



Universidad Politécnica  
de Madrid

**Escuela Técnica Superior de  
Ingenieros Informáticos**



Grado en INGENIERÍA INFORMÁTICA

Trabajo Fin de Grado

**Interacción entre Personas y Robots  
Aéreos mediante Técnicas de Visión  
Artificial**

Autor: Marko Mikael Isotupa Cañizal  
Tutor: Martín Molina González

Madrid, Enero - 2024

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

*Trabajo Fin de Grado*

**Grado en INGENIERÍA INFORMÁTICA**

**Título:** Interacción entre Personas y Robots Aéreos mediante Técnicas de Visión Artificial

Enero - 2024

**Autor:** Marko Mikael Isotupa Cañizal

**Tutor:** Martín Molina González

Departamento de Inteligencia Artificial

Escuela Técnica Superior de Ingenieros Informáticos

Universidad Politécnica de Madrid

# Índice general

<b>1. Introducción</b>	<b>3</b>
1.1. Objetivos . . . . .	4
1.2. Estructura del documento . . . . .	4
<b>2. Revisión de técnicas</b>	<b>5</b>
2.1. Algoritmos de visión computacional . . . . .	5
2.1.1. Detección de personas . . . . .	5
2.1.2. Reconocimiento de gestos . . . . .	8
2.2. Operación con robots aéreos . . . . .	9
2.2.1. Software de robótica . . . . .	11
2.3. Trabajos relacionados . . . . .	11
<b>3. Diseño</b>	<b>13</b>
3.1. Controlador del dron . . . . .	16
3.2. Instrucciones . . . . .	17
3.2.1. Búfer de gestos . . . . .	22
3.3. Detección de manos y de pose . . . . .	22
3.4. Identificación de gestos . . . . .	23
3.4.1. Arquitectura de la red neuronal . . . . .	23
3.5. Interfaz gráfica . . . . .	25
<b>4. Programación</b>	<b>28</b>
4.1. Estructura del proyecto . . . . .	28
4.2. Despliegue rápido y fácil con Docker . . . . .	32
4.3. Calidad del software . . . . .	32
4.3.1. Control de versiones con Git . . . . .	33
4.3.2. Estilo de programación . . . . .	33
<b>5. Experimentación y resultados</b>	<b>34</b>
5.1. Pruebas . . . . .	34
5.1.1. Pruebas de la red neuronal . . . . .	34
5.1.2. Pruebas de MediaPipe con MediaPipe Studio . . . . .	34
5.1.3. Integración de la visión computacional y los drones . . . . .	36
5.1.4. Pruebas de Docker y del sistema . . . . .	36
5.2. Esfuerzo de desarrollo . . . . .	38
<b>6. Conclusiones</b>	<b>40</b>
6.1. Líneas futuras . . . . .	41

**Bibliografía**

**42**

# Índice de Figuras

2.1. Incluso en posturas complicadas, MediaPipe logra detectar la pose correctamente. . . . .	6
2.2. A la izquierda, los resultados estándar de MediaPipe. A la derecha, los de YOLOv7 Pose. Como se puede apreciar, el primero detecta más detalles en las poses, lo cual permite hacer más aplicaciones. . . . .	7
2.3. El cuadrado verde indica la sección de la imagen sobre la que se aplica la detección de manos. . . . .	9
2.4. Detección de la postura de la mano realizada por MediaPipe. . . . .	10
3.1. El sistema es un intermediario entre dron y usuario. . . . .	14
3.2. Cada paso con un ejemplo de una imagen. . . . .	15
3.3. Estructura UML de las clases del dron. . . . .	16
3.4. Todos los gestos que permite la última versión del proyecto. . . . .	18
3.5. Perfil de velocidades. . . . .	20
3.6. Modelo del esqueleto que detecta MediaPipe Hands. . . . .	23
3.7. Arquitectura de la red neuronal, visualizada con la herramienta pydot. . . . .	25
3.8. División de datos en pruebas (azul) y entrenamiento (naranja). . . . .	26
3.9. Captura de la interfaz gráfica en un uso típico. . . . .	27
5.1. Matriz de confusión de los resultados del entrenamiento. . . . .	35
5.2. Prueba del funcionamiento de MediaPipe Pose usando MediaPipe Studio. . . . .	37
5.3. Algunas de las imágenes de prueba usadas para probar los gestos. . . . .	37
5.4. Imagen del dron usado para las pruebas de este trabajo. . . . .	38

# Índice de Tablas

2.1. Comparación del rendimiento de YOLOv7 y MediaPipe en distintas situaciones . . . . .	8
3.1. Coordenadas de referencia . . . . .	23
3.2. Conversión a coordenadas relativas desde ID : 0 . . . . .	24
3.3. Conversión a vector de una dimensión . . . . .	24
3.4. Normalización al valor máximo . . . . .	24
5.1. Clasificación de los resultados de las pruebas de la red neuronal. . . . .	36
5.2. Conteo de las líneas de código por lenguaje. . . . .	38
5.3. Conteo de líneas de código por módulo. . . . .	39

# Resumen

En la última década, la convergencia de la visión computacional y la tecnología de robots aéreos ha abierto un fascinante abanico de posibilidades en diversos campos, desde la agricultura y la vigilancia hasta la exploración de entornos inaccesibles. Este trabajo se sumerge en esta combinación para mejorar sus capacidades y explorar maneras en las que los usuarios interactúan con ellas.

El trabajo presentado es un sistema intermedio altamente personalizable para el control de drones. Este sistema es capaz de identificar al usuario y realizar un seguimiento en tres dimensiones. Además, puede extraer la postura de la mano del usuario identificado mediante algoritmos de aprendizaje profundo y permite controlar al dron con los gestos identificados.

El objetivo de este trabajo es explorar las posibilidades que surgen al combinar la visión computacional y los UAVs en un sistema de código abierto que pueda ser aplicado a diversos campos.

**Palabras Clave:** UAV, dron, visión computacional, aprendizaje automático

# **Abstract**

In the last decade, the convergence of computer vision and aerial robotics technology has opened up a vast array of possibilities in various fields, from agriculture and surveillance to the exploration of inaccessible environments. The present work delves into this combination to enhance its capabilities and explore further ways in which users could interact with it.

The work presents a highly customizable middleware system for drone control. This system can identify the user and track them in three dimensions. Additionally, it can extract the user's hand posture through deep learning algorithms, allowing control of the drone with the identified gestures.

The objective of this work is to explore the possibilities that arise from combining computer vision and UAVs in an open-source system that can be applied to an array of fields.

**Keywords:** UAV, drone, computational vision, machine learning

# **Capítulo 1**

## **Introducción**

La presente memoria describe el trabajo de fin de grado desarrollado entre septiembre y diciembre de 2023. El trabajo trata dos temas esenciales, cuya comprensión es clave para el seguimiento de la memoria: UAV y visión computacional, los cuales se van a explicar en profundidad a continuación.

Los UAV, acrónimo de Unmanned Aerial Vehicle, es un término general que hace referencia a un vehículo aéreo sin tripulante humano. La forma física de un UAV o dron puede variar mucho, desde sistemas con uno o varios rotores a robots con alas fijas. Aun así, los más comunes son cuadricópteros que pueden moverse en todas las direcciones y hasta rotar, y es precisamente el tipo de dron utilizado en este proyecto.

En las últimas décadas, los avances en el campo de los drones, así como una reducción en su precio, han impulsado su adopción por distintas industrias. Destaca, por ejemplo, su uso en el sector de la agricultura, donde han pasado a jugar un rol crucial para supervisar el uso del agua, la salud de las cosechas y la calidad de la tierra. Otro sector en el que han brillado los drones, especialmente en los últimos años, es en la guerra, donde se ha visto un auge en sus aplicaciones debido al bajo coste, efectividad y facilidad de uso, las mismas cualidades que han popularizado estos sistemas en tantos otros campos.

Por otro lado, la visión computacional es una rama de la inteligencia artificial que trata de dotar a los computadores de la capacidad de extraer información útil de las imágenes, vídeos y otros medios visuales, y ejecutar acciones o hacer recomendaciones basadas en esa información. Resumido por IBM<sup>1</sup>: "si la inteligencia artificial permite a los computadores pensar, la visión computacional permite ver, observar y comprender". Es la tecnología clave en tecnologías tan variadas como los automóviles de conducción autónoma, la clasificación de imágenes automática o la capacidad de Google Translate de traducir el texto en fotografías.

Por otro lado, cabe destacar los avances en la combinación de estas dos tecnologías para aplicarlo a infinidad de industrias. Además del sector de la agricultura, ya mencionado, la industria de la telecomunicación ha adoptado los drones dotados de algoritmos de visión computacional para el mantenimiento e inspección de infraestructuras como antenas. Como consecuencia, la inspección de las antenas de telefonía móvil mediante drones ha evitado una actividad que entre 2013 y 2016 causó 34

---

<sup>1</sup><https://www.ibm.com/topics/computer-vision>

muertes<sup>2</sup>.

En este contexto, el trabajo explora las tecnologías que pueden surgir de la unión entre visión computacional y robots aéreos, especialmente en el campo de la interacción entre operador y dron. Para ello, el sistema implementa tecnologías de seguimiento de objetos y de detección y clasificación de gestos.

El presente trabajo se ha desarrollado en el seno del grupo de investigación CVAR (Computer Vision and Aerial Robotics) de la Universidad Politécnica de Madrid. CVAR es un grupo interdisciplinario que se especializa en la convergencia de la visión por computador y la robótica aérea. Durante años, este grupo ha estado a la vanguardia de la investigación en el incremento de la autonomía de los UAV a base de usar la visión como el sensor clave y poder reconocer el entorno de forma autónoma.

### **1.1. Objetivos**

El propósito de este estudio consiste en explorar y poner en práctica las capacidades de las técnicas de visión artificial. Se busca facilitar la comunicación visual entre seres humanos y drones, con el fin de posibilitar la ejecución de acciones a través de gestos, así como permitir que los robots aéreos sigan visualmente a personas u objetos en movimiento. Para alcanzar este propósito, se han establecido los siguientes objetivos específicos:

- Analizar el problema de la comunicación entre operador y dron.
- Diseñar soluciones para dicho problema.
- Programación de las soluciones mediante el uso del software necesario y las herramientas disponibles.
- Evaluar el funcionamiento de los sistemas desarrollados.

### **1.2. Estructura del documento**

La presente memoria trata de describir el funcionamiento del trabajo presentado y las razones detrás de su diseño, desde una visión más general a una más específica. En el capítulo 2 se presenta una visión más específica de los algoritmos de visión computacional y por qué se han elegido los que se han elegido, así como las funcionalidades específicas del trabajo. A continuación, en el capítulo 3, se describen los módulos que implementan dichas funcionalidades, y cómo se comunican entre sí. El capítulo 4 trata los aspectos más prácticos: la estructura de los ficheros, la integración con Docker y Python3, y cómo se ha obtenido una alta calidad del software. La descripción de las pruebas y de los resultados se tratan en el capítulo 5, y finalmente, las conclusiones, en el 6.

---

<sup>2</sup><https://emerj.com/ai-sector-overviews/industrial-uses-of-drones-applications/>

## **Capítulo 2**

# **Revisión de técnicas**

Este capítulo pretende dar una visión general de los conceptos más importantes de la presente memoria: la visión computacional y los drones. Además, se repasan ciertos sistemas que tienen características en común con este trabajo.

### **2.1. Algoritmos de visión computacional**

El trabajo necesita varios tipos de tecnologías de visión computacional y de redes neuronales. Primero, cuando la imagen es transmitida por el dron, hay que detectar si hay una persona. Una vez identificada, hay que extraer la información de la mano derecha para reconocer el gesto y clasificarlo como uno de los que corresponden a una instrucción. Los siguientes subapartados describen en profundidad las tecnologías disponibles que se adapten a las necesidades del trabajo.

#### **2.1.1. Detección de personas**

Extraer las personas de una imagen es un problema con muchas soluciones. Destacan los algoritmos de HOG[1], un enfoque basado en la extracción de características locales basadas en gradientes de intensidad de píxeles, los algoritmos basados en redes neuronales convolucionales y los modelos de segmentación semántica. Sin embargo, el presente trabajo presenta dos necesidades específicas que eliminan la opción de usar algunos de ellos.

En primer lugar, es un requerimiento del trabajo que la aplicación pueda ser ejecutada en un máximo de máquinas y con facilidad. Para alcanzar este objetivo, es prudente explorar alternativas que requieran un mayor uso de la unidad central de procesamiento (CPU) en vez de la unidad de procesamiento gráfico (GPU). Esto se debe a que, aunque la GPU tiende a ofrecer un rendimiento superior a la CPU en aplicaciones de inteligencia artificial, no todos los dispositivos cuentan con una GPU, mientras que la presencia de una CPU es más común en la mayoría de los ordenadores. Además, la mayoría de librerías que se ejecutan en la CPU pueden ser fácilmente traducidas a GPU, pero lo contrario no sucede.

En segundo lugar, debido a que el objetivo del sistema es también reconocer gestos, es preferible que, al detectar la persona, también se identifique dónde están sus manos, lo cual ahorra recursos computacionales en la siguiente etapa. Por ello, el tipo



Figura 2.1: Incluso en posturas complicadas, MediaPipe logra detectar la pose correctamente.

de detección que se desea obtener es la *detección de poses*, típicamente conseguida a base del aprendizaje profundo. La mayoría de estos sistemas funcionan en dos etapas: la detección de personas y la localización de puntos clave.

En base a las necesidades descritas, se podrían considerar las siguientes opciones:

- **MediaPipe**[2]: es una librería desarrollada por Google que contiene soluciones para múltiples necesidades de aprendizaje automático, como detección de objetos, segmentación de imágenes o clasificación de texto. La librería está diseñada para su uso en dispositivos móviles, por lo que no necesita una GPU, y en CPU es extremadamente eficiente con modelos relativamente ligeros. Además, posee una documentación extensa y muy detallada.

La solución relevante para este proyecto que ofrece la librería es MediaPipe Pose, un marco de estimación de posturas para la detección de una sola persona. Detecta la topología BlazePose[3], que consiste en 33 puntos clave, como se puede apreciar en la figura 2.2. Funciona en dos etapas: detección y seguimiento, lo cual significa que como la detección no se procesa en todos los fotogramas, la inferencia se realiza más rápido. MediaPipe Pose tiene tres modelos para la estimación de poses: BlazePose GHUM Heavy, BlazePose GHUM Full y BlazePose GHUM Lite, cada cual más ligero pero menos preciso.

La librería tiene dos desventajas principales: primero, para detectar más de una persona, la complejidad aumenta exponencialmente y el rendimiento disminuye considerablemente. Segundo, los modelos que se pueden usar no son personalizables, lo cual limita la flexibilidad de sus usos. Sin embargo, como se puede ver en la figura 2.1, MediaPipe Pose es muy preciso incluso en situaciones complejas. Esto, junto al alto rendimiento en CPU, la facilidad de uso y la eficiencia hacen a esta librería la más indicada para el trabajo.

- **OpenPose**[4][5][6][7]: es una biblioteca escrita en C++ para la detección de poses en imágenes. Es capaz de identificar la pose, cara y manos (en total 135 puntos clave) de varias personas con un alto rendimiento. Su funcionamiento se basa en dos etapas: en la primera, se detectan las personas en la imagen a través de una red convolucional, y, a continuación, se estiman y asignan esqueletos de

## Revisión de técnicas

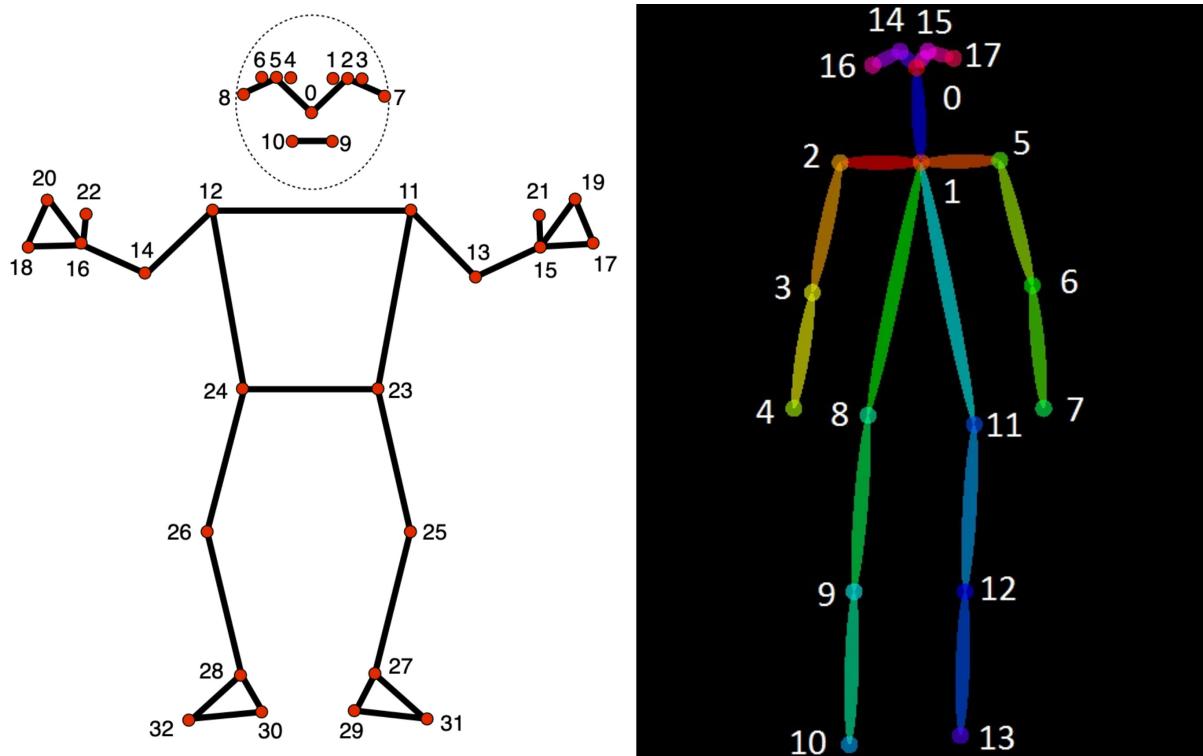


Figura 2.2: A la izquierda, los resultados estándar de MediaPipe. A la derecha, los de YOLOv7 Pose. Como se puede apreciar, el primero detecta más detalles en las poses, lo cual permite hacer más aplicaciones.

postura a cada persona detectada.

La ventaja principal de esta librería es que la complejidad de la ejecución es independiente del número de personas que haya en la imagen, a diferencia de otras librerías. No obstante, el algoritmo se ejecuta en todas las imágenes, lo cual, para vídeo, hace que la precisión y el rendimiento se vean reducidos. Por último, la librería está optimizada para GPU, y para CPU tiene un soporte mínimo. Todas estas razones hacen de OpenPose una opción inviable para el presente trabajo.

- **YOLOv7 Pose**[8]: YOLO, acrónimo de You Only Look Once ("solo miras una vez"), es un algoritmo desarrollado en 2016 para la detección de objetos en general. La séptima y última versión, como su nombre indica, se introdujo en 2022 y consiste en la detección de objetos en una sola etapa. Esta etapa está basada en una CNN (red neuronal convolucional), cuya reducción en complejidad, comparada con otros sistemas de dos etapas, obtiene un rendimiento mucho mayor a costa de una precisión reducida.

YOLOv7 Pose es la implementación del algoritmo YOLO al ser entrenado con el conjunto de datos COCO, que tiene 17 puntos clave, como se puede apreciar en la figura 2.2. A diferencia de MediaPipe, la librería YOLO es capaz de detectar varias personas en una misma imagen, y en vez de realizar una detección inicial y posteriormente un seguimiento, aplica la detección en todas las imágenes de un vídeo, lo cual lo hace menos eficiente.

A pesar de que en CPU YOLOv7 tiene un rendimiento excelente, está optimizado para GPU, y como tal, no puede competir con MediaPipe, que está diseñado para CPU de forma nativa. Un análisis en profundidad realizado se puede ver en la tabla inferior, donde se demuestra que en GPU, YOLOv7 Pose obtiene resultados constantemente mejores en términos de imágenes correctamente procesadas por segundo. Sin embargo, en CPU, MediaPipe reconoce personas y sus posturas mejor con un rendimiento hasta 30 veces mayor.

Prueba	FPS MediaPipe	FPS YOLOv7
Varias personas en la imagen(CPU)	12.57	3.34
Configuración inicial(CPU)	31.65	0.76
Configuración optimizada(CPU)	29.2	8.1
Luz baja(CPU)	29.2	8.1
Personas parcialmente tapadas(CPU)	30.0	8.0
Posturas complicadas(GPU)	29.0	83.39
Varias personas en la imagen(GPU)	6.77	78.93

Cuadro 2.1: Comparación del rendimiento de YOLOv7 y MediaPipe en distintas situaciones<sup>2</sup>.

### 2.1.2. Reconocimiento de gestos

Una vez obtenida la pose de la persona, hay que aplicar una detección de la postura de la mano. Es posible hacerlo con la imagen completa, pero las pruebas realizadas demostraron que los resultados son considerablemente peores de esta forma. La solución: con la información obtenida de la etapa anterior, ejecutar la detección de la mano únicamente en la sección de la imagen que contenga la mano (ver figura 2.3)

El siguiente paso es transformar la sección de la imagen que contiene una mano en el nombre de un gesto. Para ello, hay dos formas de proceder. La primera opción consiste en crear una red neuronal entrenada con muchas imágenes con los gestos que se quiere que se reconozcan. Esta opción es la más difícil de implementar, pues requiere entrenar la red con imágenes en todo tipo de iluminación, con manos distintas y con muchos casos de prueba para obtener un rendimiento decente. Además, el entrenamiento de dicha red es un proceso computacionalmente caro.

Para evitar estas desventajas, la segunda solución, y la elegida, consiste en separar la detección de la mano del reconocimiento de gestos. Para la primera parte de la solución, la detección de la mano, existen librerías como YOLO-Hand-Detection<sup>3</sup> o MediaPipe Hands, que identifican la postura de la mano, como se puede ver en la figura 2.4. El funcionamiento de ambas librerías es análogo a las de la pose del cuerpo, como se describió en el apartado anterior, y por las mismas razones, se usó MediaPipe para el trabajo.

Con los puntos identificados por la detección de manos, se aplica el reconocimiento de gestos, que consiste en clasificar dichos puntos como uno de los 12 gestos que soporta el sistema en la última versión. Para ello, el sistema tiene una red neuronal creada desde cero que toma los puntos normalizados y predice qué gesto es. Una descripción en profundidad del funcionamiento de dicha red se puede encontrar en el capítulo 3.

---

<sup>3</sup><https://github.com/cansik/yolo-hand-detection>

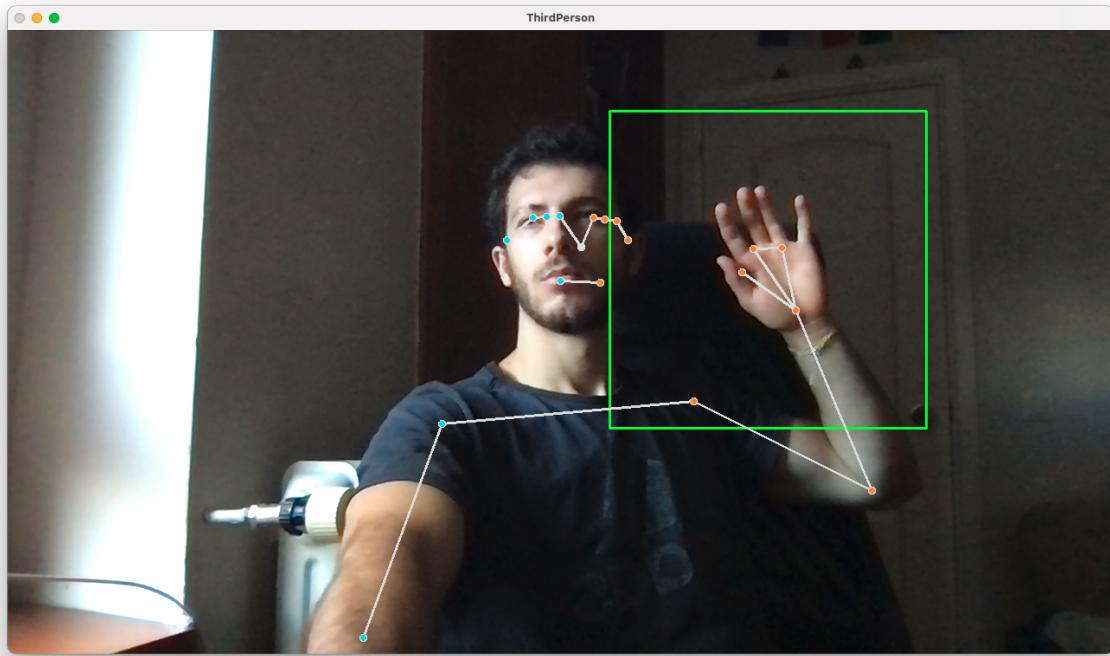


Figura 2.3: El cuadrado verde indica la sección de la imagen sobre la que se aplica la detección de manos.

La mayor ventaja de este método es que, con MediaPipe realizando la detección de la postura de la mano, no es necesario tener en cuenta los aspectos más físicos de la imagen, como la iluminación de la escena, ya que la librería lo hace automáticamente. Además, la red neuronal creada es capaz de obtener con datos limitados una alta precisión en las predicciones, y permite flexibilidad para líneas futuras como las mencionadas en el capítulo 6.1

## 2.2. Operación con robots aéreos

El avance tecnológico ha impulsado el desarrollo y la utilización de vehículos aéreos no tripulados (UAV), comúnmente denominados drones, que abarcan una amplia diversidad de formas y funcionalidades. Estos dispositivos han revolucionado numerosos campos, desde la fotografía y el entretenimiento hasta aplicaciones industriales y misiones de búsqueda y rescate.

Los drones, o UAV, pueden tener muchas formas, desde aquellos con alas fijas a multirotores. Los más habituales son los cuadricópteros, que pueden moverse en cuatro ejes: arriba/abajo, derecha/izquierda, delante/detrás y rotar en ambas direcciones, además de permanecer estáticos. Pueden tener una amplia gama de sensores que lo permiten volar con más autonomía o transmitir imágenes.

Los robots aéreos han evolucionado con distintos niveles de autonomía, cada uno con características y capacidades específicas:

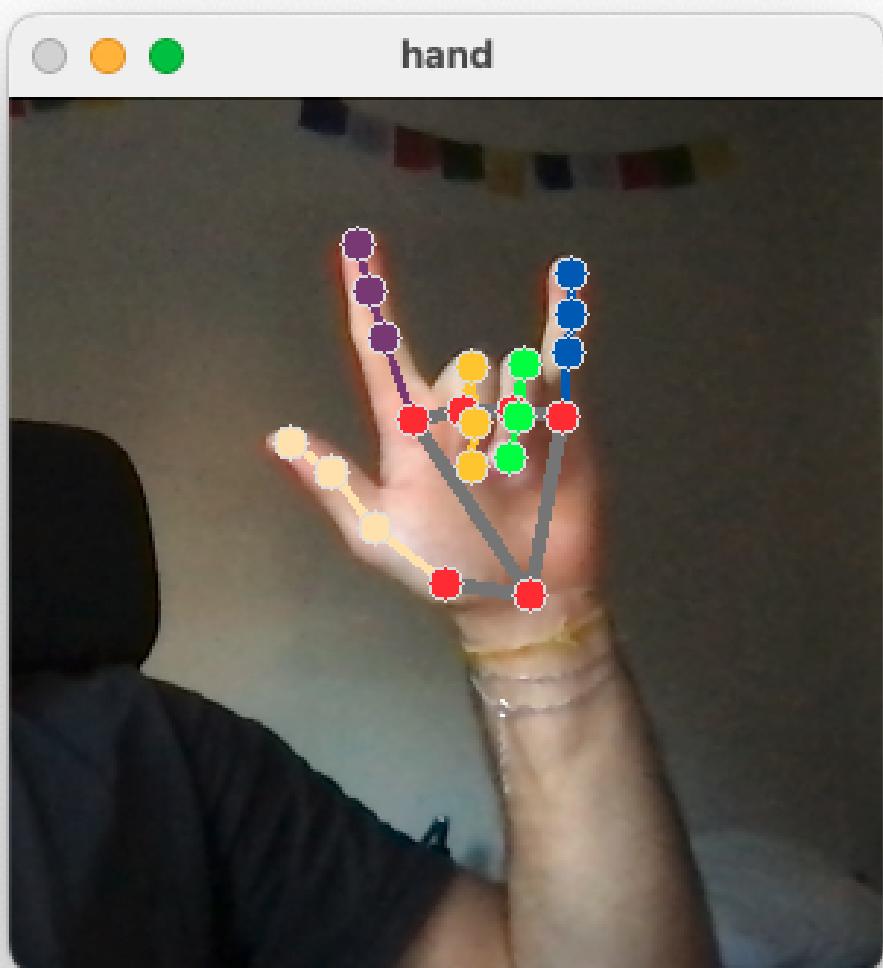


Figura 2.4: Detección de la postura de la mano realizada por MediaPipe.

- **Autonomía básica:** Drones de consumidor como el DJI Phantom<sup>4</sup> pueden seguir rutas predefinidas o vuelos programados con puntos de referencia establecidos. Estos drones son capaces de despegar, volar y aterrizar automáticamente, pero requieren de supervisión humana para tomar decisiones más complejas.
- **Drones semiautónomos:** Algunos drones comerciales, como los utilizados en la agricultura de precisión, están equipados con sensores y software que les permiten evitar obstáculos o mantener una altitud constante de manera autónoma. Estos drones pueden ajustar su ruta o realizar ciertas tareas específicas sin intervención directa del operador.
- **Autonomía plena:** Ejemplos de investigación y desarrollo en este nivel incluyen sistemas UAV para misiones de búsqueda y rescate o exploración espacial. Estos robots están equipados con algoritmos de inteligencia artificial avanzada que les permiten tomar decisiones complejas en tiempo real, adaptarse a entornos desconocidos y resolver problemas sin la necesidad de una supervisión humana constante.

### 2.2.1. Software de robótica

El software de robótica desempeña un papel fundamental en el control y la operación de robots aéreos. Plataformas como Robot Operating System 2[9] (ROS2) han revolucionado la forma en que se programan y controlan los robots, proporcionando una arquitectura flexible y modular que permite la creación de sistemas robóticos avanzados. Su capacidad para la comunicación entre componentes distribuidos ha elevado las capacidades de control y operación de drones y otros robots aéreos, convirtiéndolo en un recurso esencial en la industria de la robótica.

Dentro del ámbito específico de los drones, han emergido software como Aerostack2[10], que es un marco de trabajo de código abierto especializado en el desarrollo y control de estas aeronaves no tripuladas. Desarrollado por el grupo de investigación CVAR, cuenta con herramientas avanzadas para la planificación de trayectorias, navegación autónoma y control de vuelo. Aerostack2, construido sobre ROS2, capacita a los operadores de drones para realizar misiones precisas y complejas con un alto nivel de seguridad.

Adicionalmente, la API de Tello para Python es una interfaz que proporciona acceso a las funcionalidades de los drones DJI Tello. Esta API facilita la programación de movimientos, la captura de imágenes y la interacción con estos drones, permitiendo a entusiastas y desarrolladores integrar fácilmente drones en una amplia gama de aplicaciones personalizadas. Estas herramientas de software representan avances significativos en el control y la operación de robots aéreos, brindando versatilidad y capacidades mejoradas a quienes trabajan con este tipo de tecnología.

## 2.3. Trabajos relacionados

A pesar de que el trabajo presentado es único, partes de éste se apoyan en productos y proyectos desarrollados previamente. La capacidad de seguimiento de personas en un dron destaca en la marca DJI<sup>5</sup>, que desde 2013 ha desarrollado drones capaces de

---

<sup>4</sup><https://www.dji.com/es/phantom>

<sup>5</sup><https://www.dji.com/es>

## **2.3. Trabajos relacionados**

---

detectar y seguir a usuarios con el objetivo de hacer grabaciones de alta calidad con un equipo mínimo. Sin embargo, el acceso al código usado por los drones es cerrado y no está disponible a aquellos que no estén dispuestos a comprar los productos de la marca.

La funcionalidad de drones controlados por gestos es también un campo muy activo[11][12][13] en el sector de la visión computacional. Gran parte de estos artículos de investigación se centran en el control de drones mediante gestos identificados a base de algoritmos de aprendizaje profundo. Por ejemplo, un proyecto destacado de Google<sup>6</sup>, donde se usa el detector de gestos de MediaPipe para identificar 8 gestos con el objetivo de controlar un dron Tello.

Por último, cabe destacar el artículo publicado por el laboratorio CVAR, "Natural User Interfaces for Human-Drone Multi-Modal Interaction"[14], donde se implementan varios algoritmos de interacción con robots aéreos mediante aprendizaje automático y visión computacional. El sistema desarrollado es capaz, entre otras cosas, de controlar el movimiento de un dron mediante gestos y voces, es decir, introduciendo técnicas de NUI (Natural User Interfaces).

---

<sup>6</sup><https://developers.googleblog.com/2021/09/drone-control-via-gestures-using-mediamipe-hands.html>

# **Capítulo 3**

## **Diseño**

El presente capítulo clasifica las distintas funcionalidades que presenta el trabajo en cinco secciones descritas en profundidad. Cada sección corresponde, como se verá en el siguiente capítulo, con uno de los módulos programados, e implementa funciones específicas, asegurando así la gestión eficiente de las tareas y la optimización de los procesos involucrados. Están organizados desde los términos más generales a los más específicos.

El sistema actúa como un intermediario entre el usuario y el dron, como se puede apreciar en la figura 3.1. Dicho sistema se ejecuta en un ordenador conectado al dron a través del protocolo Wi-Fi, que realiza el cálculo asociado a la imagen transmitida por el dron y transmite de vuelta una instrucción al dron. Esta arquitectura asegura una comunicación bidireccional con un mínimo de fallos y de pérdida de información.

Cada módulo tiene una función específica, por lo que el orden en el que se ejecuta cada uno es crucial para asegurar un funcionamiento adecuado. En la figura 3.2 se puede apreciar cómo fluyen los datos en conjunto en cada iteración del bucle principal:

1. Se obtiene la imagen transmitida por el dron o una cámara.
2. De la imagen, se extrae la pose corporal.
3. Con la información de la pose corporal, se extrae la sección de la imagen que contiene la región de la mano derecha.
4. Se aplica la detección de manos y extracción de pose de la mano de la sección de la imagen del paso 3.
5. Se normalizan los puntos de la mano, la red neuronal predice qué gesto es.
6. En base al gesto, se calcula el siguiente movimiento y se transmite al dron.
7. Finalmente, se muestra toda la información en una interfaz gráfica.

En las siguientes secciones se describen en profundidad los algoritmos y los sistemas mencionados en los pasos. La división de funciones en distintos módulos asegura la modularidad, y si solo se necesitara parte del trabajo, se puede importar únicamente la librería necesaria.

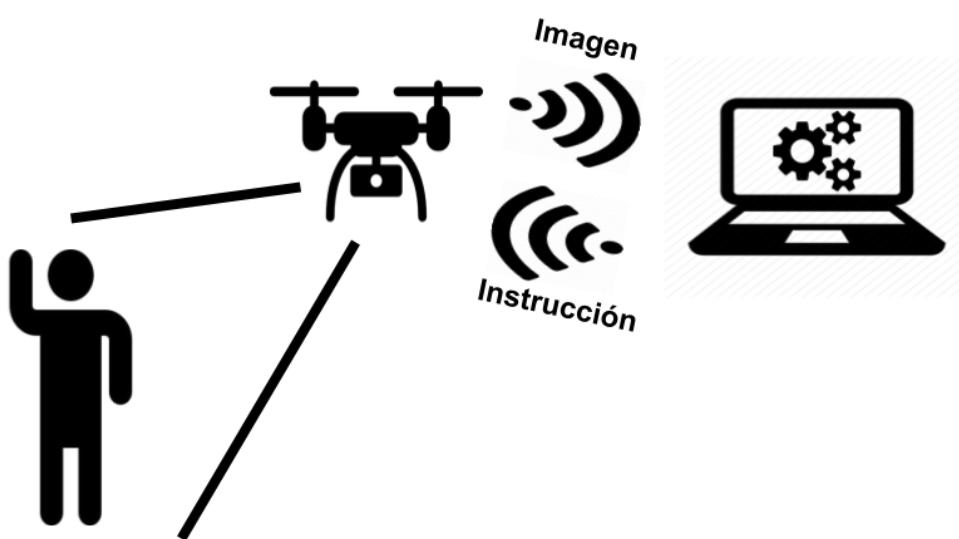


Figura 3.1: El sistema es un intermediario entre dron y usuario.

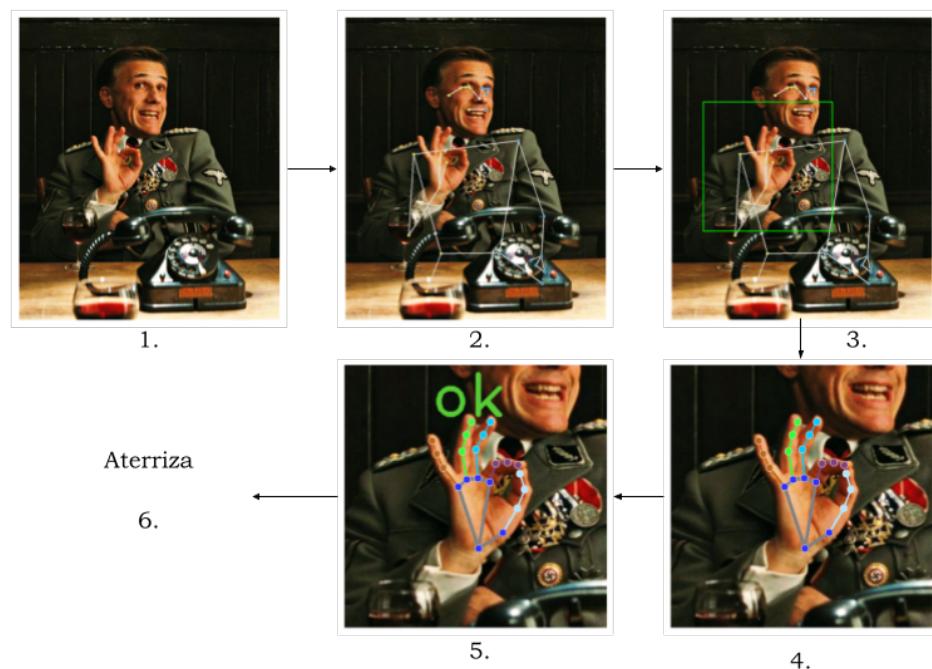


Figura 3.2: Cada paso con un ejemplo de una imagen.

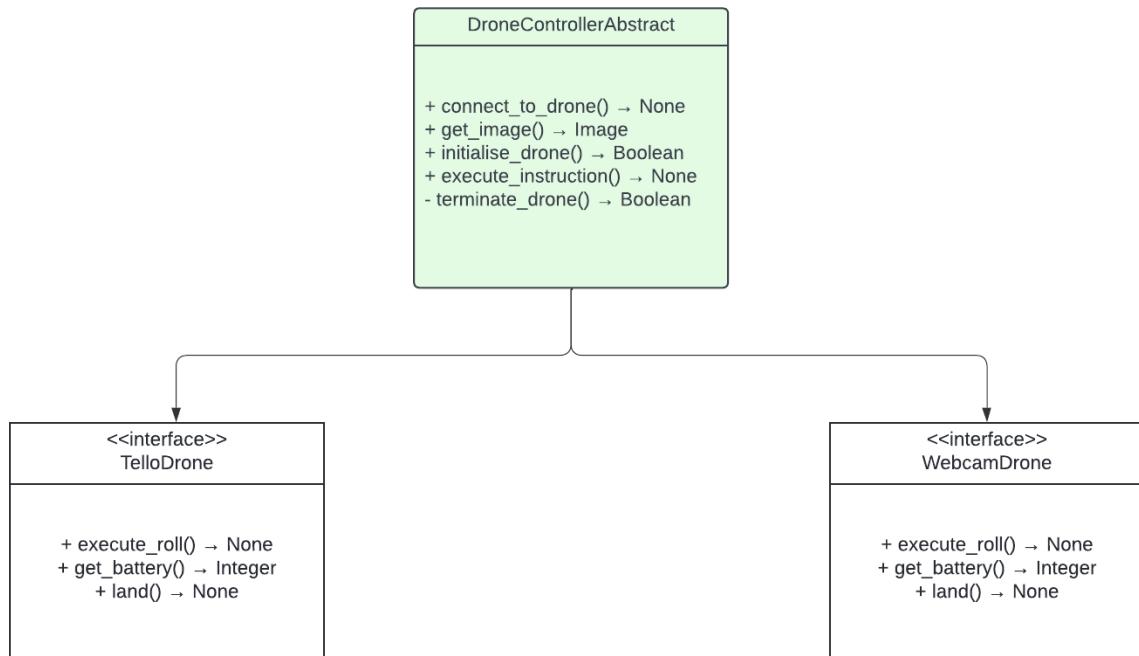


Figura 3.3: Estructura UML de las clases del dron.

## 3.1. Controlador del dron

Dependiendo del dron, las instrucciones para moverlo pueden tener una sintaxis distinta, por lo que es necesario crear una manera estándar para conectar y controlarlo, lo suficientemente flexible para que funcione con varias marcas de drones y con el trabajo. Para ello, el controlador del dron actúa como un intermediario entre la aplicación y el dron, que transforma las instrucciones generadas por la aplicación en órdenes que muevan al dron. En la figura 3.2 se puede apreciar la arquitectura de clases.

Tiene dos elementos principales: la clase abstracta, `DroneControllerAbstract`, que dicta los métodos básicos y necesarios de los que todos los drones deberían proveer, y los paquetes que implementan la clase abstracta. Estos métodos incluyen: obtención de la imagen de la cámara, conexión al dron, inicializar el dron, ejecutar una instrucción y eliminar los recursos reservados por el dron. Respecto a la cámara del dron, la frecuencia con la que transmite la imagen es más que probable que no esté sincronizada con la del bucle del sistema, lo cual puede crear problemas a la hora de procesar y mostrar la imagen en la pantalla. Como solución, el sistema recibe la imagen en un búfer, que puede ser accedido en cualquier momento para procesarlo.

En el caso de la figura, hay dos implementaciones, `TelloDrone`, que, además de las clases que hereda, tiene funciones para que el dron haga una voltereta en el aire, obtenga el porcentaje de la batería y una función para aterrizar. Esta es la implementación que se conecta a los drones Tello, usados en las pruebas de este trabajo.

La otra clase, `WebcamDrone`, es la utilizada para la simulación. En vez de conectarse a un dron, usa la librería OpenCV para acceder a la cámara del ordenador que ejecuta

el programa y recibe esas imágenes. Las instrucciones enviadas a esta clase son ignoradas, pues no hay ningún dron que pueda ejecutarlas. Es una clase diseñada para hacer las pruebas de la parte de visión computacional, sin la necesidad de un objeto físico que necesite estar cargado o se pueda romper, lo cual acelera el tiempo empleado en realizar pruebas.

### **3.2. Instrucciones**

Con una manera de controlar el dron, es necesario un sistema que calcule los movimientos a ejecutar. Estos movimientos se calculan en base al estado del dron, que puede ser si está en modo de seguimiento, si este ha despegado y las distancias a las que se quiere que se sitúe el dron, en los tres ejes.

El sistema usa una sintaxis de movimiento basado en dos partes: el tipo de movimiento, que puede ser 'despegar', 'aterrizar', 'voltereta' o 'mover', y las especificaciones del movimiento. Para los dos primeros tipos de movimientos, la especificación indica el tiempo, en segundos, a ejecutar la instrucción. La especificación de 'voltereta' indica la dirección en la que hacer una vuelta de 180 grados sobre el dron. Por último, 'mover' se describe en cuatro números que indican la velocidad, en un rango de -100 a 100, que representan el movimiento en los cuatro posibles ejes del dron: delante/detrás, arriba/abajo, derecha/izquierda y sentido horario/antihorario. Los cuatro valores nulos representan al dron manteniéndose en el aire sin moverse.

El cálculo del movimiento a realizar se basa en el gesto identificado, la pose y la imagen. Todos los gestos que la red neuronal está preparada para identificar están listadas en la figura 3.4. El efecto que el gesto tiene depende de si el sistema está en el modo de seguimiento o no. En el modo seguimiento, los gestos cambian las distancias por defecto a las que se desea que se sitúe el dron de la persona detectada. Si el sistema está en el modo de no seguimiento, los gestos simplemente indican la instrucción a ejecutar.

Por ejemplo el gesto de la palma con los dedos extendidos hacia la cámara, en modo seguimiento, disminuye la distancia a la que el dron debería estar del usuario, y en no seguimiento, transmite al controlador del dron la instrucción de hacer que el dron avance hacia delante. Para cambiar el modo de seguimiento, basta con señalar al dron. Cuando se detecte el gesto, se cambia el estado que representa el seguimiento a "no seguir" o viceversa.

La lista de gestos detectados y su instrucción asociada es la siguiente:

- Palma hacia la cámara: dron se acerca a la persona.
- Palma con dedos hacia la cámara: el dron se detiene.
- Pulgar y meñique extendidos: una voltereta hacia la izquierda.
- Puño con nudillos hacia la cámara: sin instrucción asociada.
- Pulgar e índice extendidos: el dron describe un semicírculo alrededor de la persona detectada.
- Dedo señalando a la cámara: cambiar entre modo de seguimiento y de no seguimiento.

### 3.2. Instrucciones



Figura 3.4: Todos los gestos que permite la última versión del proyecto.

## Diseño

---

- Pulgar hacia la derecha: el dron se mueve hacia la derecha del usuario.
- Pulgar hacia la izquierda: el dron se mueve hacia la izquierda del usuario.
- Pulgar hacia arriba: aumentar la altura.
- "ok": aterrizar.
- Pulgar hacia abajo: disminuir la altura.
- Mano extendida con nudillos hacia la cámara: el dron se aleja de la persona.

El seguimiento del usuario se realiza con un cálculo basado en la imagen, la pose detectada y el movimiento anterior, que producen una instrucción que moverían al dron en una dirección en la que el usuario estaría más cerca de los valores deseados. Para ello, el seguimiento se produce en los tres ejes:

- **Arriba/abajo:** el seguimiento se calcula en base a la posición de ciertos puntos en la imagen. Si el punto que representa la nariz de la pose detectada está visible, se ejecuta un código como el siguiente:

```
1     function calcular_velocidad_vertical(posicion_actual,
2         alto_de_la_imagen):
3         centro_de_la_imagen = alto_de_la_imagen / 2
4         velocidad_maxima = 60
5         distancia_del_centro = abs(centro_de_la_imagen -
6             posicion_actual)
7         velocidad = velocidad_maxima * (distancia_del_centro /
8             centro_de_la_imagen)
9     return int(velocidad)
```

El código resulta en que, cuanto más alejada esté la nariz del centro de la pantalla, mayor es la velocidad para aumentar o disminuir la altura. Para conseguir un seguimiento vertical más natural, se definen umbrales desde el centro de la imagen para crear una zona que no produce cambios de altura. Todo esto resulta en un perfil de velocidades como el siguiente, asumiendo una imagen con un alto de 720 píxeles, un umbral de 60 y una velocidad máxima de 100:

No obstante, la velocidad máxima puede variar para asegurar un funcionamiento correcto del dron, pues una velocidad máxima demasiado alta podría provocar daños en el UAV.

Finalmente, es importante resaltar un algoritmo que optimiza significativamente el seguimiento. En casos en los que el sujeto se desplaza a una velocidad excesiva, el sistema de seguimiento podría no ser capaz de mantenerlo dentro de los límites de la imagen, resultando en la pérdida de información esencial para la continuación del seguimiento. Para solucionar esto, se implementa la persistencia de puntos, que consiste en almacenar el último punto detectado, y cuando se pierde el seguimiento, continuarlo con el último punto detectado hasta que el sujeto vuelva a aparecer en la imagen.

- **Derecha/izquierda:** el seguimiento horizontal funciona de manera similar al vertical; es decir, se basa en la distancia de ciertos puntos detectados del centro de la imagen. Sin embargo, en este caso se trata del centro del ancho de la imagen, como se ve en el siguiente pseudocódigo:

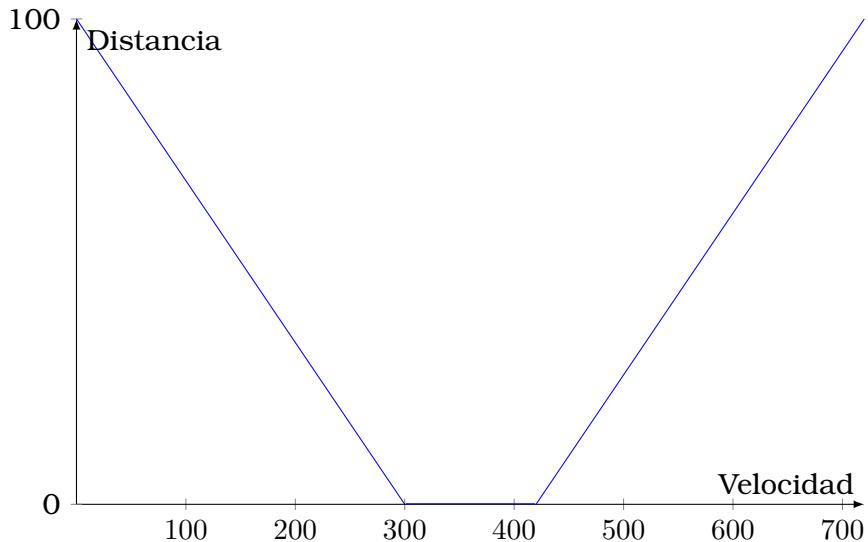


Figura 3.5: Perfil de velocidades.

```

1     function calcular_velocidad_horizontal(posicion_actual,
2         ancho_de_la_imagen):
3             centro_de_la_imagen = ancho_de_la_imagen / 2
4             velocidad_maxima = 60
5             distancia_del_centro = abs(centro_de_la_imagen -
6                 posicion_actual)
7             velocidad = velocidad_maxima * (distancia_del_centro /
8                 centro_de_la_imagen)
9             return int(velocidad)

```

Este pseudocódigo resulta en un perfil de velocidades igual que en el de la figura 3.2. La principal diferencia respecto del seguimiento vertical es que, además del movimiento lateral, se introduce un factor de rotación, que produce un seguimiento más fiable y reduce la probabilidad de un choque contra una pared.

- **Delante/detrás:** Este ha sido sin duda el eje que más dificultad ha supuesto, pues la percepción de profundidad es un desafío para las máquinas. En una primera aproximación, se planteó usar un algoritmo de percepción de profundidad, pero se comprobó que ralentizaba demasiado el programa y no sería práctico. Como solución, se plantearon tres métodos usando la información ya disponible, la pose.

El primer método calcula la distancia media entre los puntos de la nariz y de los hombros usando la siguiente fórmula:

$$\text{velocidad} = \frac{\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} + \sqrt{(x_1 - x_3)^2 + (y_1 - y_3)^2}}{2}$$

Donde el subíndice 1 representa el punto de la nariz, el 2 el del hombro derecho y 3 el del hombro izquierdo. No obstante, esta solución fallaba cuando el usuario estaba de lado a la cámara, pues las distancias verticales cambiaban tanto que

resultaba en el dron acercándose o alejándose demasiado. Para intentar remediar el problema, el siguiente método usaba la altura del triángulo generado por los puntos de la nariz y los hombros. Así, aunque el usuario se situara de lado a la perspectiva de la cámara, la altura del triángulo no variaba, acorde a la siguiente fórmula:

$$\text{velocidad} = \sqrt{(x_1 - x_4)^2 + (y_1 - y_4)^2}$$

Donde  $x_4$  es el valor  $x$  del hombro izquierdo y  $y_4$  es igual a la media de los valores verticales entre el hombro derecho y el izquierdo, es decir:

$$y_4 = \frac{y_2 + y_3}{2}$$

A pesar de que el funcionamiento es suficientemente bueno para la aplicación, se puede mejorar un aspecto: cuando el usuario agacha la cabeza, el dron se acerca, pues detecta un cambio en la altura del triángulo.

Como solución, se asumen las distancias verticales y horizontales entre la nariz y el hombro, y se determina la distancia en base a la variación de estas. En resumen, siendo  $x$  la distancia de la nariz al hombro, si  $x$  es menor a la distancia por defecto, el dron está demasiado lejos, por lo que se transmite la instrucción de "hacia delante". En el caso contrario; es decir, si  $x$  es mayor a la distancia por defecto, el dron está demasiado cerca y se transmite la instrucción "hacia detrás". Para obtener un movimiento suave, se siguen los siguientes pasos:

1. Si la distancia, calculada por el algoritmo del punto anterior es menor que la distancia deseada menos un umbral, la velocidad transmitida se define como:

$$\text{velocidad} = -\max(-v_{\text{máx}}, -v_{\text{máx}} * (\frac{d - d_{\text{deseada}}}{d_{\text{máx}} - d_{\text{deseada}} - \text{umbral}}))$$

Donde  $v$  denota la velocidad y  $d$  la distancia.

2. Si la distancia es mayor que la distancia deseada más un umbral, la velocidad transmitida se define como:

$$\text{velocidad} = \min(v_{\text{máx}}, v_{\text{máx}} * (\frac{d - d_{\text{deseada}}}{d_{\text{máx}} - d_{\text{deseada}} - \text{umbral}}))$$

3. En cualquier otro caso, la velocidad será 0.

Estas fórmulas normalizan la diferencia entre la distancia actual y la deseada, asegurando que la velocidad resultante esté en el rango [0,1]. Para garantizar que la velocidad no supere el límite establecido, se realiza el máximo o el mínimo. El umbral puede ser aumentado para obtener una región en la que la velocidad sea nula, haciendo el seguimiento menos preciso, o disminuido para un seguimiento más estricto.

Con esta implementación, se obtiene un seguimiento en los tres ejes, con una velocidad variable que permite un rastreo incluso si el sujeto se mueve rápido o corre.

#### 3.2.1. Búfer de gestos

El bucle principal completa cerca de 20 iteraciones cada segundo, lo que significa que en cada imagen en la que aparezcan manos se detectan cerca de 20 gestos cada segundo. Como cada gesto ejecuta una instrucción de mover al dron, se enviarían muchas instrucciones erróneas que el usuario no quiere transmitir. Para solucionar esto, hay implementados dos sistemas de seguridad.

El primero consiste en que se realiza la detección de manos únicamente cuando la muñeca del usuario esté más alta que su codo, desde el punto de vista de la cámara del dron. Así, solo se realiza la detección de gestos cuando el sujeto levanta la mano.

El otro sistema es un búfer circular de gestos. Cada vez que se confirma la detección de un gesto, se añade al búfer. Si todas las posiciones de dicho búfer son iguales, se ejecuta el gesto, y en caso contrario, no. Esto resulta en que, para maximizar la seguridad de que un gesto realmente se ha gesticulado, se puede aumentar el tamaño del búfer.

### 3.3. Detección de manos y de pose

El módulo encargado de la detección de manos y de la pose funciona con la librería MediaPipe, descrita en profundidad en la sección 2.1.1. Una vez obtenida una imagen, se ejecutan dos etapas, diseñado así para asegurar la modularidad y facilidad de pruebas en la aplicación.

En la primera etapa, MediaPipe Pose identifica al sujeto presente en la imagen, asignándole un esqueleto de 33 puntos de referencia, visto en la figura 2.2. Esta identificación se produce con una serie de modelos que predicen los puntos de referencia. El primer modelo detecta la presencia de cuerpos humanos en una imagen, y el segundo asigna los puntos a las regiones correspondientes en la imagen.

En la segunda etapa, MediaPipe Hands identifica la mano presente en la imagen y le asigna un esqueleto de 21 puntos de referencia, como se puede ver en la figura. Análogo a la etapa anterior, la identificación es el resultado de dos modelos: el primero, identifica la mano en la imagen, y el segundo asigna los puntos de referencia a las regiones correspondientes en la imagen.

No obstante, el modelo de MediaPipe Hands está entrenado para identificar manos cercanas a la cámara, a una distancia de unos 30 centímetros, pero la aplicación requiere una manera de detectar los gestos a varios metros de la cámara, lo cual impide la detección. Para solucionarlo, se introduce un algoritmo que, a partir de la información de la pose, extrae una región de la imagen que contiene la mano, sobre la que se aplica el modelo. Dicho algoritmo funciona de la siguiente manera:

1. Si la pose o la imagen no están disponibles, el resultado es una matriz nula.
2. Se normalizan los puntos de la pose que corresponden a los de la mano derecha. Esta normalización se explica en detalle en la siguiente sección.
3. Se calcula la distancia euclídea entre los dos puntos más alejados de los puntos correspondientes a la mano derecha. En notación matemática:

$$d = \left\| \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} - \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} \right\|$$

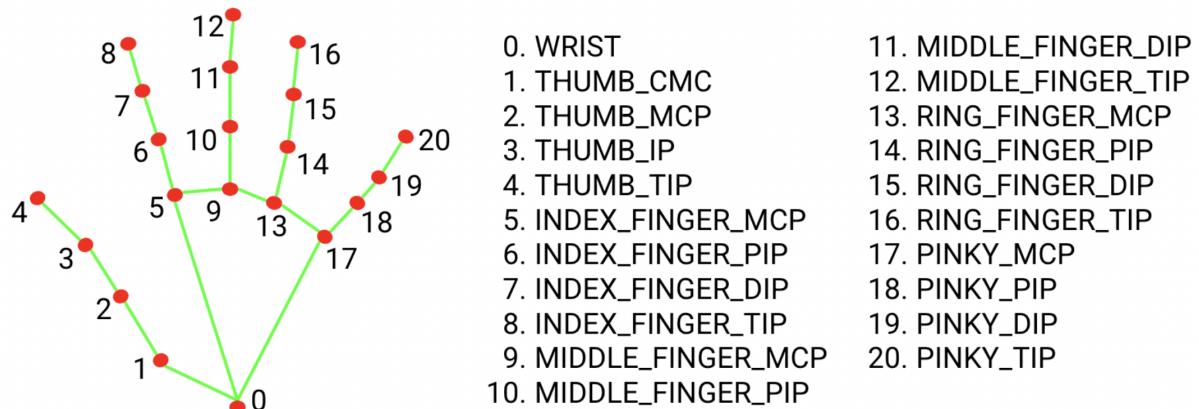


Figura 3.6: Modelo del esqueleto que detecta MediaPipe Hands<sup>1</sup>.

4.  $d$  determina el ancho y el alto del cuadrado, pero se le suma un valor para asegurar que dicho cuadrado contenga todos los píxeles de la mano.
5. Se calcula el centro de la mano con una media de todos los puntos correspondientes a la mano. Alrededor de este centro se forma el cuadrado.
6. Se extraen los píxeles contenidos por el cuadrado y se almacenan.
7. Para obtener una región siempre del mismo tamaño, se cambia el tamaño a 400 por 400 píxeles usando una interpolación bilineal, la cual obtiene los resultados más estándares.

Aplicando el modelo de MediaPipe Hands a la región extraída, se simula una mano que está mucho más cerca de la cámara de lo que realmente está, que permite el uso óptimo de la librería. Así, se obtienen 21 puntos de referencia de la mano derecha, que serán analizadas y procesadas en la siguiente sección.

## 3.4. Identificación de gestos

Para poder transformar los puntos extraídos por la detección de manos en un gesto, es necesario recurrir a una red neuronal. Esta red debe ser capaz de categorizar los 21 puntos detectados en uno de los 12 gestos disponibles en la aplicación.

### 3.4.1. Arquitectura de la red neuronal

MediaPipe Hands devuelve 21 puntos de la mano derecha detectada. Para poder ser usados, deben en primer lugar ser procesados y normalizados:

1. Se reciben los puntos  $(x, y)$  que representan las coordenadas del píxel de la imagen correspondiente a cada punto:

Cuadro 3.1: Coordenadas de referencia

ID : 0	ID : 1	ID : 2	...	ID : 17	ID : 18	ID : 19	ID : 20
[551,465]	[485,428]	[439,362]	...	[633,315]	[668,261]	[687,225]	[702,188]

### 3.4. Identificación de gestos

---

2. Cada punto se convierte a coordenadas relativas desde ID : 0, que significa que a cada coordenada  $(x_n, y_n)$  se le resta  $(x_0, y_0)$ :

Cuadro 3.2: Conversión a coordenadas relativas desde ID : 0

ID : 0	ID : 1	ID : 2	...	ID : 17	ID : 18	ID : 19	ID : 20
[0,0]	[-66,-37]	[-112,-103]	...	[82,-150]	[117,-204]	[136,-240]	[151,-277]

3. Se convierte la matriz a un vector de una dimensión:

Cuadro 3.3: Conversión a vector de una dimensión

ID : 0	ID : 1	ID : 2	...	ID : 17	ID : 18	ID : 19	ID : 20
0	0	-66	-37	-112	-103	...	82 -150 117 -204 136 -240 151 -277

4. Se normalizan los valores al valor absoluto del valor máximo, es decir,  $\frac{x_n}{|x_{\max}|}$ . En el ejemplo,  $x_{\max}$  es el último valor de ID: 20, -277.

Cuadro 3.4: Normalización al valor máximo

ID : 0	ID : 1	ID : 2	...	ID : 18	ID : 19	ID : 20	
0	0	-0.24	-0.13	-0.4	-0.37	...	0.422 -0.74 0.491 -0.87 0.545 -1

Este procesamiento resulta en que los datos pasan a no depender del tamaño de la imagen, de forma que la predicción es independiente de la cámara desde la que se hace la detección. Con estos datos procesados, se pasan a la red neuronal, que tiene la arquitectura de la figura 3.7.

Esta figura representa una red neuronal con cinco capas ocultas. Como entrada, toma un vector de tamaño 42, que se conecta a la capa número 2, con activación ReLU (Rectified Linear Unit), seleccionada debido a su simplicidad y capacidad para introducir no linealidades en el modelo. Las dos siguientes capas son iguales. Por último, la salida, dada como un número que representa 1 de los 12 gestos, se procesa a través de una capa con activación SoftMax, la cual asigna probabilidades a cada clase. Así, la red produce una distribución de probabilidad sobre las posibles clases, facilitando la interpretación de la salida como la probabilidad de pertenecer a cada clase.

Esta simple arquitectura resulta en una red neuronal con un rendimiento excelente incluso entrenada con menos de 100 casos de prueba por gesto. Dichos datos de entrenamiento son creados mediante un programa que permite añadir decenas de casos de entrenamiento por segundo a partir de la cámara del ordenador. Este programa permite indicar el índice en el que se van a guardar los gestos, y con la barra de espacio, guarda el gesto procesado en un archivo de tipo CSV. Este sencillo programa permite crear cerca de 30 casos de prueba por segundo, haciendo trivial el añadir casos de prueba o nuevos gestos.

Para entrenar la red neuronal, se creó un programa que lee los datos, los divide en datos de entrenamiento y de prueba, y comienza el entrenamiento. A base de prueba y error, se decidió usar 1000 epochs y 64 de tamaño de batch, parámetros que obtienen resultados excelentes. Como optimización, se introdujo la detención temprana, la cual que hace que se obtenga un detenga el entrenamiento alrededor de los 330 epochs.

## Diseño

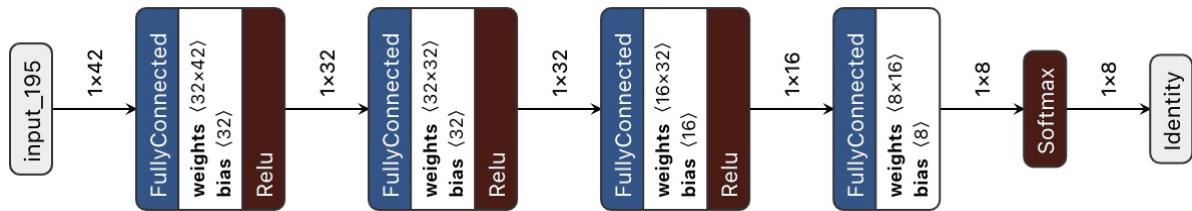


Figura 3.7: Arquitectura de la red neuronal, visualizada con la herramienta pydot<sup>2</sup>.

Con los gestos que aparecen en el gráfico de la figura 3.4 se hizo el modelo entrenado actual, pero está comprobado que incluso con 100 casos de prueba por clase se pueden obtener resultados aceptables, debido a la simple arquitectura de la red neuronal.

## 3.5. Interfaz gráfica

La relevancia de una interfaz gráfica que represente de manera visual los resultados de la aplicación ha sido fundamental. Esta interfaz, empleando OpenCV, se enfoca en mostrar toda la información esencial y facilitar la depuración de errores sin depender de una terminal. OpenCV, reconocida por su especialización en visión computacional, también posibilita la apertura, manipulación y cierre de ventanas. Como se aprecia en la figura 3.9, en lugar de exhibir todo en una sola ventana, se han dispuesto tres ventanas distintas para otorgar mayor flexibilidad al usuario al momento de organizar y visualizar el contenido en pantallas de distintas resoluciones.

La ventana grande izquierda muestra la imagen transmitida por el dron con la pose detectada dibujada por encima. El recuadro verde marca la región de interés sobre la que se realiza el procesamiento de la detección de la mano derecha, cuyos resultados se muestran en la ventana de arriba a la derecha. Si no se dan las condiciones necesarias para que se detecte la mano, la pantalla se presenta en negro. Estas condiciones incluyen, entre otros, que el punto de la muñeca se encuentre por encima del hombro, creando así una región en la que no se transmitan comandos al dron.

Por último, la información del estado del sistema se muestra en la ventana a la derecha abajo. Esta información incluye, en orden:

1. El estado de seguimiento (Following o Not Following)
2. La última instrucción transmitida al dron
3. La batería, en porcentaje
4. El nombre del gesto detectado, o 'None' si no se detecta ningún gesto.

El motor gráfico es un envoltorio ('wrapper') de OpenCV que trabaja como un búfer de gráficos, lo cual asegura un funcionamiento correcto y un rendimiento alto. Tras inicializar la interfaz, se puede cambiar el contenido de los búferes correspondientes a las tres ventanas, pero solo cuando se indique se actualizará lo mostrado en pantalla. Estos dos últimos pasos se ejecutan en cada iteración del bucle. Una vez que no se necesiten las ventanas, se cierran, liberando los recursos consumidos. En resumen:

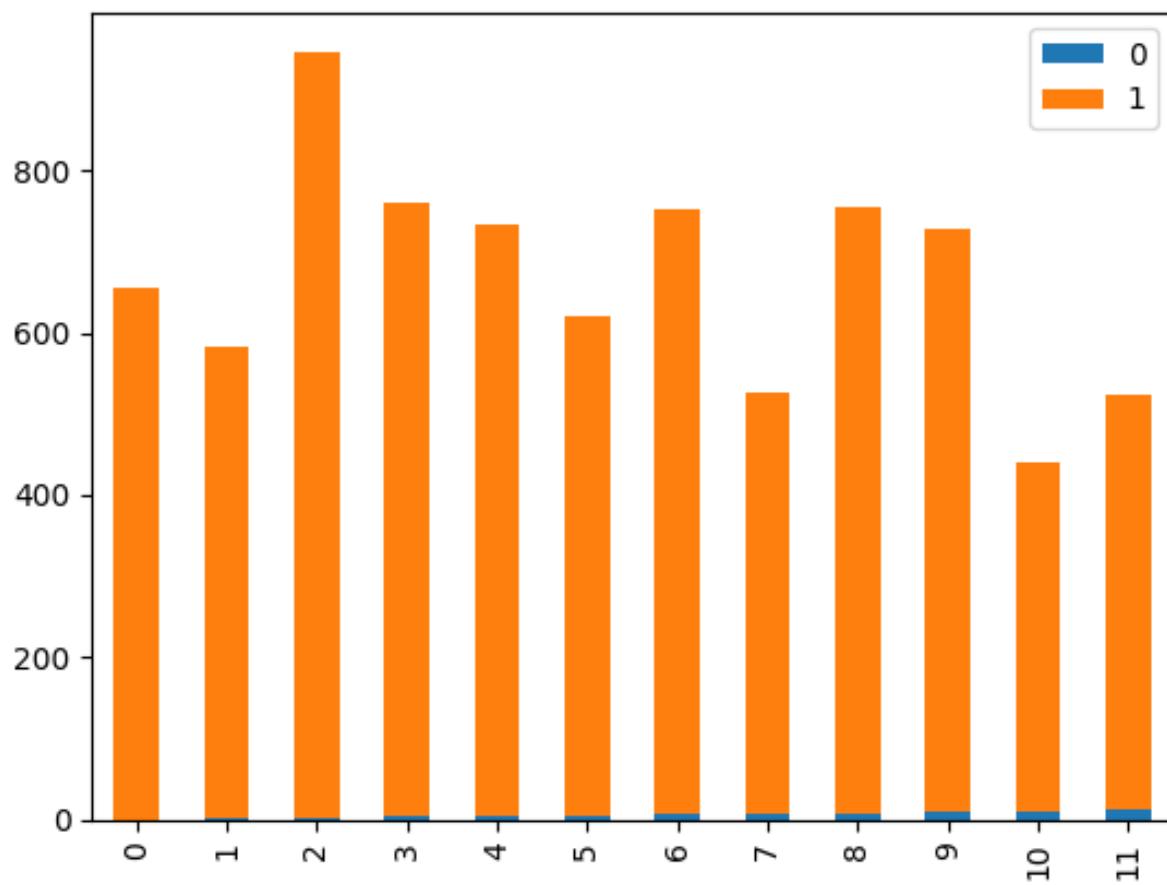


Figura 3.8: División de datos en pruebas (azul) y entrenamiento (naranja).

## Diseño

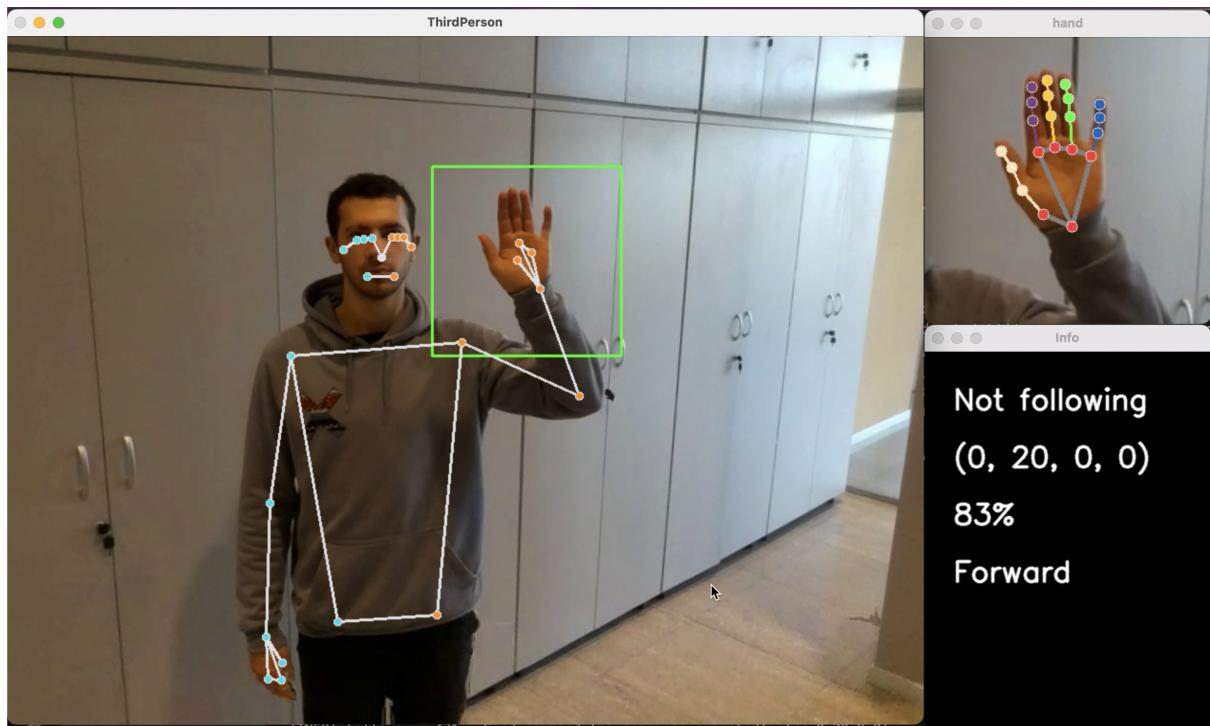


Figura 3.9: Captura de la interfaz gráfica en un uso típico.

1. Se inicializan las ventanas
2. Por cada iteración del bucle,
  - a. se actualiza el contenido de los búferes,
  - b. se refrescan las ventanas, mostrando el nuevo contenido
3. Se cierran las ventanas, liberando los recursos

Con el objetivo de dotar de mayor flexibilidad a las herramientas gráficas, el gestor detecta teclas presionadas, que se pueden usar para añadir funcionalidades adicionales. Un ejemplo de uso es la capacidad de controlar al dron mediante el teclado si se saliera de control. A la hora de hacer pruebas, esta funcionalidad resulta extremadamente útil.

## Capítulo 4

# Programación

En el presente capítulo se van a tratar los aspectos más prácticos: cómo se programó la aplicación, la estructura de esta y otros aspectos importantes.

### 4.1. Estructura del proyecto

El trabajo está programado en su mayoría en Python 3, que, a pesar de tener un menor rendimiento respecto a otros lenguajes como C++, tiene un soporte de librerías de inteligencia artificial excelentes. De librerías usadas, además de las esenciales como un lector de archivos CSV o implementaciones de estructuras de datos, se usan principalmente cuatro:

- **OpenCV**: es una librería especializada en visión computacional pero que goza de un excelente soporte para operaciones de acceso a la cámara del ordenador, usado en la funcionalidad de simulación del trabajo, y para la gestión de ventanas, usado en la interfaz de usuario.
- **TensorFlow**: es la base de todas las operaciones de las redes neuronales usadas en el trabajo. Es una biblioteca amplia y flexible que se utiliza principalmente para construir y entrenar modelos de redes neuronales y otros modelos de aprendizaje profundo.
- **Scikit-learn**: se centra en algoritmos clásicos de aprendizaje supervisado y no supervisado, así como en herramientas para la preparación de datos, evaluación de modelos y selección de características.
- **MediaPipe**: como ya se ha detallado en secciones previas, cumple las funciones de detección de personas y pose y de detección de manos.
- **Numpy**: simplifica algunas operaciones matriciales, específicamente las relacionadas con la manipulación de imágenes.

Para una lista de las versiones específicas de las librerías usadas, véase el archivo `requirements.txt`, que contiene esta información en un formato que facilita la instalación con herramientas como pip.

El proyecto está estructurado de tal manera que cada módulo cumple una función específica, lo cual permite usar funcionalidades concretas sin que sea necesario importar todos los módulos. Como se puede ver en el árbol inferior, el proyecto está es-

## Programación

---

tructurado en cinco carpetas, que contienen las cinco funcionalidades y una carpeta que contiene los archivos necesarios para ejecutar la aplicación, como los modelos. Para almacenar los datos, como los datos de entrenamiento o el nombre de los gestos, se usan archivos CSV, los cuales simplifican en gran medida la labor de abrir y leer los datos.

```
1 ThirdPerson/
2 | -- requirements.txt
3 | -- main.py
4 | -- config.json
5 | -- Dockerfile
6 | -- instructions/
7 |   | -- __init__.py
8 |   | -- gesture_buffer.py
9 |   | -- gesture_instructions.py
10 | -- neural_network/
11 |   | -- __init__.py
12 |   | -- gesture_recognition.py
13 | -- drone_controller/
14 |   | -- __init__.py
15 |   | -- drone_interface.py
16 |   | -- webcam_drone.py
17 |   | -- tello_drone.py
18 | -- mp_utils/
19 |   | -- __init__.py
20 |   | -- mp_hands.py
21 |   | -- mp_pose.py
22 |   | -- pose_hands.py
23 | -- gui/
24 |   | -- __init__.py
25 |   | -- gui.py
26 | -- model/
27 |   | -- add_gesture.py
28 |   | -- model_training.ipynb
29 |   | -- keypoint_data.csv
30 |   | -- hand_landmarker.task
31 |   | -- keypoint_classifier_label.csv
32 |   | -- pose_landmarker_full.task
33 |   | -- pose_landmarker_heavy.task
34 |   | -- keypoint_classifier.tflite
```

Para iniciar la ejecución del programa, se ejecuta `main.py`, que contiene la función principal y coordina los archivos necesarios. El siguiente pseudocódigo representa el funcionamiento de `main.py`:

```
Cargar ajustes de configuración
Inicializar el controlador del dron
Inicializar la interfaz gráfica
Conectarse al dron
```

Inicializar detección de manos y pose

Inicializar instrucciones

Inicializar búfer de gestos

Inicializar reconocedor de gestos

Mientras no se pulse el botón de salir o el dron aterrice:

    Obtener imagen del dron

    Extraer pose de la imagen

    Extraer región de interés (ROI) de la mano derecha

    Extraer el esqueleto de la mano de la ROI

    Identificar el gesto

    Añadir el gesto al búfer

    Calcular el movimiento a ejecutar en base al gesto

    Dibujar la información en las imágenes correspondientes

    Actualizar ventanas

    Ejecutar instrucción

Terminar el dron

Cerrar la detección de pose y manos

Cerrar la interfaz gráfica

A su vez, para ejecutar el archivo principal, se necesita config.json, el cual contiene los parámetros y las configuraciones necesarias para editar el funcionamiento del programa de manera rápida, lo cual supone una mejora crucial a la hora de optimizar el rendimiento. El tipo de archivo JSON ofrece un excelente soporte mediante la librería de Python del mismo nombre, que permite leer y transformar en variables los contenidos de dicho tipo de archivo. Las configuraciones disponibles más importantes son las siguientes:

- Las rutas, todas relativas a la localización de la aplicación, son configurables, para poder cambiar de versión de modelos si hay problemas o mejoras disponibles. Las rutas son:
  - pose\_landmarker: contiene la ruta relativa a la aplicación con el archivo del modelo entrenado para detectar poses con MediaPipe.
  - gesture\_recogniser: es la ruta al archivo que contiene el modelo para distinguir gestos.
  - hand\_landmarker: es la ruta al archivo entrenado para detectar manos con MediaPipe.
  - keypoint\_classifier\_labels: es la ruta al archivo CSV con la lista de los nombres de los gestos.
- Los índices de confianza de los modelos se dan como un valor del 0 al 1:
  - min\_pose\_detection\_confidence: es la confianza mínima de la detección de la persona.
  - min\_pose\_presence\_confidence: es la confianza mínima de la presencia de la pose.
  - min\_pose\_tracking\_confidence: es la confianza mínima del seguimiento

de la pose.

- `min_hand_detection_confidence`: es la confianza mínima de la detección de la mano.
- `min_hand_presence_confidence`: es la confianza mínima de la presencia de la pose de la mano.
- `min_pose_tracking_confidence`: es la confianza mínima del seguimiento de la pose de la mano.
- `show_battery`: muestra el porcentaje de la batería en la terminal.
- `following`: dicta el estado inicial de la aplicación en cuanto al modo de seguimiento.
- `safe_zone`: si esta opción está activa, se realiza la detección de gestos solo cuando la muñeca del sujeto está por encima de su codo.
- `buffer_length`: determina la longitud del búfer circular que almacena los gestos detectados. Cuanto más largo es el búfer, más se tarda en transmitir la instrucción.
- `speed`: establece la velocidad de los movimientos del dron.
- `simulation`: si la opción está activada, las imágenes se obtienen de la cámara del ordenador en vez del de un dron.

Por último, el archivo `Dockerfile` permite coordinar el programa con la ejecución de contenedores de Docker, de lo que se habla en profundidad en la sección 4.2

Las seis carpetas actúan como módulos que contienen los archivos que implementan cada una de las funcionalidades presentadas en el capítulo anterior. Cada módulo, a excepción de `model`, que no lo necesita, contiene un archivo llamado `__init__.py`, usado para definir el paquete. A continuación, se lista cada paquete con una breve descripción y la referencia a la sección que describe en profundidad su funcionamiento.

- `instructions`: corresponde a la sección 3.3; es decir, el módulo encargado de almacenar el estado del dron, y de calcular la siguiente instrucción a ejecutar, dependiendo de si está en el modo de seguir o no. El archivo `gesture_buffer.py` crea el búfer circular mencionado en 3.2.1.
- `neural_network`: como su propio nombre indica, es la red neuronal, descrita en la sección 5 del capítulo 3. Carga la red preentrenada, recibe una matriz correspondiente a la mano detectada, procesa los datos y devuelve el índice de la lista en `/model`.
- `drone_controller`: contiene la clase abstracta de drones, así como dos implementaciones: una para el control de Tellos y otra para una simulación mediante la cámara del ordenador. Para más información, véase la sección 3.1.
- `mp_utils`: es el paquete que permite la detección tanto de poses como de manos. `pose_hands.py` es una clase que junta las funcionalidades de `mp_hands.py` y las de `mp_pose.py` en un solo paquete para simplificar la sintaxis a la hora de ser usadas en un programa llamante. Véase 3.4 para más detalles.

- **gui**: simplemente implementa la interfaz gráfica del usuario. Su funcionamiento está descrito en la sección 3.2.
- **model**: en esta carpeta se encuentran tanto los archivos de los modelos de las redes neuronales preentrenadas, las de Google y las propias, como los dos programas para entrenar la red neuronal propia, `add_gesture.py` para añadir casos de prueba rápidamente al CSV y `model_training.ipynb` para entrenar a la red. Descrito en el 3.5.

## **4.2. Despliegue rápido y fácil con Docker**

Uno de los objetivos de este trabajo fue conseguir que el sistema tuviera soporte para muchos sistemas operativos y arquitecturas. Con este fin, se recurrió a Docker, una plataforma de software que permite crear, probar e implementar aplicaciones rápidamente. Este empaqueta software en unidades estandarizadas llamadas contenedores que incluyen todo lo necesario para que el software se ejecute, incluidas bibliotecas, herramientas de sistema, código y tiempo de ejecución. Con Docker, se puede implementar y ajustar la escala de aplicaciones rápidamente en cualquier entorno con la certeza de saber que el código se ejecutará.

Para coordinar ThirdPerson con Docker, solo son necesarios los archivos de `requirements.txt` y `Dockerfile`. Este último contiene lo siguiente:

```
1 FROM python:3.9
2
3 WORKDIR /app
4
5 COPY . /app
6
7 RUN pip install -r requirements.txt
8
9 ENV DISPLAY=:0
10
11 CMD ["python", "main.py"]
```

El objetivo de este código es crear un contenedor con la versión 3.9 de Python, instalar en dicho contenedor las librerías listadas en `requirements.txt`, habilitar los gráficos para poder mostrar la interfaz gráfica y ejecutar el programa.

Así, para usar ThirdPerson desde cualquier sistema, los pasos se reducen a:

1. Clonar de GitHub el repositorio.
2. Con la última versión de Docker instalado en el sistema, inicializar los contenedores.
3. Ejecutar ThirdPerson.

## **4.3. Calidad del software**

Con el objetivo de crear un sistema que funcionara correctamente, fue necesario que en el procesamiento que se realiza a cada imagen transmitida por el dron se pudiera procesar lo suficientemente rápido como para que el sistema general funcionara en

tiempo real con un mínimo de retraso. Para conseguir esto, fue imprescindible desde el principio asegurar una calidad del software lo suficientemente buena.

### 4.3.1. Control de versiones con Git

Como este proyecto se desarrolló desde distintos ordenadores, Git, el sistema de control de versiones, jugó un papel esencial para coordinar y mantener cada sistema al día de la última versión del trabajo. Como servidor remoto, se usó GitHub<sup>1</sup>. A lo largo de los cuatro meses del desarrollo del presente trabajo se hicieron cerca de 30 pushes.

### 4.3.2. Estilo de programación

El código se lee más de lo que se escribe. Para mejorar la legibilidad del código en el caso de que sea usado por otras personas en futuros proyectos, se ha seguido el estándar especificado en la documentación de Python (<https://peps.python.org/pep-0008/>). Para conseguir este estilo con el mínimo de esfuerzo, se usó la extensión de Visual Studio Code llamada ".<sup>a</sup>utopep8", desarrollada por Microsoft.

---

<sup>1</sup><https://github.com/isotupa/ThirdPerson>

## **Capítulo 5**

# **Experimentación y resultados**

### **5.1. Pruebas**

El objetivo de las pruebas realizadas es asegurar el correcto funcionamiento de las funcionalidades integrantes de la aplicación final. Las pruebas realizadas se dividen en dos secciones: la visión computacional y la interacción con el dron. La primera se puede hacer con el modo simulación y programáticamente, pero para la segunda es necesario hacerlo con un dron.

#### **5.1.1. Pruebas de la red neuronal**

Antes de entrenar el modelo de la red neuronal, se dividen los datos disponibles en dos categorías: datos de entrenamiento y datos de prueba. Esta división permite comprobar el rendimiento y la capacidad de predicción de forma precisa y programáticamente. Para ello, se alimentan los datos de prueba a la red neuronal, y se compara el resultado con la etiqueta correcta. El porcentaje de respuestas correctas determina el funcionamiento de la red. Con una división de un 25 % de casos de prueba, una prueba de inferencia revela predicciones correctas en el 100 % de los casos, un valor excelente. No obstante, un valor tan alto puede revelar un sobreajuste en el entrenamiento. Para comprobar esto, se crean unos 100 nuevos casos de prueba por cada gesto, cuya precisión vuelve a revelar un 100 % de éxito, confirmando el funcionamiento correcto. Véase en la figura 5.1 la matriz de confusión asociada a los resultados del entrenamiento.

Para obtener los mejores resultados, es importante proporcionar casos de prueba en los que el gesto se vea desde todos los ángulos posibles. Como se puede ver en la figura, el conjunto de datos usado demuestra unos resultados casi libres de errores (precisión >97 % para todos las clases).

#### **5.1.2. Pruebas de MediaPipe con MediaPipe Studio**

La evaluación del funcionamiento adecuado de las soluciones de MediaPipe se puede realizar con la herramienta creada por Google llamada MediaPipe Studio<sup>1</sup>. Esta página web permite comprobar y personalizar las soluciones usadas. Para ello, se

---

<sup>1</sup><https://developers.google.com/mediapipe/solutions/studio>

## Experimentación y resultados

---

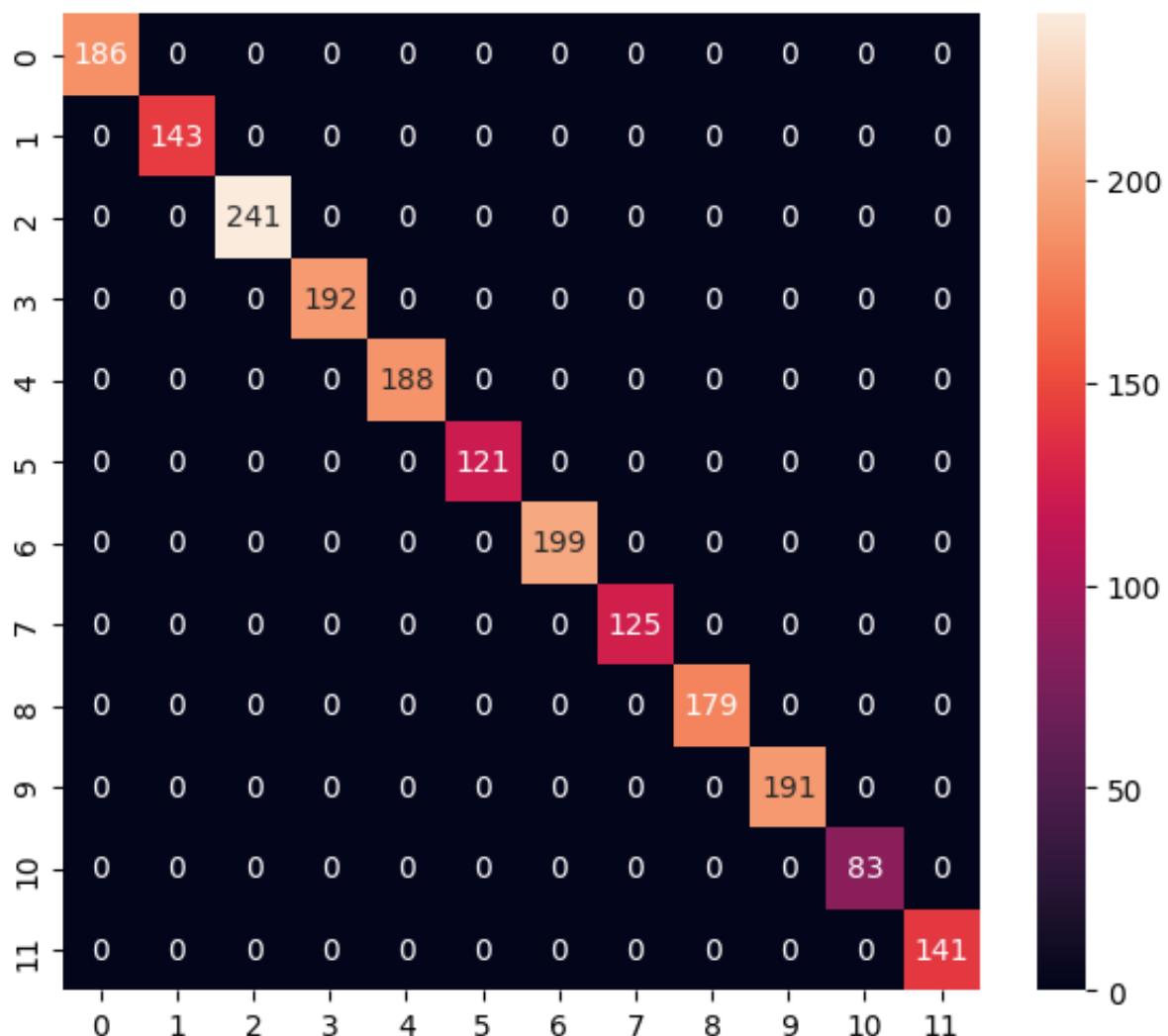


Figura 5.1: Matriz de confusión de los resultados del entrenamiento.

<b>Class</b>	<b>Precision</b>	<b>Recall</b>	<b>F1-Score</b>	<b>Support</b>
0	1.00	1.00	1.00	186
1	1.00	1.00	1.00	143
2	1.00	1.00	1.00	241
3	1.00	1.00	1.00	192
4	1.00	1.00	1.00	188
5	1.00	1.00	1.00	121
6	1.00	1.00	1.00	199
7	1.00	1.00	1.00	125
8	1.00	1.00	1.00	179
9	1.00	1.00	1.00	191
10	1.00	1.00	1.00	83
11	1.00	1.00	1.00	141
<b>Accuracy</b>	1.00			
<b>Macro Avg</b>	1.00			
<b>Weighted Avg</b>	1.00			
<b>Total Support</b>	1989			

Cuadro 5.1: Clasificación de los resultados de las pruebas de la red neuronal.

proporcionan imágenes con personas y se comprueba si la postura detectada es correcta. Véase en la figura 5.2 una prueba usando la cámara del ordenador y en la 5.3 algunas de las imágenes usadas para probar el funcionamiento. Las imágenes usadas para las pruebas son extraídas de la base de datos HaGRID<sup>2</sup>.

### 5.1.3. Integración de la visión computacional y los drones

El dron elegido para las pruebas de este trabajo es el cuadricóptero de la empresa Ryze llamado Tello. Este pequeño dron, que se puede ver en la imagen 5.4, es un modelo bastante básico con tres sensores principales: un giroscopio, detector de altitud y una cámara. Este sencillo sistema basta para volar autónomamente hasta 13 minutos, con una batería cambiante que permite extender este tiempo rápidamente. Este es el dron disponible en el laboratorio en el que se ha desarrollado el presente trabajo, ya que la API<sup>3</sup> abierta ofrece las funciones necesarias para comprobar el funcionamiento del dron.

Una de estas funciones es un método que ejecuta una serie de pruebas: verifica la conexión con el dron, lo despega, lo mueve y lo aterriza. Con estas pruebas, se asegura el funcionamiento de la transmisión de instrucciones al dron y la calibración de este.

### 5.1.4. Pruebas de Docker y del sistema

Por último, para valorar si la aplicación puede funcionar en otros sistemas, se ha realizado una instalación de la aplicación en tres sistemas diferentes: primero, en un MacBook Pro de 16GB de RAM. El segundo, en un sistema operativo Windows 10 con 4GB de RAM. El tercero, Ubuntu 20.04 con 8GB de RAM. En los tres sistemas

<sup>2</sup><https://github.com/hukenovs/hagrid>

<sup>3</sup><https://djtellopy.readthedocs.io/en/latest/tello/>

## Experimentación y resultados

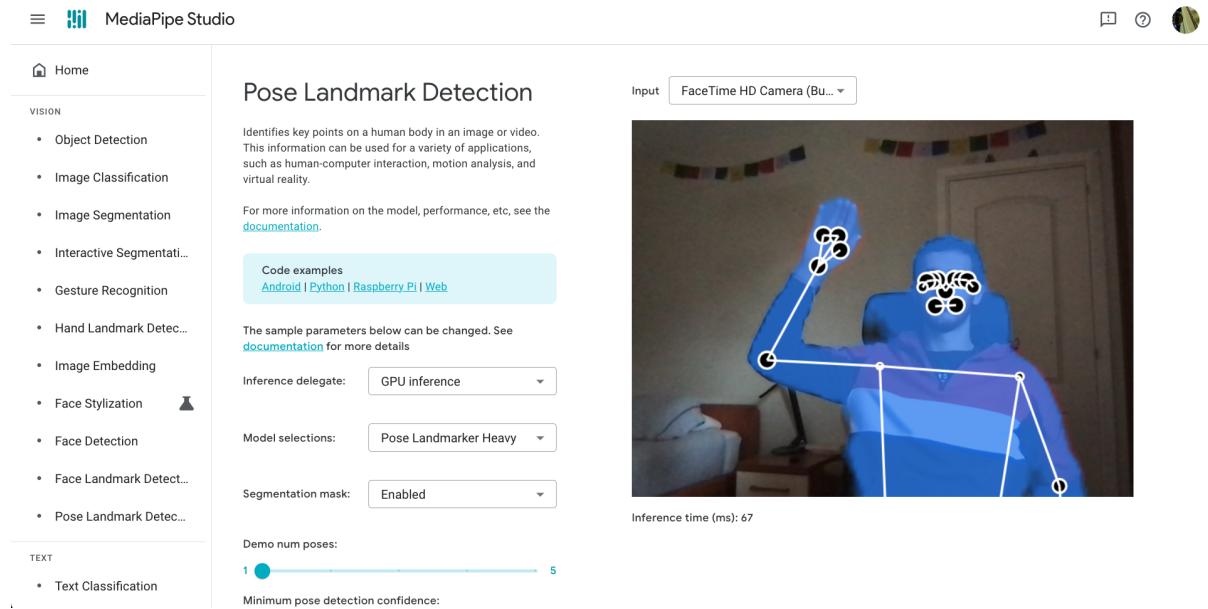


Figura 5.2: Prueba del funcionamiento de MediaPipe Pose usando MediaPipe Studio.

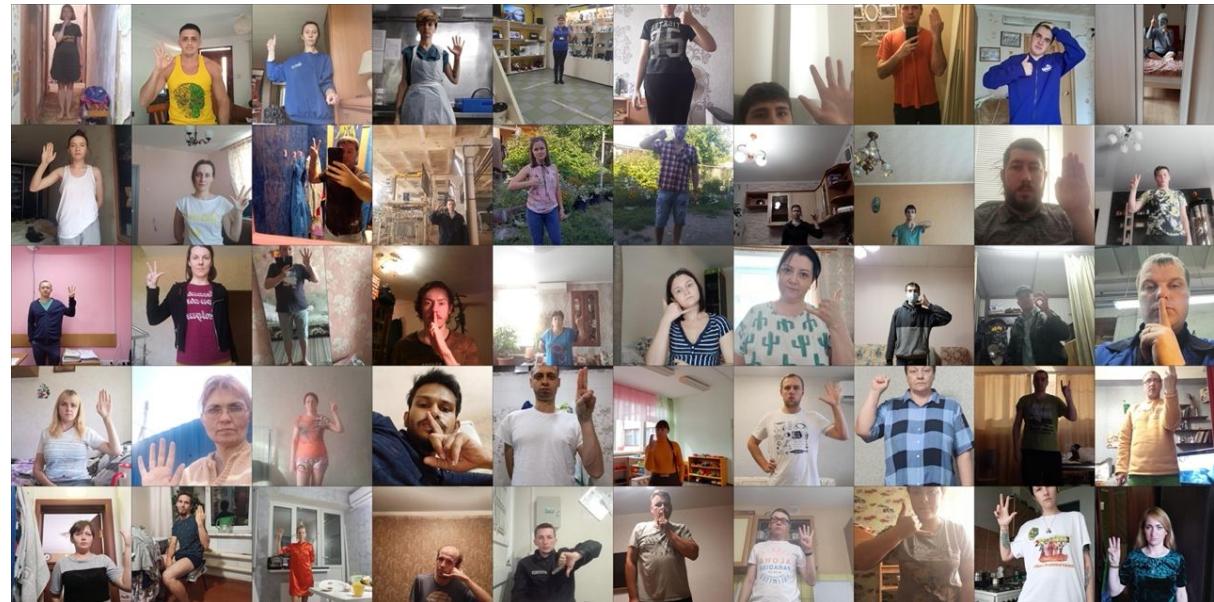


Figura 5.3: Algunas de las imágenes de prueba usadas para probar los gestos.



Figura 5.4: Imagen del dron usado para las pruebas de este trabajo<sup>4</sup>.

se consiguió instalar y superar las pruebas mencionadas en las secciones anteriores. Sin embargo, el rendimiento varió entre sistemas en gran medida. Respectivamente, se obtuvieron cifras en torno a los 25, 8 y 13 FPS, lo que indica que la usabilidad de la aplicación depende de la capacidad de procesamiento de cada sistema.

## 5.2. Esfuerzo de desarrollo

En total, el desarrollo de este trabajo duró 4 meses, de septiembre a diciembre. En este periodo, se escribieron cerca de 5000 líneas en los diversos programas, pero el resultado final son más de 1300 líneas solo en Python, como se puede ver por el conteo realizado por la extensión de VS Code Counter.

Lenguaje	Archivos	Líneas de código	Comentarios	Líneas en blanco	Total
CSV	3	19,979	0	3	19982
JSON	2	3,788	0	4	3,792
Python	18	887	126	270	1,283
Docker	1	6	6	6	18
pip requirements	1	5	0	0	5

Cuadro 5.2: Conteo de las líneas de código por lenguaje.

Finalmente, se consiguió un sistema que coordina tres niveles de redes neuronales para extraer una persona, su mano derecha, y el gesto de una imagen transmitida por un dron, todo a cerca de 20 imágenes por segundo en un MacBook Pro con 16

## **Experimentación y resultados**

---

Path	Files	Code	Comment	Blank	Total
.	25	24,665	132	283	25,080
. (Files)	4	147	6	29	182
drone_controller	4	90	1	29	120
gui	2	45	4	15	64
instructions	3	164	40	40	244
legacy_models	1	71	21	38	130
model	5	23,852	23	41	23,916
mp_utils	4	220	28	55	303
neural_network	2	76	9	36	121

Cuadro 5.3: Conteo de líneas de código por módulo.

GB de RAM. Un vídeo con los resultados está disponible en Google Drive<sup>5</sup>.

---

<sup>5</sup><https://drive.google.com/file/d/1r-PhZ3m6j1w3SDJnh3irRgPy7qEReQmQ/view?usp=sharing>

# **Capítulo 6**

## **Conclusiones**

La evolución continua de la inteligencia artificial ha impulsado el desarrollo de tecnologías innovadoras como la visión computacional. Lo que antes eran considerados proyectos de alta complejidad reservados solo para especialistas, ahora se han convertido en un campo accesible y aplicable a diversas áreas. El objetivo primordial de este trabajo fue demostrar, a través de una implementación práctica, las capacidades que las técnicas de visión artificial brindan para facilitar la interacción visual entre seres humanos y drones. Para lograrlo, se delinearon cuatro metas principales.

El análisis de la comunicación entre el operador humano y el dron constituyó un proceso fundamental para comprender los requisitos esenciales que esta interacción implicaba. Se llevó a cabo una evaluación exhaustiva de los elementos clave que intervienen en esta comunicación, como la transmisión de datos, la interpretación de comandos y la retroalimentación en tiempo real entre el dron y el operador.

En la siguiente fase, se emprendió una investigación para explorar y evaluar las potenciales soluciones. Este análisis exhaustivo abarcó una variedad de herramientas de vanguardia, como MediaPipe, OpenCV y YOLO, entre otras tecnologías de visión artificial y procesamiento de imágenes. Cada una de estas herramientas se sometió a pruebas para determinar si eran adecuados para la resolución de los retos identificados en la comunicación operador-dron.

Una vez identificadas las soluciones más prometedoras, se procedió a su implementación práctica. Esto implicó el aprovechamiento de una gama de tecnologías disponibles, entre las que se incluyen, pero no se limitan a, Python, Docker y Jupyter. La utilización efectiva de estas herramientas permitió la traducción de los conceptos teóricos en una aplicación concreta y funcional.

La etapa final de este proceso fue la realización de pruebas exhaustivas y rigurosas para validar la efectividad y la fiabilidad de los sistemas implementados en condiciones de tiempo real. Estas pruebas involucraron escenarios diversos para comprobar la capacidad del sistema en situaciones dinámicas y variables, asegurando así su funcionamiento correcto y su adaptabilidad a diferentes entornos.

Como resultado de este proceso, se logró la creación de un sistema altamente modular capaz de conectarse a un dron, analizar y procesar imágenes en tiempo real, así como llevar a cabo un seguimiento de personas y el control del dron mediante gestos. Este trabajo se ha bautizado como ThirdPerson, como referencia a la cámara

## Conclusiones

---

de películas y videojuegos que persiguen a personajes, y está disponible para su uso libre en GitHub<sup>1</sup>.

### 6.1. Líneas futuras

El trabajo explora las posibilidades que surgen de la combinación de la visión computacional con los UAV. Como tal, las aplicaciones que puede llegar a tener este proyecto son varias, e incluyen, entre otras:

- **Más drones:** esta mejora, aunque es la que más trabajo requeriría, también es la que más posibilidades abriría. En la última versión del trabajo, se pueden usar dos o más drones simultáneamente. Sin embargo, con ciertos cambios en la programación, hasta se podrían abrir vías de comunicación entre los drones. Esto permitiría, por ejemplo, grabar a varios sujetos a la vez, o grabar a uno solo pero desde distintos ángulos.
- **Más gestos y más acciones:** la última versión de la red neuronal preentrenada permite el reconocimiento de 12 gestos. Sin embargo, no todos tienen una acción asignada. Posibles mejoras en la aplicación incluirían acciones más sencillas, como "mover el dron en círculo", hasta acciones preprogramadas, como "aléjate 5 metros, mantente quieto 5 segundos y vuelve a la posición de inicio". Otra mejora en los gestos sería hacerlos dinámicos; es decir, que en vez de que un gesto sea, por ejemplo, levantar el pulgar, fuera mover la mano hacia arriba. Por un lado, esto haría que los gestos fueran más naturales y más fáciles de recordar, pero por otro, la detección se complicaría, debido a que el reconocimiento de manos no siempre es fiable.
- **Detección de colisiones:** uno de los principales problemas en este momento es que, si se le da al dron una instrucción equivocada, este no dudará en estrellarse contra una pared o el techo. Para solventar esto, se podría añadir la capacidad de detectar dónde están las paredes. Para ello, una opción sencilla sería usar un segmentador de imágenes, y que, antes de ejecutar una instrucción, detecte si tiene un objeto cerca. Por supuesto, esto no evitaría todas las colisiones, pero por lo menos pararía algunas de las más obvias.
- **Detección de profundidad:** para un mejor seguimiento en el eje delante/detrás, se puede añadir una detección de profundidad más fiable. Sistemas como Monodepth2[15] o ZoeDepth[16], entre otros, hacen uso de los avances en las redes convolucionales para dotar a los ordenadores de comprensión de profundidad, que puede mejorar las capacidades de seguimiento de robots aéreos.

---

<sup>1</sup><https://github.com/isotupa/ThirdPerson>

# Bibliografía

- [1] N. Dalal y B. Triggs. «Histograms of oriented gradients for human detection». En: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. Vol. 1. 2005, 886-893 vol. 1. DOI: 10.1109/CVPR.2005.177.
- [2] Camillo Lugaressi et al. *MediaPipe: A Framework for Building Perception Pipelines*. 2019. arXiv: 1906.08172 [cs.DC].
- [3] Valentin Bazarevsky et al. *BlazePose: On-device Real-time Body Pose tracking*. 2020. arXiv: 2006.10204 [cs.CV].
- [4] Zhe Cao et al. «Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields». En: *CVPR*. 2017.
- [5] Tomas Simon et al. «Hand Keypoint Detection in Single Images using Multiview Bootstrapping». En: *CVPR*. 2017.
- [6] Shih-En Wei et al. «Convolutional pose machines». En: *CVPR*. 2016.
- [7] Z. Cao et al. «OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields». En: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2019).
- [8] Chien-Yao Wang, Alexey Bochkovskiy y Hong-Yuan Mark Liao. *YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors*. 2022. arXiv: 2207.02696 [cs.CV].
- [9] Steven Macenski et al. «Robot Operating System 2: Design, architecture, and uses in the wild». En: *Science Robotics* 7.66 (2022), eabm6074. DOI: 10.1126/scirobotics.abm6074. URL: <https://www.science.org/doi/abs/10.1126/scirobotics.abm6074>.
- [10] Miguel Fernandez-Cortizas et al. *Aerostack2: A Software Framework for Developing Multi-robot Aerial Systems*. 2023. arXiv: 2303.18237 [cs.RO].
- [11] Bin Hu y Jiacun Wang. «Deep learning based hand gesture recognition and UAV flight controls». En: *International Journal of Automation and Computing* 17.1 (2020), págs. 17-29.
- [12] Chang Liu y Tamás Szirányi. «Real-time human detection and gesture recognition for on-board UAV rescue». En: *Sensors* 21.6 (2021), pág. 2180.
- [13] Chang Liu y Tamás Szirányi. «Gesture Recognition for UAV-based Rescue Operation based on Deep Learning.» En: *Improve*. 2021, págs. 180-187.
- [14] Ramon A. Suárez Fernández et al. «Natural user interfaces for human-drone multi-modal interaction». En: *2016 International Conference on Unmanned Aircraft Systems (ICUAS)*. 2016, págs. 1013-1022. DOI: 10.1109/ICUAS.2016.7502665.
- [15] Clément Godard et al. «Digging into Self-Supervised Monocular Depth Prediction». En: (oct. de 2019).

## BIBLIOGRAFÍA

---

- [16] Shariq Farooq Bhat et al. *ZoeDepth: Zero-shot Transfer by Combining Relative and Metric Depth*. 2023. arXiv: 2302.12288 [cs.CV].