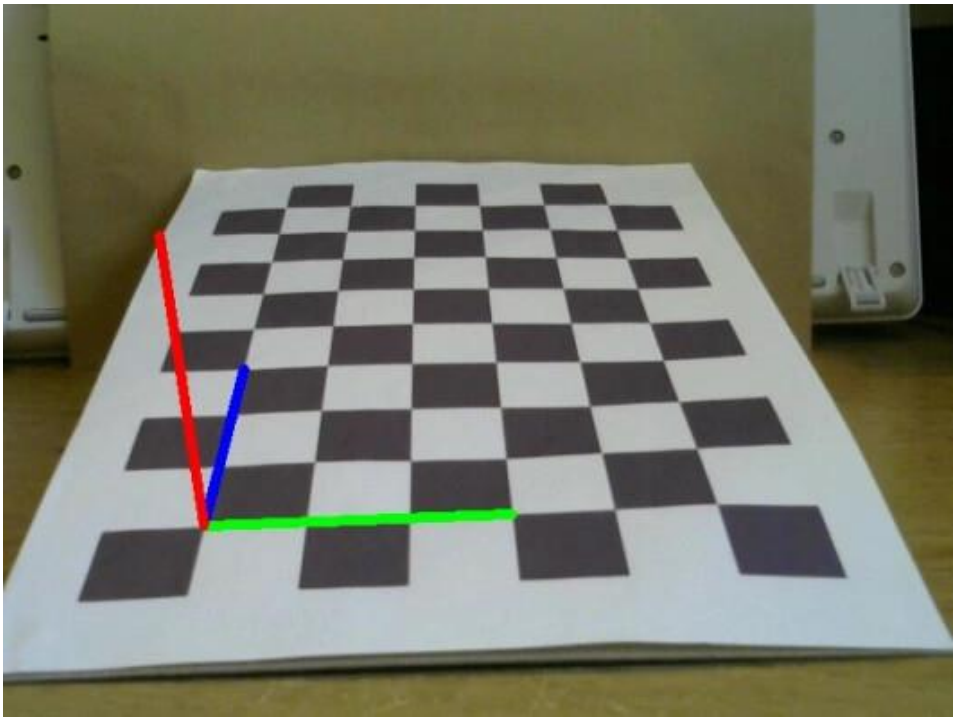


Figura 43. Ejes de coordenadas para los object points.

FUENTE: HTTPS://FORUM.OPENCV.ORG/T/FUNDAMENTAL-OF-ROTATION-AND-TRANSLATION-IN-CALIBRATECAMERA/2388

Sin embargo, las coordenadas de los puntos imagen deben de calcularse a partir de las esquinas detectadas. Para ello se ha utilizado la función de OpenCV `cv.cornerSubPix()` para refinar las coordenadas en la imagen de las esquinas encontradas y guardarlas en la variable *corners2*. Estas coordenadas serán las que se añadan al vector de puntos imagen *imgpoints*.

Por último, se mostrará la imagen con el reconocimiento de esquinas mediante `cv.drawChessboardCorners(…)` y `cv.imshow("img", img)` y se esperará durante un segundo antes de analizar la siguiente imagen para que el usuario pueda visualizar la detección de esquinas tal y como se ve en la **Figura 44**.

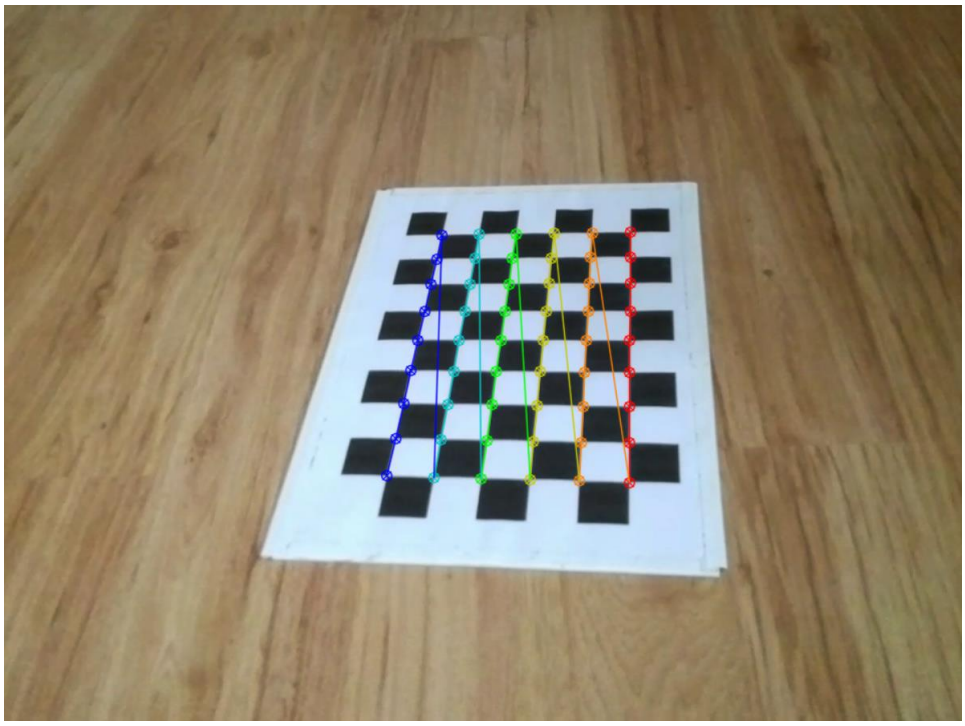


Figura 44. Visualización de los puntos característicos del patrón de calibración.

FUENTE: ELABORACIÓN PROPIA

Una vez terminado el proceso se elimina la ventana de visualización con `cv.destroyAllWindows()`.

6.1.2.1. Calibración

Una vez obtenidos los puntos objeto y los puntos imagen se puede obtener la matriz de calibración y los vectores de distorsión mediante la función `cv.calibrateCamera(…)` tal y como se muestra en el **Código 5**.

```
65  (
66      ret,
67      cameraMatrix,
68      dist,
69      rvecs,
70      tvecs,
71  ) = cv.calibrateCamera(  # Obtiene la matriz de calibración y el vector de características
72      objpoints,
73      imgpoints,
74      frameSize,
75      None,
76      None,  # La función necesita como entrada los object points e image points obtenidos p
77  )
78  print("cameraMatrix: {}".format(cameraMatrix))
79  print("dist: {}".format(dist))
```

Código 5. Código de calibración.

Como se puede observar, la función tiene como parámetros de entrada los puntos imagen y los puntos objeto obtenidos previamente, además del tamaño del fotograma en pixeles. Esta función se encarga de, a partir de **cotejar** los puntos objeto con sus puntos imagen correspondientes, obtener las matrices de la cámara y su pose, siendo capaz de devolver así la **matriz característica** de la cámara y, además, los **parámetros de distorsión**. Estos valores se almacenan en las variables *cameraMatrix* y *dist* respectivamente y se muestran en la consola con la función `print(…)`. Las variables *rvecs* y *tvecs*, servirán posteriormente para saber lo fiable que es la calibración mediante **reproyección**.

Por último, se guardan los valores de *cameraMatrix* y *dist* para ser utilizados en el programa principal con la función de la librería **pickle** `pickle.dump(…)`, tal como se muestra en el **Código 6**.

```
82  pickle.dump((cameraMatrix, dist), open("calibration.pkl", "wb"))
83  pickle.dump(cameraMatrix, open("cameraMatrix.pkl", "wb"))
84  pickle.dump(dist, open("dist.pkl", "wb"))
```

Código 6. Guardado de los parámetros cameraMatrix y dist.

6.1.2.2. Corrección de la distorsión y error de reproyección

Aunque en este punto no es necesario realizar una corrección de distorsión, es interesante realizarlo para comprobar, aunque sea visualmente que esta verdaderamente funciona. Para ello se utilizará el **Código 7.**

```python
# CORRECCIÓN DE DISTORSIÓN

img = cv.imread(
    "Imagenes/img5.png"
)  # Se lee una imagen para poder realizar una comprobación visual de la corrección de di
h = img.shape[0]
w = img.shape[1]

(
    newCameraMatrix,
    roi,
) = cv.getOptimalNewCameraMatrix(  # Se refina la matriz de calibración y se almacena en
    cameraMatrix, dist, (w, h), 1, (w, h)
)

# Undistort
dst = cv.undistort(
    img, cameraMatrix, dist, None, newCameraMatrix
)  # Se corrige la distorsión de la imagen con "undistort" para la comprobación visual

# Se recorta la imagen puesto que al corregir la distorsión se deforman los bordes de la
x, y, w, h = roi
dst = dst[y : y + h, x : x + w]
cv.imwrite("caliResult1.png", dst)
```

Código 7. Comprobación de la corrección de la distorsión.

En este código se utiliza una de las imágenes previamente cargadas (img5.png) como candidata para corregir su distorsión y se obtiene su tamaño en pixeles mediante la función `img.shape[]`.

Acto seguido, se refina la matriz de la cámara mediante la función `cv.getOptimalNewCameraMatrix(…)` la cual devuelve la matriz optimizada y unos parámetros que servirán para ajustar la imagen al corregir la distorsión.

Por último, se corrigen las distorsiones con la función `cv.undistort(…)`, la cual a partir de la imagen original, las matrices de la cámara obtenidas y el vector de distorsión, devuelve la imagen corregida en la variable *dts,* y se recortan los bordes para evitar deformaciones. Puesto que la cámara utilizada no genera distorsiones muy notables se han adjuntado imágenes que representan este efecto provenientes de otras fuentes (**Figura 45**). En ciertas cámaras como las cámaras de focal cercana al **ojo de pez** hay una gran diferencia entre calibrar la cámara o no hacerlo.

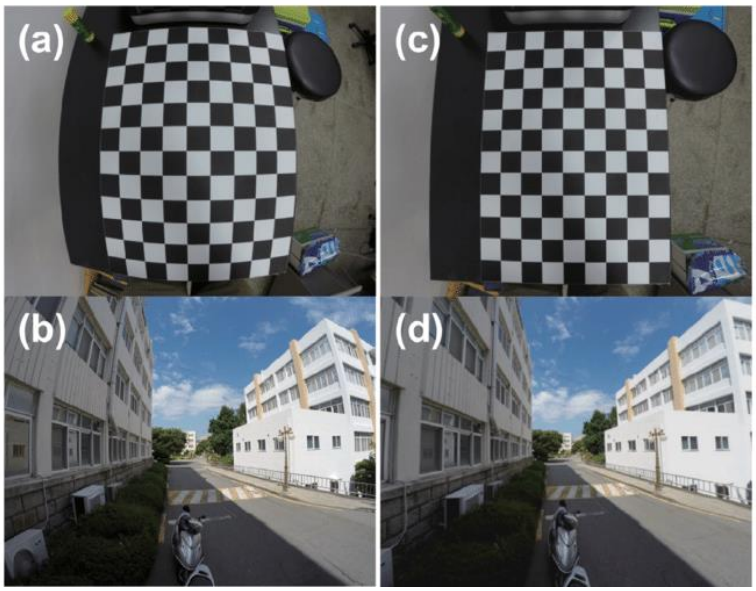


Figura 45. Efectos de la calibración. A la izquierda, sin calibrar. A la derecha, calibrada.

FUENTE: HTTPS://WWW.RESEARCHGATE.NET/FIGURE/THE-PREDETERMINED-SIZE-CALIBRATION-PATTERN-5-5-CM-PER-RECTANGLE-USED-TO-CORRECT-THE_FIG6_336215709

Por otro lado, en el **Código 8** se ha realizado la comprobación de la fiabilidad de la calibración. Para esto se ha calculado el **error de reproyección** de la imagen.

```
112  # Cálculo del error de reproyección
113  mean_error = 0
114
115  for i in range(len(objpoints)):
116      imgpoints2, _ = cv.projectPoints(
117          objpoints[i], rvecs[i], tvecs[i], cameraMatrix, dist
118      )
119      error = cv.norm(imgpoints[i], imgpoints2, cv.NORM_L2) / len(imgpoints2)
120      mean_error += error
121
122  print("total error: {}".format(mean_error / len(objpoints)))
```

Código 8. Cálculo del error de reproyección.

En este código se calcula la proyección de los puntos imagen de cada fotografía a partir del modelo de la cámara obtenido en la calibración, de la pose en la fotografía (rvecs y tvecs) y de los puntos objeto. Posteriormente se obtiene la diferencia entre esos puntos y los puntos calculados originalmente y se almacena en la variable *error* con la función `cv.norm(…)`. Este error calculado se suma al error del resto de imágenes. Por último, se imprime por pantalla la media del error de

todas las imágenes. En la **Figura 46** se puede ver una representación del significado del error de reproyección.

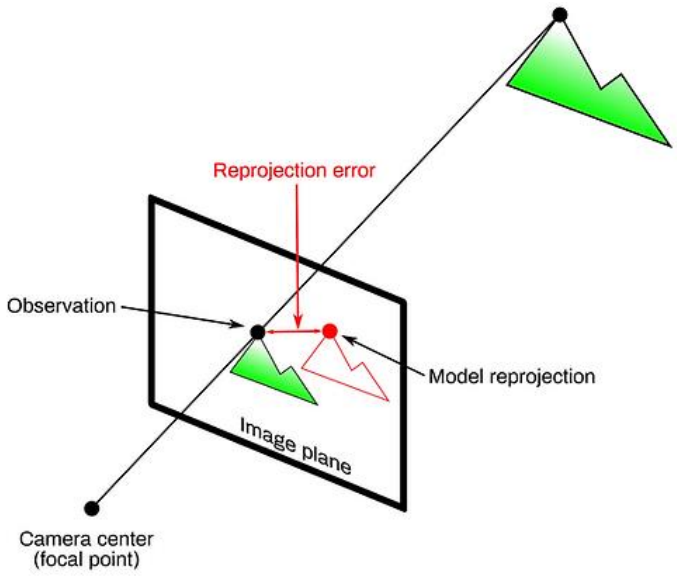


Figura 46. Esquema del error de reproyección.

FUENTE: HTTPS://WWW.CAMCALIB.IO/POST/WHAT-IS-THE-REPROJECTION-ERROR

Así pues, el error de reproyección obtenido en la calibración del proyecto es de: total error: 0.025202390741132362, lo que significa que la desviación media al proyectar la imagen con respecto del cálculo real es de 0.025 pixeles, un error más que aceptable.

6.1.3. Final.py

El programa Final.py tiene el código encargado de **filtrar el objeto** del fondo, **corregir las distorsiones** de la imagen, obtener las coordenadas del **centro de gravedad** del objeto en pixeles y traducir estas **coordenadas** al mundo real. Posteriormente, representa estas coordenadas en un gráfico.

Este programa utiliza prácticamente todas las librerías que se han visto anteriormente, esto se puede observar en la sección del **Código 9.**

```
5   import numpy as np
6   import cv2 as cv
7   import pickle
8   import sympy as sp
9   import math
10  import matplotlib.pyplot as plt
11  import time
12  from djitellopy import Tello
```

Código 9. Librerías Final.py.

Para empezar la explicación del código se realizará un recorrido por las distintas funciones declaradas. Estas son la función **CentroGravedad(),** la función **update_graph()** y por último la función **pinHole()**.

6.1.3.1. CentroGravedad()

Esta función (**Código 10**) es la encargada de, a partir del contorno del objetivo que se le pasa como argumento, obtener el centro de gravedad de este, es decir, el punto promedio del contorno en pixeles.

```
138 def CentroGravedad(Contorno):
139     M = cv.moments(Contorno)
140     if M["m00"] == 0:
141         M["m00"] = 1
142     u = int(M["m10"] / M["m00"])
143     v = int(M["m01"] / M["m00"])
144     return u, v
```

Código 10. Función CentroGravedad().

Este código se basa en los momentos del contorno, obtenidos por la función de OpenCV `cv.moments(Contorno)`, para obtener el centro de gravedad.

Primeramente, se comprueba que el momento de orden cero `M["m00"]`, que representa el área contenida por el contorno, no sea cero, puesto que, en el caso de serlo, se sustituirá por un área de 1 para evitar errores de división por cero en los siguientes cálculos.

Por otro lado, los momentos `M["m10"]` y `M["m01"]` son los momentos de primer orden que representan las coordenadas x e y del

contorno. Así pues, dividiendo cada una de estas coordenadas por el área del contorno se obtiene el **punto promedio** del contorno en x e y, es decir, el centro de gravedad *u* y *v*. Estas coordenadas son las que retornan de la función.

6.1.3.2. Update_graph()

La función *Update_graph(),* mostrada en el **Código 11**, recoge como argumento las coordenadas (X,Y) y las recopila en dos vectores, *x_values* e *y_values*, que representan el histórico de las coordenadas en las que se ha detectado el objetivo.

```
108  def update_graph(X, Y):
109      # Agregamos el valor de 'x' y de 'y' a la lista
110      x_values.append(X)
111      y_values.append(Y)
112
113      # Marcas en el suelo en metros
114      marcas_x = [0, 20, 46, 20]
115      marcas_y = [0, -5, 0, 5]
116
117      # Recorrido predefinido
118      x_predef = [0, 20, 46, 20, 0]
119      y_predef = [0, -5, 0, 5, 0]
120
121      # Actualizamos la gráfica con los nuevos datos
122      ax1.plot(x_values, y_values, "b-")
123      ax1.plot(x_predef, y_predef, "r-")
124      ax1.scatter(marcas_x, marcas_y, color="red", marker="v")
125
126      ax1.set_aspect("equal")
127
128      # Establecemos los límites de los ejes 'x' e 'y' en metros
129      ax1.set_xlim(0, 50)
130      ax1.set_ylim(-10, 10)
131
132      # Título y etiquetas de los ejes
133      ax1.set_title("Objetivo")
134      ax1.set_xlabel("X (metros)")
135      ax1.set_ylabel("Y (metros)")
```

Código 11. Función update_graph().

Los vectores *marcas_x* y *marcas_y* representan algunos puntos característicos del mundo real, se hace así para tener unos **puntos de referencia** para hacer una prueba de validez del algoritmo. Mientras

tanto, los vectores *x_predef* e *y_predef* sirven para dibujar un **recorrido predefinido** entre puntos que resulte fácil de replicar y comparar con la detección real del objetivo. Este experimento se explicará mejor en el **apartado 7.**

A continuación, se representa en la gráfica mostrando en azul y unido por una línea continua, `ax1.plot(x_values, y_values, "b-")`, el recorrido del objetivo detectado por la aeronave mientras que se mostrará en rojo y al igual unido por una línea continua, `ax1.plot(x_predef, y_predef, "r-")`, el recorrido predefinido. Por otro lado, se representará por marcas, pero sin unir los puntos fijos en el mundo real mediante la función `ax1.scatter(marcas_x, marcas_y, color="red", marker="v")`. En la **Figura 47** se muestra un ejemplo de la representación, sin detectar ningún objetivo, con los puntos de referencia y el recorrido predefinido del **Código 11.**

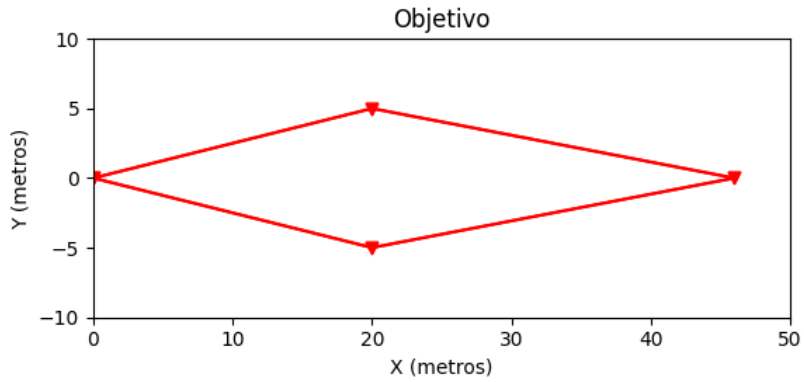


Figura 47. Ejemplo de recorrido predefinido.

FUENTE: ELABORACIÓN PROPIA

Es importante hacer hincapié en que los puntos de referencia son únicamente para que sea sencillo seguir el recorrido en el mundo real, estos no tienen ninguna relevancia en los cálculos realizados por el algoritmo.

Las siguientes líneas del código son para establecer que la distancia en metros de los ejes sea la misma y fijar los límites de estos, además de escribir el título del gráfico y los nombres de los ejes.

6.1.3.3. pinHole()

La función *pinHole()* es la función más importante de todo el proyecto. Esta es la encargada de a partir de las **coordenadas en pixeles** del centro de gravedad del objetivo (*pixw* y *pixh)*, de ciertos **parámetros de la cámara** (*kw, kh* y *f*) y ciertos **parámetros geométricos** (*Z* y *pose*) traducir las coordenadas en pixeles del objetivo a coordenadas respecto de unos ejes de coordenadas fijos del mundo real.

Para poder abordar la explicación de esta función es importante entender los ejes de referencia que se han tomado y las **transformaciones** que se han realizado, además de entender la importancia de cada una de las variables que se mencionan. En la **Figura 48** se ha realizado un esquema que recopila todas estas transformaciones entre los ejes.

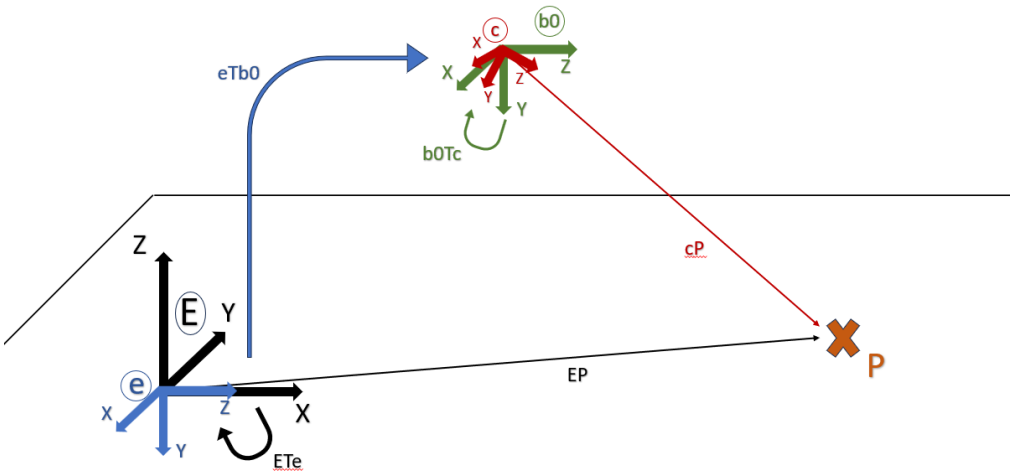


Figura 48. Esquema de los ejes y transformaciones necesarias para la traducción de coordenadas.

FUENTE: ELABORACIÓN PROPIA

En la figura se pueden observar los ejes *E* y *c* que representan los **ejes globales** y los **ejes de la cámara** respectivamente, mientras que los ejes *e* y *b0* representan pasos intermedios. El objetivo final es poder obtener la matriz de transformación ${}^{E}T_{c}$ para traducir las coordenadas obtenidas por las ecuaciones de *Pin Hole* ${}^{c}P$ a las coordenadas respecto

del eje fijo ${}^{E}P$, siendo P el objetivo a localizar. De esta forma obtenemos que:

$$ {}^{E}P = {}^{E}T_c * {}^{c}P \rightarrow {}^{c}P = {}^{c}T_E * {}^{c}P = \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} $$

Por otro lado, según el modelo Pin Hole, del cual se muestran sus parámetros en la **Figura 49,** también se puede obtener una función del valor de x_c, y_c y z_c.

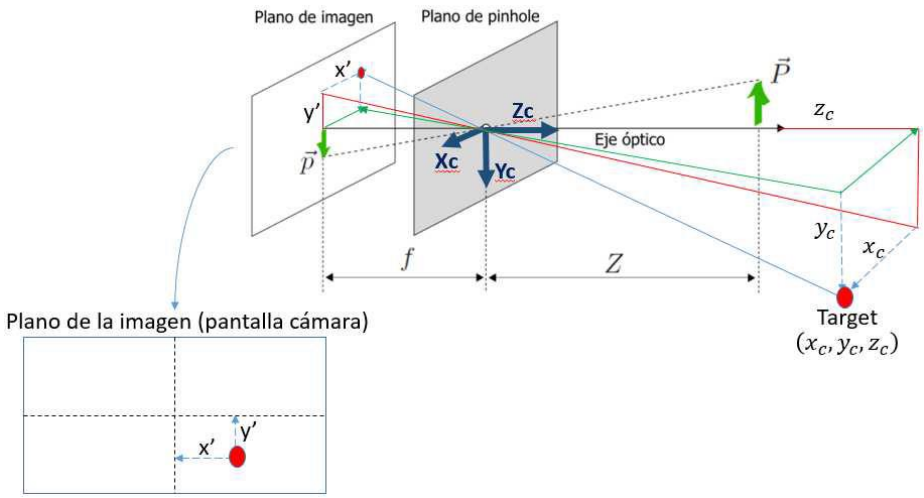


Figura 49. Parámetros del modelo Pin Hole.

FUENTE: APUNTES "ROBÓTICA AÉREA TEMA 2".

Así pues, para obtener *x'* e *y'*, siendo estas la posición del píxel en cuestión en mm ubicado en el sensor, es necesario conocer primeramente *kw* y *kh,* siendo estas el número de pixeles del fotograma dividido entre el tamaño del sensor: $kw = \frac{Pixeles\ X}{Tamaño\ Horizontal}$; $kh = \frac{Pixeles\ Y}{Tamaño\ Vertical}$. A partir de esto es posible calcular *x'* e *y' de la siguiente forma:*

$$ x' = \frac{\delta w}{kw} \quad ; \quad y' = \frac{\delta h}{kh} $$

Siendo δw y δh la posición del píxel con respecto del centro de la imagen.

Una vez obtenidos estos parámetros se puede obtener por relación de triángulos equivalentes que:

$$\frac{x'}{f} = \frac{x_c}{z_c} \quad ; \quad \frac{y'}{f} = \frac{y_c}{z_c} \quad \rightarrow \quad f * \frac{x_c}{z_c} - x' = 0 \quad ; \quad f * \frac{y_c}{z_c} - y' = 0$$

Así pues, sustituyendo x_c, y_c y z_c con los valores obtenidos de las matrices de transformación previamente (${}^{c}P$), y resolviendo las ecuaciones se pueden obtener las incógnitas X e Y.

En el **Código 12** se pueden observar la inicialización de variables, la definición de las **variables simbólicas** y el cálculo de los parámetros del plano imagen que se han mencionado.

```
39  def pinHole(pixw, pixh, kw, kh, f, Z, pose):
40      # Valores obtenidos fuera de la funcion (Tamano en pixeles de la imagen)
41      global numPixY, numPixX
42
43      # Definicion de los valores simbolicos necesarios
44      X, Y, yc = sp.symbols("X Y yc")
45
46      # Calculos de parametros de la camara y coordenadas del punto imagen
47      dh = pixh - (numPixY / 2)
48      dw = pixw - (numPixX / 2)
49      yp = dh / kh
50      xp = dw / kw
```

Código 12. Inicialización y cálculos previos de la función pinHole().

A continuación, en el **Código 13**, se muestra el cálculo realizado para obtener la matriz de transformación ${}^{E}T_c$ y el vector ${}^{E}P$ a resolver con sus **variables simbólicas**. Estas transformaciones contemplan la translación y rotación entre los ejes, además de la rotación propia del dron (roll, pitch y yaw). Es importante comprender que Z en este vector será constante y no una variable a resolver, puesto que el objetivo siempre estará a la **misma altura Z** de los ejes. La información sobre matrices de transformación homogéneas y rotaciones ha sido explicada en el **apartado 3.2**

```python
    # Declaracion de las matrices simbolicas necesarias para la transformacion de coordena

    EP = sp.Matrix([X, Y, Z, 1])
    ETe = sp.Matrix(
        [[0, 0, 1, 0], [-1, 0, 0, 0], [0, -1, 0, 0], [0, 0, 0, 1]]
    )  # Giro para orientar ejes mundo a ejes dron de referencia
    eTb0 = sp.Matrix(
        [[1, 0, 0, pose[0]], [0, 1, 0, pose[1]], [0, 0, 1, pose[2]], [0, 0, 0, 1]]
    )  # Translacion de nuevos los ejes al body del dron
    b0Tb1 = sp.Matrix(  # Rotacion roll
        [
            [math.cos(pose[3]), -math.sin(pose[3]), 0, 0],
            [math.sin(pose[3]), math.cos(pose[3]), 0, 0],
            [0, 0, 1, 0],
            [0, 0, 0, 1],
        ]
    )
    b1Tb2 = sp.Matrix(  # Rotacion pitch
        [
            [1, 0, 0, 0],
            [0, math.cos(pose[4]), -math.sin(pose[4]), 0],
            [0, math.sin(pose[4]), math.cos(pose[4]), 0],
            [0, 0, 0, 1],
        ]
    )
    b2Tc = sp.Matrix(  # Rotacion yaw
        [
            [math.cos(pose[5]), 0, math.sin(pose[5]), 0],
            [0, 1, 0, 0],
            [-math.sin(pose[5]), 0, math.cos(pose[5]), 0],
            [0, 0, 0, 1],
        ]
    )
    ETc = ETe * eTb0 * b0Tb1 * b1Tb2 * b2Tc
```

Código 13. Matrices de transformación de la función pinHole().

Por último, la función se encarga de obtener el vector ${}^{c}P$ a partir de ${}^{E}T_{c}$ y ${}^{E}P$, tal y como se ha explicado previamente, se asignan los valores a cada una de las variables x_c, y_c y z_c, se resuelven las ecuaciones de Pin Hole con estos valores y se devuelven los valores resueltos de X e Y. El código que incluye estos pasos es el **Código 14.**

```
87      cP = ETc.inv() * EP
88
89      # Ecuaciones simbolicas que relacionan las coordenadas mundo con las coorde
90      xc = cP[0]
91      yc = cP[1]
92      zc = cP[2]
93
94      # Resolucion de las ecuaciones a partir de las formulas del modelo PINHOLE
95      sol = sp.solve([(f * yc / zc) - yp, (f * xc / zc) - xp], dict=True)
96      # print(sol)
97
98      # Llamamos a la función para actualizar la gráfica del objetivo
99      update_graph(sol[0][X], sol[0][Y])
100     # Forzamos la actualización de las gráficas
101     plt.draw()
102     plt.pause(0.001)
103     # Se devuelven las coordenadas mundo y se dibujan
104     return sol[0][X], sol[0][Y], Z
```

Código 14. Resolución de las ecuaciones simbólicas de pinHole().

Es en este punto también donde se llama a la función `update_graph(sol[0][X], sol[0][Y])`, la cual añade estos nuevos valores al histórico de la posición del objetivo y los representa, tal y como se ha explicado en el punto anterior.

6.1.3.4. Código principal

Una vez explicadas las funciones, se procede a la explicación del código principal.

```
14  # Estableciendo conexion con el Tello
15  tello = Tello()
16
17  # Inicialización de variables de control de vuelo
18  offset_yaw = 0
19  t_old = 0
20  periodo = 500000000  # Tiempo entre cada bucle main (ns)
21
22  tello.connect()
23  tello.streamon()
24
25  # Parametros intrinsecos de la camara obtenidos de las especificaciones tecnicas de la
26  f = 3.8 / 1000  # Longitud Focal    4.2
27  H = 6.17 / 1000  # Tamano del sensor Horizontal 1/2.3"
28  V = 4.55 / 1000  # Tamano del sensor vertical 1/2.3"
29
30  # Altura del objeto con respecto de los ejes de referencia E
31  Altura = 100 / 100  # en metros
32
33  # Creamos listas vacía para almacenar los valores de 'x' y de 'y' del objetivo y de la
34  x_values = []
35  y_values = []
```

Código 15. Inicialización Final.py.

Como se puede observar en el **Código 15**, el código principal empieza con una breve inicialización de variables como el **offset** de yaw, del que se hablará más adelante, la inicialización de la variable del tiempo *t_old* o la asignación del **periodo de temporalización** del bucle principal, además de la inicialización de la librería Tello y la conexión con el dron que se explicó en el **punto 6.1.1**.

Además, se asignan los valores de los **parámetros de la cámara** tales como la distancia focal (*f*) o el tamaño del sensor (*H* y *V*) y el parámetro que asigna la altura del objeto con respecto de los ejes de referencia (*Altura*).

Por último, se crean las listas que contendrán el histórico de la posición del objetivo llamadas *x_values* e *y_values*.

El siguiente paso en el código principal es cargar los parámetros de distorsión y de la matriz de la cámara obtenidos en la calibración. Para ello se utilizará la librería **pickle**, la cual a través de la función `pickle.load(…)`, recuperará estos datos tal y como se muestra en el **Código 16** y los almacenará en las variables *cameraMatrix* y *dist*.

```
148   cameraMatrix = pickle.load(open("cameraMatrix.pkl", "rb"))
149   dist = pickle.load(open("dist.pkl", "rb"))
```

Código 16. Carga de los parámetros obtenidos en la calibración.

A continuación, se crearán los filtros de color necesarios para diferenciar el objetivo del fondo. Para ello se ha utilizado el **espacio de color** conocido como **HSV** (del inglés Hue, Saturation, Value), el cual se basa en tres parámetros para identificar un color: Tonalidad (Hue), Saturación (Saturation) y Valor (Value) (Rodas Jordá, s. f.). En la **Figura 50** se puede observar una representación del espacio HSV.

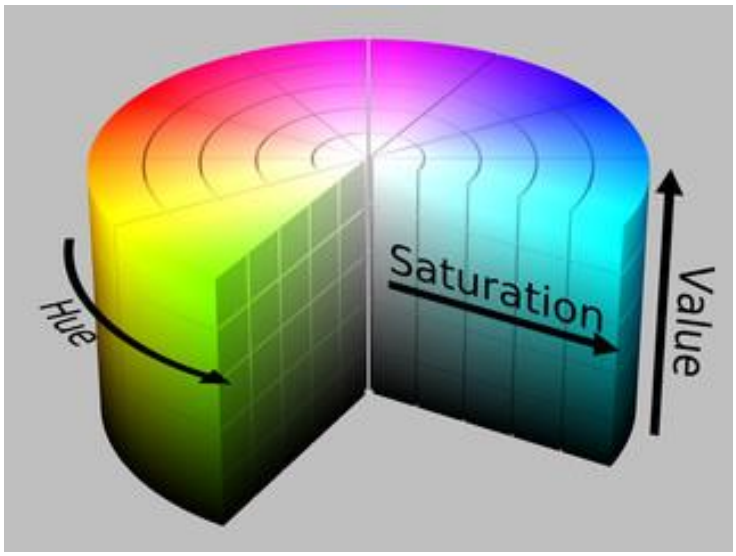


Figura 50. Modelo del espacio de color HSV.

FUENTE: HTTPS://DOCS.OPENCV.ORG/3.4/DA/D97/TUTORIAL_THRESHOLD_INRANGE.HTML

En OpenCv los valores de tonalidad se encuentran entre 0 y 179 grados, mientras que la saturación y el valor se encuentran entre 0 y 255. Para poder asignar un rango de valores de color como objetivo a filtrar es necesario fijarse en la escala de la **Figura 51.**

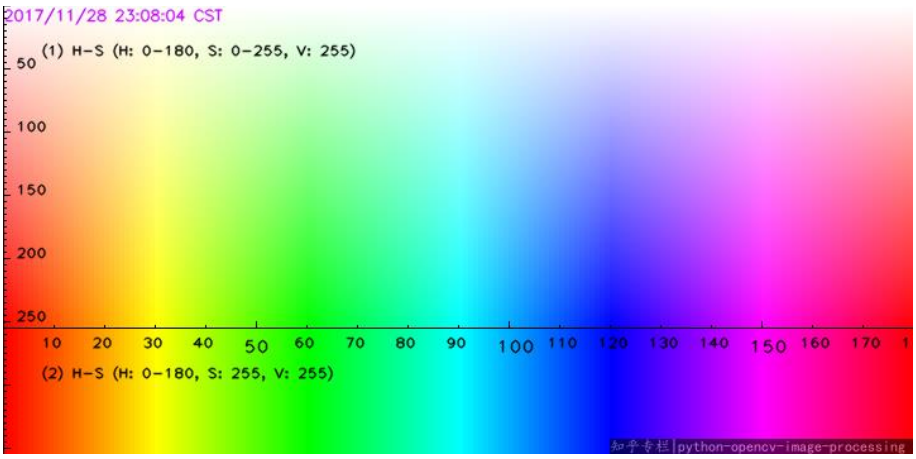


Figura 51. Valores de HSV con OpenCV.

FUENTE: HTTPS://OMES-VA.COM/DETECCION-DE-COLORES/

En el ejemplo del **Código 17** se ha utilizado un objetivo de color **azul** por lo que la tonalidad debería estar entre 100 y 125. Los valores de saturación y valor máximos están fijados a 255 mientras que los mínimos deben ajustarse de manera experimental según el objetivo sea más pálido (disminuye la saturación) o más oscuro (disminuye el valor).

```
152   # Azul
153   colorBajo = np.array([100, 100, 70], np.uint8)
154   colorAlto = np.array([125, 255, 255], np.uint8)
```

Código 17. Filtros de color.

Con este filtro se obtienen el filtrado mostrado en la **Figura 52** a partir del objeto mostrado en la **Figura 53.**

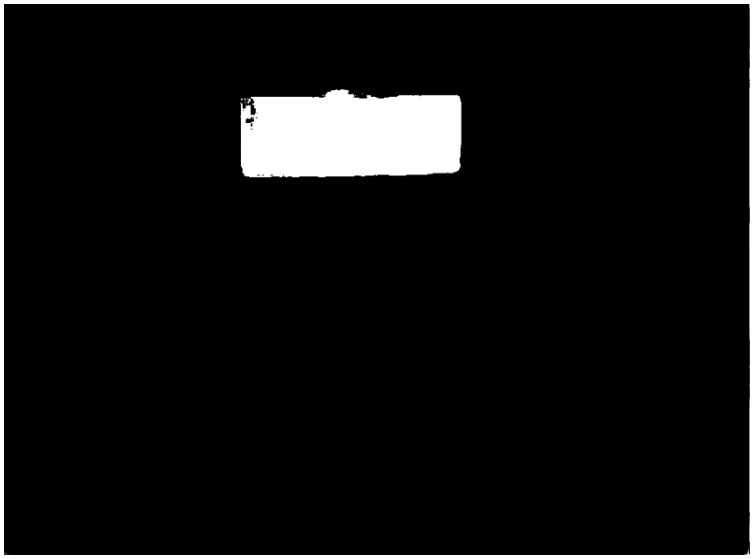


Figura 52. Filtrado de objetivo con HSV utilizando una máscara para el color azul.

FUENTE: ELABORACIÓN PROPIA

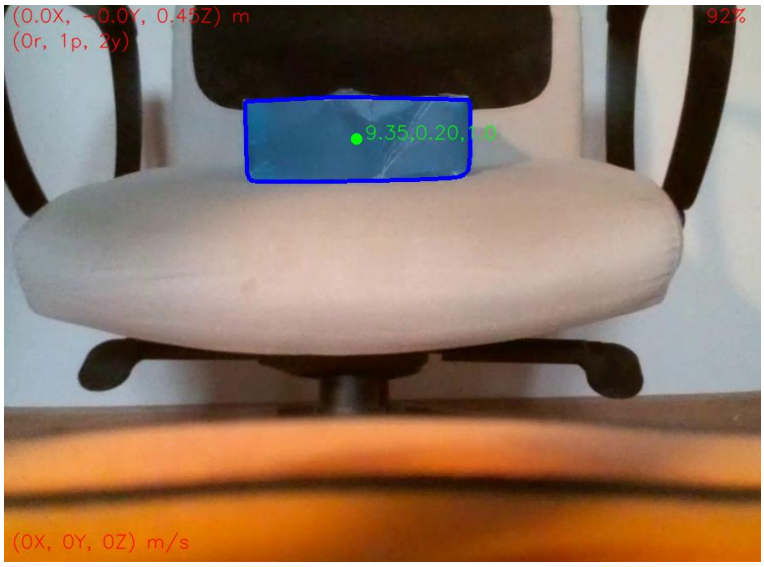


Figura 53. Objetivo azul.

FUENTE: ELABORACIÓN PROPIA

Luego, se crean los **elementos estructurales** necesarios para poder realizar las **operaciones morfológicas** de apertura y cierre sobre la máscara, de forma que se filtre mejor el ruido de la imagen que pueda pasar un simple filtrado de color (Rodas Jordá, s. f.). Estos elementos estructurales tendrán forma de elipse y serán de distinto tamaño según la operación que se realice (**Código 18**).

```
168   kernel = cv.getStructuringElement(cv.MORPH_ELLIPSE, (10, 10))
169   kernel2 = cv.getStructuringElement(cv.MORPH_ELLIPSE, (20, 20))
```

Código 18. Creación de elementos estructurales.

Como último paso previo a el bucle principal se habilita el modo interactivo de las gráficas, se crea una estructura (*subplot*) donde situarlas y se muestran en una ventana mediante el **Código 19**.

```
172   # Habilitamos el modo interactivo de la grafica
173   plt.ion()
174
175   # Creamos la figura para las gráficas
176   fig, ax1 = plt.subplots()
177
178   # Mostramos las gráficas en una ventana
179   plt.show()
```

Código 19. Creación de la gráfica.

Al inicio del bucle principal, `while True:`, los primeros pasos que se realizan son la lectura de teclas y la lectura de la IMU. Para la lectura de teclas se utiliza el mismo comando que se ha explicado en el punto **6.1.1.** Acto seguido se comprueba si esa tecla corresponde con la *t* o la *o*. En el caso de ser la primera se iniciará un despegue automático del dron, esto es así por si no fuese posible realizar el despegue desde el dispositivo de control, mientras que la tecla *o* sirve para reiniciar el ángulo yaw, por si el ángulo inicial está orientado en una dirección equivocada. La lectura de la IMU se realiza mediante las instrucciones integradas en la librería **djitellopy.** Es en este punto donde se aplica la corrección de yaw al restarle el offset asignado al pulsar la tecla *o*.

Además se toma este punto como el inicio de la temporalización del bucle mediante la instrucción `t_old = time.time_ns()`. Todo este proceso se puede comprobar en el **Código 20**.

```
183     k = cv.waitKey(5)
184     # Despega el dron
185     if k == ord("t"):
186         tello.takeoff()
187     if k == ord("o"):
188         offset_yaw = tello.get_yaw()
189
190     # Obtencion de la IMU
191     imu_r = tello.get_roll()
192     imu_p = tello.get_pitch()
193     imu_y = tello.get_yaw() - offset_yaw
194
195     t_old = time.time_ns()
```

Código 20. Lectura de IMU y corrección de la guiñada.

El siguiente paso que realiza el **Código 21** es la obtención de la pose de la cámara respecto de los ejes de coordenadas *e* (**Figura 48**), a partir de los datos de la IMU y del telémetro de altura.

```
197     # Pose de la camara con respecto de los ejes de referencia e
198     xb = 0
199     yb = -tello.get_distance_tof()/100
200     zb = 0
201     roll = imu_r * math.pi / 180
202     pitch = (-15 + imu_p) * math.pi / 180
203     yaw = imu_y * math.pi / 180
204
205     pose = np.array([xb, yb, zb, roll, pitch, yaw])
```

Código 21. Obtención de la pose de la cámara.

Algo a tener en cuenta es que en este punto se deben traducir los ángulos a radianes, además que el ángulo *pitch* tiene un "offset" de 15 grados puesto que esta es la inclinación de la cámara con respecto del resto de la aeronave. Además, se puede observar que la distancia en *x* y *z* (siempre respecto de los ejes *e*) siempre será nula puesto que la aeronave no se va a desplazar en horizontal, pero sí en vertical.

En el **Código 22** se puede observar como el siguiente paso es leer la imagen transmitida por la aeronave y corregir sus distorsiones tal como se explicó en el **apartado 6.1.2.2.**

```
207     frame = tello.get_frame_read().frame
208
209     # Obtencion del tamano de la imagen
210     numPixY = frame.shape[0]
211     numPixX = frame.shape[1]
212
213     newCameraMatrix, roi = cv.getOptimalNewCameraMatrix(
214         cameraMatrix, dist, (numPixX, numPixY), 1, (numPixX, numPixY)
215     )
216
217     # Corrección de la distorsion de la imagen
218     dst = cv.undistort(frame, cameraMatrix, dist, None, newCameraMatrix)
219
220     # Recorte de la imagen y obtencion del nuevo tamano
221     x, y, numPixX, numPixY = roi
222     dst = dst[y : y + numPixY, x : x + numPixX]
223     numPixY = dst.shape[0]
224     numPixX = dst.shape[1]
```

Código 22. Lectura de la imagen y corrección de distorsiones.

A continuación, en el **Código 23**, se realiza el cálculo de *kw* y *kh*, tal y como se ha explicado en el **punto 6.1.3.3. Figura 49.** Posteriormente se asigna la imagen al espacio de color *HSV* mediante la función `cv.cvtColor(dst, cv.COLOR_BGR2HSV)`, se filtra la imagen entre los valores de *colorBajo* y *colorAlto* con la instrucción `cv.inRange(frameHSV, colorBajo, colorAlto)` y se realizan las operaciones de apertura y cierre pertinentes con la función `cv.morphologyEx(…)`.

Una vez se tiene la imagen filtrada, se analiza el contorno de esta mediante `cv.findContours(…)` y en el caso de haber varios contornos que superen un área determinada (en este caso de 500 pixeles), se comprueba cual es el contorno con mayor área mediante el bucle for.

```python
    # Calculo de kh y kw necesarios para la realizacion de PinHole
    kw = numPixX / H
    kh = numPixY / V

    # Cambio de espacio de color para facilitar la umbralizacion y calculo de contor
    frameHSV = cv.cvtColor(dst, cv.COLOR_BGR2HSV)
    mask = cv.inRange(frameHSV, colorBajo, colorAlto)
    mask2 = cv.morphologyEx(mask, cv.MORPH_OPEN, kernel)
    mask2 = cv.morphologyEx(mask2, cv.MORPH_CLOSE, kernel2)
    contornos, _ = cv.findContours(mask2, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)

    # Se reinician las variables que marcan el area y el contorno más relevante en c
    mayor_contorno = None
    mayor_area = 0

    # para no dibujar todos los contornos y evitar ruidos:
    for c in contornos:  # Miramos los contornos uno por uno
        area = cv.contourArea(c)
        if (
            area > 500 and area > mayor_area
        ):  # comprobamos que el area es mayor a un valor límite y al area detectand
            mayor_area = area
            mayor_contorno = c
```

Código 23. Análisis del objetivo mediante funciones de OpenCv.

En la **Figura 54** se puede observar la diferencia del objetivo filtrado sin y con operaciones morfológicas (izquierda y derecha respectivamente).

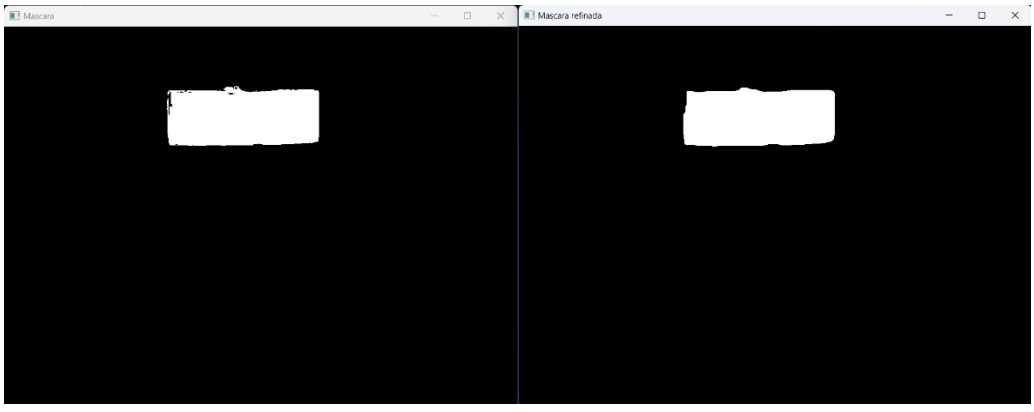


Figura 54. Diferencias al aplicar operaciones morfológicas.

FUENTE: ELABORACIÓN PROPIA

Acto seguido, si hay algún contorno que ha superado el filtro, se ejecuta el **Código 24.**

```
250     if (
251         mayor_contorno is not None
252     ):  # Se comprueba que hay algún contorno que ha superado el filtro
253         nuevoContorno = cv.convexHull(mayor_contorno)  # Suaviza el contorno filtrado
254         # Calculo del centro de gavedad del contorno
255         u, v = CentroGravedad(nuevoContorno)
256 
257         if yb < 0:  # Si el dron está en el aire
258             # Llamada a la funcion PINHOLE declarada al principio del codigo
259             X, Y, Z = pinHole(u, v, kw, kh, f, Altura, pose)
260 
261         cv.drawContours(
262             dst, [nuevoContorno], 0, (255, 0, 0), 3
263         )  # Dibujamos solo los contornos que cumplen esta especificacion
264 
265         # Dibujar el centro de gravedad y sus coordenadas
266         cv.circle(dst, (u, v), 7, (0, 255, 0), -1)
267         cv.putText(
268             dst,
269             "{},{},{}".format(round(X, 2), round(Y, 2), round(Z, 2)),
270             (u + 10, v),
271             font,
272             0.75,
273             (0, 255, 0),
274             1,
275             cv.LINE_AA,
276         )
```

Código 24. Traducción a coordenadas mundo.

Este código calcula el **centro de gravedad** del contorno gracias a la función *CentroGravedad()* explicada en el **apartado 6.1.3.1.** A continuación, comprueba si la altura del dron es distinta de 0, puesto que si lo fuera los cálculos darían error, y llama a la función *pinHole(),* la cual se encarga de traducir las coordenadas en pixeles del centro de gravedad del contorno a **coordenadas mundo**.

Una vez obtenidas las coordenadas deseadas, se dibuja el contorno con una línea azul alrededor del objetivo, se señala el centro de gravedad de este con un punto verde y se escriben sus coordenadas mediante las líneas 261 a 276 del **Código 24**.

A continuación, se escriben datos interesantes en la misma pantalla que la imagen mediante varias instrucciones de `cv.putText(…)`, tales como la posición del dron, su orientación o la batería restante, y se muestra la imagen por pantalla con `cv.imshow("Camara", dst)`.

Por último, en el **Código 25** se comprueba si la tecla pulsada es la tecla *esc* y si lo es se rompe el bucle y sale del programa, guardando la gráfica que representa la posición del objetivo y cerrando las pestañas.

De no pulsarse esta tecla, se comprueba el tiempo que ha estado realizando el bucle principal (**tiempo de procesamiento**) y de ser menor que el tiempo de bucle asignado al inicio del código principal, se realiza una **espera** para que se cumpla el tiempo restante. De esta forma se produce una medida siempre cada **medio segundo**, ahorrando recursos de procesamiento al no pedir un procesamiento constante de la imagen.

```
318     # Si se pulsa la tecla esc se rompe el bucle y acaba el codigo
319     if k == 27:
320         # tello.land()
321         break
322     # Temporalización de los cálculos
323     dt = time.time_ns() - t_old  # nsec
324     if periodo > dt:
325         time.sleep((periodo - dt) / 1000000000)
326         print("Tiempo de procesamiento {} nsec", dt / 1000000000)
327     else:
328         print("Error: tiempo de cálculo demasiado alto")
329 
330 plt.savefig("Objetivo.png")  # Guarda una imagen del recorrido del objetivo
331 cv.destroyAllWindows()
```

Código 25. Temporalización y cierre del programa principal.

6.2. Justificación de la solución adoptada

Para la realización del proyecto se optó, como se ha mencionado anteriormente, por utilizar un dron prefabricado con fácil acceso y programación, puesto que el trabajo no busca entrar en profundidad sobre la construcción de la aeronave y su control de bajo nivel, sino que pretende realizar un diseño propio del algoritmo de alto nivel encargado de localizar el objetivo. Es por ello por lo que lo prioritario a la hora de buscar opciones en el mercado para este proyecto es encontrar una aeronave que pueda ofrecer estabilidad y una transmisión de imágenes y datos de *pose* de fácil acceso y de manera fiable. Además, el factor precio es importante puesto que no se dispone de un presupuesto elevado para la realización del proyecto. Es por ello por lo que se eligió esta aeronave sencilla y barata, pero fiable.

Es importante también reconocer las limitaciones que esta tiene. No tiene localización GPS, no funciona en ambientes con poca luz o superficies lisas, es muy ligera y por lo tanto muy susceptible a turbulencias por viento, la distancia a la que se extiende la conexión no es demasiado alta, no tiene

estabilización de imagen física (gimbal)… No obstante, se ha elegido esta aeronave pensando en que este es un proyecto a pequeña escala que pretende demostrar la viabilidad para ser ejecutado en proyectos más avanzados. Con más presupuesto, una conexión de largo alcance, una aeronave con autonomía elevada y mayor peso, localización GPS y una cámara térmica con gimbal, el proyecto podría alcanzar un nivel muy superior al que se pretende llegar con este trabajo.

6.3. Valoración de soluciones alternativas

En las etapas más tempranas del proyecto, la idea principal era desarrollar un dispositivo que se pudiera acoplar a una aeronave sencilla para que realizase todo el objetivo planteado. Esta idea en principio es factible puesto que, ya que no tiene por qué comunicarse con la aeronave, este dispositivo podría encargarse independientemente de captar las imágenes, extraer su pose mediante una IMU y un sensor barométrico, realizar el procesamiento *on board* y enviar directamente las coordenadas del objetivo a la estación tierra.

La idea principal era realizar el procesamiento y la lectura de datos mediante una Raspberry Pi 3 (**Figura 55**), una cámara para Raspberry (RaspiCam), una IMU de 6 DoF, un barómetro y un GPS de manera opcional.

Figura 55. Raspberry Pi 3.

FUENTE: HTTPS://WWW.PCCOMPONENTES.COM/RASPBERRY-PI-3-MODELO-BPLUS

Este formato se rechazó debido al elevado precio que significaban todos los componentes sin contar con la aeronave, además de que implicaba un dron con la suficiente *MTOM* (Maximun Take Off Mass) para elevar su propio peso y el del dispositivo. Estos factores hicieron contemplar la alternativa que finalmente ha adoptado el proyecto

7. Resultados

Para poner a prueba el algoritmo se han realizado algunas pruebas y se han analizado los resultados obtenidos de estas.

Las primeras pruebas realizadas han sido enfocadas en el reconocimiento del objetivo. Como se puede comprobar en el **Código 26,** se han realizado pruebas con diferentes colores para asegurar una correcta detección del objetivo.

```
151  # Definicion de umbrales de color para detectar el objetivo en HSV
152  # Azul
153  colorBajo = np.array([100, 100, 70], np.uint8)
154  colorAlto = np.array([125, 255, 255], np.uint8)
155
156  # Verde
157  #colorBajo = np.array([30, 50, 70], np.uint8)
158  #colorAlto = np.array([60, 255, 255], np.uint8)
159
160  # Morado
161  #colorBajo = np.array([130, 80, 70], np.uint8)
162  #colorAlto = np.array([160, 255, 255], np.uint8)
163
164  # Rosa
165  #colorBajo = np.array([140, 50, 70], np.uint8)
166  #colorAlto = np.array([160, 255, 255], np.uint8)
```

Código 26. Umbrales de color utilizados.

Entre estos cuatro colores (Azul, Verde, Morado y Rosa) los colores que han resultado más satisfactorios a la hora de ser identificados son el **Azul y el Morado**, puesto que son colores poco presentes en los objetos que se encuentran en las escenas y además son colores bastante saturados. Por otro lado, el color rosa fue el que peor resultado obtuvo debido a que el objeto utilizado tenía un color rosa pálido, poco saturado y que, en ambientes con una

luz difusa, como lo suelen ser las escenas en exteriores, puede confundirse con colores como grises o blancos con mucha facilidad. En cuanto al ruido de color, ha sido un impedimento bastante grande en la detección de objetivos con poca luz, condición en la que el ruido crece mucho al aumentar el **ISO** de la cámara. Es por ello por lo que se decidieron utilizar las operaciones morfológicas y el filtrado por área para evitar confundir este ruido con el objetivo. Por estos motivos se recomienda utilizar colores de alta saturación en ambientes con suficiente luz.

En la siguiente experiencia se puso a prueba la capacidad del algoritmo para seguir y trazar la posición de un objetivo en movimiento. Para ello se marcaron unas localizaciones en el suelo, a unas distancias conocidas como se muestra en la **Figura 56**.

Figura 56. Marcas para la prueba de seguimiento.

FUENTE: ELABORACIÓN PROPIA

Estas mismas marcas se utilizan como puntos de referencia en la función *Update_graph()*, obteniendo un recorrido tal y como se muestra en la **Figura 57.**

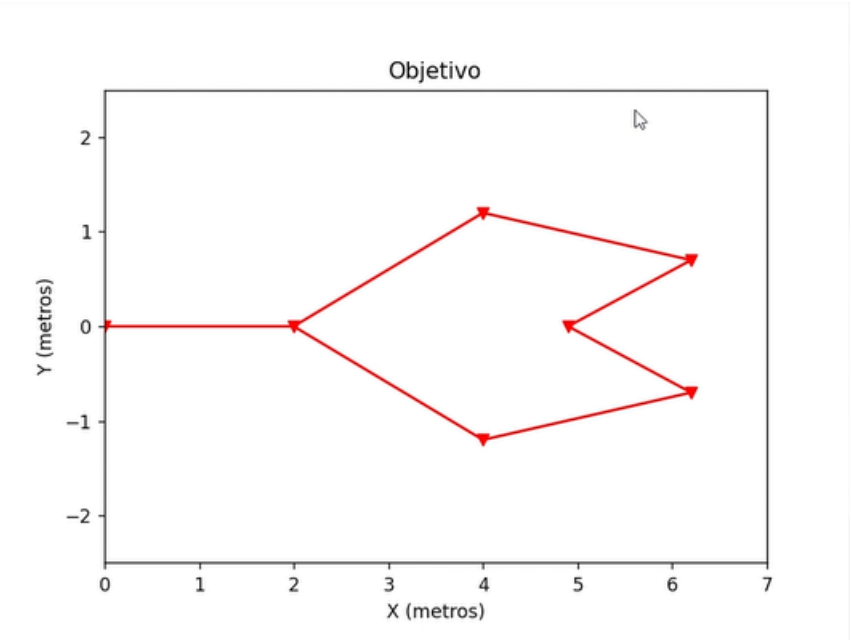


Figura 57. Recorrido predefinido de la prueba de seguimiento.

FUENTE: ELABORACIÓN PROPIA

La finalidad de esta prueba es realizar el recorrido predefinido con un **robot terrestre** al que se le ha acoplado un objetivo reconocible, de tal forma que el dron sea capaz de **identificar** su posición en diferentes momentos, añadiendo estas al histórico de la posición. Finalmente, se pretende superponer el recorrido detectado sobre el recorrido predefinido y observar las diferencias.

En esta prueba el objetivo será un pequeño robot móvil de fabricación propia con un objetivo azul, tal y como se muestra en la **Figura 58.**

Figura 58. Objetivo móvil.

FUENTE: ELABORACIÓN PROPIA

En la **Figura 59** se ha realizado una captura de pantalla al programa mientras el robot terrestre realizaba el recorrido.

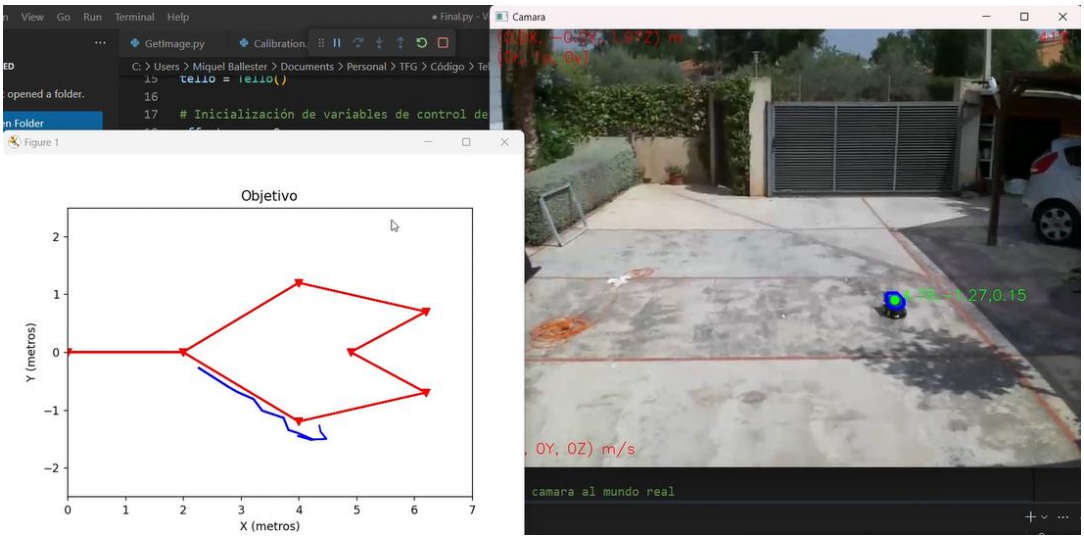


Figura 59. Muestra del funcionamiento del programa principal.

FUENTE: ELABORACIÓN PROPIA

Por último, en la **Figura 60** se muestra el resultado final al superponer ambos recorridos.

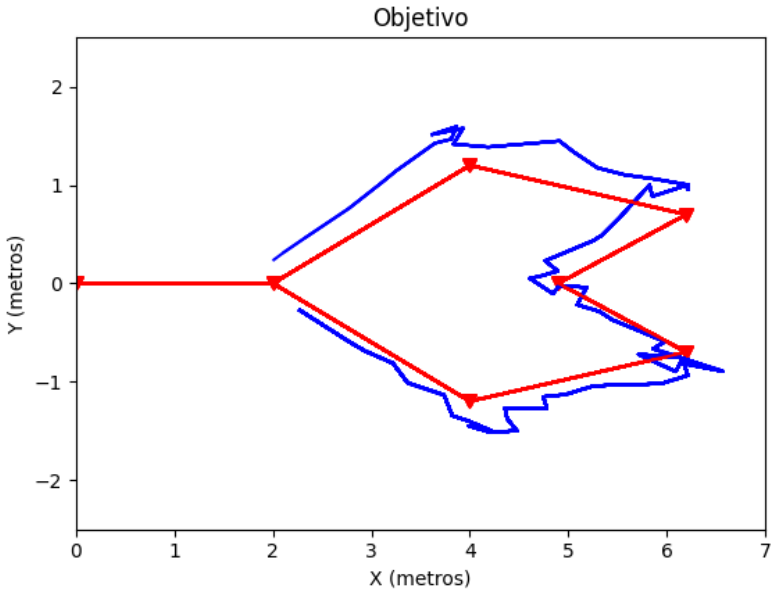


Figura 60. Diferencia entre el recorrido predefinido (Rojo) y el recorrido detectado (Azul).

FUENTE: ELABORACIÓN PROPIA

Lo primero que llama la atención es que el recorrido detectado no está completo, pero esto tiene una explicación muy sencilla. Esto es debido a que la aeronave está sobre el punto (0,0), elevada a casi **2 metros de altura** y la cámara tiene un ángulo de visión limitado. Estas condiciones hacen que los valores más pequeños de X no entren dentro de la imagen.

En cuanto al seguimiento detectado, se puede decir que se aproxima bastante al recorrido real. Aunque es cierto que el recorrido detectado tiene ciertas desviaciones, no se puede negar que el resultado es **muy satisfactorio**. Como conclusiones de esta experiencia se obtiene que el error máximo observado es de aproximadamente 0.4 m, error asumible teniendo en cuenta que la distancia máxima detectada es de 6 metros. Además, se puede observar que la mayor parte del error se produce en el eje Y, puesto que las distancias en el eje X suelen ser muy próximas a las reales.

La última prueba realizada busca demostrar la fiabilidad del algoritmo a largas distancias. Para ello se ha buscado un terreno plano con distancias despejadas de decenas de metros y se ha probado a medir un objetivo a la vez con el dron y con un telemetro laser de la marca **Oubel** (**Figura 61**).

Figura 61. Telemetro utilizado para contrastar la distancia medida.

FUENTE: HTTPS://WWW.DESERTCART.IN/PRODUCTS/228522416-OUBEL-GOLF-RANGEFINDER-800-1200-YARDS-HUNTING-RANGEFINDER-WITH-SCAN-MODE-6-X-4-MODES-RANGEFINDER-WITH-SLOPE-FLAG-POLE-LOCKING-VIBRATION-SCAN-MODE-TRACK-PREY

En el momento de la toma de la medida la aeronave mostraba una distancia en módulo al eje de coordenadas de $\sqrt{28.8^2 + (-11.63)^2} = 31.06\,m$ tal y como se muestra en la **Figura 62**.

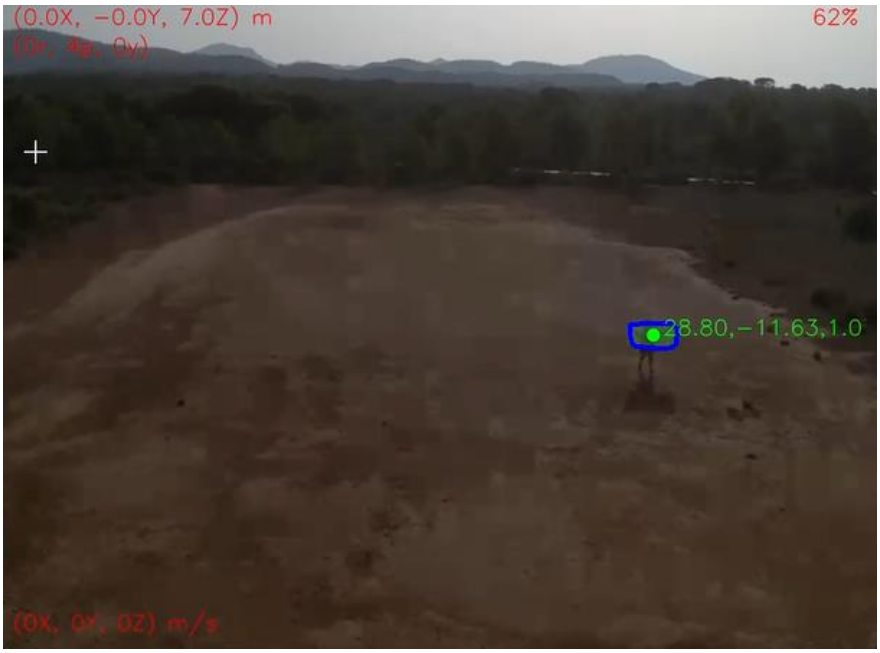


Figura 62. Medida de larga distancia desde la aeronave.

FUENTE: ELABORACIÓN PROPIA

A la vez que se tomaba esta medida, la distancia obtenida desde el telémetro mostraba 29 metros. Distancia cercana a la del telemetro y que demuestra unos **resultados validos** a una distancia considerable.

8. Conclusión

A continuación, se muestran las conclusiones obtenidas tras la realización de este trabajo. Además, se exponen algunas ideas y caminos que seguir para poder avanzar a partir del algoritmo obtenido.

Como conclusión principal, se puede decir que el resultado del proyecto ha sido satisfactorio, puesto que ha cumplido los objetivos esperados y ha ofrecido datos muy próximos a los esperados. Así pues, el proyecto ha permitido, más allá de comprender conceptos ya conocidos y aplicarlos a la programación, diseñar un nuevo algoritmo desde cero capaz de cumplir los objetivos marcados y del que se tiene la esperanza de que pueda ser desarrollado, mejorado y aplicado en un futuro en ámbitos del mundo real. Además, se ha desarrollado el algoritmo en un nuevo lenguaje de programación como lo es Python, y se ha utilizado una herramienta muy potente en el ámbito de la visión por computador, pero a la vez fácil de comprender como lo es OpenCv. Solamente con estas herramientas y algunas librerías de apoyo se han podido obtener unos resultados muy satisfactorios.

Primeramente, el trabajo se ha centrado en la obtención de los parámetros de la cámara mediante la calibración. Este proceso se ha realizado con varias cámaras para observar las diferencias (la cámara del ordenador personal, la Raspicam y la cámara del DJI Tello). Se ha podido observar que de estas tres cámaras la Raspicam era la que mayor distorsión radial tenía, y por lo tanto a la que más le afectaba la calibración y la corrección de distorsiones.

En segundo lugar, se ha centrado el proyecto en filtrar el objetivo del fondo. Este proceso empezó con un filtrado simple por color, pero acabó necesitando unos pocos procesos más para poder ofrecer un filtro fiable, tal y como se ha mostrado en el **apartado 6.1.3 y 7**.

Por último, se ha diseñado el algoritmo pertinente a partir de los conocimientos expuestos en el **apartado 3** y se ha puesto a prueba a partir de

ciertas experiencias en entornos semicontrolados las cuales han terminado ofreciendo unos **resultados favorables**.

No obstante, se han encontrado algunas limitaciones dentro del algoritmo. Es por ello por lo que los entornos deben ser **semicontrolados**, puesto que deben cumplir una característica esencial para el correcto funcionamiento. Esta característica se trata de que, puesto que el objetivo debe de encontrarse siempre a una misma altura Z del eje de referencia fijo, la superficie del entorno debe ser lo más plana posible.

En cuanto a las limitaciones ofrecidas por el material utilizado, a continuación, se exponen las más importantes:

- Alcance de la conexión debido al poco rango de la zona Wi-Fi generada.
- Poca estabilización de la imagen debido a la falta de una estabilización física o gimbal.
- Poca estabilidad de la aeronave ante ráfagas de viento debido a su peso y tamaño reducido.
- Limitación del alcance y la visión de la cámara.

Todos estos factores refuerzan la idea de que a partir de un hardware más avanzado se podrían obtener resultados todavía mejores y a mayor distancia.

Tal y como se ha expuesto en el **apartado 1.2**, la idea del proyecto surge a partir de la motivación por crear nuevas herramientas útiles para el sector de los profesionales del rescate, como pueden ser socorristas o bomberos. A raíz de esto, se plantean algunos caminos en los que se podría evolucionar el proyecto.

Una buena mejora para avanzar en el proyecto podría ser utilizar una cámara térmica. Con esta mejora, una aeronave y una conexión más estable y un sistema de posicionamiento global fiable (GPS), se podría utilizar este algoritmo para encontrar y localizar personas extraviadas en el mar, puesto que la temperatura de este, aunque cada vez más alta, sigue dando margen para diferenciar una persona por su huella térmica. No obstante, el desafío de diferencia a una persona en una masa de agua de gran tamaño con corrientes y variaciones de temperatura es bastante considerable.

Por otro lado, en el caso de la localización de focos de calor en incendios forestales, el desafío que se plantea se encuentra en el relieve presente en los terrenos donde suelen ocurrir estos. Para poder afrentar este desafío, se ha planteado como posible camino la utilización de herramientas como un telémetro laser con movimiento solidario con la cámara. De esta forma, el dato obtenido pasaría de ser la altura entre la aeronave y el objetivo el módulo de la distancia entre estos. Otro camino planteado para obtener esta misma distancia sería utilizar herramientas matemáticas para, a partir de varias imágenes de un mismo objetivo desde distintos puntos de vista, obtener este mismo módulo de la distancia (Paralaje).

9. Alineación del trabajo con los ODS

En este punto, se va a realizar una justificación sobre la relación entre el proyecto y los **Objetivos de Desarrollo Sostenible (ODS).** En la siguiente tabla se muestra el grado de relación:

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

A continuación, se hará una breve descripción de la relación del proyecto con los ODS con los que tiene más relación, es decir, el 13, el 15 y el 17:

Figura 63. ODS 13.

FUENTE: HTTPS://WWW.PACTOMUNDIAL.ORG/ODS/13-ACCION-POR-EL-CLIMA/

- ODS 13: Acción por el clima. En relación con este objetivo, el proyecto tiene mucha relación con la **meta 13.1** "Fortalecer la resiliencia y la capacidad de adaptación a los riesgos relacionados con el clima y los desastres naturales en todos los países". Esto es así puesto que el proyecto podría ser de gran ayuda para paliar las consecuencias de desastres naturales, por ejemplo, facilitando el rescate de vidas humanas.

- ODS 15: Vida de ecosistemas terrestres. El proyecto tiene una relación muy estrecha con este ODS, aunque sobre todo es importante mencionar su relación con la **meta 15.4** "Para 2030, velar por la conservación de los ecosistemas montañosos, incluida su diversidad biológica, a fin de mejorar su capacidad de proporcionar beneficios esenciales para el desarrollo sostenible". Esto es así debido a la proyección del proyecto en la lucha contra los incendios forestales.

Figura 64. ODS 15.

FUENTE: HTTPS://WWW.PACTOMUNDIAL.ORG/ODS/15-VIDA-DE-ECOSISTEMAS-TERRESTRES/

Figura 65. ODS 17.

FUENTE: HTTPS://WWW.PACTOMUNDIAL.ORG/ODS/17-ALIANZAS-PARA-LOGRAR-LOS-OBJETIVOS/

- ODS 17: Alianzas para lograr objetivos. En este ODS el proyecto se hace más notable en la **meta 17.6** "Mejorar la cooperación regional e internacional Norte-Sur, Sur-Sur y triangular en materia de ciencia, tecnología e innovación y el acceso a estas, y aumentar el intercambio de conocimientos en condiciones mutuamente convenidas, incluso mejorando la coordinación entre los mecanismos existentes, en particular a nivel de las Naciones Unidas, y mediante un mecanismo mundial de facilitación de la tecnología". Este proyecto, además de necesitar la colaboración de múltiples disciplinas para avanzar, pretende ser una idea que inspire a la creación de tecnologías mucho más avanzadas capaces de salvar vidas y ecosistemas.

10. Bibliografía

Armesto Ángel, L. (2023). Asignatura de Sistemas Robotizados, «*Presente, Pasado y Futuro*». Universidad Politécncia de Valencia; Presentaciones y pdf.

Barrientos Sotelo, V. R., García Sánchez, J. R., & Silva Ortigoza, R. (2007). Robots Móviles: Evolución y Estado del Arte. Polibits, (35), 12-17.

Bynau. (2021). ¿Qué es la distancia focal? Recuperado el 06 de julio del 2023, de https://bynau.com/que-es-la-distancia-focal/

Cut the knot, (s. f.). Cartesian Coordinate System. Recuperado el 05 de julio del 2023, de http://www.cut-the-knot.org/Curriculum/Calculus/Coordinates.shtml

De Blois, A. (s. f.). ¿Qué es una cámara estenopeica? Todo sobre fotografía estenopeica. Blog del fotógrafo. Recuperado el 06 de julio de 2023, de https://www.blogdelfotografo.com/camara-estenopeica/

Dronecasero. (s. f.). Qué es el FAILSAFE. Recuperado el 05 de julio del 2023, de http://dronecasero.blogspot.com/2014/08/que-es-el-failsafe.html

El blog de la fotografía. (s. f.). Distancia focal de tu objetivo: ¿Qué es? Recuperado el 06 de julio del 2023, de https://www.elblogdelafotografia.com/distancia-focal/

Eliteradiocontrol. (s. f.). Drones híbridos, usos y capacidades para estos super drones. Recuperado el 03 de Julio del 2023, de https://eliteradiocontrol.com/drones/drone-hibrido/

González Sorribes, A. (2023). Asignatura de Robótica Aérea, «Introducción a los UAVs». Universidad Politécncia de Valencia; Presentaciones y pdf.

González Sorribes, A. (2023). Asignatura de Robótica Aérea, «Aviónica y componentes básicos». Universidad Politécncia de Valencia; Presentaciones y pdf.

González Sorribes, A. (2023). Asignatura de Robótica Aérea, «Modelado cinemático y localización espacial». Universidad Politécncia de Valencia; Presentaciones y pdf.

Mathworks. (s. f.). What is camera calibration? Recuperado el 10 de julio del 2023, de https://es.mathworks.com/help/vision/ug/camera-calibration.html

Matplotlib. (s. f.). Matplotlib: Visualization with Python. Recuperado el 13 de julio del 2023, de https://matplotlib.org/

Nielsen, N. (2021). Learn Camera Calibration in Python with OpenCV: Complete Step-by-Step Guide with Python Script. Recuperado el 17 de julio del 2023, de https://www.youtube.com/watch?v=3h7wgR5fYik

Numpy. (s. f.). What is NumPy? Recuperado el 13 de julio del 2023, de https://numpy.org/doc/stable/user/whatisnumpy.html

OpenCV. (s. f.). Camera calibration an 3D reconstruction. Recuperado el 10 de julio del 2023, de https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html

OpenCV. (s. f.). Optical flow. Recuperado el 11 de julio del 2023, de https://docs.opencv.org/3.4/d4/dee/tutorial_optical_flow.html

OpenCV. (s. f.). About. Recuperado el 13 de julio del 2023, de https://opencv.org/about/

Python. (s. f.). What is Python? Executive summary. Recuperado el 12 de julio del 2023, de https://www.python.org/doc/essays/blurb/

Python. (s. f.). La biblioteca estándar de Pyhton. Recuperado el 13 de julio del 2023, de https://docs.python.org/es/3/library/index.html

Real Academia Española. (s. f.). Avionica. Recuperado el 03 de julio del 2023, de https://dle.rae.es/avi%C3%B3nica

Real Academia Española. (s. f.). Óptico. Recuperado el 12 de julio del 2023, de https://dle.rae.es/avi%C3%B3nica

Rodas Jordá, A. (2023). Asignatura de Robótica Inteligente «Segmentación de imagenes». Universidad Politécncia de Valencia; Presentaciones y pdf.

Rodas Jordá, A. (2023). Asignatura de Robótica Inteligente «Procesamiento de imagenes». Universidad Politécncia de Valencia; Presentaciones y pdf.

Rodriguez, A. (2023). Robots subacuaticos, qué son y cómo funcionan. Naova. Recuperado el 29 de junio de 2023, de https://nanova.org/robots-subacuaticos-que-son-y-como-funcionan/

Sympy. (s. f.). About. Recuperado el 13 de julio del 2023, de https://www.sympy.org/en/index.html

Visual Studio Code. (s. f.). Getting started. Recuperado el 13 de julio del 2023, de https://code.visualstudio.com/docs#vscode

Wayback Machine. (s. f.). Principio de funcionamiento del IMU. Recuperado el 05 de julio del 2023, de https://web.archive.org/web/20171107221625/http:/www.technaid.com/es/soporte/investigacion/principio-de-funcionamiento-del-imu/

Zhang, Z. (2008). A flexible new technique for camera calibration. Microsoft. Recuperado el 10 de julio del 2023, de https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/tr98-71.pdf

11. Apéndices

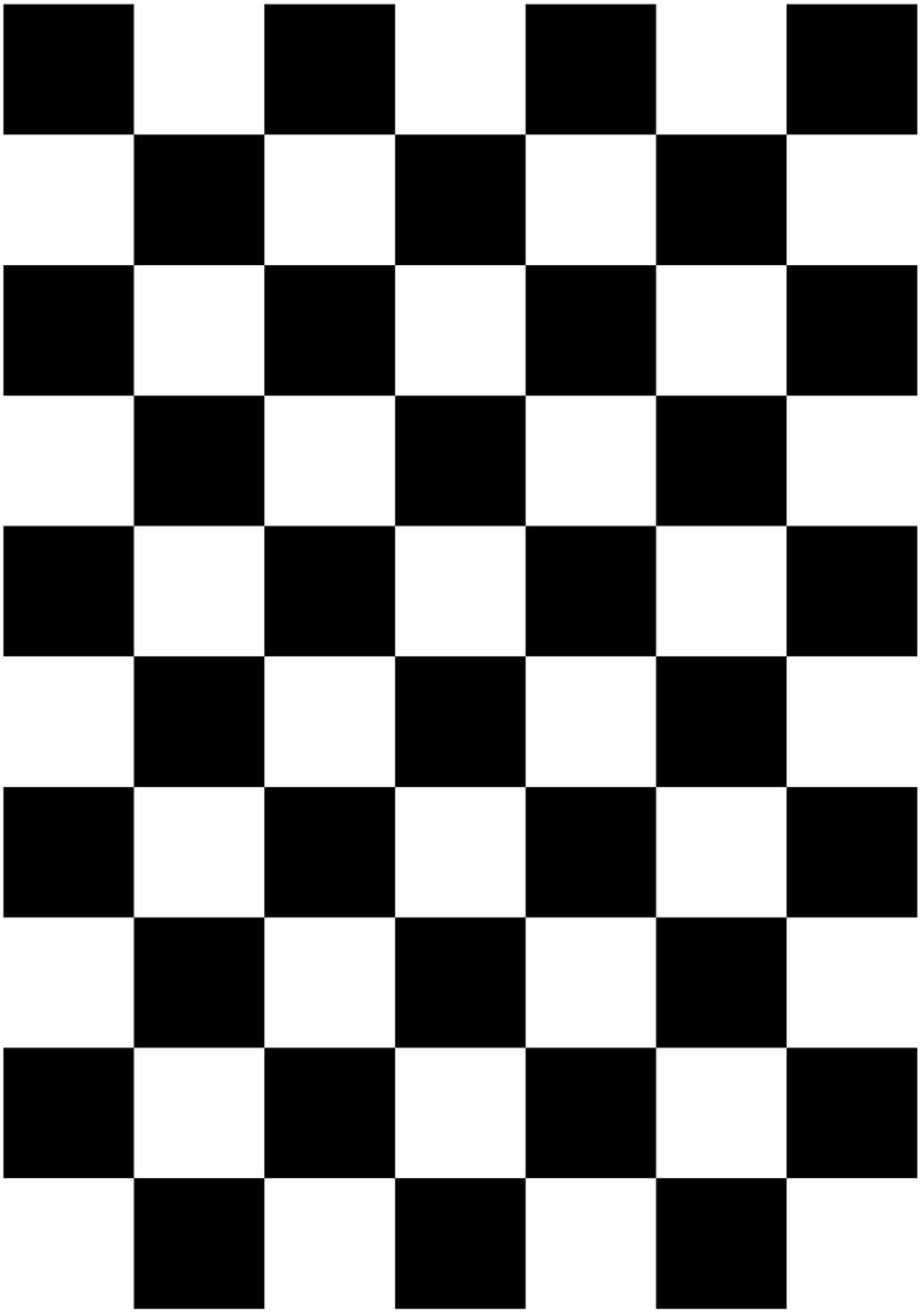
11.1.Índice de figuras

11.2.Índice de códigos

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Anexo I. Patrón de calibración utilizado para calibrar la cámara. Chessboard.



UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Anexo II. Código completo para la obtención de imágenes con Python. GetImage.py

```
################################################
##### CÓDIGO PARA GUARDAR IMAGENES EN PYTHON #####
################################################

from djitellopy import Tello
import cv2

tello = Tello()

# Conexión con el dron e inicialización de transmisión de
imagenes
tello.connect()
tello.streamon()
num = 0 # Número de la primera foto

while True:
    frame = tello.get_frame_read().frame
    k = cv2.waitKey(5)  # Se le asigna a la variable k la tecla
que se pulse

    # Si esta tecla es "esc" se romperá el bucle
    if k == 27:
        break

    # Si es "s" se guardará una imagen png del frame en una
carpeta llamada "Imagenes" con el número de la imagen
    elif k == ord("s"):
        cv2.imwrite("Imagenes/img" + str(num) + ".png", frame)
        print("image saved!")
        num += 1

    cv2.imshow(
        "Img",
        frame,  # Muestra en la pantalla la imagen que está
obteniendo en tiempo real.
    )

# Se liberán y destruyen todas las ventanas antes de terminar el
código.
cv2.destroyAllWindows()
```

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Anexo III. Código completo para la calibración de la cámara con Python.
Calibration.py

```
#####################################################################
##### CÓDIGO PARA CALIBRAR LA CÁMARA CON TABLERO DE AJEDREZ ####
#####################################################################

import numpy as np
import cv2 as cv
import glob
import pickle

# OBTENCIÓN DE OBJECT POINTS Y IMAGE POINTS A PARTIR DEL TABLERO
DE AJEDREZ

chessboardSize = (
    9,
    6,
)  # Tamaño del tablero de ajedrez, contando las esquinas
comunes entre cuatro cuadrados
frameSize = (960, 720)  # Tamaño del frame

# termination criteria
criteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER,
30, 0.001)

# Prepara la matriz de object points
objp = np.zeros((chessboardSize[0] * chessboardSize[1], 3),
np.float32)
objp[:, :2] = np.mgrid[0 : chessboardSize[0], 0 :
chessboardSize[1]].T.reshape(-1, 2)

size_of_chessboard_squares_mm = 25
objp = objp * size_of_chessboard_squares_mm

# Vector para almacenar los object points y los image points de
todas las imagenes
objpoints = []  # Puntos 3D en el mundo real
imgpoints = []  # Puntos 2D en el plano imagen

# Almacena la ruta a todas las imagenes guardadas previamente en
la carpeta Imagenes
images = glob.glob("Imagenes/*.png")
```

```
for (
    image
) in (
    images
):  # Analiza las imagenes contenidas en las rutas de la
variable "images" una a una dentro de este for
    img = cv.imread(image)
    gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)  # Convierte la
imagen a escala de grises

    # Busca las esquinas del tablero
    ret, corners = cv.findChessboardCorners(gray,
chessboardSize, None)

    # Si las encuentra añade los object points correspondientes
y los image points tras refinar los primeros
    if ret == True:
        objpoints.append(objp)
        corners2 = cv.cornerSubPix(gray, corners, (11, 11), (-1,
-1), criteria)
        imgpoints.append(corners2)

        # Muestra las esquinas en una imagen
        cv.drawChessboardCorners(img, chessboardSize, corners2,
ret)
        cv.imshow("img", img)
        cv.waitKey(
            1000
        )  # Espera un segundo, esto solo sirve para dar tiempo
a ver la imagen

cv.destroyAllWindows()
```

```
# CALIBRACIÓN

(
    ret,
    cameraMatrix,
    dist,
    rvecs,
    tvecs,
) = cv.calibrateCamera(  # Obtiene la matriz de calibración y el
vector de características
    objpoints,
    imgpoints,
    frameSize,
    None,
    None,  # La función necesita como entrada los object points
e image points obtenidos previamente
)
print("cameraMatrix: {}".format(cameraMatrix))
print("dist: {}".format(dist))

# Guarda los resultados de la calibración en archivos .plk para
poder utilizarlos más adelante en otros códigos
pickle.dump((cameraMatrix, dist), open("calibration.pkl", "wb"))
pickle.dump(cameraMatrix, open("cameraMatrix.pkl", "wb"))
pickle.dump(dist, open("dist.pkl", "wb"))

# CORRECCIÓN DE DISTORSIÓN

img = cv.imread(
    "Imagenes/img5.png"
)  # Se lee una imagen para poder realizar una comprobación
visual de la corrección de distorsión y se comprueba su tamaño
h = img.shape[0]
w = img.shape[1]

(
    newCameraMatrix,
    roi,
) = cv.getOptimalNewCameraMatrix(  # Se refina la matriz de
calibración y se almacena en "newCameraMatrix"
    cameraMatrix, dist, (w, h), 1, (w, h)
)
```

```python
# Undistort
dst = cv.undistort(
    img, cameraMatrix, dist, None, newCameraMatrix
)  # Se corrige la distorsión de la imagen con "undistort" para la comprobación visual

# Se recorta la imagen puesto que al corregir la distorsión se deforman los bordes de la imagen
x, y, w, h = roi
dst = dst[y : y + h, x : x + w]
cv.imwrite("caliResult1.png", dst)

# Cálculo del error de reproyección
mean_error = 0

for i in range(len(objpoints)):
    imgpoints2, _ = cv.projectPoints(
        objpoints[i], rvecs[i], tvecs[i], cameraMatrix, dist
    )
    error = cv.norm(imgpoints[i], imgpoints2, cv.NORM_L2) / len(imgpoints2)
    mean_error += error

print("total error: {}".format(mean_error / len(objpoints)))
```

Anexo IV. Código completo para la detección y localización de objetos remotos mediante visión por computador con Python. Final.py

```
#################################################################
######################
##### CODIGO PARA LA IDENTIFICACION DEL OBJETIVO Y EL CALCULO DE
SU LOCALIZACION #####
#################################################################
######################

import numpy as np
import cv2 as cv
import pickle
import sympy as sp
import math
import matplotlib.pyplot as plt
import time
from djitellopy import Tello

# Estableciendo conexion con el Tello
tello = Tello()

# Inicialización de variables de control de vuelo
offset_yaw = 0
t_old = 0
periodo = 500000000  # Tiempo entre cada bucle main (ns)

tello.connect()
tello.streamon()

# Parametros intrinsecos de la camara obtenidos de las
especificaciones tecnicas de la camara del dron en metros
f = 4.2 / 1000  # Longitud Focal    4.2
H = 6.17 / 1000  # Tamano del sensor Horizontal 1/2.3"
V = 4.55 / 1000  # Tamano del sensor vertical 1/2.3"

# Altura del objeto con respecto de los ejes de referencia E
Altura = 10 / 100  # en metros

# Creamos listas vacía para almacenar los valores de 'x' y de
'y' del objetivo y de la posición de la aeronave
x_values = []
y_values = []
```

```
# Funcion PinHole, traduce las coordenadas de la camara al mundo
real
def pinHole(pixw, pixh, kw, kh, f, Z, pose):
    # Valores obtenidos fuera de la funcion (Tamano en pixeles
de la imagen)
    global numPixY, numPixX

    # Definicion de los valores simbolicos necesarios
    X, Y, yc = sp.symbols("X Y yc")

    # Calculos de parametros de la camara y coordenadas del
punto imagen
    dh = pixh - (numPixY / 2)
    dw = pixw - (numPixX / 2)
    yp = dh / kh
    xp = dw / kw

    # Declaracion de las matrices simbolicas necesarias para la
transformacion de coordenadas imagen a coordenadas mundo

    EP = sp.Matrix([X, Y, Z, 1])
    ETe = sp.Matrix(
        [[0, 0, 1, 0], [-1, 0, 0, 0], [0, -1, 0, 0], [0, 0, 0,
1]]
    )  # Giro para orientar ejes mundo a ejes dron de referencia
    eTb0 = sp.Matrix(
        [[1, 0, 0, pose[0]], [0, 1, 0, pose[1]], [0, 0, 1,
pose[2]], [0, 0, 0, 1]]
    )  # Translacion de nuevos los ejes al body del dron
    b0Tb1 = sp.Matrix(  # Rotacion roll
        [
            [math.cos(pose[3]), -math.sin(pose[3]), 0, 0],
            [math.sin(pose[3]), math.cos(pose[3]), 0, 0],
            [0, 0, 1, 0],
            [0, 0, 0, 1],
        ]
    )
```

```
    b1Tb2 = sp.Matrix(  # Rotacion pitch
        [
            [1, 0, 0, 0],
            [0, math.cos(pose[4]), -math.sin(pose[4]), 0],
            [0, math.sin(pose[4]), math.cos(pose[4]), 0],
            [0, 0, 0, 1],
        ]
    )
    b2Tc = sp.Matrix(  # Rotacion yaw
        [
            [math.cos(pose[5]), 0, math.sin(pose[5]), 0],
            [0, 1, 0, 0],
            [-math.sin(pose[5]), 0, math.cos(pose[5]), 0],
            [0, 0, 0, 1],
        ]
    )
    ETc = ETe * eTb0 * b0Tb1 * b1Tb2 * b2Tc

    cP = ETc.inv() * EP

    # Ecuaciones simbolicas que relacionan las coordenadas mundo
con las coordenadas imagen
    xc = cP[0]
    yc = cP[1]
    zc = cP[2]

    # Resolucion de las ecuaciones a partir de las formulas del
modelo PINHOLE
    sol = sp.solve([(f * yc / zc) - yp, (f * xc / zc) - xp],
dict=True)
    # print(sol)

    # Llamamos a la función para actualizar la gráfica del
objetivo
    update_graph(sol[0][X], sol[0][Y])
    # Forzamos la actualización de las gráficas
    plt.draw()
    plt.pause(0.001)
    # Se devuelven las coordenadas mundo y se dibujan
    return sol[0][X], sol[0][Y], Z
```

```python
# Definimos la función para actualizar la gráfica
def update_graph(X, Y):
    # Agregamos el valor de 'x' y de 'y' a la lista
    x_values.append(X)
    y_values.append(Y)

    # Marcas en el suelo en metros
    marcas_x = [0, 20, 46, 20]
    marcas_y = [0, -5, 0, 5]

    # Recorrido predefinido
    x_predef = [0, 20, 46, 20, 0]
    y_predef = [0, -5, 0, 5, 0]

    # Actualizamos la gráfica con los nuevos datos
    ax1.plot(x_values, y_values, "b-")
    ax1.plot(x_predef, y_predef, "r-")
    ax1.scatter(marcas_x, marcas_y, color="red", marker="v")

    ax1.set_aspect("equal")

    # Establecemos los límites de los ejes 'x' e 'y' en metros
    ax1.set_xlim(0, 50)
    ax1.set_ylim(-10, 10)

    # Título y etiquetas de los ejes
    ax1.set_title("Objetivo")
    ax1.set_xlabel("X (metros)")
    ax1.set_ylabel("Y (metros)")


def CentroGravedad(Contorno):
    M = cv.moments(Contorno)
    if M["m00"] == 0:
        M["m00"] = 1
    u = int(M["m10"] / M["m00"])
    v = int(M["m01"] / M["m00"])
    return u, v
```

```
# Recuperacion de los datos de la calibracion obtenidos
previamente y guardados en archivos .pkl
cameraMatrix = pickle.load(open("cameraMatrix.pkl", "rb"))
dist = pickle.load(open("dist.pkl", "rb"))

# Definicion de umbrales de color para detectar el objetivo en
HSV, kernel para apertura y cierre y fuente para escritura
# Azul
colorBajo = np.array([100, 100, 70], np.uint8)
colorAlto = np.array([125, 255, 255], np.uint8)

# Verde
#colorBajo = np.array([30, 50, 70], np.uint8)
#colorAlto = np.array([60, 255, 255], np.uint8)

# Morado
#colorBajo = np.array([130, 80, 70], np.uint8)
#colorAlto = np.array([160, 255, 255], np.uint8)

# Rosa
#colorBajo = np.array([140, 50, 70], np.uint8)
#colorAlto = np.array([160, 255, 255], np.uint8)

kernel = cv.getStructuringElement(cv.MORPH_ELLIPSE, (10, 10))
kernel2 = cv.getStructuringElement(cv.MORPH_ELLIPSE, (20, 20))
font = cv.FONT_HERSHEY_SIMPLEX

# Habilitamos el modo interactivo de la grafica
plt.ion()

# Creamos la figura para las gráficas
fig, ax1 = plt.subplots()

# Mostramos las gráficas en una ventana
plt.show()
```

```
# Bucle principal
while True:
    k = cv.waitKey(5)
    # Despega el dron
    if k == ord("t"):
        tello.takeoff()
    if k == ord("o"):
        offset_yaw = tello.get_yaw()

    # Obtencion de la IMU
    imu_r = tello.get_roll()
    imu_p = tello.get_pitch()
    imu_y = tello.get_yaw() - offset_yaw

    t_old = time.time_ns()

    # Pose de la camara con respecto de los ejes de referencia e
(distancias en metros, angulos en radianes)
    xb = 0
    yb = -tello.get_distance_tof()/100
    zb = 0
    roll = imu_r * math.pi / 180
    pitch = (-15 + imu_p) * math.pi / 180
    yaw = imu_y * math.pi / 180

    pose = np.array([xb, yb, zb, roll, pitch, yaw])

    frame = tello.get_frame_read().frame

    # Obtencion del tamano de la imagen
    numPixY = frame.shape[0]
    numPixX = frame.shape[1]

    newCameraMatrix, roi = cv.getOptimalNewCameraMatrix(
        cameraMatrix, dist, (numPixX, numPixY), 1, (numPixX,
numPixY)
    )
```

```
    # Corrección de la distorsion de la imagen
    dst = cv.undistort(frame, cameraMatrix, dist, None,
newCameraMatrix)

    # Recorte de la imagen y obtencion del nuevo tamano
    x, y, numPixX, numPixY = roi
    dst = dst[y : y + numPixY, x : x + numPixX]
    numPixY = dst.shape[0]
    numPixX = dst.shape[1]

    # Calculo de kh y kw necesarios para la realizacion de
PinHole
    kw = numPixX / H
    kh = numPixY / V

    # Cambio de espacio de color para facilitar la umbralizacion
y calculo de contornos del objeto
    frameHSV = cv.cvtColor(dst, cv.COLOR_BGR2HSV)
    mask = cv.inRange(frameHSV, colorBajo, colorAlto)
    mask2 = cv.morphologyEx(mask, cv.MORPH_OPEN, kernel)
    mask2 = cv.morphologyEx(mask2, cv.MORPH_CLOSE, kernel2)
    contornos, _ = cv.findContours(mask2, cv.RETR_EXTERNAL,
cv.CHAIN_APPROX_SIMPLE)

    # Se reinician las variables que marcan el area y el
contorno más relevante en cuanto al tamaño
    mayor_contorno = None
    mayor_area = 0

    # para no dibujar todos los contornos y evitar ruidos:
    for c in contornos:  # Miramos los contornos uno por uno
        area = cv.contourArea(c)
        if (
            area > 500 and area > mayor_area
        ):  # comprobamos que el area es mayor a un valor límite
y al area detectanda de mayor valor
            mayor_area = area
            mayor_contorno = c
```

```
    if (
        mayor_contorno is not None
    ):  # Se comprueba que hay algún contorno que ha superado el
filtro
        nuevoContorno = cv.convexHull(mayor_contorno)  # Suaviza
el contorno filtrado
        # Calculo del centro de gavedad del contorno
        u, v = CentroGravedad(nuevoContorno)

        if yb < 0:  # Si el dron está en el aire
            # Llamada a la funcion PINHOLE declarada al
principio del codigo
            X, Y, Z = pinHole(u, v, kw, kh, f, Altura, pose)

        cv.drawContours(
            dst, [nuevoContorno], 0, (255, 0, 0), 3
        )  # Dibujamos solo los contornos que cumplen esta
especificacion

        # Dibujar el centro de gravedad y sus coordenadas
        cv.circle(dst, (u, v), 7, (0, 255, 0), -1)
        cv.putText(
            dst,
            "{},{},{}".format(round(X, 2), round(Y, 2), round(Z,
2)),
            (u + 10, v),
            font,
            0.75,
            (0, 255, 0),
            1,
            cv.LINE_AA,
        )
```

```
    # Escribir en la esquina superior izquierda las coordenadas
del dron y su orientación
    cv.putText(
        dst,
        "({}X, {}Y, {}Z) m".format(
            round(pose[2], 2), round(-pose[0], 2), round(-
pose[1], 2)
        ),
        (10, 20),
        font,
        0.75,
        (0, 0, 255),
        1,
        cv.LINE_AA,
    )
    cv.putText(
        dst,
        "({}r, {}p, {}y)".format(round(imu_r, 2), round(imu_p,
2), round(imu_y, 2)),
        (10, 50),
        font,
        0.75,
        (0, 0, 255),
        1,
        cv.LINE_AA,
    )

    # Escribir en la esquina superior derecha la bateria del
dron
    cv.putText(
        dst,
        "{}%".format(tello.get_battery()),
        (840, 20),
        font,
        0.75,
        (0, 0, 255),
        1,
        cv.LINE_AA,
    )
```

```
    # Mostrar imagen
    cv.imshow("Camara", dst)
    #cv.imshow("Mascara", mask)
    #cv.imshow("Mascara refinada", mask2)

    # Si se pulsa la tecla esc se rompe el bucle y acaba el
codigo
    if k == 27:
        # tello.land()
        break
    # Temporalización de los cálculos
    dt = time.time_ns() - t_old  # nsec
    if periodo > dt:
        time.sleep((periodo - dt) / 1000000000)
        print("Tiempo de procesamiento {} nsec", dt /
1000000000)
    else:
        print("Error: tiempo de cálculo demasiado alto")

plt.savefig("Objetivo.png")  # Guarda una imagen del recorrido
del objetivo
cv.destroyAllWindows()
```

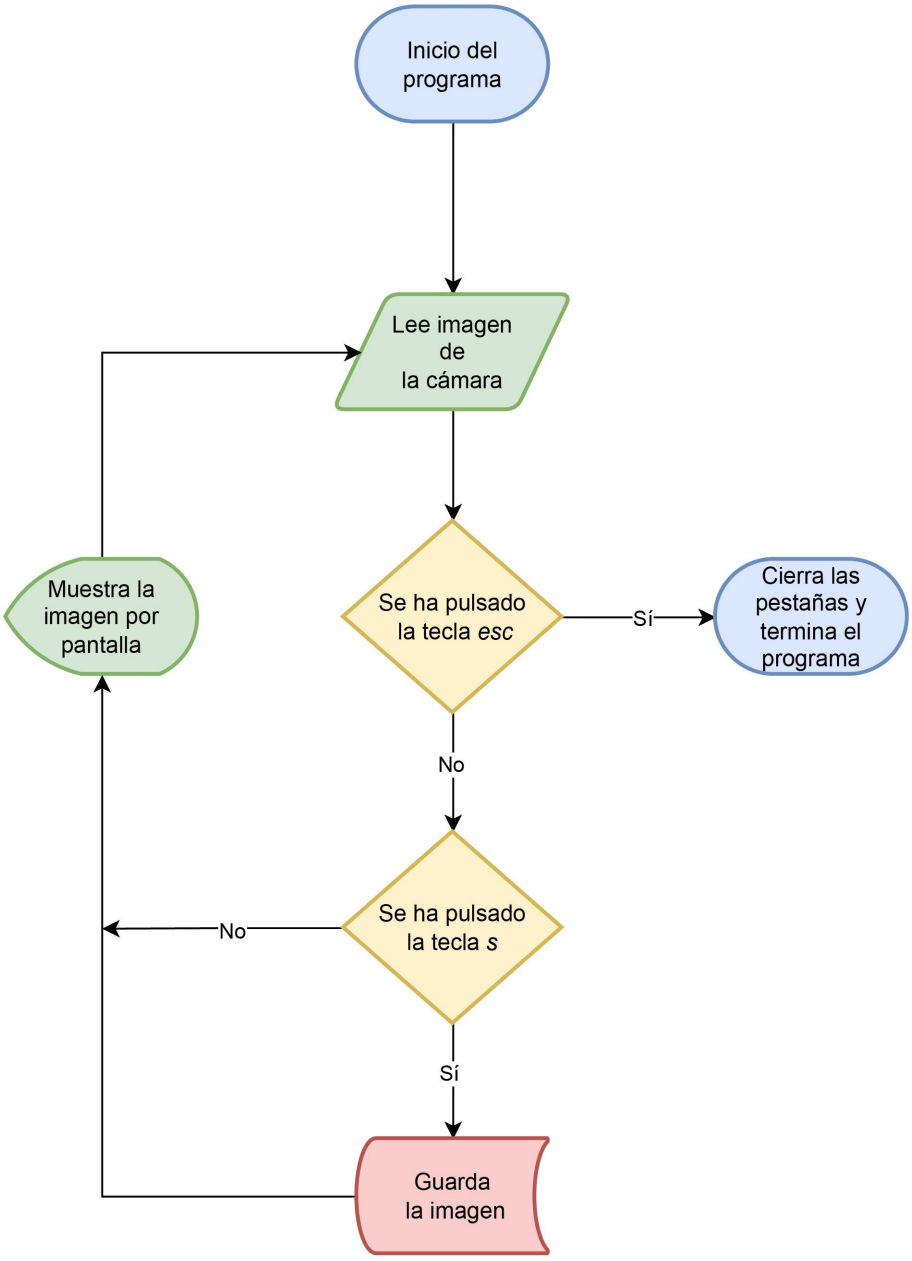
UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Trabajo Fin de Grado

PLANOS

ÍNDICE DE LOS PLANOS



TÍTULO DEL PROYECTO:
Detección y localización de objetos remotos mediante visión por computador en vehículos aéreos.

AUTOR DEL PLANO: Miquel Ballester Sanchis

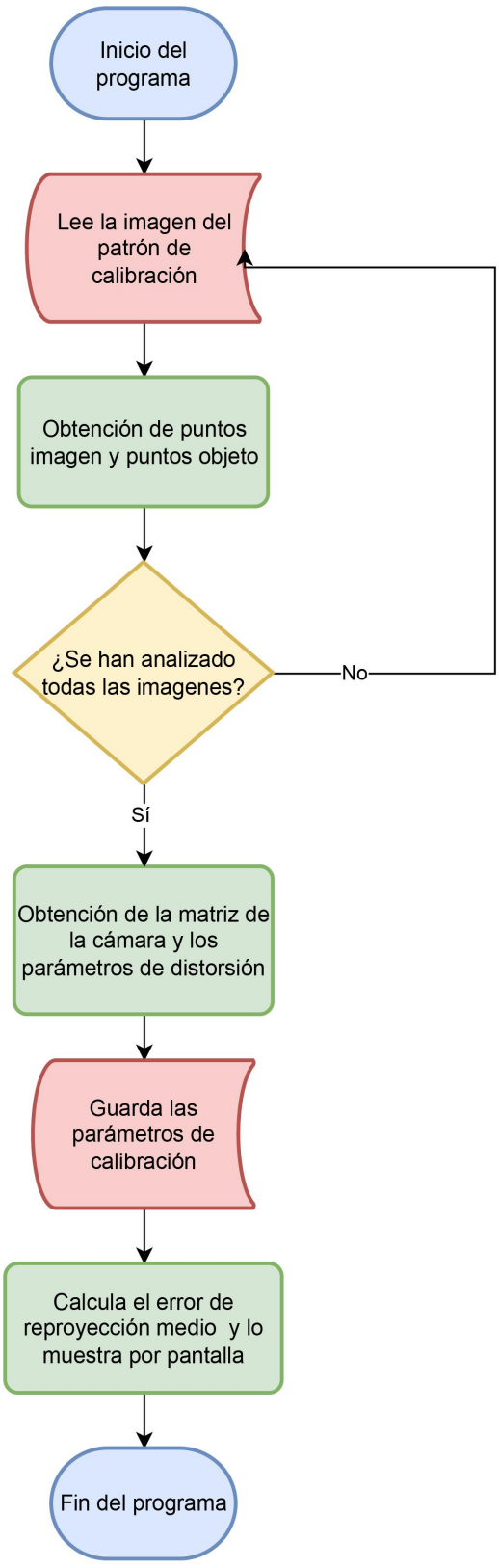
FECHA: 20/07/2023

ESCALA: S/E

Escuela Técnica Superior de Ingeniería del Diseño

TÍTULO DEL PLANO: Flujograma GetImage.py

Nº PLANO: 01



TÍTULO DEL PROYECTO:
Detección y localización de objetos remotos mediante visión por computador en vehículos aéreos.

AUTOR DEL PLANO: Miquel Ballester Sanchis

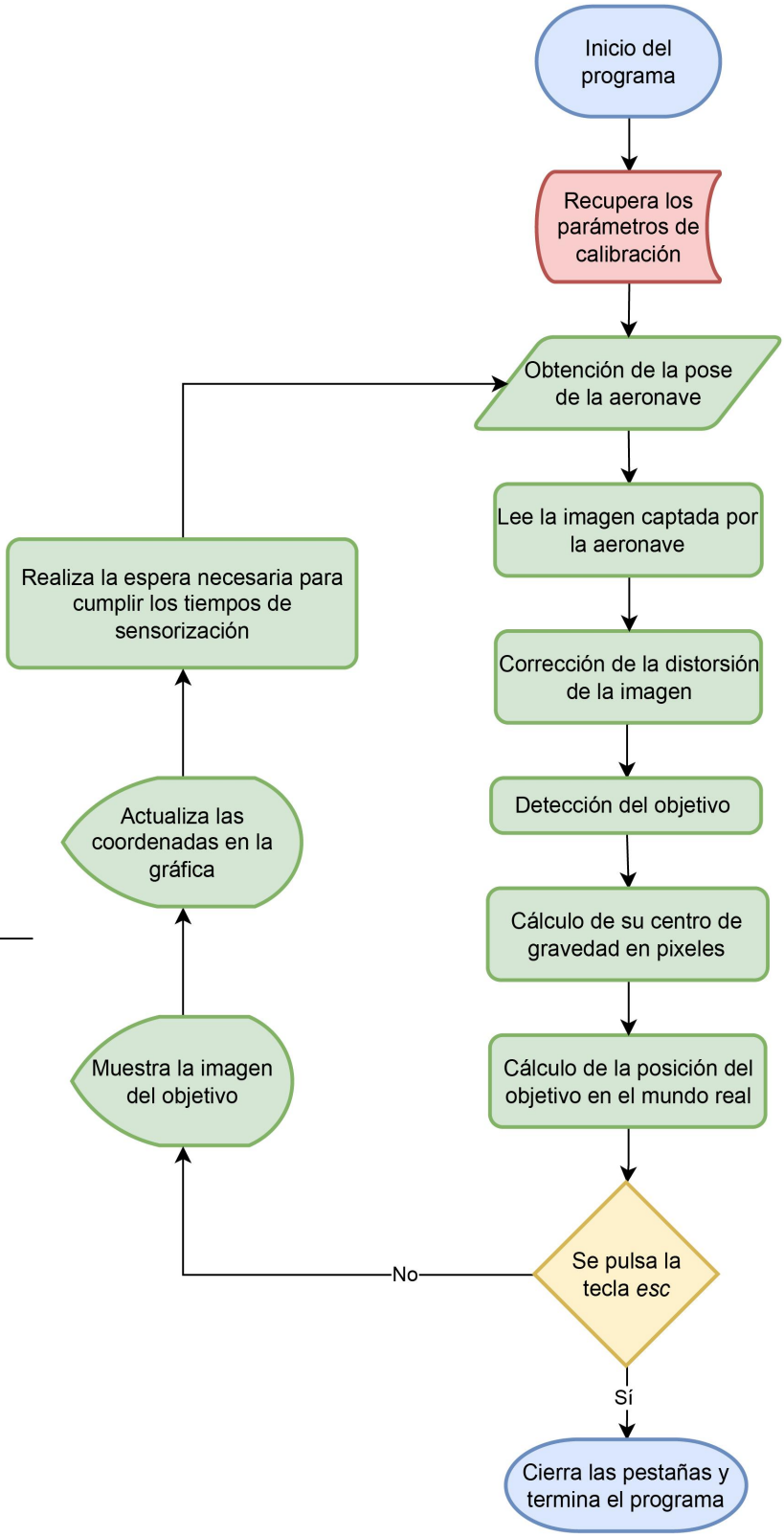
FECHA: 20/07/2023

ESCALA: S/E

Escuela Técnica Superior de Ingeniería del Diseño

TÍTULO DEL PLANO: Flujograma Calibration.py

Nº PLANO: 02



TÍTULO DEL PROYECTO:
Detección y localización de objetos remotos mediante visión por computador en vehículos aéreos.

AUTOR DEL PLANO: Miquel Ballester Sanchis

FECHA: 20/07/2023

ESCALA: S/E

Escuela Técnica Superior de Ingeniería del Diseño

TÍTULO DEL PLANO: Flujograma Final.py

Nº PLANO: 03

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Trabajo Fin de Grado

PLIEGO DE CONDICIONES

ÍNDICE DEL PLIEGO DE CONDICIONES

1. Objeto

El presente documento tiene como objetivo exponer los diferentes puntos a tener en cuenta en la realización y puesta a prueba del algoritmo diseñado para la localización de objetos remotos mediante vehículos aéreos. Algunos puntos a tener en cuenta son las especificaciones técnicas de los componentes del proyecto, así como las características que se deben cumplir en la realización de las pruebas y las normativas a tener en cuenta.

2. Normativa

Este proyecto debe realizarse teniendo en cuenta las siguientes normativas:

- Normativa europea (Reglamento 2019/947 y Reglamento 2019/945) la cual regula el uso de drones y cuyo organismo encargado de hacerla cumplir en España es la Agencia Estatal de Seguridad Aérea (AESA).
- Reglamento (UE) 2016/679 del parlamento europeo y del consejo de 27 de abril de 2016 (Reglamento General de Protección de Datos). En este documento se recoge todo aquello relativo a la protección de las personas físicas en lo que respecta al tratamiento de datos personales.

Además de estas normativas es muy importante leer la “Renuncia de responsabilidad y directrices de seguridad” del dron Tello.

3. Especificaciones de materiales

En este apartado se desarrollan las especificaciones técnicas de los materiales tecnológicos utilizados.

- Ordenador **HP series 15s-fq1xxx** con un sistema operativo Windows 11, procesador Intel Core i3 de decima generación, 8 Gb de RAM y conexión Wi-Fi 5G.
- Teléfono **Xiaomi Redmi Note 10S** con Android 13 y MIUI 14.0.2, 6+2 Gb de RAM, 128 Gb de almacenamiento y Wi-Fi 5G.

- Dron DJI/Ryze **Tello** de aproximadamente 80 gr de peso, con Sensor telemétrico, barómetro, LED, sistema de visión y Wi-Fi 2.4.
- **Python 3.11.1** con todas las librerías necesarias instaladas.
- App **TELLO 1.6.4.0** de Shenzhen RYZE Tech Co.Ltd.

4. Condiciones de ejecución

Es importante para la ejecución de las pruebas que se cumplan algunas características en el entorno:

- La superficie sobre la que se vuela el dron y en la que descansa el objetivo debe ser plana y sin inclinación. Esto es debido a que un dato que debe conocerse obligatoriamente es la altura del objetivo con respecto del sistema de referencia. Al establecer una superficie plana, esta altura siempre será la misma y se podrá fijar el dato.
- La superficie sobre la que se sitúe el dron no puede ser lisa, reflectante o estar en movimiento, puesto que si no se cumple esta característica el sistema de posicionamiento visual de la aeronave no funcionará debidamente.
- Debe de haber una buena iluminación de la escena. Si no se cumple esta característica, el sistema de posicionamiento visual fallará y además es probable que el sistema de reconocimiento de objetivos también falle debido a la poca distinción del color en baja iluminación.
- La meteorología debe de ser favorable en exteriores. La aeronave necesita unas condiciones favorables para poder mantener su estabilidad y dar buenas prestaciones.

5. Prueba de servicio

5.1. Prueba de conexión con la estación tierra

Esta prueba se puede realizar con el código GetImage.py del Anexo II.

El primer paso a realizar es el encendido de la aeronave pulsando el botón de encendido ubicado en la parte lateral derecha de la aeronave. En la

parte frontal de esta se encenderá un LED en distintos colores hasta que se quede parpadeando en amarillo. Una vez suceda esto se deberá buscar la Wifi con nombre TELLOXXXX en el teléfono. Es importante que cuando se realice esta conexión el teléfono debe estar desconectado de las redes móviles. Una vez realizada la conexión se abrirá la aplicación Tello y se podrá observar la imagen obtenida con el dron. A continuación, se procede a conectar el ordenador a la misma WIFI y ejecutar el código mencionado. Cuando se ejecute el código, la imagen mostrada en el teléfono se pondrá en negro y se podrá ver en el ordenados. Es importante tener en cuenta que aunque en el teléfono no se pueda observar la imagen, este sigue pudiendo controlar la aeronave.

5.2. Prueba de calibración.

Para realizar esta prueba se deben haber obtenido las imágenes del patrón de calibración en distintos ángulos y posiciones previamente mediante el programa GetImage.py.

Una vez obtenidas estas imágenes se ejecutará el código Calibration.py (Anexo III) el cual tras realizar la calibración pertinente mostrará el error de reproyección medio obtenido. Un error de reproyección de menos de 0.1 pixeles es considerado un muy buen resultado. Hay que tener en cuenta que esto también depende de la resolución de la imagen puesto que, a mayor resolución, mayor puede ser el error de reproyección.

5.3. Prueba de localización y seguimiento.

Para la realización de esta prueba se utilizará el código Final.py (Anexo IV). Es importante que esta prueba se realice tras las dos pruebas expuestas anteriormente y en las condiciones adecuadas expuestas en el apartado de *Condiciones de Ejecución.*

Para la realización de esta prueba es importante marcar unos ejes globales con la misma orientación que los ejes de referencia globales (E). Una vez marcada la orientación y el origen de estos se deben marcar unos puntos fáciles de diferenciar en estos ejes teniendo en cuenta que las mejores distancias para marcarlos son, en X entre 2 y 10 metros y en Y entre -3 y 3

metros aproximadamente (estos valores dependen de la altura a la que se eleve el dron, en este caso a una altura de 2 metros). Con estos puntos fijados se debe trazar un recorrido a realizar que pase por estos puntos y los una en línea recta. Con los puntos y el recorrido marcados, estos se deben asignar a los vectores pertinentes (*marcas_x/marcas_y* y *x_predef/y_predef*) de la función *update_graph().* Es importante también añadir la altura a la que se ubicará el centro de gravedad del objetivo respecto del sistema en la variable global *Altura*.

Por último, se realizará la conexión tal y como se ha explicado en la *Prueba de conexión con la estación tierra*, se elevará el dron hasta la altura pertinente (2 metros) sin desplazarlo horizontalmente y se ejecutará el programa. Es importante que durante la prueba el ángulo yaw 0 se cumpla cuando la cámara está alineada con el eje X del sistema de referencia E, si esto no se cumple, se debe alinear y asignar yaw 0 a esa posición mediante la tecla “o”. El último paso es realizar el recorrido designado entre los puntos de referencia con el objetivo, recordando en todo momento que este se debe mantener siempre a la misma altura.

Una vez terminada la prueba, se termina el programa pulsando la tecla “esc”. Esto guardará la gráfica que superpone del recorrido detectado al predefinido como “Objetivo.png”, facilitando así su análisis a posteriori.

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Trabajo Fin de Grado

PRESUPUESTO

ÍNDICE DEL PRESUPUESTO

1. Cuadro de precios elementales

Para la realización de este documento se han recopilado todos los materiales necesarios, así como el software utilizado y la retribución de la mano de obra en la siguiente tabla de precios elementales. No se ha realizado una descomposición mayor de materiales como la aeronave o el vehículo terrestre puesto que para este proyecto se han adquirido ya ensamblados.

1. Cuadro de precios elementales			
Ref	ud.	Descripción	Precio (€)
Materiales y Software			
m1	ud.	Dron DJI Tello Boost Combo	169,00
m2	ud.	Portátil HP 15s-fq1153ns	395,00
m3	ud.	Teléfono Xiaomi Redmi Note 10S	199,95
m4	ud.	Impresión del patrón de calibración	0,03
m5	ud.	Robot móvil terrestre	130,00
m6	ud.	Spray de pintura azul	5,86
m7	ud.	Spray de pintura morada	5,32
m8	días	Software Python	0,00
m9	días	Software App Tello	0,00
m10	días	Software AutoCAD (291€ / mes)	9,70
M.O.D			
h1	h	Ingeniero	23,00
Medios auxiliares			
%		M.A. sobre costes directos	8%

2. Precios descompuestos

En esta tabla de precios descompuestos es importante observar que la partida *d4* esta compuesta por ciertos materiales sin necesidad de realizar ningún trabajo por parte del operario. Esto se ha realizado para tener una única partida como “Estación Tierra” y que fuese más sencillo y claro al añadir esta a la partida de “Realización de las pruebas pertinentes”.

2. Cuadro de precios descompuestos					
Ref	ud.	Descripción	Precio (€)	Cantidad	Parcial
d1	ud.	Diseño del algoritmo en cuestión			
Materiales y Software					
m10	días	Software AutoCAD (291€ / mes)	9,70	2	19,40
M.O.D					
h1	h	Ingeniero	23,00	35	805,00
Medios auxiliares					
%		M.A. sobre costes directos	8%	824,40	65,95

				Precio	870,95

Ref	ud.	Descripción	Precio (€)	Cantidad	Parcial
d2	ud.	Programación			
		Materiales y Software			
m8	días	Software Python	0,00	7	0,00
		M.O.D			
h1	h	Ingeniero	23,00	26	598,00
		Medios auxiliares			
%		M.A. sobre costes directos	8%	598,00	47,84
				Precio	645,84

Ref	ud.	Descripción	Precio (€)	Cantidad	Parcial
d3	ud.	Redacción			
		M.O.D			
h1	h	Ingeniero	23,00	90	2070,00
		Medios auxiliares			
%		M.A. sobre costes directos	8%	2070,00	165,60
Precio					2235,60

Ref	ud.	Descripción	Precio (€)	Cantidad	Parcial
d4	ud.	Estación tierra			
		Materiales y Software			
m2	ud.	Portátil HP 15s-fq1153ns	395,00	1	395,00
m3	ud.	Teléfono Xiaomi Redmi Note 10S	199,95	1	199,95
Precio					594,95

Ref	ud.	Descripción	Precio (€)	Cantidad	Parcial
d5	ud.	Objetivo móvil			
		Materiales y Software			
m5	ud.	Robot móvil terrestre	130,00	1	130,00
m6	ud.	Spray de pintura azul	5,86	1	5,86
m7	ud.	Spray de pintura morada	5,32	1	5,32
		M.O.D			
h1	h	Ingeniero	23,00	1	23,00
		Medios auxiliares			
%		M.A. sobre costes directos	8%	164,18	13,13
Precio					177,31

Ref	ud.	Descripción	Precio (€)	Cantidad	Parcial
d6	ud.	Realización de las pruebas pertinentes			
		Materiales y Software			
m1	ud.	Dron DJI Tello Boost Combo	169,00	1	169,00
m4	ud.	Impresión del patrón de calibración	0,03	1	0,03
m8	días	Software Python	0,00	4	0,00
m9	días	Software App Tello	0,00	4	0,00

d4	ud.	Estación tierra	594,95	1	594,95
d5	ud.	Objetivo móvil	177,31	1	177,31
M.O.D					
h1	h	Ingeniero	23,00	16	368
Medios auxiliares					
%		M.A. sobre costes directos	8%	1309,29	104,743552
Precio					1414,04

3. Mediciones

3. Estado de mediciones			
Referencia	Unidad	Partida	Cantidad
d1	ud.	Diseño del algoritmo en cuestión	1
d2	ud.	Programación	1
d3	ud.	Redacción	1
d6	ud.	Realización de las pruebas pertinentes	1

4. Valoración

4. Valoración				
Referencia	Descripción	Precio	Cantidad	Total
d1	Diseño del algoritmo en cuestión	870,95	1	870,95
d2	Programación	645,84	1	645,84
d3	Redacción	2235,60	1	2235,60
d6	Realización de las pruebas pertinentes	1414,04	1	1414,04
TOTAL (PEM)				5166,43

5. Presupuesto

5. Presupuesto Total				
Referencia	Descripción	Precio	Cantidad	Total
d1	Diseño del algoritmo en cuestión	870,95	1	870,95
d2	Programación	645,84	1	645,84
d3	Redacción	2235,60	1	2235,60
d6	Realización de las pruebas pertinentes	1414,04	1	1414,04
TOTAL (PEM)				5166,43
Beneficio Industrial			13%	671,635894
Costes Indirectos			6%	309,985797
TOTAL (PEC)				6148,05
IVA			21%	1291,09085
PRESUPUESTO TOTAL				7439,14

El presupuesto total de la realización del proyecto asciende a **7439.14€.**