ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA MECÁNICA

"DESARROLLO E IMPLEMENTACIÓN DE UN SISTEMA DE VISIÓN ARTIFICIAL Y SEGUIMIENTO DE OBJETIVOS HUMANOS POR UN CUADRICÓPTERO DE EXTERIORES UTILIZANDO MATLAB"

PROYECTO PREVIO A LA OBTENCIÓN DEL TÍTULO DE MAGÍSTER EN MECATRÓNICA Y ROBÓTICA

ING. ERICK PATRICIO HERRERA GRANDA
erickherreragranda@hotmail.com

ING. JONATHAN GABRIEL LOOR BAUTISTA
joga3001@hotmail.com

DIRECTOR: ING. MARIO GERMÁN GRANJA RAMÍREZ M.Sc.
mario.granja@epn.edu.ec

Quito, Mayo, 2018

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por los señores **ERICK PATRICIO HERRERA GRANDA** y **JONATHAN GABRIEL LOOR BAUTISTA**, bajo mi supervisión.

Ing. Mario Germán Granja Ramírez M.Sc.
DIRECTOR DE PROYECTO

DECLARACIÓN

Nosotros, **ERICK PATRICIO HERRERA GRADA** y **JONATHAN GABRIEL LOOR BAUTISTA**, declaramos bajo juramento que el trabajo aquí descrito es de nuestra autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que hemos consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedemos nuestros derechos de propiedad intelectual correspondiente a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

Ing. Erick Patricio Herrera Granda	**Ing. Jonathan Gabriel Loor Bautista**

DEDICATORIA

Este proyecto va dedicado a mi querido padre José Luis Herrera, porque gracias a su incansable esfuerzo y el invaluable ejemplo de vida que me ha demostrado día a día, así como a mis hermanos Israel y Dayana, podemos ser siempre mejores personas y profesionales. Él nos ha enseñado que siempre tendremos la responsabilidad de retribuir al mundo y a la sociedad con el fruto de los dones que Dios y la vida nos han dado.

Erick

Este proyecto se lo quiero dedicar a un gran hombre que con su ejemplo de vida enseñó a toda su familia y personas que lo conocían, que la familia representa lo más valioso que una persona tiene para disfrutar durante este hermoso regalo llamado vida, a ti extrañado abuelito Vicente Gilberto Bautista Herrera (+).

Jonathan

AGRADECIMIENTOS

Agradezco a toda mi familia, especialmente a mi querida madre María Norma Granda Gavilanez, a mi padre José Luis Herrera, a mis hermanos Israel y Dayana que me han ayudado en todo el proceso que ha sido cursar esta Maestría, y me han brindado su apoyo incondicional, así mismo, la fuerza necesaria para poder seguir adelante y no permitir que me doblegue ante toda la responsabilidad y esfuerzo que ha implicado estos estudios superiores.

También agradezco el haber conocido en esta Maestría a los mejores amigos que he tenido el gusto de conocer, Geovanny Romero, Jonathan Loor y Luis Mayorga, "Los Gurús". Gracias a ustedes por ser los amigos más incondicionales ya que con la amistad tan sólida que se ha construido, hemos logrado superar todas las barreras que se nos han presentado, sin importar lo difícil que ha sido siempre hemos salido adelante como amigos.

Agradezco al Ing. Mario Granja, ya que sus enseñanzas desde el pregrado y a lo largo de la Maestría han inculcado en mí el gusto por la Robótica.

Erick

Quiero agradecer a la Facultad de Ingeniería Mecánica por los programas de Maestría que permiten que los profesionales de las diferentes áreas puedan adquirir nuevos conocimientos, y de esta forma estén preparados para los nuevos retos que exige la sociedad a los profesionales. Especialmente un agradecimiento grande al Ing. Mario Granja por su apoyo en la culminación con éxito de este proyecto, sus ideales y conocimientos han sido fundamentales en investigaciones que favorecen al beneficio de la sociedad.

Los agradecimientos a la familia nunca estarán limitados a un solo proyecto sino al día a día con las acciones. Gracias infinitas a mi madre Ana María Bautista Guerrero, ningún proyecto valdrá los agradecimientos que tengo hacia usted. Un agradecimiento a mis hermanos Andrés, Erick, Juanito y Naiara que siempre han sido de gran apoyo en todos los proyectos que he iniciado, sus palabras valen mucho para mí.

Un agradecimiento sincero a ese grupo de amigos incondicionales “Los Gurús”, Erick Herrera, Geovanny Romero y Luis Mayorga, ya que gracias a ese apoyo fraterno se ve cumplido otra meta de las muchas que nos hemos trazado como profesionales.

Jonathan

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

En el presente proyecto se desarrolló un sistema de visión artificial, con el objetivo de demostrar su aplicación para la detección de rostros humanos en tiempo real mediante una entrada de video. Mediante el procesamiento de imágenes digitales se pudo obtener las señales de control para que el dron ejecute su vuelo en tiempo real. Para alcanzar este objetivo, este proyecto se realizó por etapas. Como metodología de detección se empleó un detector en cascada empleando el algoritmo de Viola Jones. Inicialmente se seleccionaron los equipos y software necesarios para la adquisición del video capturado por la cámara a bordo del cuadricóptero. Posteriormente se desarrolló un sistema de procesamiento de imágenes que permitió mejorar la calidad de las imágenes y prepararlas para la detección. Luego por medio de la segmentación de colores de cada frame, umbralización, binarización y detección de bordes fue posible definir automáticamente una región para cada imagen en la cual se ubicó el objetivo humano. De esta manera, ya con la región donde es más probable que se encuentre el rostro, se entrenó un clasificador de cascada que permitió mediante un algoritmo de seguimiento obtener la posición y área donde se encontraba el rostro respecto al centro de la imagen. Finalmente, empleando un controlador comunicado con el dron se logró direccionar el vuelo del mismo para mantenerlo anclado al objetivo.

Palabras clave: Detección, Segmentación, Seguimiento, Umbralización, Visión artificial.

ABSTRACT

In the present project an artificial vision system was developed, with the main purpose of demonstrating that its application is possible in the detection of human faces, in real time, through a video input. By means of image processing the control signals were obtained, in order to controlling the drone in real time. To achieve this goal, this project was carried out in stages. Considering that the detection methodology was a cascade detector using the algorithm of Viola Jones. Initially, the equipment and software necessary, for the acquisition of the video captured by the camera, were selected. Later, an image processing system was developed, in order to improve the quality of the image and prepare it for detection. Then, through the segmentation of colors of each frame, thresholding, binarization and edge detection, the program was able to automatically define a region for each image in which the human objective is located. In this way, having the region where the face is most likely to be found, a cascade classifier is trained, which through a tracking algorithm, will allow to obtain the position and area where the face is located from the center of the image. Finally, using a controller connected to the drone, the flight trajectory will be controlled in order to keep it attached to the target.

Keywords: Artificial vision, Detection, Monitoring, Threshold, Segmentation.

INTRODUCCIÓN

En la actualidad, en consecuencia al gran desarrollo y comercialización de drones, se empieza a pensar en un sinnúmero de aplicaciones que estos podrían tener para el beneficio de la sociedad. Es así, que se dispone en el mercado de sistemas robotizados aéreos más conocidos como drones o UAV, los mismos que pueden ser utilizados en diferentes aplicaciones ya sean industriales, agrícolas y de seguridad. La implementación de visión artificial en estos dispositivos permitiría realizar un análisis de la información presente las imágenes para reconocer en las mismas un objeto de interés, como sería el caso de rostros humanos, y mediante este reconocimiento poder tomar las acciones de control que se establezcan. También podría aplicarse en la industria para la clasificación de objetos en tiempo real, pudiendo tomar acciones de control sobre los objetos que no cumplan las especificaciones establecidas en la producción. Además, se pueden desarrollar otro tipo de aplicaciones de entretenimiento, como el seguimiento y registro en video durante actividades deportivas, espectáculos, viajes, etc.

Las dificultades que se tienen con estos dispositivos en sus respectivas aplicaciones incluyen el tiempo de vuelo que pueden soportar bajo una carga completa de sus baterías, la vulnerabilidad a factores climáticos como el viento y lluvia, la dependencia de un operario humano que controle su vuelo, entre otras. Por otro lado, para que los drones comerciales que se comercializan actualmente realicen la tarea de seguimiento, es necesario que el objetivo vaya llevando consigo un Smartphone previamente sincronizado con el dron, de esta manera no sería posible su aplicación en seguridad, y se limita en gran medida su aplicación industrial y demás.

Objetivo General

Desarrollar e implementar un sistema de visión artificial y seguimiento de objetivos humanos por un cuadricóptero de exteriores utilizando MATLAB.

Objetivo Específicos

- Desarrollar el algoritmo para el seguimiento y detección del objetivo.
- Implementar un controlador de vuelo para un cuadricóptero.
- Realizar pruebas de funcionamiento.

1. MARCO TEÓRICO

1.1. Introducción

Cuando se habla de visión artificial se hace mención a una combinación de hardware y software que mediante la adquisición, procesamiento y análisis de imágenes permite la ejecución, repetibilidad, automatización, inspección y control de tareas, que para una persona podrían resultar complejas, y en muchos casos tediosas. Si bien la capacidad de abstracción de una imagen para una persona resulta muy superior al de un software de visión artificial, este último se destaca por su velocidad de procesamiento para la toma de decisiones en un lapso corto de tiempo, además, de su objetividad. Por otra parte la ventaja de una persona con respecto a la visión artificial radica en el análisis cualitativo del objeto en estudio. La visión artificial tiene la ventaja de realizar un análisis cuantitativo, ya que puede analizar una imagen en pequeñas celdas llamadas píxeles, lo que hace que su incorporación en un sistema de identificación se vuelva preciso.

Por otro lado, la visión artificial debe precisar cierto nivel de inteligencia mediante una retroalimentación a través de una base de datos para que el modelo sea capaz de aprender y realice clasificaciones de objetivos en tiempo real, de una forma mucho más rápida que una persona, aportando seguridad e información más completa.

1.2. Adquisición de imágenes

El primer paso para la detección de objetivos es adquirir las imágenes del medio, que luego serán procesadas y clasificadas para el reconocimiento del objetivo determinado previamente. Las imágenes son adquiridas mediante un sistema óptico de una cámara, que por medio de un sensor puede ser convertida en una unidad manejable formada por píxeles que posteriormente será procesada. Para que una imagen sea capturada básicamente se requiere de una fuente de energía que ilumine el área de interés, un objeto que refleje la energía de la fuente y un sensor que capte la energía reflejada, como se puede observar en la figura 1.1.

Para el presente proyecto el sensor que captura la energía reflejada del objeto será la cámara de un dron, por lo tanto la energía medible será la luz visible cuya longitud de onda se encuentra entre 400 y 700 nm. La energía captada por la cámara es una onda electromagnética compuesta de fotones carentes de masa, y que pueden cuantificarse mediante los parámetros:

- Energía *E*, medida en electronvoltios [eV]
- Frecuencia *f*, medida en Hertz [Hz]
- Longitud de onda λ, medida en metros [m]

Los elementos de la onda electromagnética se relacionan mediante la velocidad de la luz *c* (3 x 10^8 m/s) y la constante de Planck *h* (6,63 x 10^{-34} J) que se incluyen en la ecuación 1.1 y ecuación 1.2.

$$\lambda = \frac{c}{f} \qquad \text{Ec. [1.1]}$$

$$E = h \cdot f \qquad \text{Ec. [1.2]}$$

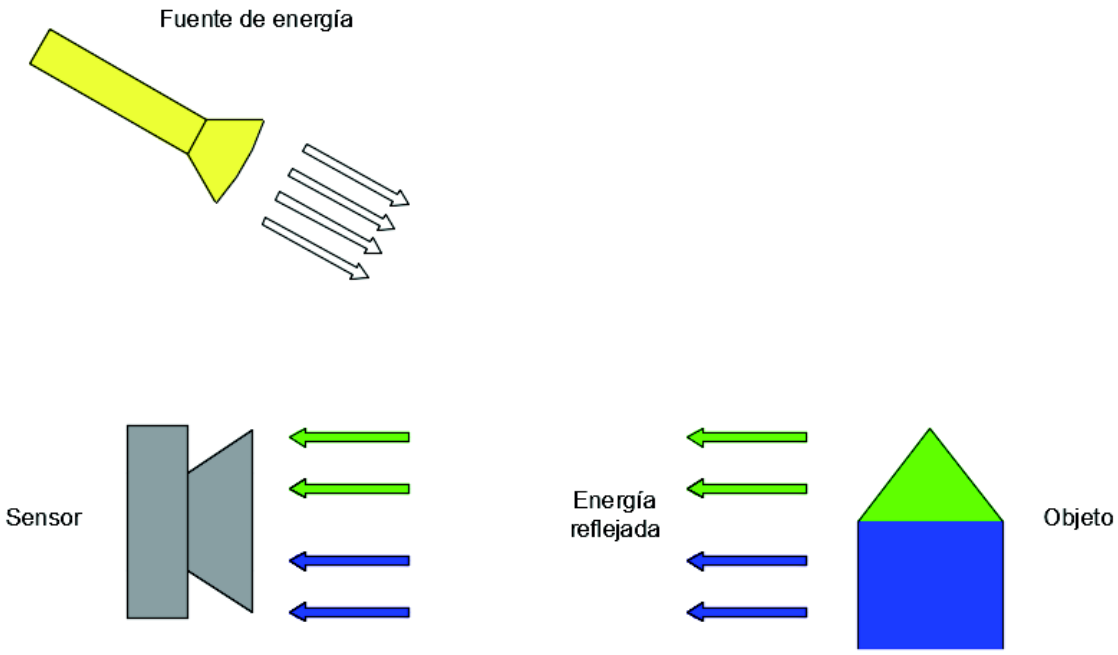


Figura 1.1. Proceso de adquisición de imágenes.

(Fuente: elaboración propia)

La luz reflejada por el objeto en estudio es enfocada por los lentes de una cámara y es grabada por un sensor de imagen. Esta imagen se construye a partir de una matriz de celdas 2D, como se observa en la figura 1.2., donde cada celda de la matriz representa un píxel que mide la intensidad de luz incidente, para posteriormente convertirlo en una señal de voltaje y codificarlo en un número digital, que será una entidad procesable y manejable por un algoritmo de reconocimiento de objetos.

A través de la herramienta “Image Acquisition Toolbox” que Matlab incluye en su librería se puede adquirir directamente imágenes y videos desde cámaras web de bajo costo hasta dispositivos industriales de alta gama.

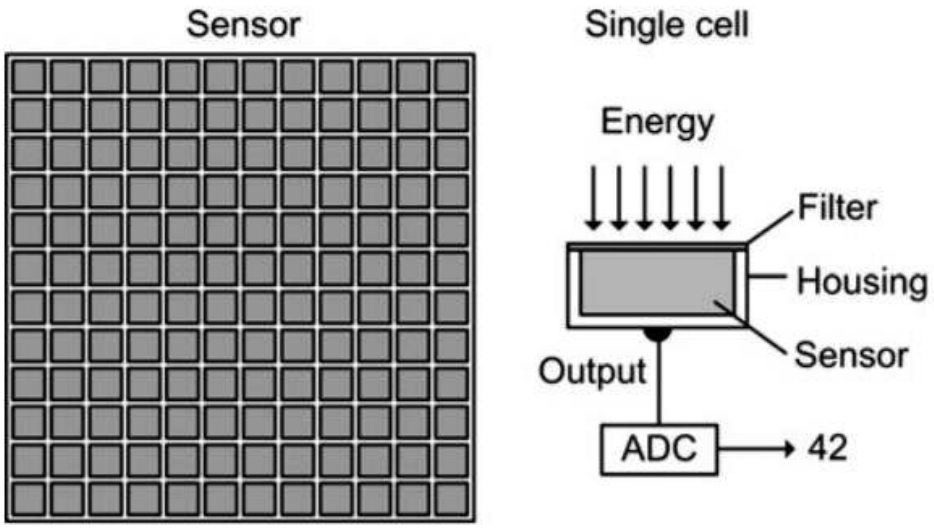


Figura 1.2. Matriz de celdas de un sensor 2D.
(Fuente: Moeslund, 2012)

1.3. Procesamiento de imágenes

Para obtener una imagen, la información que llega al sensor se convierte en un valor en pixeles en el rango [0; 255], que corresponde a la cantidad de luz que golpea cada celda durante el tiempo de exposición, y que se visualiza como un tono que varía entre negro (0) y blanco (255) en la escala de grises formando una matriz 2D, o a su vez cuando se trabaja con color RGB se forma una matriz 3D. Todas las imágenes se representan mediante un par de coordenadas que indican la posición y la intensidad del píxel analizado. Para *f(x, y)*, *x* corresponde a la posición horizontal del píxel, *y* la posición vertical y *f* la intensidad en ese punto. Por lo tanto, cuando se observa una imagen en escala de grises, lo que debe entenderse es que cada celda de la matriz representa un píxel con diferente intensidad, como se muestra en la figura 1.3.

Así Matlab, para el procesamiento de imágenes trabaja con matrices para realizar sus operaciones, donde cada valor de celda corresponde a un dato (píxel). Por ejemplo una imagen de 1 040 columnas y 720 filas tendrá un total de 748 800 datos (píxels) con los que Matlab deberá trabajar.

Cuando se opera con imágenes que contengan color, Matlab hace uso de matrices 3D para el almacenamiento de los datos (píxeles) como se observa en la figura 1.4, donde el primer plano representa la intensidad de los pixeles rojos, el segundo plano representa la intensidad de los píxeles verdes y el tercer plano representa la intensidad de los píxeles azules.

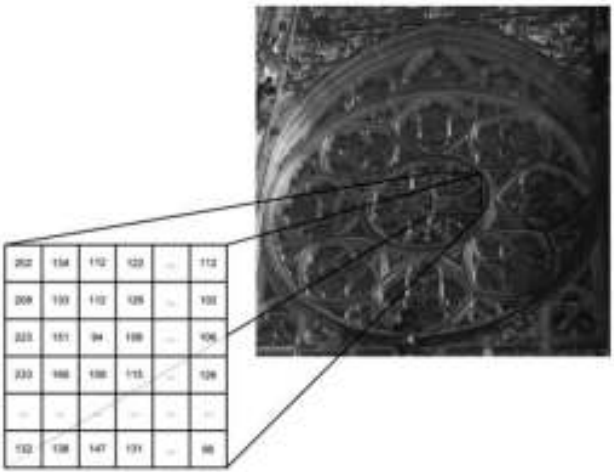


Figura 1.3. Matriz 2D en escala de grises de una porción de una imagen.

(Fuente: Moeslund, 2012)

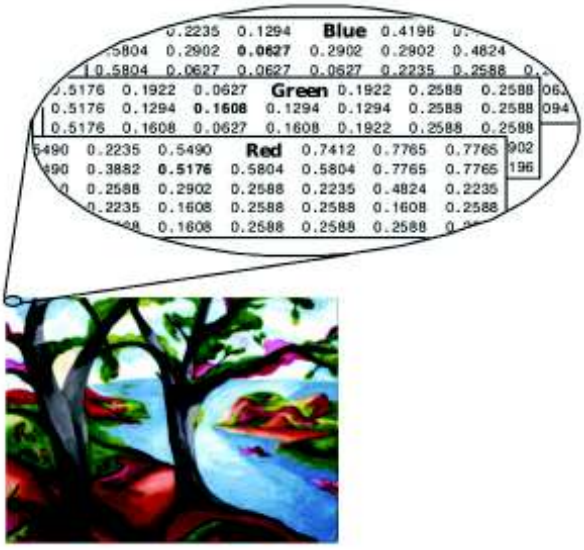


Figura 1.4. Matriz 3D de una porción de una imagen a color.

(Fuente: Matlab, 1999)

1.4. Detección de objetos

Luego del procesamiento de las imágenes, la información recibida por el software debe ser capaz de discriminar la imagen que necesita ser reconocida del medio en el cual se presenta la misma, de una manera eficaz, esto es, rápida, precisa y confiable. Se extraen los parámetros relevantes de la imagen en estudio y se elimina toda la información que no pertenece al objeto que se quiere detectar. El proceso de detección de objetos es fundamental para el preprocesamiento de las imágenes mediante técnicas de segmentación, clasificadores y matching que se describen a continuación.

1.4.1. Segmentación

Para la detección de objetos es fundamental separar las regiones de interés del entorno en el que se encuentra el objeto. Toda la imagen es separada en objetos o regiones más pequeñas, y se realizan una comparación píxel a píxel identificando si pertenece o no al objeto de interés, mediante la codificación en números binarios. Los elementos pertenecientes al objeto con un dígito 1 y los que no pertenecen al objeto con un dígito 0, obteniendo una imagen binaria como se muestra en la figura 1.5.

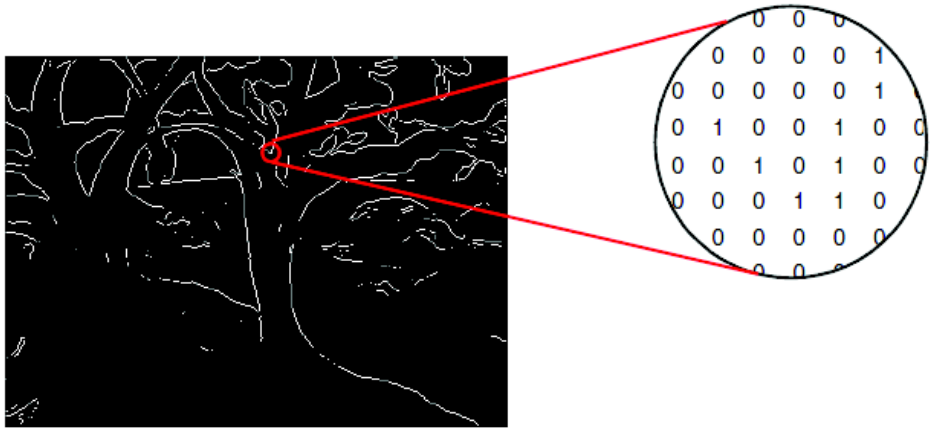


Figura 1.5. Binarización de una imagen.
(Fuente: Matlab, 1999)

1.4.2. Clasificadores

Es el proceso mediante el cual el sistema de visión artificial logra reconocer un objeto en diferentes ambientes, esto se logra mediante un entrenamiento ingresando imágenes que son parte del objeto de interés y realizando una clasificación de las imágenes que no son parte del objeto de interés como se observa en la figura 1.6. Para lograr un preciso entrenamiento del clasificador se hace uso de una red neuronal para que aprenda a reconocer el objeto de interés.

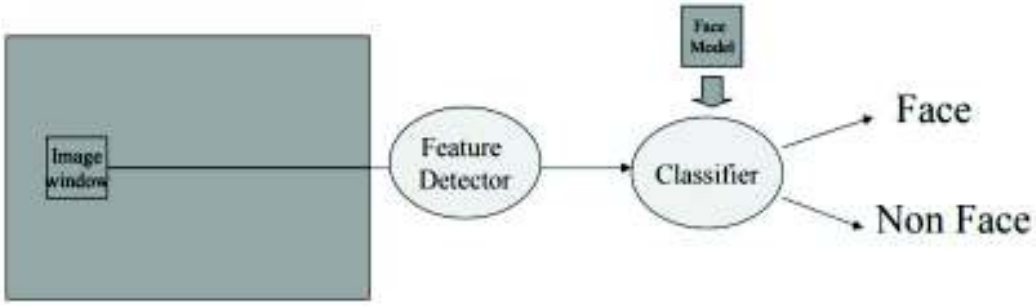


Figura 1.6. Clasificador de una imagen.
(Fuente: recuperado de http://bit.ly/2HOG11x)

1.4.3. Matching

El reconocimiento de un objeto como todo un conjunto en muchos casos resulta no ser tan fácil, ya que si se requiere reconocer un rostro por ejemplo, existen varios rostros que pueden ser similares, por esto es necesario el reconocimiento de características únicas que permitan que el objeto de interés a reconocer sea discriminado de otros objetos similares que se pueden presentar en el entorno de adquisición de imágenes. Las características propias de un objeto deben ser invariantes, por un lado cumplir con una invarianza geométrica cuando se realiza traslaciones, rotaciones o un escalamiento del objeto; así como una invarianza fotométrica al cambio de brillo, exposición, etc. Así, en la figura 1.7. se observa el reconocimiento de características invariantes de un objeto de interés.

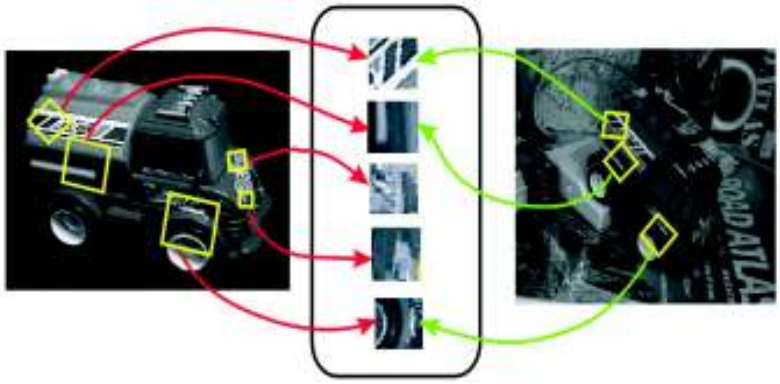


Figura 1.7. Reconocimiento de características.
(Fuente: recuperado de http://bit.ly/2u0VKZr)

1.5. Inteligencia Artificial

No fue hasta el año de 1956 en la Conferencia de Dartmouth[1] que se definió por primera vez el concepto de inteligencia artificial, así, inteligencia artificial es: “hacer que una máquina se comporte como lo haría un ser humano, de tal manera que se la podría llamar inteligente”. La visión artificial como un campo de la inteligencia artificial busca que una máquina o un sistema inteligente sean capaces de abstraer toda la información que contiene una imagen en tiempo real para la toma de decisiones, como lo haría una persona, esto se logra mediante el entrenamiento de clasificadores de objetos mediante redes neuronales, previamente hay que definir los métodos de adquisición y procesamiento de las imágenes digitales para que las mismas sean analizadas mediante los algoritmos

[1] (Banda H., 2014)

desarrollados y obtener la información suficiente y necesaria que permita que la máquina o sistema ejecuten los controles adecuados.

1.5.1. Control Inteligente

El control inteligente es una metodología donde se desarrollan técnicas de control que buscan emular el comportamiento humano para la toma de decisiones. Básicamente, se busca crear modelos inteligentes con la capacidad de aprender y tomar decisiones a partir de grandes cantidades de datos. Las áreas de aplicación mayormente desarrolladas en el campo del control inteligente incluyen las redes neuronales, control difuso, algoritmos genéticos, sistemas expertos y sistemas híbridos.

1.5.2. Redes Neuronales

Las redes neuronales son algoritmos de computadora basados en modelos matemáticos que emulan una red neuronal biológica, con el objetivo de que el modelo se vuelva inteligente y sea capaz de aprender. Esta tecnología informática es muy usada para el procesamiento de imágenes, reconocimiento de patrones y problemas de control. El perceptrón multicapa feedforward mostrado en la figura 1.8. es la red neuronal artificial con mayor cantidad de aplicaciones en los sistemas de control[2]. Esta red neuronal está compuesta por un conjunto de neuronas interconectas donde los *wi* son los pesos y *b* es el sesgo (bias) para la neurona. La señal *z* representa una señal en la neurona y *f* es la función de activación que realiza la señal al ser procesada en la neurona, siempre y cuando la señal de activación sea lo suficientemente grande.

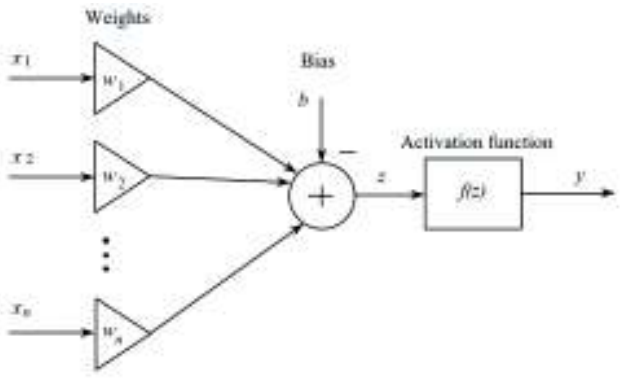


Figura 1.8. Modelo individual de una neurona.

(Fuente: Passino, 2015)

[2] (Passino, 2015)

La función de activación tiene dos posibles resultados[3]:

- Función umbral:

$$f(z) = \begin{cases} 1, & si\ z \geq 0 \\ 0, & si\ z < 0 \end{cases} \qquad \text{Ec. [1.3]}$$

- Función sigmoidea (logística):

$$f(z) = \frac{1}{1 + e^{-z}} \qquad \text{Ec. [1.4]}$$

En la figura 1.9. se observa una red neuronal perceptrón de 3 capas, ya que existen 3 etapas de procesamiento neuronal entre las entradas y las salidas. Los círculos representan las neuronas de la red, esto incluye los pesos, sesgos, señales y funciones de activación, mientras que las líneas representan las conexiones que existen entre las entradas y las salidas de las neuronas para las diferentes capas de la red. Los x_i y los y_i corresponden a las entradas y salidas de las diferentes capas respectivamente, por otra n_1 y n_2 representan el número de neuronas que existen en la primera y segunda capas ocultas, además que, el número de neuronas en la capa de salida está representada por la letra m.

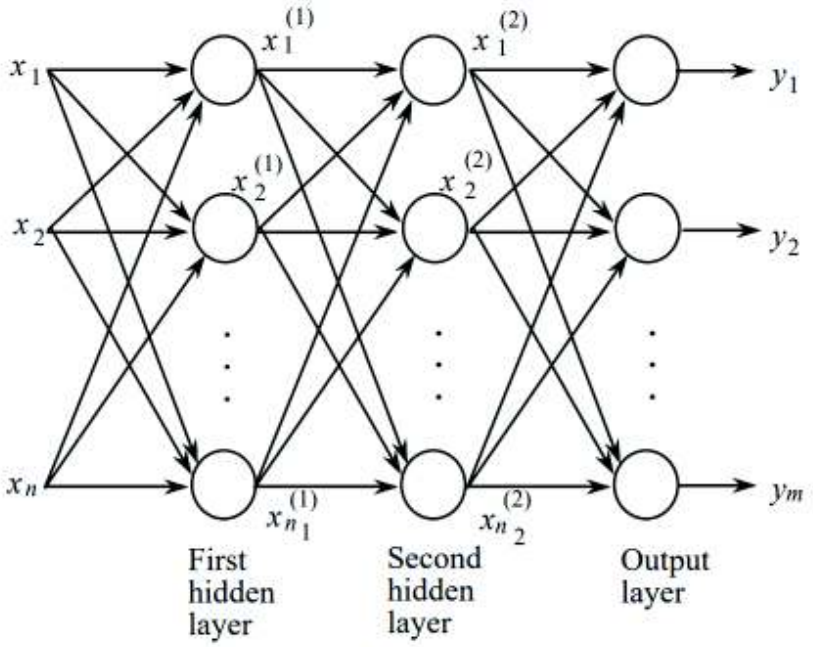


Figura 1.9. Modelo perceptrón multicapa.

(Fuente: Passino, 2015)

[3] (Passino, 2015)

1.5.3. Deep Learning

El Deep Learning o aprendizaje profundo es una forma de aprendizaje en la que un modelo neuronal es capaz de aprender en forma automática, realizando tareas de clasificación a partir de imágenes, texto y audios. Mientras que, una red neuronal puede trabajar con 2 o 3 capas en la red, las redes profundas que se presentan en el Deep Learning pueden tener cientos de capas, como se observa en la figura 1.10. El aprendizaje profundo tiene una gran ventaja sobre las acciones de control que puede tener una persona para la clasificación de objetos, ya que las redes neuronales presentes en estos sistemas inteligentes manejan grandes cantidades de datos, y el conjunto es capaz de tomar decisiones en décimas de segundo, lo que hace que los sistemas basados en aprendizaje profundo sean más eficientes y confiables. La figura 1.11. muestra que en los últimos años, el error cometido por un sistema basado en aprendizaje profundo ha ido disminuyendo notablemente, incluso por debajo del error que podría cometer una persona para una misma acción de control.

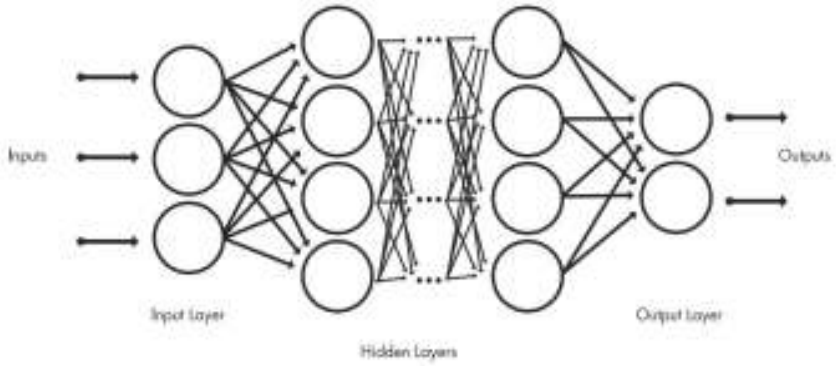


Figura 1.10. Modelo perceptrón multicapa.

(Fuente: Passino, 2015)

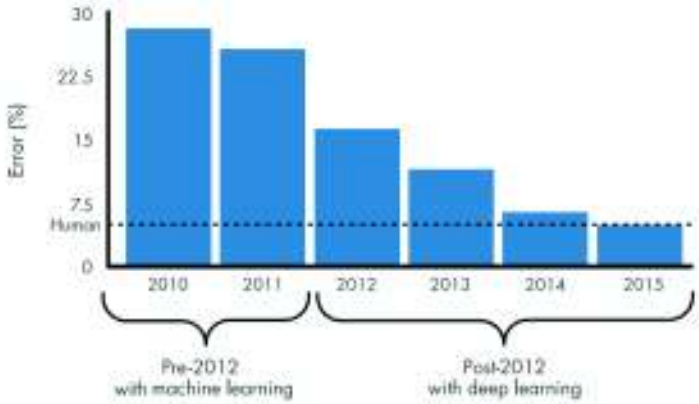


Figura 1.11. Disminución en el error cometido por un sistema basado en aprendizaje profundo.

(Fuente: Passino, 2015)

1.6. Control de vuelo de un cuadricóptero

En años recientes, el desarrollo de vehículos no tripulados (VANT) ha tenido un gran avance, debido a sus innumerables aplicaciones que van desde su uso por entretenimiento hasta investigaciones científicas. El uso de un VANT tiene muchas ventajas para realizar vuelos controlados autónomos con confiabilidad y seguridad, sin embargo una de las limitaciones que actualmente se tiene para estos sistemas aéreos es el tiempo autónomo de vuelo, debido a la capacidad de almacenamiento de energía de las baterías que se usan para su funcionamiento.

Uno de los cuadricópteros utilizados frecuentemente para el desarrollo de proyectos de investigación es el AR. Drone 2.0 de la compañía francesa Parrot que se observa en la figura 1.12., esto debido a su bajo costo y la gran cantidad de sensores con los que cuenta. También hay que destacar su estabilidad en vuelo estacionario que lo hace ideal para el presente proyecto de reconocimiento y seguimiento de objetivos humanos, además, otra ventaja importante es despegue y aterrizaje vertical, lo que permite el uso del mismo en espacios reducidos donde haya gran concurrencia de personas.

Figura 1.12. AR. Drone 2.0 de Parrot.
(Fuente: http://bit.ly/2psQYhO)

El movimiento para el cuadrotor se origina por el cambio de velocidad de los rotores, es así, que si se quiere un movimiento del dron hacia delante lo que debe realizar el controlador es aumentar la velocidad de los rotores traseros y disminuir la velocidad de los rotores delanteros. De igual forma si se requiere el desplazamiento del dron hacia la derecha deberá aumentarse la velocidad de los rotores izquierdos y disminuirse la velocidad de los rotores derechos. Para realizar un movimiento horizontal di giro horizontal debe existir un cambio en el par de torsión entre los pares de rotores, esto es, se debe

aumentar la velocidad de un par de los rotores en sentido horario y el otro par de rotores en sentido antihorario y viceversa.

El modelo dinámico del cuadricóptero considera al mismo como un sólido rígido sometido a un empuje total U_1 que es debido a la suma de las 4 fuerzas que se generan en los rotores (F_1, F_2, F_3 y F_4) y a 4 momentos (U_1, U_2, U_3 y U_4) que se generan por los cambios en los pares de torsión, debido al cambio en los ángulos: roll θ (movimiento de balanceo), pitch ϕ (movimiento de cabeceo) y yaw ψ (movimiento de guiñada), como se observa en la figura 1.13.

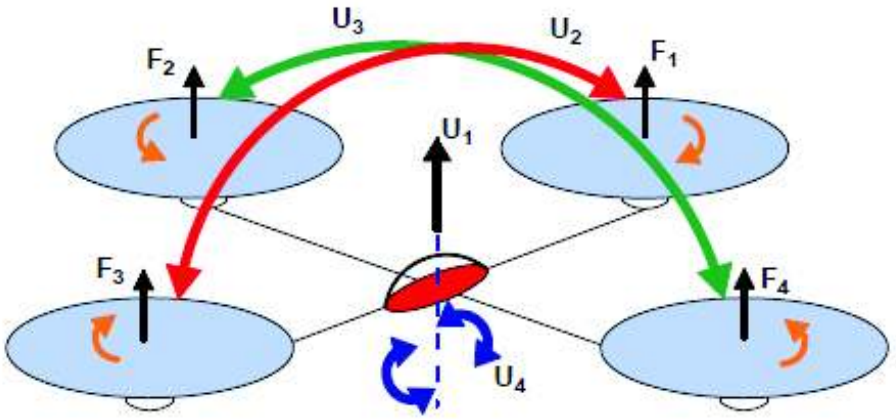


Figura 1.13. Diagrama de fuerzas y momentos a los que es sometido un cuadricóptero.
(Fuente: Raffo G., 2007)

2. METODOLOGÍA

2.1. Introducción

En el presente capítulo se detalla la consecución de los objetivos del proyecto, siendo el objetivo principal el desarrollo e implementación de un sistema de visión artificial y seguimiento de objetivos humanos por un cuadricóptero de exteriores utilizando Matlab. De esta manera las actividades principales desarrolladas en cada apartado son: la selección del hardware apropiado para el proyecto, el desarrollo del modelado dinámico del sistema a ser controlado, y finalmente el desarrollo de los algoritmos de procesamiento de imágenes, reconocimiento y seguimiento de objetivos. Con esta metodología fue posible alcanzar el objetivo de dotar de un sistema de visión artificial para el pilotaje del dron anclado a un objetivo determinado.

2.2. Selección del Hardware

Para el desarrollo del presente proyecto de entre las muchas alternativas de drones que se dispone en el mercado en la actualidad, el dron que resultó ser más eficiente fue el AR. Drone 2.0 de Parrot que se muestra en la figura 2.1. Este modelo en particular de la marca Parrot permite tener una conexión de manera estable y segura con el ordenador, presentando una fácil y estable sincronización mediante el protocolo de transmisión de los fotogramas TCP. El protocolo representa una gran ventaja en este tipo de aplicaciones, ya que garantiza el envío de datos sin errores y que los mismos se reciban en el orden que se están transmitiendo, ya que necesita confirmar que la información se esté recibiendo para poder continuar con el envío de paquetes de datos. Otra consideración importante para la selección de este dron es la gran cantidad de sensores con los que cuenta como se observa en la figura 2.2., entre los que se tienen sensores de altitud y orientación, así como:

- Giroscopio de 3 ejes
- Acelerómetro de 3 ejes
- Magnetómetro de 3 ejes con precisión de 6°
- Sensor de presión ±10 Pa
- Sensor de ultrasonido (altitud)
- Cámara QVGA vertical a 60 FPS para la medición de la velocidad respecto al piso

Por otra parte, muchos proyectos de investigación realizados hasta la fecha han empleado este dron, por lo que se dispone de gran cantidad de información referente al tipo de control a realizar, así como aplicaciones para Smartphone, PC, e incluso herramientas y toolbox para las librerías de Matlab y Linux.

Figura 2.1. AR. Drone 2.0 de Parrot, Elite Edition.

(Fuente: https://bit.ly/2uMzClU)

También se consideró que el AR. Dron 2.0 de Parrot no requiere de elementos electrónicos adicionales que representan mayor complejidad y costo para el desarrollo de aplicaciones. En la tabla 2.1. se visualizan todos los elementos con los que cuenta el AR. Dron 2.0 de Parrot . Además, este drone funciona como un punto de acceso Wi-Fi al cual se pueden conectar smartphones u ordenadores sin necesidad de un router adicional, facilitando en gran medida la tarea de sincronización de los dispositivos y mejorando la velocidad de transmisión de datos, lo que garantiza un envío y recepción de información fácil y seguro.

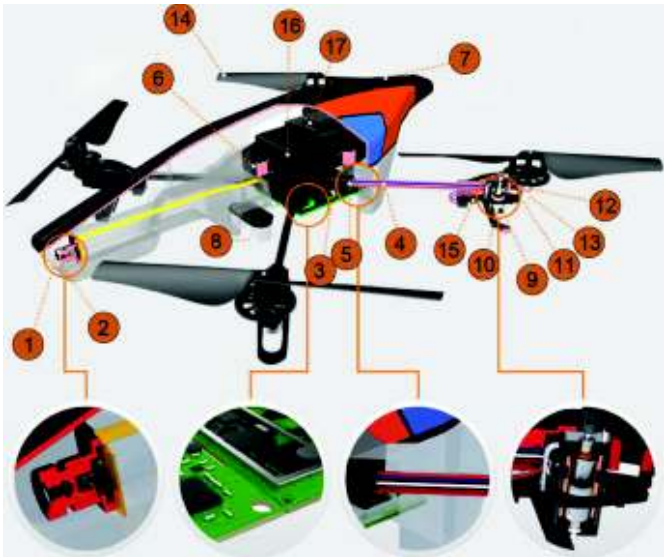


Figura 2.2. Especificaciones dron Parrot AR. 2.0.

(Fuente: https://bit.ly/2q2o1u3

Tabla 2.1. Especificaciones técnicas del dron Parrot AR. 2.0.

Número	Descripción
1	Cámara HD, 720px, 30 FPS
2	Grúa para grabación de video
3	Procesador AR M Cortex A8, 32 bits a GHz
4	Tubos de fibra de carbono
5	Piezas de plástico nylon
6	Espuma aislante la inercia de los motores
7	Casco de protección de polipropileno expandido
8	Nanorevestimiento repelente a los líquidos
9	4 motores de rotor interno sin escobillas
10	Rodamiento de bolas en miniatura
11	Engranes de Nylatron de bajo ruido
12	Eje de las hélices de acero templado
13	Cojinete de bronce autolubricante
14	Fuerza de propulsión de las hélices
15	Microcontrolador AVR de 8 MIPS
16	Batería recargable Li-Po de 1000 mAh
17	Puerto USB

(Fuente: https://bit.ly/2q2o1u3)

Por otro lado, este dron posee la ventaja de disponer de un procesador AR M Cortex A8 de 32 bits a 1 GHz con DSP de video TMS320 DMC64x a 800 MHzLinux 2.6.32, y una memoria RAM DDR2 de 1 Gb a 200 MHz, configuración de software que resulta de gran utilidad para desarrolladores, ya que en ella se pueden subir y almacenar programas de control de vuelo mediante una previa sincronización con el ordenador. En el caso particular de Matlab se dispone de librerías y toolbox desarrollados como el AR Drone Simulink Development-Kit V1.1, AR.Drone 2.0 Support from Embedded Coder, Control AR Drone Parrot 2.0 with Matlab 2015a and Vicon, entre otros, que son de gran utilidad ya que permiten subir con facilidad los controladores de vuelo desarrollados en Simulink o Scripts al procesador a bordo del dron.

Este dron emplea 4 motores de rotor interno (inrunner) sin escobillas de 14,5 a 28,5 W de potencia, que le brindan un vuelo estable y producen un ruido moderado. Cuenta con una cámara principal HD de 1080 por 720p a 30 FPS con objetivo angular de 92º y perfil de codificación H264 que presenta una difusión en tiempo real de baja latencia. Finalmente, emplea baterías recargables de LiPo de tres elementos con una autonomía de 1000 mAh

que en su versión estándar brinda un tiempo de vuelo de 12 minutos y en su versión power edition alcanza 36 minutos empleando baterías de 1500 mAh de alta densidad.
Es así, que este dron fue seleccionado por la amplia gama de virtudes antes mencionadas, especialmente debido a que es uno de los escasos drones que permite sincronización y control desde el ordenador de una manera muy estable, y cuyo precio es el más accesible en el rango de las características antes mencionadas.

2.3. Desarrollo del Controlador

Para el desarrollo del controlador de vuelo del dron fue necesario tomar como punto de partida el hecho de que este tipo de sistema es no lineal, ya que las matrices de rotación y velocidades angulares involucran funciones trigonométricas. Para controlar y generar un desplazamiento en un sistema referencial x, y, z que se muestra en la figura 2.3., se debe considerar que los cuatro motores del dron son quienes generan un desplazamiento, variando su velocidad angular en conjunto y provocando un empuje en sentido vertical en caso de que se desee desplazar en el eje vertical z, o una variación en la velocidad angular en torno a los ejes para producir un cambio en la orientación del dron. De manera que mediante rotaciones en roll θ y pitch $\emptyset$ se genera un desplazamiento en las direcciones x y y respectivamente, o mediante el incremento de la velocidad angular sincronizado de los motores se genera una rotación φ en torno al eje vertical.
Es así, que conforme con lo antes descrito se concluye que se trata de un problema que debe implicar un modelado dinámico, donde las variables a ser controladas son las posiciones x, y, z y sus respectiva velocidades $\dot{x}$, $\dot{y}$, $\dot{z}$, así como las posiciones angulares θ, $\emptyset$, φ (roll, pitch, yaw) y sus velocidades angulares $\dot{\theta}$, $\dot{\emptyset}$, $\dot{\varphi}$, que dependen únicamente de la correcta sincronización de las velocidades angulares ω_1, ω_2, ω_3 y ω_4 de los motores del cuadricóptero, y debido a que la relación entre las variables es no lineal, se concluyó que se debe emplear una técnica de control para un sistema no lineal complejo de múltiples entradas y múltiples salidas MIMO.
Finalmente, la técnica que se empleó es un modelado dinámico en el espacio de estados, que mediante un controlador linealizado realimentado realiza las acciones de control sobre un sistema no lineal empleando leyes de control no lineales.

2.3.1. Modelo dinámico

Para generar un modelo dinámico se puede considerar al cuadrotor como un sólido rígido F_b que está moviéndose en un espacio tridimensional x, y, z. De la misma forma se puede considerar al piso como un cuerpo rígido fijo en tierra F_e como se muestra en la figura 2.3.

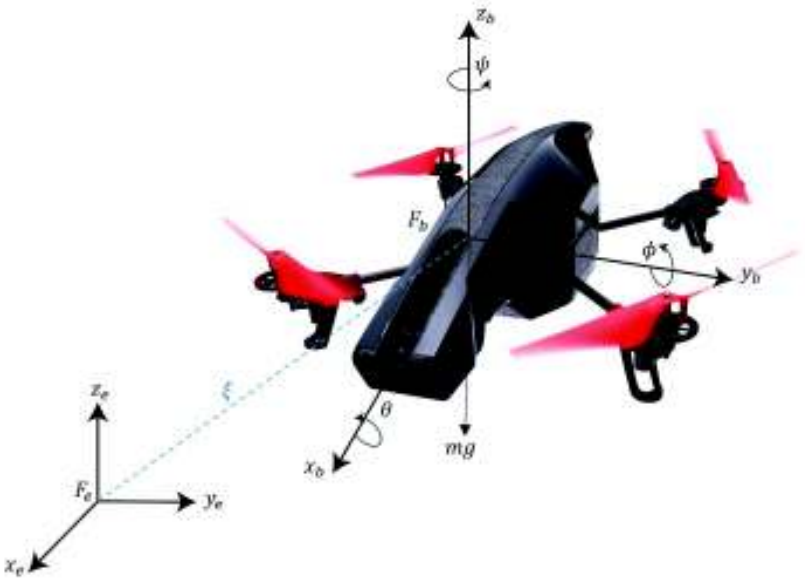


Figura 2.3. Sistema de coordenadas del cuadricóptero.

(Fuente: elaboración propia)

La posición del centro de masa del cuadrotor está en las coordenadas $\xi = [x, y, z]^T$, y su orientación está dada por los ángulos de Euler: yaw, roll y pitch, $\eta = [\psi, \theta, \phi]^T$, de esta manera las ecuaciones dinámicas 2.1 y 2.3 son las que modelan al cuadrotor[4].

$$m\ddot{\xi} = mg\boldsymbol{D} + \boldsymbol{R}F \qquad \text{Ec. [2.1]}$$

$$I\dot{\Omega} = -\Omega \times I\Omega + \tau \qquad \text{Ec. [2.2]}$$

Por otro lado, se definen las matrices $\boldsymbol{D} = [0,0,-1]^T$, $\boldsymbol{R}\ \epsilon\ SO(3)$ que es la matriz de rotación del cuadrotor F_b respecto a tierra F_e, además, el vector de fuerzas aplicado sobre el cuadrotor es $F = [0,0,u]^T$, siendo u el empuje principal que genera el dron sobre el piso para impulsarse. La masa total del cuadrotor está denominada como m y g es la aceleración gravitacional. La variable $\Omega = [r, q, p]^T$ representa la matriz de velocidades angulares del dron, I representa el momento de inercia y τ es el torque total. Tomando a las funciones $sen(*)$ y $cos(*)$ como $S*$ y $C*$ respectivamente, la matriz de rotación R queda expresada por la ecuación 2.3.

[4] (García-Carrillo, y otros, 2013)

$$R = \begin{pmatrix} C\theta C\phi & S\phi S\theta C\psi - C\phi S\psi & C\phi S\theta C\psi + S\phi S\psi \\ C\theta S\psi & S\phi S\theta S\psi + C\phi C\psi & C\phi S\theta S\psi - S\phi C\psi \\ -S\theta & S\phi C\theta & C\phi C\theta \end{pmatrix} \quad \text{Ec. [2.3]}$$

Además, se puede definir un vector adicional $\tilde{\tau} = [\tilde{\tau}_\psi, \tilde{\tau}_\theta, \tilde{\tau}_\phi]^T$ que representa el torque generalizado en el sentido de roll, pitch y yaw como muestra la ecuación 2.4.

$$\tilde{\tau} = I^{-1}W^{-1}\left(-I\dot{W}\dot{\eta} - W\dot{\eta} \times IW\dot{\eta} + \tau\right) \quad \text{Ec. [2.4]}$$

Donde I es el tensor de inercia del dron en [kg · m^2]. La equivalencia entre las velocidades angulares respecto al sistema de coordenadas y en el sistema de referencia móvil se expresa mediante la ecuación 2.5, y la matriz de transformación W mediante la ecuación 2.6.

$$\Omega = W\dot{\eta} \quad \text{Ec. [2.5]}$$

$$W = \begin{pmatrix} -sen\theta & 0 & 1 \\ cos\theta sen\phi & cos\phi & 0 \\ cos\theta cos\phi & -sen\phi & 0 \end{pmatrix} \quad \text{Ec. [2.6]}$$

Mediante la sustitución de las ecuaciones de equilibrio dinámico de fuerzas y torques, y su respectivo despeje se puede reducir el modelo dinámico del cuadrotor mediante las ecuaciones 2.7 – 2.12.

$$\ddot{x} = \frac{u}{m}(cos\psi sen\theta cos\phi + sen\psi sen\phi) \quad \text{Ec. [2.7]}$$

$$\ddot{y} = \frac{u}{m}(sen\psi sen\theta cos\phi - cos\psi sen\phi) \quad \text{Ec. [2.8]}$$

$$\ddot{z} = \frac{u}{m}(cos\theta cos\phi) - g \quad \text{Ec. [2.9]}$$

$$\ddot{\psi} = \tilde{\tau}_\psi \quad \text{Ec. [2.10]}$$

$$\ddot{\theta} = \tilde{\tau}_\theta \quad \text{Ec. [2.11]}$$

$$\ddot{\phi} = \tilde{\tau}_\phi \quad \text{Ec. [2.12]}$$

Donde:

$\ddot{x}$: aceleración en x, medida en [m/s^2]

$\ddot{y}$: aceleración en y, medida en [m/s^2]

$\ddot{z}$: aceleración en z, medida en [m/s^2]

$\ddot{\psi}$: aceleración angular en yaw, medida en [rad/s^2]

$\ddot{\theta}$: aceleración angular en roll, medida en [rad/s^2]

$\ddot{\phi}$: aceleración angular en pitch, medida en [rad/s^2]

Además, se debe tener en cuenta que el empuje principal (comúnmente conocido como gaz) y los momentos angulares $\tilde{\tau}_{\psi}$, $\tilde{\tau}_{\theta}$, $\tilde{\tau}_{\phi}$, son las entradas de control. Para poder modelar un controlador interno a bordo del dron, se proponen las ecuaciones 2.13 – 2.16 que relacionan a la aceleración vertical y aceleraciones angulares, con las entradas del sistema de control[5].

$$\ddot{z} = -a_1\dot{z} + a_3 u_z \qquad \text{Ec. [2.13]}$$

$$\ddot{\phi} = -b_1\dot{\phi} - b_2\phi + b_3 u_{\phi} \qquad \text{Ec. [2.14]}$$

$$\ddot{\theta} = -c_1\dot{\theta} - c_2\theta + c_3 u_{\theta} \qquad \text{Ec. [2.15]}$$

$$\ddot{\psi} = -b_1\dot{\psi} - b_2\psi + b_3 u_{\psi} \qquad \text{Ec. [2.16]}$$

Así, el análisis y resolución de los dos sistemas propuestos desembocarán en la obtención de las leyes de control que gobiernan al sistema. Siendo u_z, u_{ψ}, u_{θ} y u_{ϕ} las entradas de control correspondientes a las componentes del empuje pero divididas en sus diferentes funciones que corresponden a generar el ascenso del dron en el eje z y los momentos angulares en torno a cada eje, mismos que provocarán un cambio de orientación del cuadricóptero. Además, los parámetros a_i, b_i, c_i y d_i están relacionados con las ganancias de control k_i mediante las cuales se obtendrán sus respectivos valores. De esta manera relacionando el modelo dinámico con su controlador se genera el sistema de ecuaciones[6] 2.17 – 2.22.

$$\ddot{x} = (g - a_1\dot{z} + a_3\, u_z)\left(tan\theta\, cos\psi + \frac{\tan\phi}{cos\theta}\, sen\psi\right) \qquad \text{Ec. [2.17]}$$

$$\ddot{y} = (g - a_1\dot{z} + a_3\, u_z)\left(tan\theta\, sen\psi + \frac{\tan\phi}{cos\theta}\, cos\psi\right) \qquad \text{Ec. [2.18]}$$

$$\ddot{z} = -a_1\dot{z} + a_3\, u_z \qquad \text{Ec. [2.19]}$$

$$\ddot{\phi} = -b_1\dot{\phi} - b_2\phi + b_3\, u_{\phi} \qquad \text{Ec. [2.20]}$$

$$\ddot{\theta} = -c_1\dot{\theta} - c_2\theta + c_3\, u_{\theta} \qquad \text{Ec. [2.21]}$$

$$\ddot{\psi} = -d_1\dot{\psi} + d_3\, u_{\psi} \qquad \text{Ec. [2.22]}$$

Como se indicó en la sección anterior, el método a emplearse es el modelamiento por espacio de estados, donde las variables de estado corresponden a las posiciones x, y, z, y sus respectivas derivadas que corresponden a las velocidades tangenciales a los ejes $\dot{x}$, $\dot{y}$, $\dot{z}$, además las posiciones angulares θ , ϕ , ψ y sus respectivas derivadas

[5] [6] (Santiaguillo-Salinas, Rosaldo-Serrano, & Aranda-Bricaire, 20117)

correspondientes a las velocidades angulares $\dot{\theta}$, $\dot{\phi}$, $\dot{\psi}$. De manera se generan las variables de estado que se presentan en las ecuaciones 2.23 – 2.34.

$$x_1 = x \quad \text{Ec. [2.23]}$$

$$x_2 = \dot{x} \quad \text{Ec. [2.24]}$$

$$x_3 = y \quad \text{Ec. [2.25]}$$

$$x_4 = \dot{y} \quad \text{Ec. [2.26]}$$

$$x_5 = z \quad \text{Ec. [2.27]}$$

$$x_6 = \dot{z} \quad \text{Ec. [2.28]}$$

$$x_7 = \phi \quad \text{Ec. [2.29]}$$

$$x_8 = \dot{\phi} \quad \text{Ec. [2.30]}$$

$$x_9 = \theta \quad \text{Ec. [2.31]}$$

$$x_{10} = \dot{\theta} \quad \text{Ec. [2.32]}$$

$$x_{11} = \psi \quad \text{Ec. [2.33]}$$

$$x_{12} = \dot{\psi} \quad \text{Ec. [2.34]}$$

Finalmente al vincularse con las ecuaciones del modelo dinámico antes propuesto, y considerando que x_2, x_4, x_6, x_8, x_{10} y x_{12} son las derivadas de sus respectivos estados, además, de sustituir los tensores de inercia, se tiene el sistema de ecuaciones de estado 2.35 – 2.46.

$$\dot{x}_1 = x_2 \quad \text{Ec. [2.35]}$$

$$\dot{x}_2 = \left(\frac{sen(x_{11})sen(x_7) + \cos(x_{11})\, sen(x_9)\cos(x_7)}{m}\right) U(1) \quad \text{Ec. [2.36]}$$

$$\dot{x}_3 = x_4 \quad \text{Ec. [2.37]}$$

$$\dot{x}_4 = \left(\frac{-cos(x_{11})sen(x_7) + \text{sen}(x_{11})\, sen(x_9)\cos(x_7)}{m}\right) U(1) \quad \text{Ec. [2.38]}$$

$$\dot{x}_5 = x_6 \quad \text{Ec. [2.39]}$$

$$\dot{x}_6 = -g + \left(\frac{cos(x_9)cos(x_7)}{m}\right) U(1) \quad \text{Ec. [2.40]}$$

$$\dot{x}_7 = x_8 \quad \text{Ec. [2.41]}$$

$$\dot{x}_8 = \frac{I_{yy} - I_{zz}}{I_{xx}} x_{10} x_{12} - \frac{J_{tp}}{I_{xx}} x_8 \omega + \frac{U(2)}{I_{xx}} \quad \text{Ec. [2.42]}$$

$$\dot{x}_9 = x_{10} \quad \text{Ec. [2.43]}$$

$$\dot{x}_{10} = \frac{I_{zz} - I_{xx}}{I_{yy}} x_8 x_{12} - \frac{J_{tp}}{I_{yy}} x_8 \omega + \frac{U(3)}{I_{xx}} \quad \text{Ec. [2.44]}$$

$$\dot{x}_{11} = x_{12} \quad \text{Ec. [2.45]}$$

$$\dot{x}_{12} = \frac{I_{xx} - I_{yy}}{I_{zz}} x_8 x_{10} + \frac{U(4)}{I_{zz}} \quad \text{Ec. [2.46]}$$

Donde:

$U(1)$: empuje aplicado en el eje vertical, medido en [N]

$U(2)$: momento angular aplicado en dirección roll, medido en [kg · m^2/s]

$U(3)$: momento angular aplicado en dirección pitch, medido en [kg · m^2/s]

$U(4)$: momento angular aplicado en dirección yaw, medido en [kg · m^2/s]

I_{xx}: momento de inercia del cuadrotor al girar en torno al eje x, medido en [kg · m^2]

I_{yy}: momento de inercia del cuadrotor al girar en torno al eje y, medido en [kg · m^2]

I_{zz}: momento de inercia del cuadrotor al girar en torno al eje z, medido en [kg · m^2]

J_{tp}: momento de inercia total aplicado en torno al eje de propulsión, medido en [kg · m^2]

2.3.2. Formulación del controlador

Como se puede notar en las ecuaciones de estado antes propuestas, el sistema de control de vuelo es no lineal y tiene especial complejidad debido a la presencia de las funciones trigonométricas seno y coseno. Por lo tanto, la propuesta del controlador desarrollado consiste en un controlador realimentado cuyas leyes de control se obtendrán mediante un controlador equivalente linealizado en un punto de equilibrio.

Para obtener el punto de equilibrio lo primero que se realizó es igualar las derivadas de cada estado a cero donde se tiene el punto de equilibrio del sistema, como muestran las ecuaciones 2.47 – 2.58.

$$0 = x_2 \quad \text{Ec. [2.47]}$$

$$0 = \left(\frac{sen(x_{11}) sen(x_7) + cos(x_{11}) sen(x_9) cos(x_7)}{m} \right) U(1)_Q \quad \text{Ec. [2.48]}$$

$$0 = x_4 \quad \text{Ec. [2.49]}$$

$$0 = \left(\frac{-cos(x_{11})sen(x_7) + sen(x_{11})sen(x_9)cos(x_7)}{m}\right)U(1)_Q \quad \text{Ec. [2.50]}$$

$$0 = x_6 \quad \text{Ec. [2.51]}$$

$$0 = -g + \left(\frac{cos(x_9)cos(x_7)}{m}\right)U(1)_Q \quad \text{Ec. [2.52]}$$

$$0 = x_8 \quad \text{Ec. [2.53]}$$

$$0 = \frac{l_{yy} - l_{zz}}{l_{xx}}x_{10}x_{12} - \frac{J_{tp}}{l_{xx}}x_{10}\omega + \frac{U(2)_Q}{l_{xx}} \quad \text{Ec. [2.54]}$$

$$0 = x_{10} \quad \text{Ec. [2.55]}$$

$$0 = \frac{l_{zz} - l_{xx}}{l_{yy}}x_8x_{12} - \frac{J_{tp}}{l_{yy}}x_8\omega + \frac{U(3)_Q}{l_{xx}} \quad \text{Ec. [2.56]}$$

$$0 = x_{12} \quad \text{Ec. [2.57]}$$

$$0 = \frac{l_{xx} - l_{yy}}{l_{zz}}x_8x_{10} + \frac{U(4)_Q}{l_{zz}} \quad \text{Ec. [2.58]}$$

Donde:

$U(1)_Q$: empuje aplicado en el eje vertical en equilibrio, medido en [N]

$U(2)_Q$: momento angular aplicado en dirección roll en equilibrio, medido en [kg · m^2/s]

$U(3)_Q$: momento angular aplicado en dirección pitch en equilibrio, medido en [kg · m^2/s]

$U(4)_Q$: momento angular aplicado en dirección yaw en equilibrio, medido en [kg · m^2/s]

Simplificando se obtiene el sistema de ecuaciones 2.59 – 2.67.

$$0 = x_2 \quad \text{Ec. [2.59]}$$

$$0 = sen(x_{11})sen(x_7) + cos(x_{11})sen(x_9)cos(x_7) \quad \text{Ec. [2.60]}$$

$$0 = x_4 \quad \text{Ec. [2.61]}$$

$$0 = -cos(x_{11})sen(x_7) + sen(x_{11})sen(x_9)cos(x_7) \quad \text{Ec. [2.62]}$$

$$0 = x_6 \quad \text{Ec. [2.63]}$$

$$0 = -g + \left(\frac{cos(x_9)cos(x_7)}{m}\right)U(1)_Q \quad \text{Ec. [2.64]}$$

$$0 = x_8 \quad \text{Ec. [2.65]}$$

$$0 = \frac{U(2)_Q}{l_{xx}} \quad \text{Ec. [2.66]}$$

$$0 = x_{10} \quad \text{Ec. [2.67]}$$

Despejando $sen(x_9)\cos(x_7)$ de la ecuación 2.62 y reemplazando en la ecuación 2.60 se tiene el valor de x_7 en el punto de equilibrio, como muestra la ecuación 2.73.

$$sen(x_9)cos(x_7) = \frac{cos(x_{11})sen(x_7)}{sen(x_{11})} \quad \text{Ec. [2.68]}$$

$$0 = sen(x_{11})sen(x_7) + cos(x_{11})\frac{cos(x_{11})sen(x_7)}{sen(x_{11})} \quad \text{Ec. [2.69]}$$

$$0 = sen^2(x_{11})sen(x_7) + cos^2(x_{11})sen(x_7) \quad \text{Ec. [2.70]}$$

$$0 = sen(x_7)\left(sen^2(x_{11}) + cos^2(x_{11})\right) \quad \text{Ec. [2.71]}$$

$$0 = sen(x_7) \quad \text{Ec. [2.72]}$$

$$x_7 = k\pi \quad \text{Ec. [2.73]}$$

Donde:

$k = \pm 1, \pm 2, \pm 3, \pm 4, \pm 5$ ...

Reemplazando este valor en la ecuación 2.60 se obtiene el valor de x_9 y x_{11}, como indican las ecuaciones 2.75 y 2.76 respectivamente.

$$0 = sen(x_{11})sen(x_9) \quad \text{Ec. [2.74]}$$

Entonces:

$$x_9 = k\pi \quad \text{Ec. [2.75]}$$

$$x_{11} = k\pi \quad \text{Ec. [2.76]}$$

Finalmente el punto de equilibrio para las doce variables de estado del sistema se expresa mediante la ecuación 2.77.

$$xQ = \begin{bmatrix} X \\ 0 \\ Y \\ 0 \\ Z \\ 0 \\ k\pi \\ 0 \\ k\pi \\ 0 \\ k\pi \\ 0 \end{bmatrix} \quad \text{Ec. [2.77]}$$

Donde:

X: valor de x en donde se desea que se estabilice el dron

Y: valor de y en donde se desea que se estabilice el dron

Z: valor de z en donde se desea que se estabilice el dron

2.3.3. Linealización

Para la linealización del sistema se procedió a obtener las matrices A, B, C, D de un sistema de comportamiento equivalente en el punto de equilibrio, por lo que las matrices para el sistema linealizado se consiguieron siguiendo el procedimiento descrito a continuación.

El sistema en el espacio de los estados:

$$\dot{x} = Ax + Bu \quad \text{Ec. [2.78]}$$

$$\dot{y} = Cx + Du \quad \text{Ec. [2.79]}$$

Es equivalente en el punto de equilibrio con:

$$\Delta\dot{x} = \bar{A}\Delta x + \bar{B}u \quad \text{Ec. [2.80]}$$

$$\Delta\dot{y} = \bar{C}\Delta x + \bar{D}u \quad \text{Ec. [2.81]}$$

Donde:

$$\bar{A} = \left.\frac{\partial F}{\partial x}\right|_{\substack{x_Q \\ u_Q}} \quad \text{Ec. [2.82]}$$

$$\bar{B} = \left.\frac{\partial F}{\partial u}\right|_{\substack{x_Q \\ u_Q}} \quad \text{Ec. [2.83]}$$

$$\bar{C} = \frac{\partial G}{\partial x}\Big|_{\substack{x_Q \\ u_Q}} \qquad \text{Ec. [2.84]}$$

$$\bar{D} = \frac{\partial G}{\partial u}\Big|_{\substack{x_Q \\ u_Q}} \qquad \text{Ec. [2.85]}$$

Donde:

F: ecuación de estados

G: ecuación de la salida del sistema

x_Q: valores de equilibrio para los estados (donde la derivada del estado es igual a cero)

u_Q: valores de equilibrio para la entrada

Entonces el valor de la matriz de estados equivalente del sistema linealizado se determinó al obtener las derivadas parciales de cada ecuación de estados respecto a cada uno de los estados como resume la ecuación 2.86.

$$\bar{A} = \frac{\partial F}{\partial x}\Big|_{\substack{x_Q \\ u_Q}} = \begin{bmatrix} \frac{\partial F_1}{\partial x_1} & \frac{\partial F_1}{\partial x_2} & \cdots & \frac{\partial F_1}{\partial x_n} \\ \frac{\partial F_2}{\partial x_1} & \frac{\partial F_2}{\partial x_2} & \cdots & \frac{\partial F_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial F_n}{\partial x_1} & \frac{\partial F_n}{\partial x_2} & \cdots & \frac{\partial F_n}{\partial x_n} \end{bmatrix} \qquad \text{Ec. [2.86]}$$

Realizando este proceso para los doce estados se obtienen las siguientes ecuaciones 2.87 – 2.90.

$$\text{A=}\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \pm g & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -g & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -Jtp * \omega/lyy & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \qquad \text{Ec. [2.87]}$$

Donde:

ω: vector de velocidades angulares de los cuatro motores, medida en [rad/s]

$$B=\begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1/m & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1/l_{xx} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1/l_{yy} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/l_{zz} \end{pmatrix} \qquad \text{Ec. [2.88]}$$

$$C=\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \qquad \text{Ec. [2.89]}$$

$$D=\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \qquad \text{Ec. [2.90]}$$

Finalmente el espacio de estados equivalente linealizado se expresan mediante las ecuaciones 2.91 y 2.92.

$$\Delta\dot{x} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \pm g & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -g & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -Jtp * \omega/lyy & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \Delta x + \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1/m & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1/l_{xx} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1/l_{yy} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/l_{zz} \end{pmatrix} \begin{pmatrix} U(1) \\ U(2) \\ U(3) \\ U(4) \end{pmatrix}$$ Ec. [2.91]

$$\Delta\dot{y} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \Delta x + \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} U(1) \\ U(2) \\ U(3) \\ U(4) \end{pmatrix}$$ Ec. [2.92]

Una vez obtenido el sistema equivalente linealizado se procedió a aplicar un controlador realimentado lineal, el cual tiene por objetivo realizar la transformación de la variable de estado en la entrada disponible mediante una matriz de ganancias k, como expresa la ecuación 2.93.

$$u = -kx$$ Ec. [2.93]

Donde:

u: vector de entradas del sistema

k: matriz de ganancias del controlador

Entonces la matriz de ganancias propuesta presenta doce ganancias $k_1, k_2, k_3, \dots, k_{12}$, que permitieron tener control de las salidas a partir de las entradas deseadas para los doce estados, y fueron ubicadas en la matriz a partir del efecto que tiene cada entrada sobre los respectivos estados. De manera que, por ejemplo si se conoce que la entrada $U(1)$ corresponde al empuje vertical, este tendrá efecto sobre las variables de estado x_5 y x_6 que corresponden a la posición vertical z y su respectiva velocidad $\dot{z}$. La entrada $U(2)$ corresponde a un momento en roll por lo que esta tendrá efecto sobre las variables de estado x_7 y x_8 provocando un desplazamiento angular en roll θ y su respectiva velocidad angular $\dot{\theta}$. La entrada $U(3)$ corresponde a un momento en pitch por lo que esta tendrá efecto sobre las variables de estado x_9 y x_{10} provocando un desplazamiento angular en pitch $\emptyset$ y su respectiva velocidad angular $\dot{\emptyset}$. Finalmente, la entrada $U(4)$ corresponde a un momento en yaw por lo que esta tendrá efecto sobre las variables de estado x_{11} y x_{12} provocando un desplazamiento angular en yaw ψ y su respectiva velocidad angular $\dot{\psi}$.
Entonces la matriz de ganancias propuesta se expresa mediante la ecuación 2.94.

Teniendo en cuenta las ecuaciones 2.23 – 2.34 se puede construir la matriz de ganancias del cuadricóptero mediante la matriz mostrada en la ecuación 2.94[7], donde se han colocado 4 ganancias destinadas a estabilizar las variables z, $\dot{z}$, ψ y $\dot{\psi}$.

$$H(t) = \begin{bmatrix} 0 & I_2 & 0 & 0 \\ 0 & 0 & g_z(t)A(\psi_1) & 0 \\ 0 & 0 & 0 & I \\ -K_1 & -K_2 & -K_3 & -K_4 \end{bmatrix} \qquad \text{Ec. [2.94]}$$

Tomando en cuenta que en el controlador propuesto en el presente proyecto debe controlar también las variables roll, pitch y sus respectivas ganancias, se propone la matriz de ganancias de la ecuación 2.95.

$$k = \begin{pmatrix} 0 & 0 & 0 & 0 & k_5 & k_6 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & k_7 & k_8 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & k_9 & k_{10} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & k_{11} & k_{12} \end{pmatrix} \qquad \text{Ec. [2.95]}$$

Por lo que el controlador queda expresado por la matriz de la ecuación 2.97.

[7] (Santiaguillo-Salinas, Rosaldo-Serrano, & Aranda-Bricaire, 20117)

$$u = -\begin{pmatrix} 0 & 0 & 0 & 0 & k_5 & k_6 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & k_7 & k_8 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & k_9 & k_{10} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & k_{11} & k_{12} \end{pmatrix} \begin{pmatrix} x_{1\delta} \\ x_{2\delta} \\ x_{3\delta} \\ x_{4\delta} \\ x_{5\delta} \\ x_{6\delta} \\ x_{7\delta} \\ x_{8\delta} \\ x_{9\delta} \\ x_{10\delta} \\ x_{11\delta} \\ x_{12\delta} \end{pmatrix} \quad \text{Ec. [2.96]}$$

$$u = \begin{pmatrix} -k_5\, x_{5\delta} - k_6\, x_{6\delta} \\ -k_7\, x_{7\delta} - k_8\, x_{8\delta} \\ -k_9\, x_{9\delta} - k_{10}\, x_{10\delta} \\ -k_{11}\, x_{11\delta} - k_{12}\, x_{12\delta} \end{pmatrix} \quad \text{Ec. [2.97]}$$

Donde:

$x_{1\delta}, x_{2\delta}, x_{3\delta}, \dots, x_{12\delta}$: variaciones infinitesimales de los estados $x_1, x_2, x_3, \dots, x_{12}$:

Como el sistema que se desea controlar es:

$$\dot{x} = Ax + Bu \quad \text{Ec. [2.98]}$$

Entonces mediante la entrada hallada en función de las ganancias se puede reescribir como:

$$Bu = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1/m & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1/l_{xx} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1/l_{yy} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/l_{zz} \end{pmatrix} \begin{pmatrix} -k_5\, x_{5\delta} - k_6\, x_{6\delta} \\ -k_7\, x_{7\delta} - k_8\, x_{8\delta} \\ -k_9\, x_{9\delta} - k_{10}\, x_{10\delta} \\ -k_{11}\, x_{11\delta} - k_{12}\, x_{12\delta} \end{pmatrix} \quad \text{Ec. [2.99]}$$

Ec. [2.100]

$$Bu = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \dfrac{-k_5\, x_{5\delta} - k_6\, x_{6\delta}}{m} \\ 0 \\ \dfrac{-k_7\, x_{7\delta} - k_8\, x_{8\delta}}{l_{xx}} \\ 0 \\ \dfrac{-k_9\, x_{9\delta} - k_{10}\, x_{10\delta}}{l_{yy}} \\ 0 \\ \dfrac{-k_{11}\, x_{11\delta} - k_{12}\, x_{12\delta}}{l_{zz}} \end{pmatrix}$$

El sistema a controlar en función de las matrices halladas es:

$$\dot{x} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \pm g & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -g & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -Jtp * \omega/lyy & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_{1\delta} \\ x_{2\delta} \\ x_{3\delta} \\ x_{4\delta} \\ x_{5\delta} \\ x_{6\delta} \\ x_{7\delta} \\ x_{8\delta} \\ x_{9\delta} \\ x_{10\delta} \\ x_{11\delta} \\ x_{12\delta} \end{pmatrix}$$

$$+ \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \dfrac{-k_5\, x_{5\delta} - k_6\, x_{6\delta}}{m} \\ 0 \\ \dfrac{-k_7\, x_{7\delta} - k_8\, x_{8\delta}}{l_{xx}} \\ 0 \\ \dfrac{-k_9\, x_{9\delta} - k_{10}\, x_{10\delta}}{l_{yy}} \\ 0 \\ \dfrac{-k_{11}\, x_{11\delta} - k_{12}\, x_{12\delta}}{l_{zz}} \end{pmatrix}$$

Ec. [2.101]

Donde realizando la suma de matrices se obtiene:

$$\dot{x} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \pm g & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -g & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -k_5/m & -k_6/m & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -k_7/l_{xx} & -k_8/l_{xx} & 0 & -J_{tp}/l_{xx} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & w & -k_9/l_{yy} & -k_{10}/l_{yy} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -k_{11}/lzz & -k_{12}/lzz \end{pmatrix} \begin{pmatrix} x_{1\delta} \\ x_{2\delta} \\ x_{3\delta} \\ x_{4\delta} \\ x_{5\delta} \\ x_{6\delta} \\ x_{7\delta} \\ x_{8\delta} \\ x_{9\delta} \\ x_{10\delta} \\ x_{11\delta} \\ x_{12\delta} \end{pmatrix}$$

Ec. [2.102]

Además, si se limita el rango de giro sobre los ejes entre $-\pi < \emptyset < \pi$, entonces $\pm g = g$, y aplicando el punto de equilibrio, la matriz A del sistema final se obtiene:

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & g & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -g & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -k_5/m & -k_6/m & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -k_7/l_{xx} & -k_8/l_{xx} & 0 & -J_{tp}/l_{xx} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & w & -k_9/l_{yy} & -k_{10}/l_{yy} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -k_{11}/lzz & -k_{12}/lzz \end{pmatrix}$$

Ec. [2.103]

Donde A es un elemento de gran importancia para el análisis, debido que cuando se hallan los valores propios de la matriz se puede encontrar el polinomio característico, que al igualarse a los polos ubicados en el lugar geométrico apropiado permitirán una acción de control. Los valores propios y el polinomio característico se obtienen como se muestra en la ecuación 2.104.

$$(s + a)^{12} = |SI - A| \qquad \text{Ec. [2.104]}$$

Donde:

s: variable para el polinomio característico en el dominio de Laplace

a: ubicación del polo en el eje x para el semiplano negativo, donde la respuesta de control es estable

La expresión $(s + a)^{12}$ representa la ubicación de los polos del nuevo sistema estable y controlado, que estarán ubicados en la posición $s = -a$, lo que es apropiado ya que de esta manera el sistema es estable, dado que los polos se ubican todos en el semiplano

negativo para x, donde a medida que a tenga mayor valor, la estabilidad se alcanzará en menor tiempo, y $|SI - A|$ es el método por el que se obtiene el polinomio característico del sistema.

Al hacer que los polos del sistema controlado sean iguales que los polos del polinomio característico, el sistema queda controlado mediante las ganancias $k_1, k_2, k_3, \dots, k_{12}$.

Desarrollando el lado izquierdo de la igualdad se tiene el polinomio de la ecuación 2.105.

$$(s+a)^{12} = s^{12} + 12as^{11} + 66a^2s^{10} + 220a^3s^9 + 220a^3s^9 + 495a^4s^8 + 792a^5s^7 + 924a^6s^6 + 792a^7s^5 + 495a^8s^4 + 220a^9s^3 + 66a^{10}s^2 + 12a^{11}s + a^{12} \quad \text{Ec. [2.105]}$$

Dada la complejidad del sistema y las ecuaciones implicadas se desarrolló un código en el software Matlab que permitió obtener el polinomio característico. Luego a partir de los coeficientes de cada grado de la ecuación se generaron 8 ecuaciones que permitieron calcular las 8 ganancias. El programa desarrollado para llevar a cabo esta tarea tiene por código:

```
clear; clc;

syms Ixx Iyy Izz m g k1 k2 k3 k4 k5 k6 k7 k8 k9 k10 k11 k12 Jtp w s
a %Se definen las variables a ser empleadas

%Parámetros para cálculo real de ganancias

Ixx = 8.1*10^(-3);  % Momento de inercia del cuadrotor alrededor del
eje X

Iyy = 8.1*10^(-3);  % Momento de inercia del cuadrotor alrededor del
eje Y

Izz = 14.2*10^(-3);  % Momento de inercia del cuadrotor alrededor
del eje Z

Jtp = 104*10^(-6);  % Momento de inercia total en torno al eje de
propulsión

m = 1;  %Masa del cuadrotor en kg

g = 9.81;   %Aceleración gravitacional

a=10; %Ubicación de los 10 polos en el semiplano negativo

%Se ingresa la matriz de estados linealizada

A=[0 1 0 0 0 0 0 0 0 0 0 0;...
   0 0 0 0 0 0 0 0 g 0 0 0;...
   0 0 0 1 0 0 0 0 0 0 0 0;...
   0 0 0 0 0 0 -g 0 0 0 0 0;...
   0 0 0 0 0 1 0 0 0 0 0 0;...
   0 0 0 0 -k5/m -k6/m 0 0 0 0 0 0;...
```

```
   0 0 0 0 0 0 0 1 0 0 0 0;
   0 0 0 0 0 0 -k7/Ixx -k8/Ixx 0 -Jtp/Ixx 0 0;...
   0 0 0 0 0 0 0 0 0 1 0 0;...
   0 0 0 0 0 0 0 w -k9/Iyy -k10/Iyy 0 0;...
   0 0 0 0 0 0 0 0 0 0 0 1;...
   0 0 0 0 0 0 0 0 0 0 0 -k11/Izz -k12/Izz];

%Se genera una matriz identidad de dimensión 12x12 multiplicada por
S

SI=[s 0 0 0 0 0 0 0 0 0 0 0;...
    0 s 0 0 0 0 0 0 0 0 0 0;...
    0 0 s 0 0 0 0 0 0 0 0 0;...
    0 0 0 s 0 0 0 0 0 0 0 0;...
    0 0 0 0 s 0 0 0 0 0 0 0;...
    0 0 0 0 0 s 0 0 0 0 0 0;...
    0 0 0 0 0 0 s 0 0 0 0 0;...
    0 0 0 0 0 0 0 s 0 0 0 0;...
    0 0 0 0 0 0 0 0 s 0 0 0;...
    0 0 0 0 0 0 0 0 0 s 0 0;...
    0 0 0 0 0 0 0 0 0 0 s 0;...
    0 0 0 0 0 0 0 0 0 0 0 s];

SIA=SI-A %Se obtiene realizando las operaciones matriciales

polcaract=det(SIA)% El determinante de SI-A corresponde al polinomio
característico

ord=collect(polcaract,s)%Se organiza el polinomio característico de
menor a mayor grado

x=coeffs(ord,s)%Se obtiene los coeficientes del polinomio
característico en un arreglo tipo matriz

polctrl=(s+a)^12 %Se genera un polinomio equivalente de grado 12 que
permitirá reubicar los polos

y=coeffs(polctrl,s)%Se extraen los coeficientes del polinomio de
control

%El sistema compuesto de 12 ecuaciones se ensambla al igualar cada
coeficiente del polinomio característico con el coeficiente del
polinomio de control

eq1= x(1)==y(5)
eq2= x(2)==y(6)
eq3= x(3)==y(7)
eq4= x(4)==y(8)
eq5= x(5)==y(9)
eq6= x(6)==y(10)
eq7= x(7)==y(11)
eq8= x(8)==y(12)
eq9= x(9)==y(13)

%Mediante el comando solve, se solicita resolver el sistema de
ecuaciones resultantes para hallar los valores de las ganancias

[k5,k6,k7,k8,k9,k10,k11,k12]=solve(eq1,eq2,...
    eq3,eq4,eq5,eq6,eq7,eq8,eq9,'k5,k6,k7,k8,k9,k10,k11,k12')
```

De donde al ejecutarlo se obtuvieron las 8 ecuaciones del controlador, a pesar de que las ecuaciones son de grado 12 los elementos de grado s^0, s^1, s^2, s^3, y s^4 no tienen coeficientes, obteniéndose las ecuaciones 2.106 – 2.113.

$$eq1 = (k5 * k7 * k9 * k11)/(I_{xx} * I_{yy} * I_{zz} * m) = 495 * a\hat{}8 \quad \text{Ec. [2.106]}$$

$$eq2 = (k5 * k7 * k9 * k12 + k5 * k7 * k10 * k11 + k5 * k8 * k9 * k11 + k6 * k7 * k9 * k11)/(I_{xx} * I_{yy} * I_{zz} * m) == 792 * a\hat{}7 \quad \text{Ec. [2.107]}$$

$$eq3 = (k5 * k7 * k10 * k12 + k5 * k8 * k9 * k12 + k5 * k8 * k10 * k11 + k6 * k7 * k9 * k12 + k6 * k7 * k10 * k11 + k6 * k8 * k9 * k11 + k7 * k9 * k11 * m + I_{xx} * k5 * k9 * k11 + I_{yy} * k5 * k7 * k11 + I_{zz} * k5 * k7 * k9 + I_{yy} * J_{tp} * k5 * k11 * w)/(I_{xx} * I_{yy} * I_{zz} * m) == 924 * a\hat{}6 \quad \text{Ec. [2.108]}$$

$$eq4 = (k5 * k8 * k10 * k12 + k6 * k7 * k10 * k12 + k6 * k8 * k9 * k12 + k7 * k9 * k12 * m + k8 * k9 * k11 * m + I_{xx} * k5 * k9 * k12 + I_{xx} * k5 * k10 * k11 + I_{xx} * k6 * k9 * k11 + I_{yy} * k5 * k7 * k12 + I_{yy} * k5 * k8 * k11 + I_{yy} * k6 * k7 * k11 + I_{zz} * k5 * k7 * k10 + I_{zz} * k5 * k8 * k9 + I_{zz} * k6 * k7 * k9 + I_{yy} * J_{tp} * k5 * k12 * w + I_{yy} * J_{tp} * k6 * k11 * w)/(I_{xx} * I_{yy} * I_{zz} * m) == 792 * a\hat{}5 \quad \text{Ec. [2.109]}$$

$$eq5 = (k6 * k8 * k10 * k12 + k7 * k10 * k12 * m + k8 * k9 * k12 * m + k8 * k10 * k11 * m + I_{xx} * I_{yy} * k5 * k11 + I_{xx} * I_{zz} * k5 * k9 + I_{yy} * I_{zz} * k5 * k7 + I_{xx} * k5 * k10 * k12 + I_{xx} * k6 * k9 * k12 + I_{xx} * k6 * k10 * k11 + I_{yy} * k5 * k8 * k12 + I_{yy} * k6 * k7 * k12 + I_{yy} * k6 * k8 * k11 + I_{zz} * k5 * k8 * k10 + I_{zz} * k6 * k7 * k10 + I_{zz} * k6 * k8 * k9 + I_{xx} * k9 * k11 * m + I_{yy} * k7 * k11 * m + I_{zz} * k7 * k9 * m + I_{yy} * I_{zz} * J_{tp} * k5 * w + I_{yy} * J_{tp} * k6 * k12 * w + I_{yy} * J_{tp} * k11 * m * w)/(I_{xx} * I_{yy} * I_{zz} * m) == 495 * a\hat{}4 \quad \text{Ec. [2.110]}$$

$$eq6 = (k8 * k10 * k12 * m + I_{xx} * I_{yy} * k5 * k12 + I_{xx} * I_{yy} * k6 * k11 + I_{xx} * I_{zz} * k5 * k10 + I_{xx} * I_{zz} * k6 * k9 + I_{yy} * I_{zz} * k5 * k8 + I_{yy} * I_{zz} * k6 * k7 + I_{xx} * k6 * k10 * k12 + I_{yy} * k6 * k8 * k12 + I_{zz} * k6 * k8 * k10 + I_{xx} * k9 * k12 * m + I_{xx} * k10 * k11 * m + I_{yy} * k7 * k12 * m + I_{yy} * k8 * k11 * m + I_{zz} * k7 * k10 * m + I_{zz} * k8 * k9 * m + I_{yy} * I_{zz} * J_{tp} * k6 * w + I_{yy} * J_{tp} * k12 * m * w)/(I_{xx} * I_{yy} * I_{zz} * m) == 220 * a\hat{}3 \quad \text{Ec. [2.111]}$$

$$eq7 = (I_{xx} * I_{yy} * I_{zz} * k5 + I_{xx} * I_{yy} * k6 * k12 + I_{xx} * I_{zz} * k6 * k10 + I_{xx} * I_{yy} * k11 * m + I_{xx} * I_{zz} * k9 * m + I_{yy} * I_{zz} * k7 * m + I_{xx} * k10 * k12 * m + I_{yy} * k8 * k12 * m + I_{zz} * k8 * k10 * m + I_{yy} * I_{zz} * J_{tp} * m * w)/(I_{xx} * I_{yy} * I_{zz} * m) == 66 * a\hat{}2 \quad \text{Ec. [2.112]}$$

$$eq8 = (I_{xx} * I_{yy} * I_{zz} * k6 + I_{xx} * I_{yy} * k12 * m + I_{xx} * I_{zz} * k10 * m + I_{yy} * I_{zz} * k8 * m)/(I_{xx} * I_{yy} * I_{zz} * m) == 12 * a \quad \text{Ec. [2.113]}$$

Finalmente, se solicitó a Matlab que resuelva el sistema, obteniéndose el valor exacto de las ganancias del controlador, así:

```
K5 = 40.00289;
K6 = 12.051;
K7 = 30.30082;
K8 = 5.00023;
K9 = 30.30082;
K10 = 5.00023;
K11 = 30.30082;
K12 = 5.00023;
```

De esta manera las leyes de control quedan determinadas mediante las ecuaciones 2.114 – 2.117.

$$U(1) = \frac{m(g - k_5(x_5 - Z_d) - k_6 x_6}{\cos(x_9)\cos(x_7)} \quad \text{Ec. [2.114]}$$

$$U(2) = -k_7(x_7 - \theta_d) - k_8 x_8 \quad \text{Ec. [2.115]}$$

$$U(3) = -k_9(x_9 - \emptyset_d) - k_{10} x_{10} \quad \text{Ec. [2.116]}$$

$$U(4) = -k_{11}(x_{11} - \psi_d) - k_{12} x_{12} \quad \text{Ec. [2.117]}$$

2.3.4. Diseño del controlador

Una vez obtenido el modelo dinámico del sistema no lineal, y habiéndolo linealizado con el objetivo de que el sistema equivalente permita obtener el valor de las ganancias de control, mediante las que se generan las leyes de control que dotarán al sistema de la capacidad de estabilizarse en los valores deseados ubicados en las leyes de control como Z_d, θ_d, $\emptyset_d$ y ψ_d; se procedió a generar una función en Matlab que permita verificar el correcto funcionamiento del controlador, y su acción de control sobre las variables.

Esta función se diseñó con el objetivo de ser llamada mediante el comando ode45, mismo que es capaz de resolver sistemas de ecuaciones diferenciales, que en este caso empleó las 12 ecuaciones de estados como sistema, reemplazando las ganancias en las cuatro leyes de control. De manera que, este controlador tiene acción empleando las ganancias y recibiendo una realimentación de los estados x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11} y x_{12}, que corresponden a la altura y velocidad de acenso, roll, pitch, yaw, y sus respectivas velocidades angulares. De esta forma, el controlador obtiene la información de estos estados y los compara con los valores deseados para estabilizar al cuadrotor en dichos valores mediante las entradas que corresponden al empuje vertical y los momentos aplicados en roll, pitch y yaw.

Mediante estos valores posteriormente la función obtiene los valores de las velocidades angulares de cada uno de los cuatro motores como se muestra en las ecuaciones 2.118 – 2.121, para de esta manera ejecutar los actuadores que en este caso son los motores. Las ecuaciones que permiten obtener la velocidad angular requerida en cada uno de los motores ante las solicitaciones de control se obtuvieron en función de la distribución geométrica y sincronización de la dirección de giro de cada motor, como se observa en la figura 2.4., además, de los factores de empuje b y arrastre d proporcionados en las especificaciones técnicas del dron, que son valores característicos del diseño del cuadrotor obtenidos experimentalmente durante el vuelo que reflejan la resistencia del fluido aire al desplazamiento vertical (b) y longitudinal (d).

$$\omega_1{}^2 = \frac{b}{4}U(1) + \frac{bl}{2}U(3) - \frac{d}{4}U(4) \qquad \text{Ec. [2.118]}$$

$$\omega_2{}^2 = \frac{b}{4}U(1) - \frac{bl}{2}U(2) + \frac{d}{4}U(4) \qquad \text{Ec. [2.119]}$$

$$\omega_3{}^2 = \frac{b}{4}U(1) - \frac{bl}{2}U(3) - \frac{d}{4}U(4) \qquad \text{Ec. [2.120]}$$

$$\omega_4{}^2 = \frac{b}{4}U(1) + \frac{bl}{2}U(2) + \frac{d}{4}U(4) \qquad \text{Ec. [2.121]}$$

Donde:

b: factor de empuje

d: factor de arrastre

$\omega_i{}^2$: cuadrado de la velocidad angular de cada motor

l: distancia al centro de masa del cuadrotor

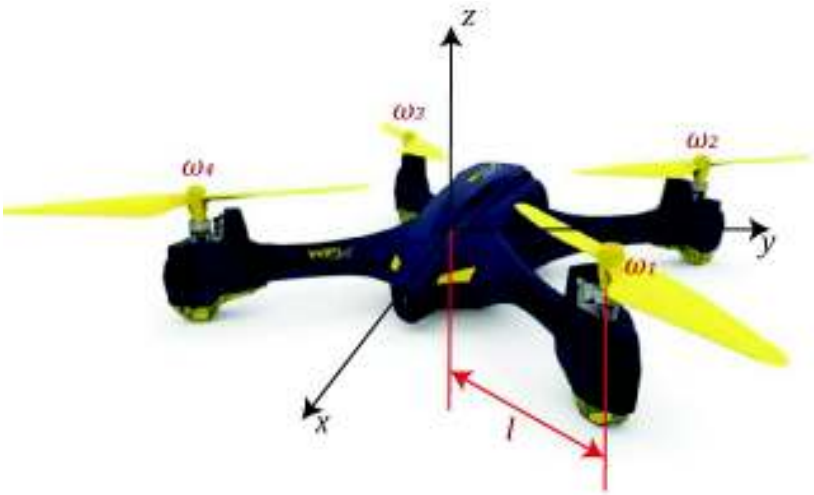


Figura 2.4. Distribución de velocidades angulares en los motores del cuadrotor.

(Fuente: elaboración propia)

Entonces, mediante los elementos antes descritos se generó la función de control "controlquadfn", cuyo código se detalla a continuación:

```
function [xdot] = controlquadfn(t,x)

%Declaración de las variables

global Jtp Ixx Iyy Izz b d l m g ZdF PhidF ThetadF PsidF ztime phitime
thetatime psitime Zinit Phiinit Thetainit Psiinit


% Factores
```

```
Ixx = 8.1*10^(-3);  %Momento de inercia del cuadrotor alrededor del
eje X

Iyy = 8.1*10^(-3);  %Momento de inercia del cuadrotor alrededor del
eje Y

Izz = 14.2*10^(-3);  %Momento de inercia del cuadrotor alrededor del
eje Z

Jtp = 104*10^(-6);  %Momento de inercia total en torno al eje de
propulsión

b = 54.2*10^(5-6);  %Factor de empuje
d = 1.1*10^(-6);  %Factor de arrastre
l = 0.24;  %Distancia al centro de masa del cuadrotor
m = 1;  %Masa del cuadrotor en kg
g = 9.81;   %Aceleración gravitacional

%Se ingresan las ganancias K calculadas

K5 = 40;
K6 = 12;
K7 = 30;
K8 = 5;
K9 = 30;
K10 = 5;
K11 = 30;
K12 = 5;

%Valores deseados para las variables de control

Zd=1;

Rolldeseado=45; %Ingresar ángulo roll deseado en grados

Pitchdeseado=45; %Ingresar ángulo pitch deseado en grados

Yawdeseado=45; %Ingresar ángulo yaw deseado en grados

Rolld=Rolldeseado;

Pitchd=Pitchdeseado;

Yawd=Yawdeseado;

%Se crean las leyes de control

U = [];    %Vector de señales de control

    U(1) = m*(g -K5*(x(5)-Zd) - K6*(x(6)))/(cos(x(9))*cos(x(7)));
%Ley de control para el empuje en z

    U(2) = (-K7*(x(7)-Rolld) - K8*(x(8)));   %Ley de control para el
momento en roll

    U(3) = (-K9*(x(9)-Pitchd) - K10*(x(10)));   %Ley de control para
el momento en pitch

    U(4) = (-K11*(x(11)-Yawd) - K12*(x(12)));   %Ley de control para
el momento en y
```

```
    U = real(U);
    U = [U(1);U(2);U(3);U(4)];    %El vector de control
    %Cálculo de velocidades angulares
    omegasqr(1) = (1/4*b)*U(1) + (1/2*b*l)*U(3) - (1/4*d)*U(4);
    omegasqr(2) = (1/4*b)*U(1) - (1/2*b*l)*U(2) + (1/4*d)*U(4);
    omegasqr(3) = (1/4*b)*U(1) - (1/2*b*l)*U(3) - (1/4*d)*U(4);
    omegasqr(4) = (1/4*b)*U(1) + (1/2*b*l)*U(2) + (1/4*d)*U(4);
    omegasqr = real(omegasqr);
    omega(1) = sqrt(omegasqr(1));
    omega(2) = sqrt(omegasqr(2));
    omega(3) = sqrt(omegasqr(3));
    omega(4) = sqrt(omegasqr(4))
        %Limitando las velocidades angulares
    for j = 1:4
        if omega(j) > 523
            omega(j) = 523;
        end
        if omega(j) < 125
            omega(j) = 125;
        end
    end
    omegasqr(1) = (omegasqr(1))^2;
    omegasqr(2) = (omegasqr(2))^2;
    omegasqr(3) = (omegasqr(3))^2;
    omegasqr(4) = (omegasqr(4))^2;
    Omega   =   d*(-   sqrt(omegasqr(1))   +   sqrt(omegasqr(2))   -
sqrt(omegasqr(3)) + sqrt(omegasqr(4)));
%Evaluando el espacio de los estados
    xdot = [x(2);... %x punto
        (sin(x(11))*sin(x(7))                                    +
cos(x(11))*sin(x(9))*cos(x(7)))*(U(1)/m);... %x dos puntos
```

```
        x(4); ... %y punto

        (-cos(x(11))*sin(x(7))                                          +
sin(x(11))*sin(x(9))*cos(x(7)))*(U(1)/m);... %y dos puntos

        x(6);... %z punto

        - g + (cos(x(9))*cos(x(7)))*(U(1)/m);... %z dos puntos

        x(8);... %pitch punto

        ((Iyy  -  Izz)/Ixx)*x(10)*x(12)  -  (Jtp/Ixx)*x(10)*Omega  +
(U(2)/Ixx);... %pitch dos puntos

        x(10);... % roll punto

        ((Izz  -  Ixx)/Iyy)*x(8)*x(12)  +  (Jtp/Iyy)*x(8)*Omega  +
(U(3)/Iyy);... %roll dos puntos

        x(12);... %yaw punto

        ((Ixx  -  Iyy)/Izz)*x(8)*x(10)  +  (U(4)/Izz)];... % yaw dos
puntos
```

Finalmente, esta función permite ingresar las condiciones de vuelo Z_d, θ_d, $\emptyset_d$ y ψ_d, y controlarlas entorno al espacio de estados creado. Para ejecutarla y realizar pruebas de su correcto funcionamiento se elaboró un script en Matlab que llama a la función y ejecuta una simulación del control de las variables y la trayectoria descrita por el dron, cuyo código para el caso del controlador aplicado al sistema no lineal se detalla a continuación:

```
clear; clc;

%Se define un tiempo de simulación y valores iniciales para las
variables de estado

t0=0; tfinal=3; tspan=[t0 tfinal]; y0=[0 0 0 0 0 0 0 0 0 0 0 0]';
%Se resuelve el sistema de ecuaciones diferenciales contenido en
'controlquadfn' mediante el comando ode45

[t,y]=ode45('controlquadfn',tspan,y0);
%Se realiza un ploteo del comportamiento de las variables de control:
altura roll pitch y yaw

subplot(3,2,1);plot(t,y(:,1));

xlabel('Time(seconds)','FontSize',10);

title('Posición x','FontSize',10);

subplot(3,2,2);plot(t,y(:,3));

xlabel('Time(seconds)','FontSize',10);

title('Posición y','FontSize',10);
```

```
subplot(3,2,3);plot(t,y(:,5));

xlabel('Time(seconds)','FontSize',10);

title('Posición z','FontSize',10);

subplot(3,2,4);plot(t,y(:,7)*(45/0.01));

xlabel('Time(seconds)','FontSize',10);

title('Ángulo roll','FontSize',10);

subplot(3,2,5);plot(t,y(:,9)*(45/0.01));

xlabel('Time(seconds)','FontSize',10);

title('Ángulo pitch','FontSize',10);

subplot(3,2,6);plot(t,y(:,11)*(45/0.01));

xlabel('Time(seconds)','FontSize',10);

title('Ángulo yaw','FontSize',10);

%Se realiza una simulación de la trayectoria de vuelo descrita por
el cuadrotor

figure(2)

plot3(y(:,1),y(:,3),y(:,5))

title('Simulación de vuelo','FontSize',10);
```

Adicionalmente, un elemento relevante que permite verificar el correcto comportamiento del controlador es el hecho que este fue desarrollado bajo la suposición que cerca del punto de equilibrio el sistema linealizado se comporta de manera idéntica al sistema no lineal real que representa el cuadrotor. Por lo que, para poder validar este efecto se desarrolló una función en Matlab “controllinealquadfn” que permite simular el vuelo del cuadrotor empleando el sistema linealizado cuyo código se presenta a continuación:

```
function [xdot] = controllinealquadfn(t,x)


%Declaración de las variables

global Jtp Ixx Iyy Izz b d l m g


% Factores

Ixx = 8.1*10^(-3);  %Momento de inercia del cuadrotor alrededor del
eje X
```

```
Iyy = 8.1*10^(-3);  %Momento de inercia del cuadrotor alrededor del
eje Y

Izz = 14.2*10^(-3);  %Momento de inercia del cuadrotor alrededor del
eje Z

Jtp = 104*10^(-6);  %Momento de inercia total en torno al eje de
propulsión

b = 54.2*10^(5-6);  %Factor de empuje
d = 1.1*10^(-6);  %Factor de arrastre
l = 0.24;  %Distancia al centro de masa del cuadrotor
m = 1;  %Masa del cuadrotor en kg
g = 9.81;   %Aceleración gravitacional

%Se ingresan las ganancias K calculadas

K5 = 40.00289;
K6 = 12.051;
K7 = 30.30082;
K8 = 5.00023;
K9 = 30.30082;
K10 = 5.00023;
K11 = 30.30082;
K12 = 5.00023;

%Valores deseados para las variables de control

Zd=1;

Rolldeseado=45; %Ingresar ángulo roll deseado en grados

Pitchdeseado=45; %Ingresar ángulo pithch deseado en grados

Yawdeseado=45; %Ingresar ángulo yaw deseado en grados

Rolld=Rolldeseado*(0.01/45);

Pitchd=Pitchdeseado*(0.01/45);

Yawd=Yawdeseado*(0.01/45);

%Se crean las leyes de control

U = [];    %Vector de señales de control

    U(1)  = m*(g  -K5*(x(5)-Zd)  -  K6*(x(6)))/(cos(x(9))*cos(x(7)));
%Ley de control para el empuje en z

    U(2) = (-K7*(x(7)-Rolld) - K8*(x(8)));   %Ley de control para el
momento en roll

    U(3) = (-K9*(x(9)-Pitchd) - K10*(x(10)));   %Ley de control para
el momento en pitch

    U(4) = (-K11*(x(11)-Yawd) - K12*(x(12)));   %Ley de control para
el momento en yaw

    U = real(U);
```

```
    U = [U(1);U(2);U(3);U(4)];    %Vector control

    %Cálculo de velocidades angulares

    omegasqr(1) = (1/4*b)*U(1) + (1/2*b*l)*U(3) - (1/4*d)*U(4);

    omegasqr(2) = (1/4*b)*U(1) - (1/2*b*l)*U(2) + (1/4*d)*U(4);

    omegasqr(3) = (1/4*b)*U(1) - (1/2*b*l)*U(3) - (1/4*d)*U(4);

    omegasqr(4) = (1/4*b)*U(1) + (1/2*b*l)*U(2) + (1/4*d)*U(4);

    omegasqr = real(omegasqr);

    omega(1) = sqrt(omegasqr(1));

    omega(2) = sqrt(omegasqr(2));

    omega(3) = sqrt(omegasqr(3));

    omega(4) = sqrt(omegasqr(4))

        %Limitando las velocidades angulares

    for j = 1:4

        if omega(j) > 523

            omega(j) = 523;

        end

        if omega(j) < 125

            omega(j) = 125;

        end

    end

    omegasqr(1) = (omegasqr(1))^2;

    omegasqr(2) = (omegasqr(2))^2;

    omegasqr(3) = (omegasqr(3))^2;

    omegasqr(4) = (omegasqr(4))^2;


    Omega   =   d*(-   sqrt(omegasqr(1))   +   sqrt(omegasqr(2))   -
sqrt(omegasqr(3)) + sqrt(omegasqr(4)));

% Evaluando el espacio de los estados
%Matrices

A=[0 1 0 0 0 0 0 0 0 0 0 0;0 0 0 0 0 0 0 0 g 0 0 0;0 0 0 1 0 0 0 0 0
0 0 0;...
    0 0 0 0 0 0 -g 0 0 0 0 0;...
    0 0 0 0 0 1 0 0 0 0 0 0;0 0 0 0 0 0 0 0 ...
    0 0 0 0;0 0 0 0 0 0 0 1 0 0 0 0;0 0 ...
```

```
    0 0 0 0 0 0 0 -Jtp/Ixx ...
    0 0;0 0 0 0 0 0 0 0 0 1 0 0;0 0 0 ...
    0 0 0 0 Omega 0 0 0 ...
    0;0 0 0 0 0 0 0 0 0 0 0 1;0 0 0 0 0 0 0 0 0 0 0 0];

B=[0 0 0 0;0 0 0 0;0 0 0 0;0 0 0 0;0 0 0 0;1/m 0 0 0;...
    0 0 0 0;0 1/Ixx 0 0;0 0 0 0;0 0 1/Iyy 0;0 0 0 0;...
    0 0 0 1/Izz];

estados=[x(1);x(2);x(3);x(4);x(5);x(6);x(7);x(8);x(9);x(10);x(11);x
(12)];

    xdot = A*estados+B*U;
```

Como se puede notar en comparación con la función desarrollada para el controlador no lineal, en esta el espacio de estados “xdot = A*estados + B*U” se generó empleando las matrices linealizadas A y B, por lo que este sería el sistema equivalente linealizado.

2.3.5. Simulación del controlador

Al ejecutar el programa de simulación en el caso del sistema no lineal, los parámetros de entrada deseados para este ejemplo se indican en las ecuaciones 2.122 – 2.125.

$$Z_d = 1\,m \qquad \text{Ec. [2.122]}$$

$$\theta_d = 45^0 \qquad \text{Ec. [2.123]}$$

$$\emptyset_d = 45^0 \qquad \text{Ec. [2.124]}$$

$$\psi_d = 45^0 \qquad \text{Ec. [2.125]}$$

Por lo que, se espera que el controlador sea capaz de estabilizar el vuelo del dron a un metro de altura (posición 1). Como se emplea un desplazamiento angular en yaw de 45º, el resultado esperado es que el dron gire en torno al eje vertical 45º (posición 2). Además, como se emplea un desplazamiento angular pitch de 45º, el resultado esperado es que el dron tome una orientación de 45º en posición de avance, por lo que en los ejes trasladados se espera que el dron sea capaz de continuar a la altura descrita desplazándose en dirección del eje x. Finalmente, como se emplea un ángulo roll de 45º, el resultado esperado es que el dron se desplace en el sentido del eje y al mismo tiempo y con la misma magnitud con la que se desplaza en el eje x (posición 3). Por lo que se espera un desplazamiento total resultante en dirección diagonal a 45º en el plano xy mientras conserva la altura de 1 metro de manera estable, como se puede observar en la figura 2.5.

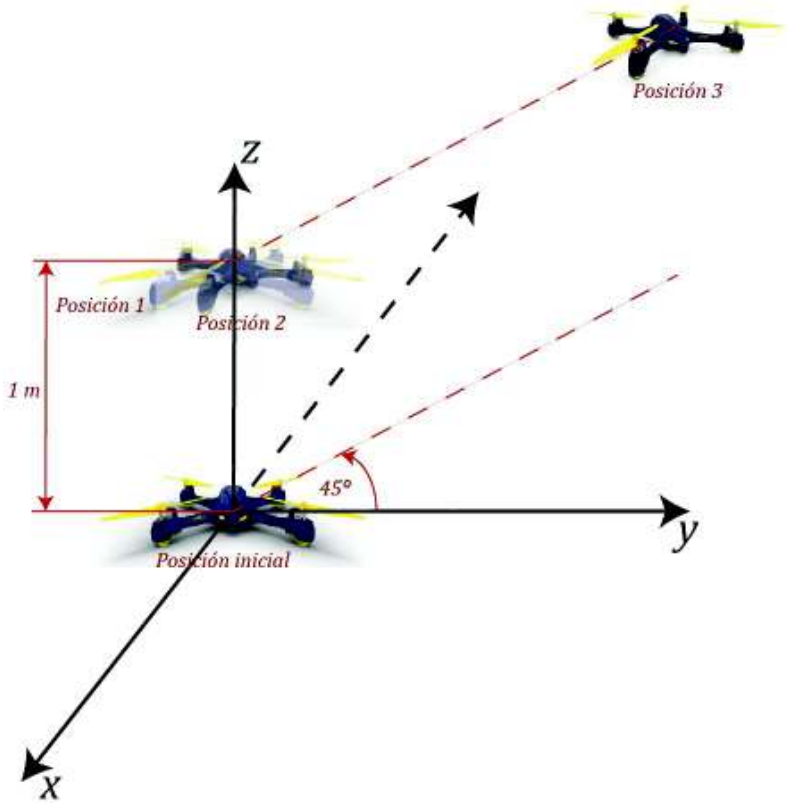


Figura 2.5. Resultados esperados de la simulación de vuelo del controlador ante los valores de entrada deseados.

(Fuente: elaboración propia)

Entonces, al ejecutar el script “simulacionnolineal” los resultados obtenidos mediante la simulación en las gráficas de la interfaz de Matlab son las que se muestran en las figuras 2.6. – 2.12.

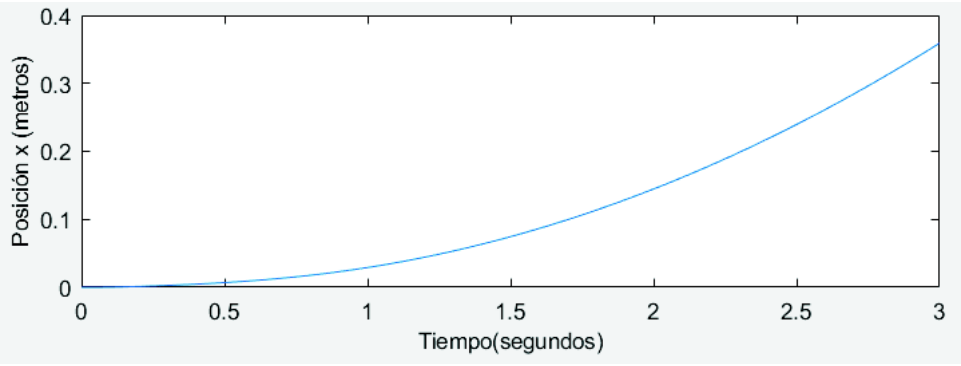


Figura 2.6. Desplazamiento en x del cuadrotor con el controlador no lineal aplicado al sistema no lineal.

(Fuente: elaboración propia)

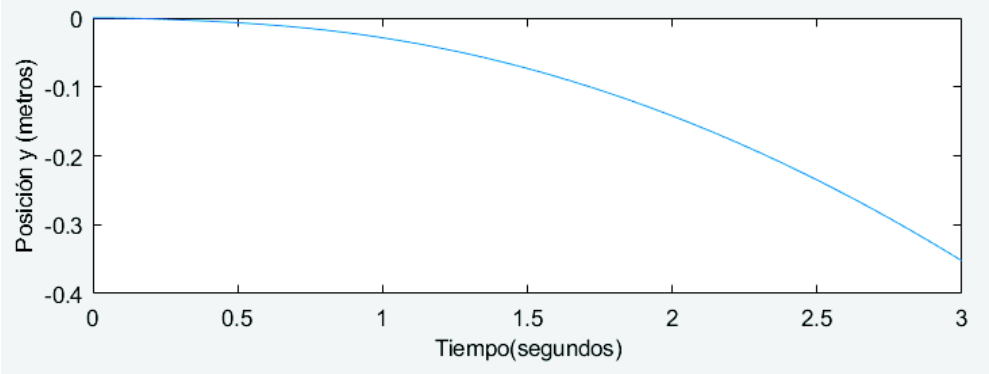


Figura 2.7. Desplazamiento en y del cuadrotor con el controlador no lineal aplicado al sistema no lineal.

(Fuente: elaboración propia)

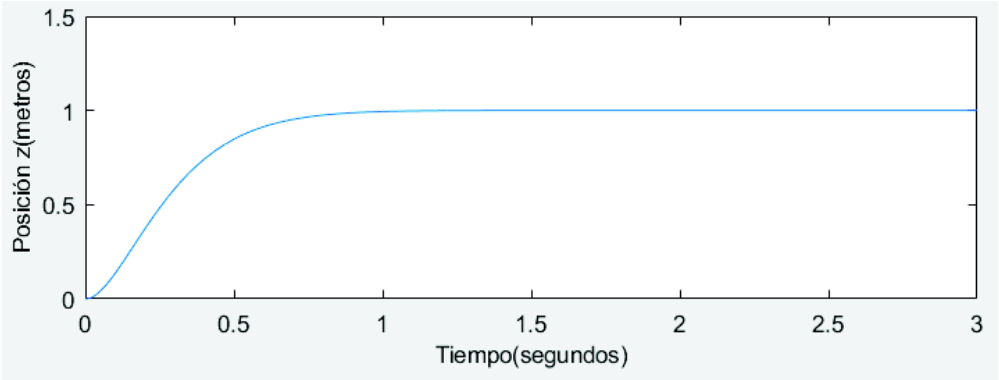


Figura 2.8. Desplazamiento en z del cuadrotor con el controlador no lineal aplicado al sistema no lineal.

(Fuente: elaboración propia)

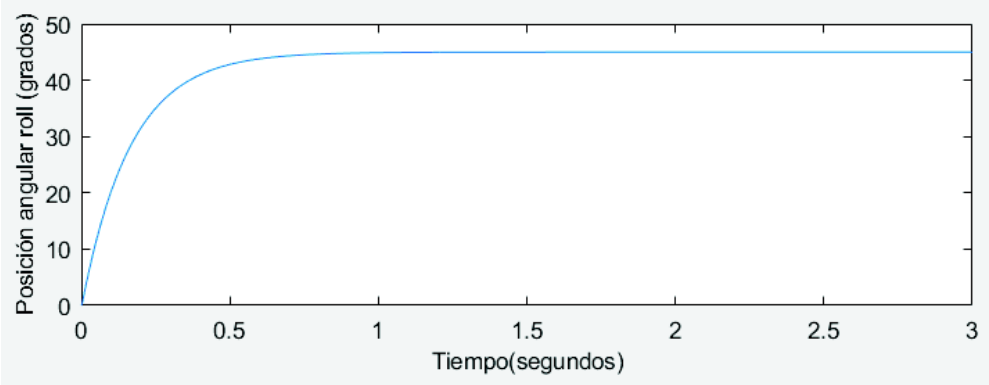


Figura 2.9. Desplazamiento angular roll del cuadrotor con el controlador no lineal aplicado al sistema no lineal.

(Fuente: elaboración propia)

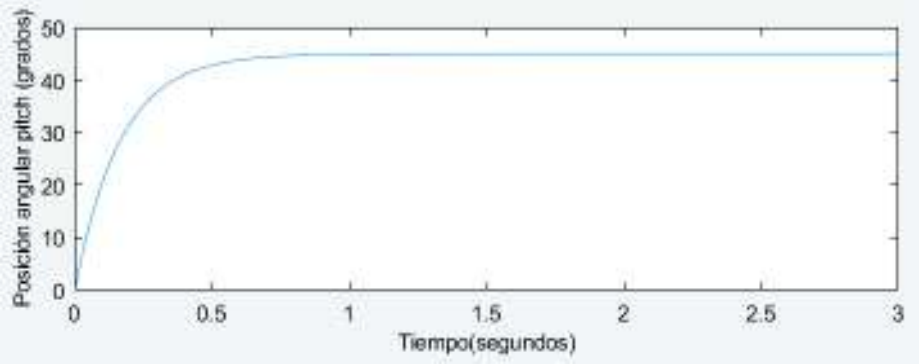


Figura 2.10. Desplazamiento angular pitch del cuadrotor con el controlador no lineal aplicado al sistema no lineal.

(Fuente: elaboración propia)

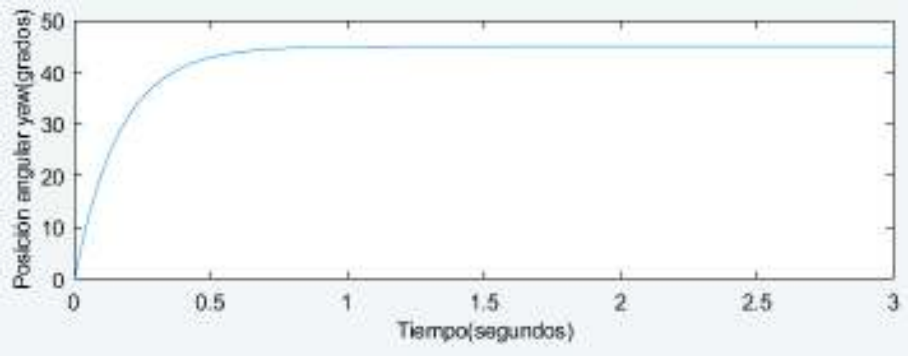


Figura 2.11. Desplazamiento angular pitch del cuadrotor con el controlador no lineal aplicado al sistema no lineal.

(Fuente: elaboración propia)

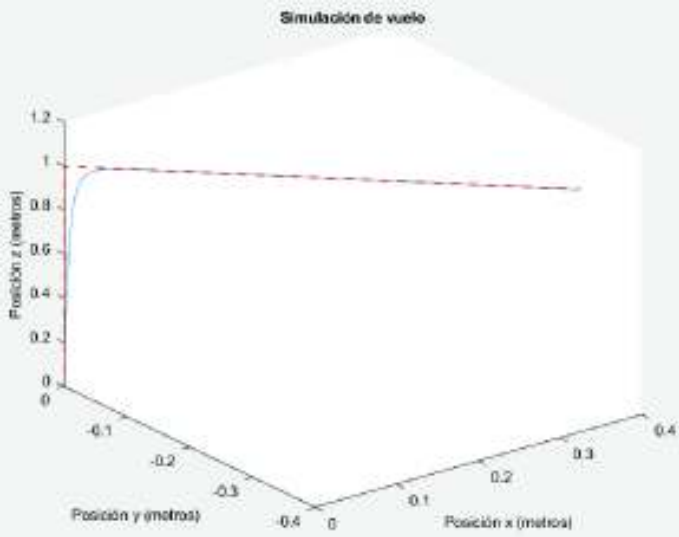


Figura 2.12. Simulación de vuelo del cuadrotor con el controlador no lineal aplicado al sistema no lineal.

(Fuente: elaboración propia)

Como se puede apreciar, los resultados entregados por la simulación de vuelo del cuadrotor coinciden con los esperados, y se puede notar una excelente y veloz respuesta del controlador, ya que en la mayoría de las variables controladas se nota que alcanzan una estabilización en el valor deseado en menos 0,5 segundos y todas se estabilizan en menos de un segundo.

Además, se puede verificar que el controlador funciona para cualquier conjunto de parámetros de entrada deseados, es así, que se realizaron varias pruebas de funcionamiento. Por ejemplo para los parámetros deseados de las ecuaciones 2.126 – 2.129.

$$Z_d = 1\,m \qquad \text{Ec. [2.126]}$$

$$\theta_d = 0^0 \qquad \text{Ec. [2.127]}$$

$$\emptyset_d = 0^0 \qquad \text{Ec. [2.128]}$$

$$\psi_d = 0^0 \qquad \text{Ec. [2.129]}$$

Se espera que el controlador al recibir únicamente un valor de entrada para la altura deseada, sin ningún ángulo de rotación, el resultado esperado es que el dron se eleve y únicamente se estabilice a una altura de un metro manteniendo su posición, como muestran las figuras 2.13. y 2.14.

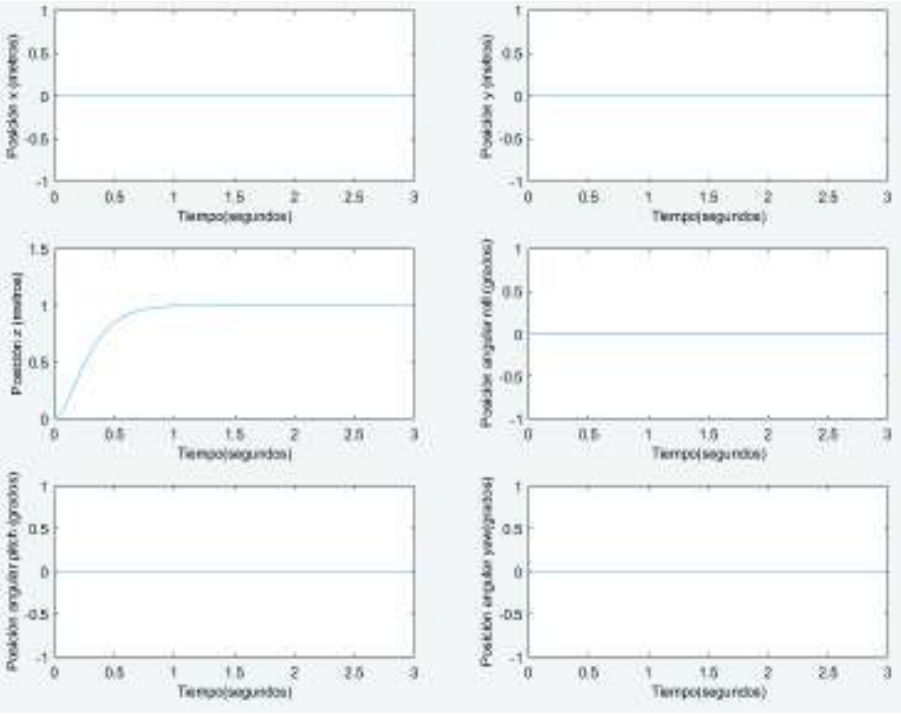


Figura 2.13. Simulación del controlador no lineal para una altura de 1 metro sin rotaciones en roll, pitch y yaw.

(Fuente: elaboración propia)

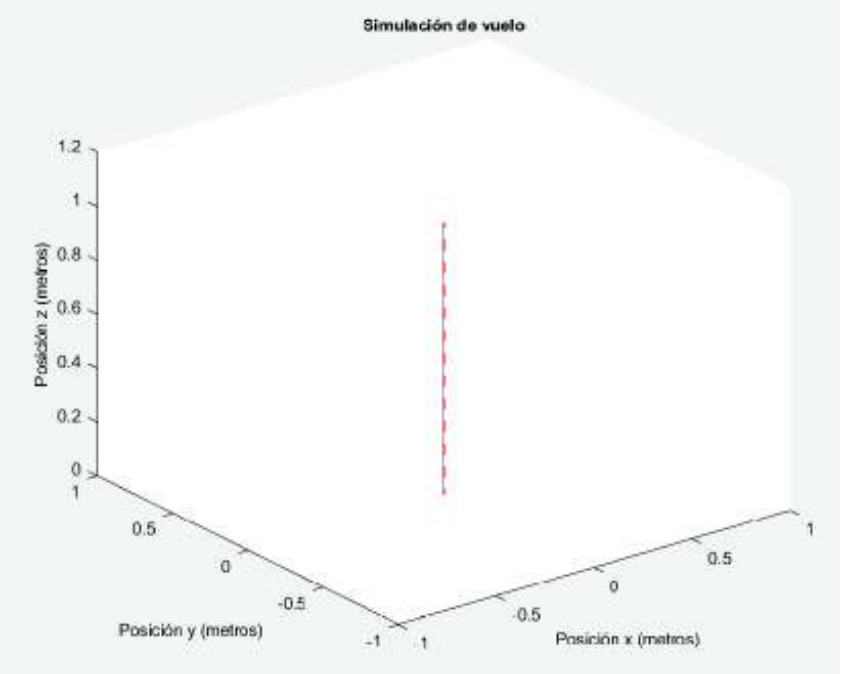


Figura 2.14. Simulación de vuelo del controlador no lineal para una altura de 1 metro sin rotaciones en roll, pitch, yaw.

(Fuente: elaboración propia)

Finalmente, se muestra un ejemplo de los resultados obtenidos para la simulación del controlador actuando bajo los parámetros de entrada de las ecuaciones 2.130 – 2.133.

$$Z_d = 1\,m \qquad \text{Ec. [2.130]}$$

$$\theta_d = 45^0 \qquad \text{Ec. [2.131]}$$

$$\emptyset_d = 0^0 \qquad \text{Ec. [2.132]}$$

$$\psi_d = 0^0 \qquad \text{Ec. [2.133]}$$

Por lo que se espera que realice un cambio de orientación del dron, generando un desplazamiento a lo largo del eje y, como muestran las figuras 2.15. y 2.16.

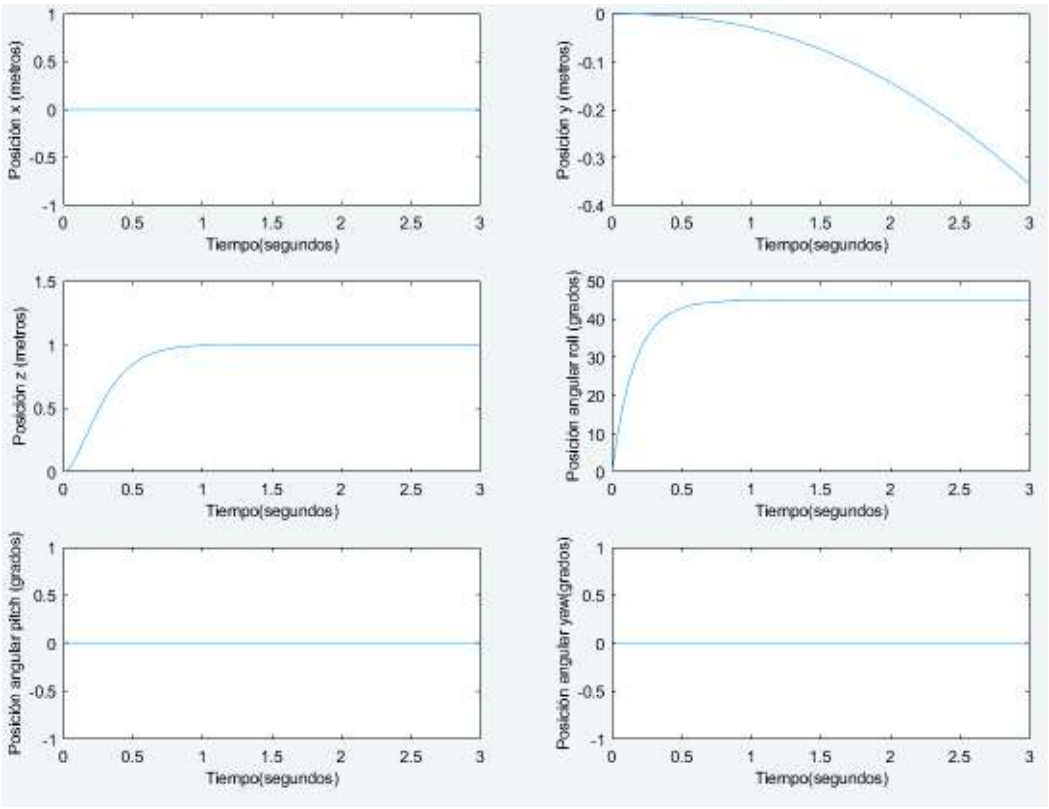


Figura 2.15. Simulación del controlador no lineal para una altura de 1 metro con rotación de 45º en roll.

(Fuente: elaboración propia)

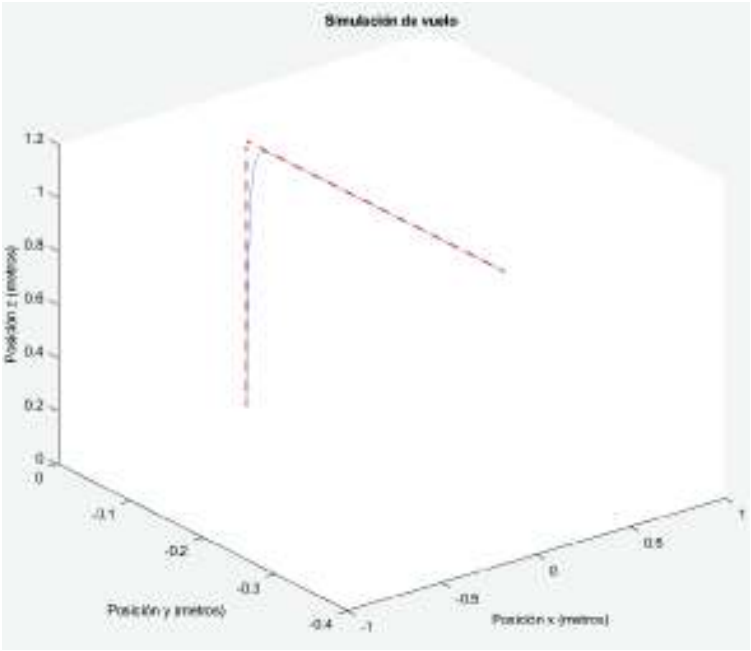


Figura 2.16. Simulación de vuelo del controlador no lineal para una altura de 1 metro con rotación de 45º en roll.

(Fuente: elaboración propia)

Como se puede apreciar, en esta simulación se solicitó al controlador accionar los motores hasta llegar a una altura de un metro, y a partir de ese momento efectuar un desplazamiento angular de 45º en torno al eje x, que corresponde a roll como se observa en la figura 2.17.

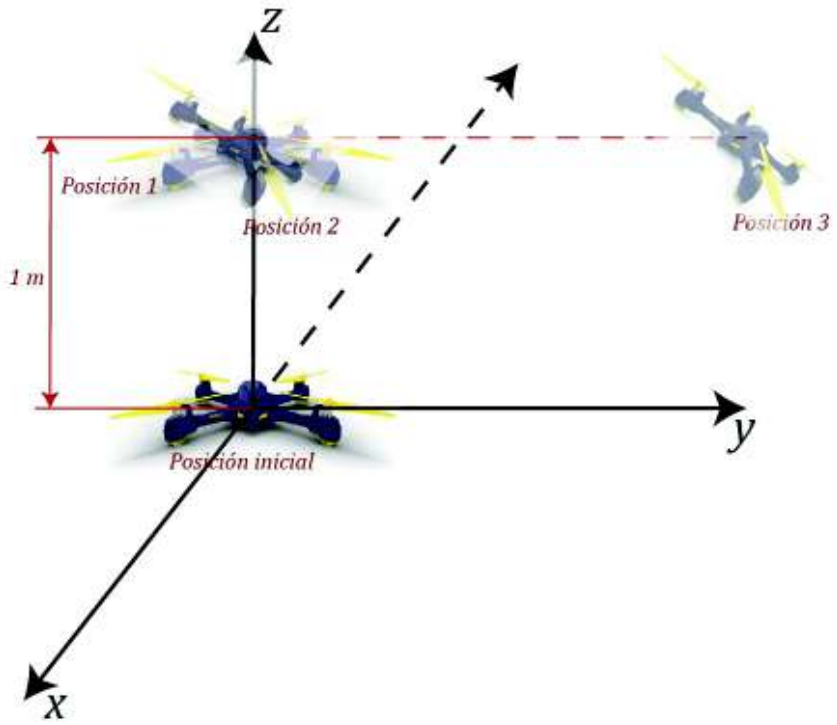


Figura 2.17. Resultados esperados para la simulación del cuadrotor con entradas 1 metro de altura y 45º en roll.

(Fuente: elaboración propia)

Como se pudo apreciar en los tres ejemplos anteriores, el controlador funcionó perfectamente ante los valores de entrada solicitados teniendo un tiempo de estabilización muy bueno y manteniendo al dron estable en los valores deseados.

2.3.6. Comparación de resultados entre el sistema lineal y no lineal

Como se mencionó en la sección anterior, un elemento de interés que permite evaluar si el controlador obtenido está actuando de manera apropiada es la comparación de la acción del mismo aplicado sobre los sistemas lineal y no lineal, ya que de esta manera se puede evaluar si el sistema linealizado corresponde con la realidad del sistema no lineal, teniendo en cuenta que las ganancias que permiten controlar al sistema se obtuvieron a partir del sistema linealizado. Es decir, verificar el correcto funcionamiento del sistema linealizado

permite comprobar que la suposición sobre la cual se obtuvo las ganancias es válida. Es así, que por ejemplo se validan los parámetros deseados de las ecuaciones 2.134 – 2.137.

$$Z_d = 1\,m \qquad \text{Ec. [2.134]}$$

$$\theta_d = 45^0 \qquad \text{Ec. [2.135]}$$

$$\emptyset_d = 45^0 \qquad \text{Ec. [2.136]}$$

$$\psi_d = 45^0 \qquad \text{Ec. [2.137]}$$

La respuesta de simulación del controlador sobre el sistema equivalente no linealizado empleando la función “controllinealfn” se observan en las figuras 2.18. – 2.24.

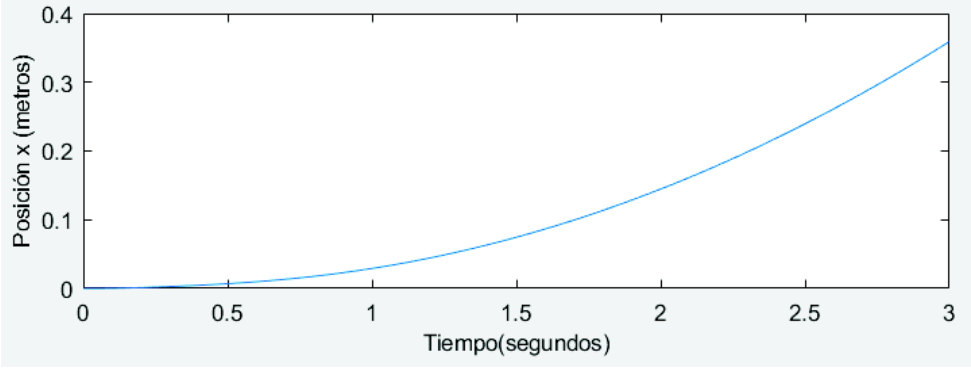


Figura 2.18. Desplazamiento en x del cuadrotor con el controlador no lineal aplicado al sistema linealizado.

(Fuente: elaboración propia)

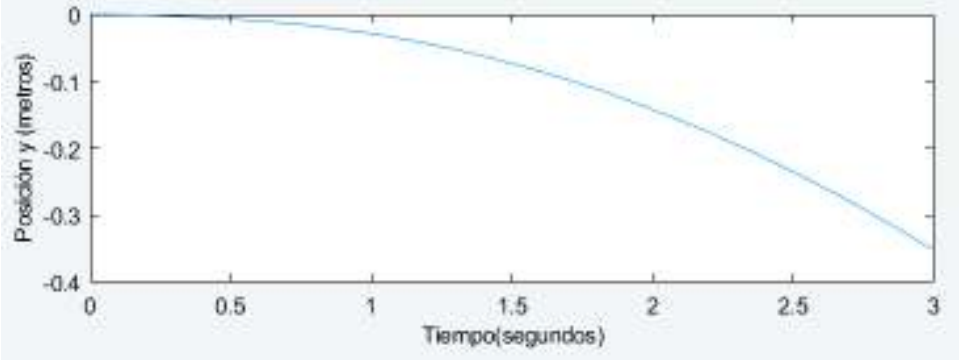


Figura 2.19. Desplazamiento en y del cuadrotor con el controlador no lineal aplicado al sistema linealizado.

(Fuente: elaboración propia)

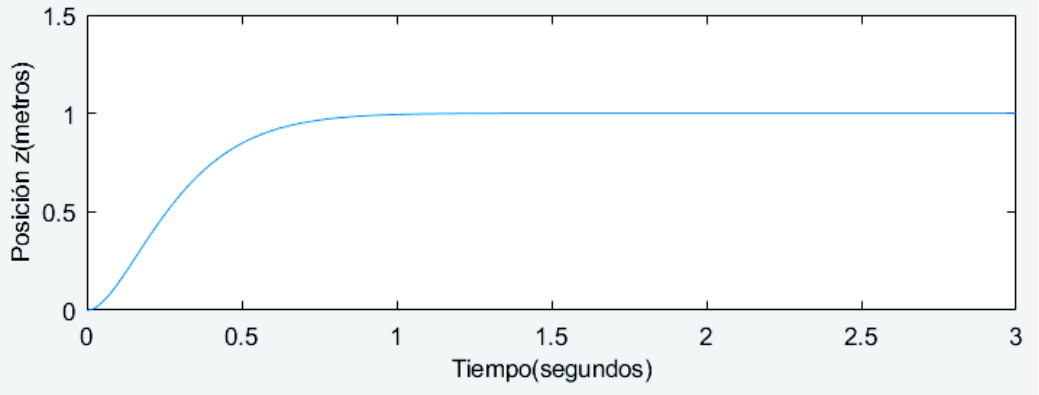


Figura 2.20. Desplazamiento en z del cuadrotor con el controlador no lineal aplicado al sistema linealizado.

(Fuente: elaboración propia)

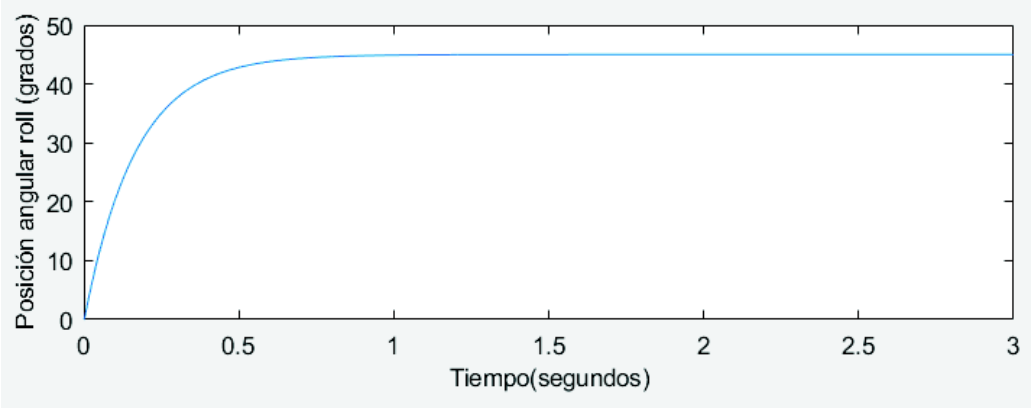


Figura 2.21. Desplazamiento angular en roll del cuadrotor con el controlador no lineal aplicado al sistema linealizado.

(Fuente: elaboración propia)

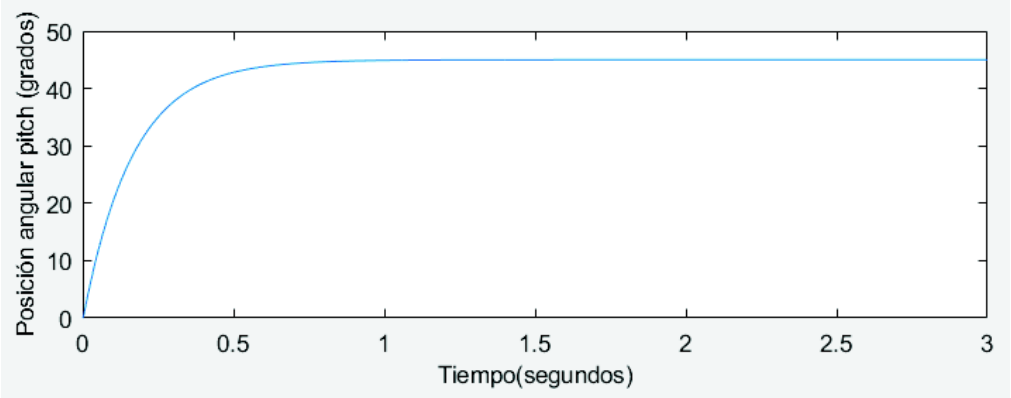


Figura 2.22. Desplazamiento angular en pitch del cuadrotor con el controlador no lineal aplicado al sistema linealizado.

(Fuente: elaboración propia)

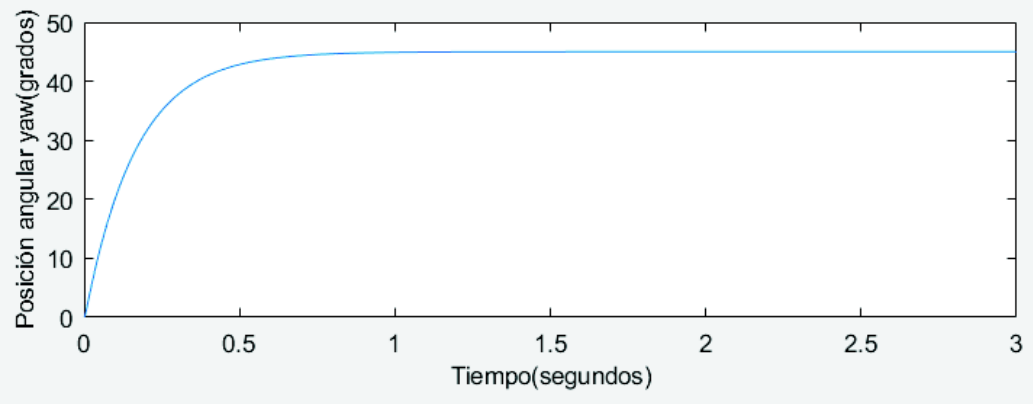


Figura 2.23. Desplazamiento angular en yaw del cuadrotor con el controlador no lineal aplicado al sistema linealizado.

(Fuente: elaboración propia)

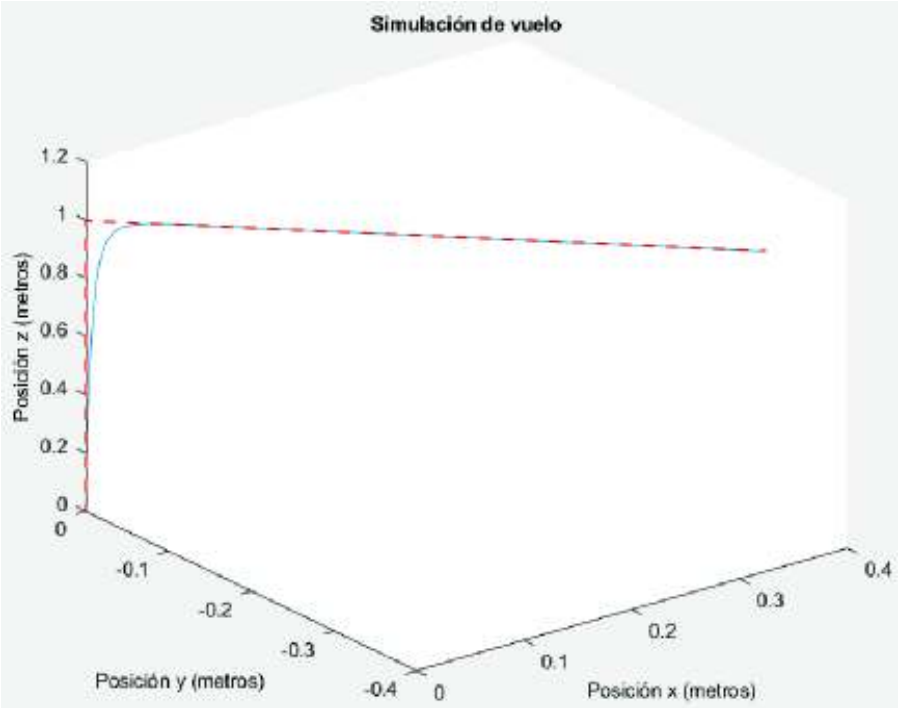


Figura 2.24. Simulación de vuelo del cuadrotor con el controlador no lineal aplicado al sistema linealizado.

(Fuente: elaboración propia)

Como se pudo visualizar en los resultados de la simulación obtenidos, la respuesta del controlador aplicado al sistema linealizado, fueron idénticos a los resultados obtenidos previamente en el sistema no lineal, lo que ratifica todo el proceso de linealización y que el valor de las 12 ganancias de control calculadas para el controlador retroalimentado son correctas y realizan una excelente acción de control sobre el cuadrotor. Cabe recalcar que el comportamiento del controlador sobre el sistema linealizado será óptima siempre y cuando se realicen acciones de control cercanas al punto de equilibrio, ya que fuera de

este rango la linealización no tendría efecto, es por eso, que el controlador que se utiliza finalmente es empleando el sistema no lineal, que no solo tiene una respuesta adecuada en el punto de equilibrio, sino ante cualquier acción que esté dentro de los límites considerados por la variables en el espacio de estados.

2.4. Desarrollo del algoritmo de procesamiento de imágenes

Esta sección del desarrollo del proyecto se enfoca en el procesamiento de las imágenes que se obtienen mediante la cámara a bordo del dron. Inicialmente para el desarrollo del algoritmo de procesamiento de imágenes se tiene en cuenta que el proyecto desarrollado también busca dotar del sistema de visión artificial a drones comerciales, y de ser posible realizarlo, este será aplicado en los dispositivos más accesibles, de manera que si posteriormente se desea implementar en aplicaciones industriales, de seguridad o entretenimiento, su implementación pueda ser realizada de la manera más económica y sencilla posible.

Además, es de interés para el presente proyecto explorar los métodos de adquisición de imágenes más comunes en los drones comerciales, que en la actualidad cuentan con la ventaja de disponer de Wi-Fi FPV (First Person View) en la mayoría de ellos. Este mecanismo de adquisición de imágenes también conocido como pilotaje con visión remota, permite al piloto del dron visualizar las imágenes que se están adquiriendo mediante la cámara del dron, en tiempo real y con alta fidelidad mientras el piloto maneja el dron. La diferencia principal que existe en la mayoría de drones comerciales que presentan FPV es que la mayoría de ellos están orientados con fines de entretenimiento y enfocados principalmente en la grabación del video en una unidad de almacenamiento a bordo, ya sea mediante el uso de un USB o micro SD, o a su vez en la memoria del smartphone que se emplea como controlador de vuelo del dron.

Es así, que son muy escasos los drones de bajo costo que presentan la ventaja de poder ser pilotados directamente desde una PC como el Parrot AR. 2.0, siendo la mayoría de softwares de vuelo de drones desarrollados para las plataformas Android y IOS como el Hubsan H507a. Esto representa una gran limitación dado que en dispositivos móviles no se cuenta con la misma capacidad de procesamiento que en un ordenador. Es por esto, que la propuesta de desarrollo de los algoritmos a continuación son enfocados a emplear la gran capacidad de procesamiento disponible en un ordenador portátil, pero por los motivos antes descritos se desarrollaron algoritmos de procesamiento de imágenes para ambos casos. El caso 1 enfocado en el procesamiento para un dron que puede ser pilotado directamente desde Windows mediante el ordenador, donde se tomó como referencia al

Parrot AR. 2.0, y en el caso 2 para los drones más comunes se empleó una consola virtual emuladora de Android, que permitió pilotar al dron mediante el ordenador corriendo simultáneamente Matlab.

Es por esto, que el diseño de los algoritmos, y la técnica de adquisición de imágenes difiere desde su esencia en cada caso y se desarrollaron programas diferentes para cada uno.

2.4.1. Algoritmo de adquisición de imágenes para el Parrot AR. 2.0

En el caso del Parrot AR. 2.0, el mecanismo de adquisición de imágenes presentó una gran ventaja debido a su amplia aplicación previa en proyectos de investigación en el área de los UAVs. Además, se dispone de una amplia variedad de softwares de pilotaje como: ARD 2 PC Flight, Auto Flight, AR.Drone, Windev AR.Drone, Windows AR.Drone .NET, AR.Drone playground, AR.Drone control simple, AR. Free Flight 2.0, eDrone Project y 4Focus. De los cuales se seleccionó el software ARD 2 PC Flight, ya que fue el que presentó una conexión TCP más estable respecto a los demás.

Además, para el Parrot AR. Drone 2.0 se dispone de librerías de Matlab como: AR.Drone 2.0 Support from Embedded Coder, Control AR Drone Parrot 2.0 with Matlab 2015a and Vicon, Control Multiple AR.Drone 2.0 with Vicon Feedback, entre otros; que facilitan la adquisición de la imagen desde la cámara y los sensores del dron, y ofrecen herramientas y toolbox de control y comunicación con el procesador del dron como también bloques para simulink que permiten el desarrollo de nuevos programas de control para el dron.

Es así que, el código de adquisición de imágenes a partir de la cámara del dron se desarrolló de manera que una vez conectado el dron con el Wi-Fi del computador, el programa ARD 2 PC Flight obtenga la imagen de la cámara por protocolo TCP, mediante la URL “tcp://192.168.1.1:5555/video.mjpg”, entonces el software ARD 2 PC Flight que emplea el decodificador ffdshow, presenta la función grab, que permite almacenar cada frame que ha sido decodificado en el directorio “C:\Users\ \MATLAB\data\base\”. La figura 2.25. muestra la configuración empleada para la adquisición de los frames mediante el decodificador ffdshow.

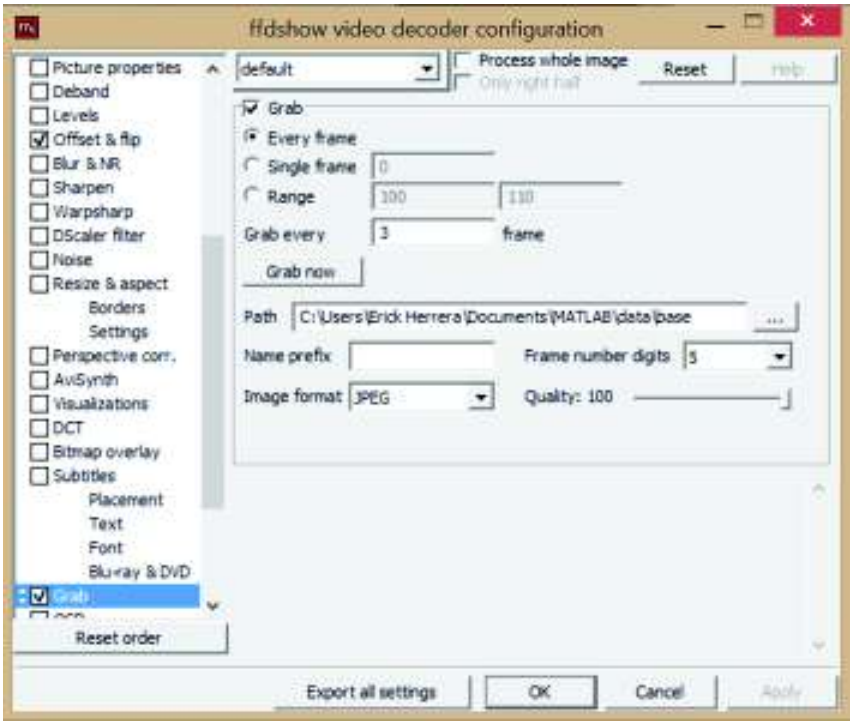


Figura 2.25. Configuración empleada para la adquisición de los frames en ffdshow.

(Fuente: elaboración propia)

En la figura 2.25. se puede visualizar que la herramienta grab permite obtener cada frame que fue decodificado a partir de la cámara del dron, además, en este ejemplo mediante el textbox “Grab every” se está obteniendo uno de cada tres frames para mejorar la velocidad de procesamiento. Si no se ingresa un valor en este casillero las imágenes procesadas ingresarían a una velocidad de 30 fps (frames por segundo), lo que implicaría mayor gasto computacional, por lo que en este caso se obtiene una velocidad de captura de 10 fps. Se ingresa el path donde se almacenarán los frames para el procesamiento, y mediante el textbox “Frame number digits” se establece que el nombre del archivo de cada frame se componga de cinco dígitos que irán guardándose de manera ascendente a medida que son capturados.

A continuación, el código desarrollado en Matlab se encargará de leer la base de datos que contienen las imágenes almacenada en la dirección “C:\Users\ \MATLAB\data\base\”, de esta manera, se generará una lista de todos los archivos (frames) capturados mediante la cámara contenidos en la base de datos. A partir de esta lista, el código de adquisición de imágenes de la cámara del dron determina cuál es el frame más reciente, obteniendo el nombre de cada frame almacenado en la base de datos, convirtiéndolo en formato numérico (double), y comparando este número con todos los demás contenidos en la carpeta; de manera que el frame más reciente será cuyo número identificador sea mayor.

El código se encargará de leer el último frame adquirido en una variable temporal de Matlab, para posteriormente eliminar todos los frames anteriores mediante el comando "delete" y quedarse únicamente con el frame actual que obtiene la cámara. De esta manera, el código permite ingresar la imagen del frame más reciente en Matlab, permitiendo que Matlab pueda procesar posteriormente el video en tiempo real.

A continuación se detalla el código de adquisición de imágenes para el Parrot AR. Drone 2.0:

```
lista=dir('base');  %Se obtiene una lista de todos los frames
almacenados en la base de datos

w=size(lista); %Se obtiene el tamaño de la matriz lista de nombres y
propiedades de los frames

    for i=1:1:w(1)-2 %Se realiza este proceso hasta el tamaño de la
    lista menos dos, ya que los dos primeros elementos de la lista
    son elementos de cabecera, mas no frames

        x=lista(i+2); %Se inicia la lectura de frames desde el tercer
        elemento de la lista

        y=x.name; %Se extrae el nombre de cada frame a manera de
        arreglo matricial

        num=strcat(y(1),y(2),y(3),y(4),y(5));  %Se reensambla el
        nombre de cada frame como string

        A(i) = str2double(num); %Se transforma el nombre en tipo
        número y se almacena en una matriz
    end

z=find(A==max((A)))+2; %Se encuentra el archivo cuyo número en el
nombre es mayor, lo que indica que es el frame adquirido más reciente

filename=lista(z).name; %Se obtiene el nombre del frame más reciente

im=imread(strcat('C:\Users\Erick
Herrera\Documents\MATLAB\data\base\',filename)); %Se lee el frame
más reciente

delete('C:\Users\Erick  Herrera\Documents\MATLAB\data\base\*.jpg');
%Se borran todos los frames anteriores al más reciente

imwrite(im,'C:\Users\Erick
Herrera\Documents\MATLAB\data\base\00001.jpg'); %Se guarda el frame
más reciente con el nombre 00001 para luego compararlo en el matching

imshow(im);
```

El resultado obtenido permite adquirir cada frame que está siendo enfocado por la cámara del Parrot AR. 2.0 en tiempo real, para posteriormente procesar cada imagen extrayendo su información, y presentar este efecto en la interfaz gráfica de Matlab (GUIDE) reensamblándola como un video. La figura 2.26. resume el proceso de adquisición de

imágenes hacia Matlab empleado para el Parrot AR. 2.0, mientras que la figura 2.27. muestra las imágenes obtenidas a partir del código de adquisición.

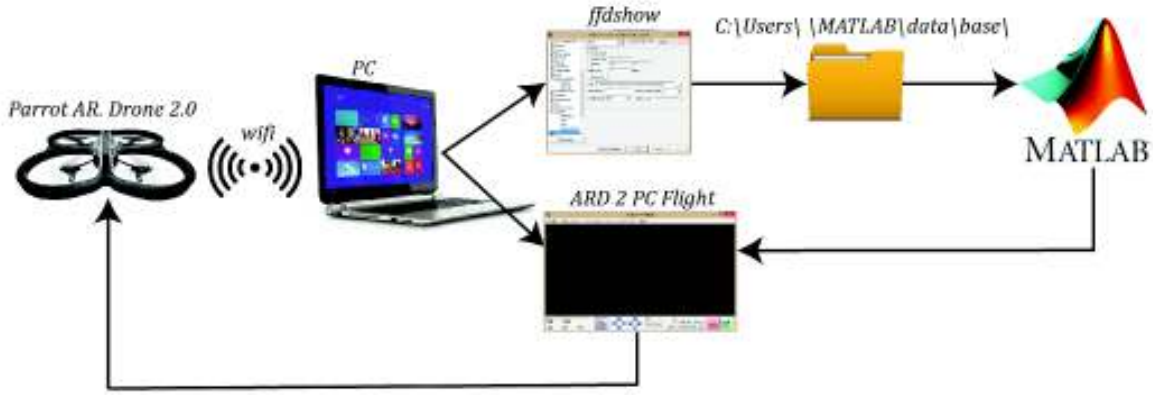


Figura 2.26. Esquema de adquisición de imágenes y conexión para el Parrot AR. Drone 2.0.
(Fuente: elaboración propia)

Figura 2.27. Resultado generado a partir del código de adquisición Parrot AR. Drone 2.0.
(Fuente: elaboración propia)

Cabe mencionar que en el caso del Parrot AR. 2.0 también se puede adquirir la imagen empleando otros métodos y toolbox de Matlab como el “hebicam”, que funcionan de manera similar a ”winvideo”, que es un add on que dispone el paquete de adquisición de imágenes empleado para cámaras USB, pero aplicado a cámaras IP. Pero en el caso de “hebicam” presenta la ventaja de poder adquirir el video proveniente de una cámara IP en protocolo TCP con la ventaja de ser el único add on de Matlab con soporte para los formatos MJPEG y h264 que emplea la cámara del Parrot. Sin embargo, este método no fue aplicado, ya que la conexión con la cámara no era estable y no permitía realizar acciones de control posteriores hacia el dron, es decir, presentaba interferencia con la

señal de control generada. El código que se puede emplear para establecer comunicación con la cámara del dron empleando hebicam se presenta a continuación:

```
clear cam;

url = 'tcp://192.168.31.111:8866/video.mjpg'; %Se ingresa la URL de
la cámara del Parrot

cam=HebiCam(url);
%imshow(cam.getsnapshot()); %%Se puede emplear para presentar una
sola imagen

figure(); %Se genera una figura

img=imshow(getsnapshot(cam)); %Se captura cada frame de la cámara

while true

    set(img, 'CData', getsnapshot(cam)); %Se reescribe la variable
    CData que almacena cada frame

    drawnow; %Se imprime la imagen para que imshow la vuelva a
    presentar

end
```

2.4.2. Algoritmo de adquisición de imágenes para el Hubsan H507a

En el caso de drones más genéricos, económicos y de fácil adquisición como el Hubsan H507a, el mecanismo de adquisición de imágenes en Windows resulta un tanto más elaborado que en el caso del Parrot AR. Drone 2.0, ya que al no contar con software previamente diseñado para el pilotaje del dron y adquisición de la imagen para poder emplear el software de pilotaje que proporciona el fabricante, es necesario emplear la plataforma Android.

Este hecho provoca que la adquisición de la imagen sea un tanto más complicada pero igualmente efectiva, y se podría decir que resulta de mucha utilidad haberla desarrollado previamente ya que de esta manera, el presente proyecto puede extender su rango de aplicación para futuros desarrollos hacia una mayor gama de drones comerciales.

El proceso de adquisición de imágenes para el Hubsan H507a inicia al emparejar el dron con el ordenador, donde el dron funciona como un punto de acceso al cual la PC puede emparejarse vía Wi-Fi. Posteriormente se ejecuta una máquina virtual de Android, en este caso se empleó el emulador NoxPlayer en su versión 6.0.5.3 que permite manejar una máquina virtual de Android en su versión 4.4.2 mediante el ordenador.

Dentro de la máquina virtual Nox se ejecuta el software de pilotaje del fabricante, que en este caso es la aplicación X-Hubsan que puede ser descargada de manera gratuita desde la Play Store de Google como muestra la figura 2.28.

Figura 2.28. Nox Player 6.0.5.3 máquina virtual Android, instalando la aplicación de pilotaje del Hubsan 507a.

(Fuente: elaboración propia)

Posteriormente, una vez en la aplicación la imagen adquirida por la cámara a bordo del Hubsan H507a se presentará dentro de la máquina virtual Android. Para poder llevar esta imagen a Matlab se lo puede realizar de diversas maneras, como por ejemplo tomando capturas de las imágenes presentadas dentro de la máquina virtual y almacenándolas en una base de datos para la PC, o transmitiendo la imagen mediante una IP local para luego leerla en el ordenador, como se realizó en el caso del Parrot.

Pero en el caso del Hubsan H507a se optó por otra alternativa, debido a que el funcionamiento de una máquina virtual reduce los recursos de Windows en general. Por lo tanto, se empleó un software adicional llamado “Manicam” que permite a partir de la imagen que se presenta en una ventana de Windows, emular una cámara web de tipo USB para que Matlab la pueda reconocer mediante el toolbox de adquisición de imágenes. Esta alternativa resulta muy conveniente debido a que se reduce en gran medida el costo computacional, ya que los frames no se están almacenando adicionalmente, sino que la imagen solo se lee como si se tratase de una cámara web conectada al ordenador de manera virtual.

Luego Matlab, lee la secuencia de frames como una cámara web mediante el toolbox de adquisición de video empleando una entrada de tipo “winvideo” en el canal 2, ya que el primero estará ocupado por la cámara web del ordenador, si la tuviera. Entonces, Matlab ingresa la imagen que obtiene la cámara del dron a manera de un objeto de video, del cual

primero se obtendrán sus parámetros de imagen largo, ancho y espacio de color. Para mediante estos valores generar una matriz de ceros de las mismas dimensiones que permitirá presentar la imagen sobrepuesta en ella. Posteriormente se define un elemento trigger que permitirá definir cuantos frames se adquieren en cada captura, y un frame grab interval que permite definir cada cuantos frames obtenidos por la cámara se efectuará la captura. Finalmente, los resultados de los frames obtenidos se presentan en una interfaz gráfica a manera de un video reensamblado a partir de una secuencia de frames.

El código desarrollado para la adquisición del video del Hubsan H507a se detalla a continuación:

```
vid=videoinput('winvideo', 2, 'RGB24_640x360'); %Se ingresa el video
adquirido mediante la cámara del dron como si se tratara de una
cámara USB

vid11=vid; %Se almacena esta entrada en una nueva variable

vidRes=vid11.VideoResolution; %Se obtiene la resolución del video

imWidth=vidRes(1); %Se lee el primer parámetro del arreglo matricial
que corresponde al ancho de la imagen en pixeles

imHeight=vidRes(2); %Se almacena el alto de la imagen en una nueva
variable

nBands=vid11.NumberOfBands; %Se lee el número de canales que tiene
la imagen que en el caso RGB son tres

hImage=imagesc(zeros(imHeight,imWidth,nBands),'parent',handles.axes
2); %Se crea una matriz de ceros del mismo tamaño y resolución de la
imagen para que contenga a la imagen real

preview(vid11, hImage); %Se ejecuta una previsualización de la
secuencia de frames a manera de video

set(vid,'ReturnedColorspace','rgb'); %Se define el espacio de color
como RGB

set(vid,'TriggerRepeat',Inf); %Se inicia un trigger (captura)

set(vid,'FramesPerTrigger',1); %Se define cuantos frames obtendrá en
cada captura

vid.FrameGrabInterval=5; %Se define cada cuantos frames se presentará
una imagen

triggerconfig(vid,'manual'); %Se define la configuración del trigger
como manual

start(vid); %Se inicia el objeto de video

tic

trigger(vid);while islogging(vid)==1, end
```

```
while r==0 %Se define un lazo while y un valor r mediante el cual se
podrá ejecutar o detener la adquisición de frames

%Adquisición de cada Frame

data=flip(getdata(vid,1),2); %Se obtiene cada frame del video para
el intervalo definido, y se voltea dos veces debido a que la imagen
que se recibe se visualiza como un espejo de la original

trigger(vid); %Se captura a partir del video

end
```

El resultado obtenido permite adquirir el video de la cámara del Hubsan H507a en tiempo real, para posteriormente capturar cada imagen, luego procesarla extrayendo su información, y presentar este efecto en la interfaz gráfica de Matlab reensamblándola mediante un video como muestran las figuras 2.29. y 2.30.

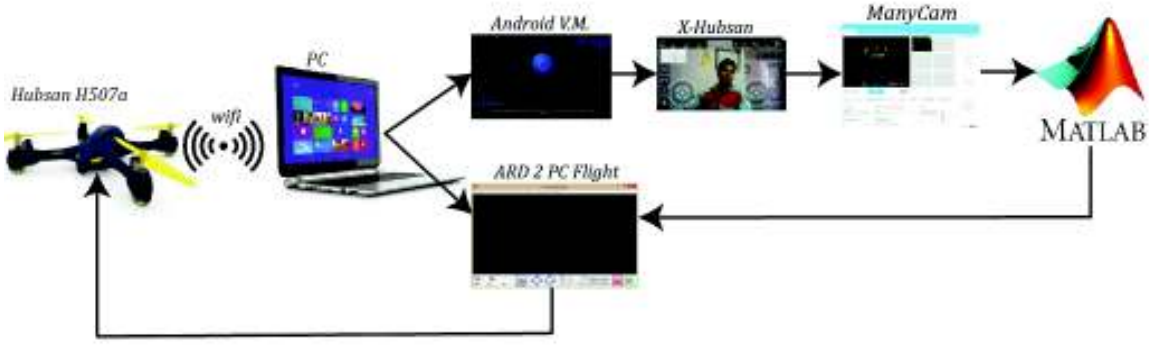


Figura 2.29. Proceso de adquisición del video hacia Matlab, empleando el Hubsan H507a.

(Fuente: elaboración propia)

Figura 2.30. Resultado generado a partir del código de adquisición para el Hubsan H507a.

(Fuente: elaboración propia)

2.4.3. Algoritmo de procesamiento de imágenes digitales

Es esta etapa del desarrollo del proyecto se de generó un código que permita a Matlab procesar las imágenes provenientes de la cámara del dron, ya que los resultados de reconocimiento facial de una persona siempre estarán sujetos a la calidad de la imagen que está llegando para aplicar el algoritmo de reconocimiento.

Un correcto procesamiento de las imágenes permite siempre obtener mejores resultados referentes a la detección de bordes y características (features) a partir de cada frame capturado mediante el algoritmo de adquisición, como se puede visualizar en la interfaz gráfica del código de procesamiento generado presentado en la figura 2.31. Además, como se ha demostrado en investigaciones previas, el resultado de reconocimiento facial depende y puede ser afectado por variaciones en las imágenes como la iluminación, expresión y pose[8]. Es así, que el factor que más afecta en la extracción de características, también conocido como reducción de dimensión, resulta ser el nivel de luminosidad, y por obvias razones la calidad de imagen que se va a procesar disminuye.

Los métodos más comúnmente aplicados en la detección de rostros son: Eigenfaces, Fisherfaces, Viola Jones, entre otros, y se encuentra que es necesario definir un valor umbral que permita discriminar las características obtenidas mediante Eigenfaces o Fisherfaces dependiendo del caso. Además, en todos los métodos citados, se puede decir que el factor común entre ellos radica en que la detección se realiza a partir de imágenes binarizadas, y en el método empleado en este caso para Matlab la correcta binarización depende de un valor umbral correctamente seleccionado.

Por lo tanto, en la sección de procesamiento de imágenes se pretende obtener un valor de umbral adecuado visualizando su correcto funcionamiento en tiempo real, y un método de mejoramiento de imágenes que permita alimentar al algoritmo de detección facial con imágenes de buena calidad.

El nivel de luminosidad se aplica en la transformación de una imagen a binaria, de manera que definiendo un valor umbral (threshold) los pixeles de una imagen que previamente ha sido transformada a escala de grises, se transformarán en unos para los valores mayores que ese umbral, y en ceros para los valores menores que ese umbral. Es así, que dependiendo de la luminosidad del entorno en el que se encuentra la imagen que se desea procesar definiendo un valor de threshold adecuado, se puede reducir el procesamiento de las imágenes a zonas de interés, donde podrían encontrarse los elementos que se desean

[8] (Ottado, 2015)

analizar, mejorando de esta manera los resultados finales obtenidos, reduciendo el costo computacional.

Dado que el presente proyecto tiene por objetivo implementar la detección de rostros a partir del procesamiento de las imágenes que capture la cámara a bordo del dron, se tomó en cuenta que la imagen obtenida dependerá del tipo de dron empleado. Es así, que se dispone de gran variedad de drones comerciales cuyo precio depende en gran medida de la cámara que llevan a bordo. Además, como es de interés reducir costos, se empleó un algoritmo que permita el mejoramiento de la imagen para drones que no presenten cámaras de gran capacidad como el Hubsan H507a, teniendo en cuenta también que a pesar de que el dron disponga de una cámara de altas prestaciones, no todas las imágenes capturadas serán de la calidad suficiente para una correcta detección.

Por lo que la implementación de un algoritmo de mejoramiento de imágenes permite disminuir en gran medida los errores en la detección de rostros, y el algoritmo implementado además permite mediante una interfaz gráfica (GUIDE) de Matlab visualizar este efecto en tiempo real, siempre con el objetivo de obtener los mejores resultados posibles en la detección.

2.4.4. Interfaz GUIDE de procesamiento de imágenes

Para el procesamiento de las imágenes se desarrolló una GUIDE en Matlab que permite al usuario configurar el procesamiento de las imágenes provenientes del dron mediante 10 pushbuttons, un radiobutton, un edittext, una slide bar, una listbox y 6 axes, como se puede visualizar en la figura 2.31.

Como se puede visualizar en la figura 2.31., la GUIDE desarrollada cuenta con 6 axes destinadas a que el usuario pueda visualizar en tiempo real el efecto que tiene la manipulación de los elementos slider1, listbox1 y radiobutton1, sobre el procesamiento de cada frame adquirido. El elemento slider1 tiene por objetivo permitir al usuario definir un valor umbral entre 0.001 y 0.251, que permita una correcta binarización de la imagen enfocada en extraer y mejorar particularmente el área de la imagen que contiene las características del rostro. Además, se dispone del elemento radiobutton1 que permite al usuario seleccionar el valor del threshold de manera automática para la binarización mediante la función “graytresh” del paquete de procesamiento de imágenes de Matlab empleando el método Otsu.

Figura 2.31. Interfaz gráfica de usuario GUIDE para la configuración del procesamiento de imágenes.

(Fuente: elaboración propia)

El elemento listbox1 en cambio tiene por objetivo que el usuario seleccione entre cuatro de las opciones de mejoramiento de imágenes más comúnmente usadas, como son:

- Gradiente Laplaciano
- Gradiente de Scharr
- Filtro Gausiano con Gradiente Laplaciano
- Filtro Gausiano con Gradiente de Scharr

Estas técnicas permiten el mejoramiento de las imágenes adquiridas, orientado a una detección adecuada de bordes mediante la aplicación de un kernel sobre toda la imagen, que se encargará de acentuar el cambio brusco de coloración, permitiendo destacar los bordes de la imagen, además, de permitir eliminar en lo posible el ruido existente en la imagen, que podría desembocar en una extracción de bordes incorrecta.

Además, en la GUIDE se dispone de 6 axes para la visualización de los efectos de la configuración del procesamiento de la imagen, mostrando a manera de video los cambios que se efectúan sobre la imagen, el área de interés delimitada y las características que se están extrayendo a partir de esta área de interés en la imagen.

Las axes empleadas y la información que están destinadas a mostrar, se detallan a continuación, las mismas que se pueden observar en la figura 2.32.

- Axes 2 (Imagen original): muestra al usuario en tiempo real la secuencia de imágenes que se están adquiriendo previas al procesamiento, cabe mencionar que en este recuadro siempre se visualizarán las imágenes invertidas horizontalmente, debido a que los drones típicamente adquieren el video de esta manera por efecto espejo.
- Axes 3 (Seteo de Grey Threshold): permite al usuario visualizar el efecto obtenido al desplazar el elemento slider1 que ajusta el valor del umbral de grises o pulsando el pushbutton1 que establece un valor automático de umbral. De esta manera muestra el efecto final sobre la imagen binarizada, donde se puede visualizar en blanco la región de interés extraída.
- Axes 4 (Selección de método de filtrado): permite al usuario visualizar el efecto del filtrado seleccionado mediante el objeto listbox 1, de esta manera el usuario podrá comparar y establecer en tiempo real cuál es el método de filtrado más adecuado con base a la correcta extracción de bordes observada, y a la visualización del método que presenta menor ruido en la imagen final.
- Axes5 (Detector de características): permite visualizar en video el proceso de matching, donde mediante una imagen pregrabada a través del código y almacenada en una base de datos, se continúa comparando con cada uno de los nuevos frames extraídos a partir del video. Es así, que permite observar la ubicación de las características extraídas visualizando el ploteo de la posición de cada característica que coincide entre las imágenes de la base de datos y cada nuevo frame que se adquiere. La intención de la implementación de esta axes es que el usuario pueda visualizar un correcto seteo del threshold, permitiendo reducir el área de extracción de características a una menor, donde se encuentra el rostro, y de esta manera mejorar la calidad de las características adquiridas para el aprendizaje, reduciendo el costo computacional.
- Axes 6 (Detector de bordes): permite que el usuario visualice el efecto final de la extracción de bordes una vez aplicados los filtros y con la imagen binarizada. Donde los píxeles que se observan en blanco corresponden a los bordes, y los píxeles en negro representan toda la información de la imagen que fue eliminada.
- Axes 7 (Áreas detectadas): permite visualizar en tiempo real, el o las áreas finales obtenidas a partir del procesamiento donde podría encontrarse un rostro. Este resultado se muestra empleando bounding boxes de color rojo etiquetadas con el

nombre de la persona con la que se está creando la base de datos. Cabe recalcar, que como se visualizó en la figura 2.31., si no se asigna un nombre para la base de datos, el software etiquetará al área como “Persona”. Además, en esta sección del procesamiento, aún se detectan elementos que no son rostros ya que en el algoritmo aún no se ha aplicado ningún clasificador de características o algoritmo de detección facial.

Figura 2.32. Elementos de la GUIDE de procesamiento de imágenes.

(Fuente: elaboración propia)

Adicionalmente en la GUIDE de procesamiento de imágenes adquiridas se dispone de diez pushbuttons, cuyas funciones son las siguientes:

- Pushbutton1 (Crear base de datos): solicita al usuario ingresar un nombre en el editbox1, al pulsar este botón el callout ejecutará el algoritmo de creación de la base de datos, almacenando 180 frames en la carpeta base del directorio de Matlab donde se está trabajando.
- Pushbutton2 (Capturar): ejecuta el código encargado de almacenar la imagen contenida en ese instante en el axes 5, en el directorio donde se está trabajando con el nombre “matching.jpg”.
- Pushbutton3 (Capturar): ejecuta el código encargado de almacenar la imagen contenida en ese instante en el axes 6, en el directorio donde se está trabajando con el nombre “bordes.jpg”.

- Pushbutton4 (Capturar): ejecuta el código encargado de almacenar la imagen contenida en ese instante en el axes 3, en el directorio donde se está trabajando con el nombre “Seteo_Graythreshold.jpg”.
- Pushbutton5 (Capturar): ejecuta el código encargado de almacenar la imagen contenida en ese instante en el axes 4, en el directorio donde se está trabajando con el nombre “Seleccion_Filtrado_Gradiente.jpg”.
- Pushbutton6 (Guardar configuración): se encarga de guardar los valores de threshold y método de procesamiento, obtenidos y ajustados en la interfaz gráfica, para que puedan ser empleados y llamados en el código del controlador.
- Pushbutton7 (Ejecutar controlador): permite llamar a ejecución a la GUIDE y al código del controlador, a su vez ejecuta el detector de rostros empleando la configuración guardada en el procesamiento.
- Pushbutton8 (Finalizar procesamiento): como el código de procesamiento de imágenes se genera a partir de una cadena de lazos while que extraen y procesan cada frame, es necesario detener esta secuencia y finalizar los lazos while para poder ejecutar otra GUIDE o cerrar la actual. Es así, que este botón permite salir del lazo de configuración de procesamiento para pasar al lazo de detección o simulación de vuelo.
- Pushbutton12 (Simulador de vuelo): permite ejecutar la GUIDE y el código referente al simulador de vuelo, el mismo que permite observar en tiempo real una simulación de la información adquirida por los sensores de vuelo del dron.
- Pushbutton13 (Resetear detector): dado que para pasar a las GUIDE del controlador o el simulador de vuelo es necesario detener el lazo de configuración de procesamiento, si se desea regresar a la interfaz de procesamiento para reajustar parámetros, es necesario reiniciar el lazo while de procesamiento. Es así, que este botón permite borrar la base de datos y parámetros de filtrado y threshold para reajustarlos.

2.4.5. Ajuste del umbral (Threshold)

El threshold como se mencionó en la sección anterior, corresponde al valor numérico entre cero y uno, mediante el cual una imagen que está en escala de grises se convierte en binaria (blanco y negro). Pero para la detección de características y reducción de la imagen a ser detectada es conveniente que este valor permita reducir el área de trabajo a regiones de la imagen donde es más probable encontrar un rostro.

Esto se puede lograr en una primera etapa descomponiendo a la imagen capturada, de la secuencia de frames provenientes de la cámara del dron, en formato RGB extrayendo su matriz de color rojo, debido al hecho de que en la composición de colores, el rostro humano en los diferentes tonos de piel presenta mayores niveles de color rojo.

Es así que la matriz R, que es la primera en el arreglo matricial de cada frame puede ser extraída, para posteriormente convertirla en escala de grises desactivando el colormap de la imagen, y finalmente, mediante un valor umbral convertirla en binaria. Este valor umbral, gracias a la GUIDE creada se puede configurar manualmente empleando el slider, o aplicar automáticamente el valor de threshold calculado mediante el método Otsu que minimiza la varianza intraclase de los píxeles blancos y negros que es obtenido a partir del histograma de frecuencias de los tonos de los píxeles. El código empleado para el seteo del Threshold se detalla a continuación:

```
%Seteo del threshold

axes(handles.axes3); %Se define el axes3 de la GUIDE como elemento
donde se presentarán los resultados

im=data; %Se almacena cada frame extraído en una nueva variable im

im1=im;

im2=imsubtract(im1(:,:,1),rgb2gray(im1)); %Se extrae el componente
rojo de la matriz rgb y se la convierte a escala de grises

im2=medfilt2(im2,[3,3]); %Se usa filtro median de 3X3 para reducir
ruido
```

Para la configuración manual se toma el valor obtenido a partir del elemento slider1, cuyo valor en la interfaz gráfica está en un intervalo de $0 \leq sldr \leq 1$, pero en la aplicación hacia la detección de rostros se encontró que el valor del umbral afecta considerablemente hasta un valor de $0{,}25$, ya que para umbrales mayores que este valor el threshold resulta en una matriz totalmente en cero. Por lo que se aplicó la función de la ecuación 2.138.

$$gtr = 0{,}25sldr + 0.001 \qquad \text{Ec. [2.138]}$$

A continuación se presenta el código para la configuración manual:

```
if auto==0 %Se emplea el valor cero para configuración manual

      gtr=sldr*0.25+0.001; %Se aplica el valor del slider en un rango
      de 0.01 a 0.26

      im3=im2bw(im2,gtr); %Se convierte la imagen de la matriz en
      escala de grises a binaria empleando el umbral
```

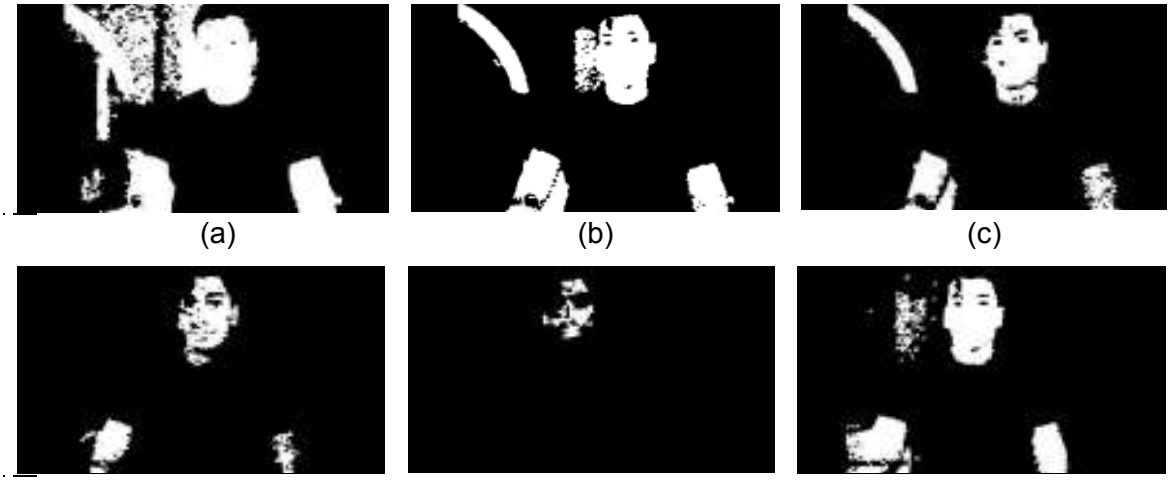
```
        im4=bwareaopen(im3,300);  %Se remueven los pixeles cuya
        frecuencia sea menor o igual a 300 para mejorar la resolución

        imshow(im4); %Se muestra el resultado
```

A continuación se presenta el código para la configuración automática:

```
    else %Cuando el radiobutton1 sea activado configuración automática

        gtr=graythresh(im2); %Se calcula automáticamente el threshold
        empleando el método Otsu

        im3=im2bw(im2,gtr); %Se convierte la imagen de la matriz en
        escala de grises a binaria

        set(handles.text3,'String',num2str(graythresh(im2)));     %Se
        imprime el valor del threshold automático en el textbox

        im4=bwareaopen(im3,20);  %Se remueven los pixeles cuya
        frecuencia sea menor o igual a 300 para mejorar la resolución

        imshow(im4);% Se muestra el resultado

    end
```

La configuración automática se obtuvo en este caso empleando la función greythresh de Matlab, pero también se puede obtener mediante el histograma de frecuencias de los píxeles si se desea conseguir valores más precisos como se detalla en la sección de detección de bordes.

En la figura 2.33. se puede visualizar los resultados que se obtienen para diferentes valores de umbral, es así que, el ajuste del umbral permite obtener una imagen binarizada enfocada únicamente en las zonas de interés, en este caso la piel y rostro basado en su luminosidad y tono. En la figura 2.33. para los casos desde a hasta e se aprecia que a medida que el umbral incrementa, el área blanca en la imagen binarizada se va reduciendo, y esto continuará hasta que el umbral sea tan elevado que todos los píxeles sean descartados, obteniéndose una matriz de ceros (negro). También se puede observar en la figura 2.33. (f) que para el ajuste automático del umbral mediante la función greythresh usado en la mayoría de aplicaciones, no es la más indicada para la detección facial. De ahí, la importancia por el que se implementó el método manual de ajuste mediante la GUIDE. En este ejemplo la configuración más apropiada obtenida corresponde al umbral de 0,076 y su efecto en el área delimitada para la futura extracción de características comparado con el ajuste automático.

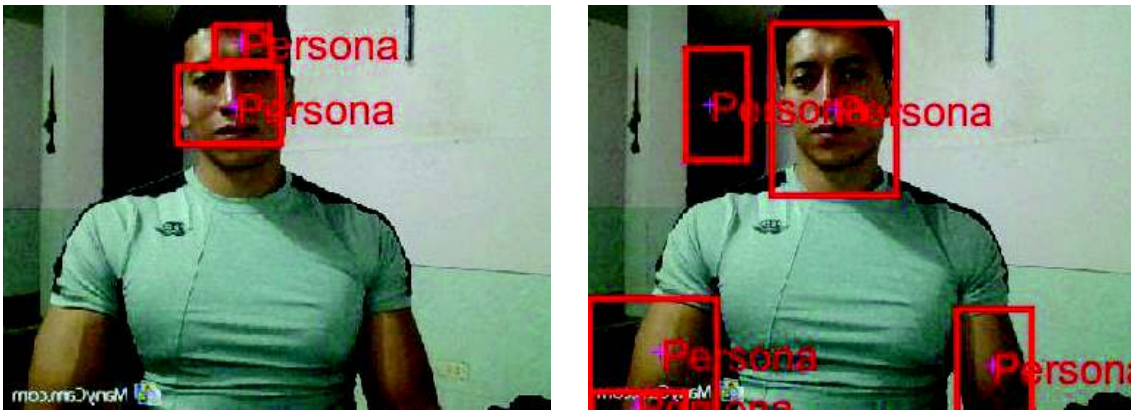


(a) (b) (c)

(d) (e) (f)

Figura 2.33. Ejemplo de imágenes binarizadas obtenidas empleando diferentes valores de umbral, (a) Threshold = 0.001, (b) Threshold = 0.033, (c) Threshold = 0.0060, (d) Threshold = 0.076, (e) Threshold = 0.099, (f) ajuste automático calculado con el método Otsu Threshold = 0.039.

(Fuente: elaboración propia)



(a) (b)

Figura 2.34. Comparación entre resultados obtenidos para el ajuste manual y automático del umbral, (a) ajuste manual Threshold = 0,076, (b) ajuste automático Threshold = 0,039.

(Fuente: elaboración propia)

En la figura 2.34. (b) se puede visualizar claramente que el ajuste greythresh para la aplicación de detección de rostros no es el mejor ya que se tienen muchos falsos positivos que generaran un mayor costo computacional en el proceso de detección al extraer características y aplicar los clasificadores. De aquí la importancia y la ventaja de disponer del método de ajuste manual que se tiene en el código desarrollado.

2.4.6. Selector de métodos de filtrado y mejoramiento de imágenes

En esta sección se desarrolló un algoritmo que mediante la GUIDE desarrollada en la sección 2.4.4. permite al usuario seleccionar el método de filtrado a ser empleado en el mejoramiento de la imagen y detección de bordes. Se proponen dos métodos de filtrado: Gausiano, Laplaciano y Scharr, que son los más comúnmente aplicados en detección de bordes por gradientes. Adicionalmente, en la siguiente sección se presentará el método de detección de bordes mediante el algoritmo Canny.

De esta manera, la segmentación de la imagen puede ser seleccionada, y gracias a la interfaz gráfica se puede visualizar el efecto en el video obtenido mediante la cámara del dron.

El algoritmo de filtardo está diseñado en función de los cuatro casos antes descritos de manera que a partir del elemento listbox de la GUIDE, y su propiedad value que varía entre 1 y 4 dependiendo del caso, se elige uno de los cuatro parámetros del lazo case del código. La propiedad value se obtiene mediante la función:

```
function listbox1_Callback(hObject, eventdata, handles)
      global list1
      ltpos=get(handles.listbox1,'value');
      list1=double(ltpos);
```

Donde, el elemento “list1” se define como variable global para que pueda ser llamado en la sección inicial del lazo while de adquisición de imágenes, y para que se pueda seleccionar el caso dentro del mismo.

Es así, que la segmentación de las imágenes para la extracción de sus bordes se realizó empleando una metodología muy similar a los pasos de Marr-Hildreth, proponiéndose de esta manera:

- Primero: aplicar un filtro Gaussiano para suavizar la imagen y eliminar el ruido en lo posible.
- Segundo: aplicar una segmentación por gradientes sea Laplaciano o Scharr.
- Tercero: aplicar un threshold en el resultado obtenido para obtener únicamente los bordes en color blanco (unos) y las regiones sin bordes en negro (ceros)

El código empleado para la selección del método de filtrado se describe a continuación:

```
%Selector de filtros y gradientes

axes(handles.axes4); %Se selecciona el axes4 como elemento donde se
presentarán los resultados
```

```
a=rgb2gray(im); %Se convierte cada frame en escala de grises

ar=double(a); %Se extiende el valor de cada pixel a 8 bytes

s=size(a); %Se obtienen las dimensiones de la matriz de la imagen

arc=ar*0; %Se genera una matriz de ceros de las mismas dimensiones
de la imagen

c=list1; %Se llama el valor del caso seleccionado en  la listbox para
que permita seleccionar el case en el switch

switch c
```

2.4.6.1. Caso 1: Gradiente Laplaciano

La aplicación de gradientes para la imagen en escala de grises permite la extracción de los bordes de la imagen ya que la segunda derivada tiene una respuesta más fuerte ante cambios de tonalidad como puntos o líneas, discontinuidades en general. En este caso, el gradiente Laplaciano corresponde a la segunda derivada bidimensional de los valores de intensidad de los píxeles, y su cambio ante la comparación con los píxeles vecinos como muestra la ecuación 2.139.

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \qquad \text{Ec. [2.139]}$$

La obtención de esta segunda derivada discreta a partir de los valores de los píxeles se puede obtener por el método de las diferencias finitas como una expansión en series de Taylor de la función $u(x,t)$ expresadas en las ecuaciones 2.40 y 2.141.

$$u(x+\Delta x,t) \approx u(x,t) + \frac{\partial u}{\partial t}(x,t)\Delta x + \frac{1}{2}\frac{\partial^2 u}{\partial x^2}(x,t)[\Delta x]^2 \qquad \text{Ec. [2.140]}$$

$$u(x-\Delta x,t) \approx u(x,t) - \frac{\partial u}{\partial t}(x,t)\Delta x + \frac{1}{2}\frac{\partial^2 u}{\partial x^2}(x,t)[\Delta x]^2 \qquad \text{Ec. [2.141]}$$

Al sumar las ecuaciones y despejar se obtiene la ecuación 2.142.

$$\frac{\partial^2 u}{\partial x^2}(x,t) \approx \frac{u(x+\Delta x,t) - 2u(x,t) + u(x-\Delta x,t)}{[\Delta x]^2} \qquad \text{Ec. [2.142]}$$

Donde, estas funciones a manera de kernel, también conocidas como máscaras o ventanas son matrices que se multiplicarán con cada conjunto posible de igual dimensión, sustituyendo el valor de cada píxel con uno procesado. Se aplicaron en dos tipos de

matrices de orden 3x3 y 5x5 obteniéndose resultados similares. Las matrices Laplacianas empleadas se detallan en las ecuaciones 2.143 y 2.144.

$$Lapl_{3\times3} = \frac{1}{16}\begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad \text{Ec. [2.143]}$$

$$Lapl_{5\times5} = \frac{1}{32}\begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 \\ 1 & 2 & -16 & 2 & 1 \\ 0 & 1 & 2 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \quad \text{Ec. [2.144]}$$

Es así, que al multiplicarse estas matrices con cada uno de los conjuntos del mismo tamaño devuelven un nuevo valor para cada píxel, donde, si no se ha producido un cambio considerable de intensidad en escala de grises no entrará en efecto el valor del término central de la matriz, por lo que de no detectarse un borde el valor resultante será el mismo que el que presentan el resto de píxeles de la imagen. El código aplicado para el case 1 del switch lógico se detalla a continuación:

```
case 1 %Gradiente Laplaciano

lapl1=[1 1 1;1 -8 1;1 1 1]/8; % Segenera el kernell laplaciano

for i=2:s(1)-1 %Se genera un lazo for que efectúe la operación
para cada fila de la imagen

    for j=2:s(2)-1 %Se genera un lazo for que efectúe la
    operación para cada columna de la imagen

        ventana=ar(i-1:i+1, j-1:j+1); %Se aísla cada matriz
        posible de la imagen de tamaño 3x3 que será afectada
        por el kernel

        prod=ventana.*lapl1; %Se multiplica la ventana por el
        laplaciano

        pix=sum(sum(prod)); %Se suman todos los valores
        obtenidos de la matriz

        arc(i,j)=(pix+255)/2; %Se sustituye cada pixel de la
        imagen or este nuevo valor

    end

end

arcc=arc;
```

2.4.6.1.1. Binarización de la imagen mediante el histograma de frecuencias

El resultado obtenido es una matriz aún en escala de grises, cuyos valores de intensidad se han homogeneizado de manera que solo quedan pocos posibles valores para cada pixel, que son los que no corresponden a un borde ya que todos llevan un único valor, y los que si corresponden a un borde llevan diferentes valores de mayor intensidad dependiendo de qué tan brusco fue el cambio para cada borde. Es así, que resulta conveniente umbralizar los valores de los píxeles de esta imagen para poder visualizar con claridad los resultados obtenidos.

Esta tarea se la puede realizar obteniendo el histograma de frecuencias de los valores de los píxeles. Por lo tanto, la imagen quedará dividida en 256 clases, ya que existen 256 posibles valores de intensidad que puede tomar cada pixel para una imagen en escala de grises. Finalmente, el valor umbral corresponderá al máximo pico en este histograma de frecuencias que concierne a la mayoría de valores de los píxeles, en otras palabras, los píxeles que no son bordes, y a partir de este valor hacia arriba se tendrán unos (bordes) o ceros (regiones).

El código que permite obtener el valor del threshold para binarizar la imagen y poder visualizar los bordes, se presenta a continuación:

```
%% Obtención de histograma

arrcc=uint8(arcc); %Se convierten los valores de los píxeles a
enteros

histR=zeros(1,256); %Se genera una matriz de ceros en una fila para
acumular las frecuencias del histograma

for f = 1:s(1) %Se inicia un lazo for para las filas de la matriz
imagen

    for c = 1:s(2) %Se inicia un lazo for para todas las columnas de
    la matriz imagen

        ngR=arrcc(f,c); %Se obtiene el nivel de gris de cada pixel
        en la matriz

        histR(ngR+1)=histR(ngR+1)+1; %Se acumula un valor de 1 en el
        histograma si el pixel coincide

    end

end

figure(3), plot(histR), title('Histograma') %Se presenta el
resultado del histograma
```

```
Prob=histR/(s(1)*s(2)); %Se transforma los valores del histograma a
valores probabilísticos entre (0 - 1)

A=Prob; %Se define una copia del histograma Probabilístico

T=find(A==max(max((Prob)))); %Se obtiene la posición del valor de
luminosidad donde se encuentra la saturación para los bordes

Tprob=(T)/255; %Se calcula el threshold en porcentaje decimal

I=uint8(arcc); %Se transforma la imagen detectada los bordes en
valores enteros

Ib=im2bw(I,Tprob); %Se binariza la imagen empleando el trhreshold
calculado mediante el histograma

EE=strel('disk',3); %Se crea un elemento disco de grosor 2 para dar
el grosor deseado a los bordes

Ie=imerode(Ib,EE); %Se expande el grosor de los bordes mediante la
resta de la imagen original menos la erosionada

Ip=abs(Ib-Ie-1); %Gradiente morfológico porque en el borde se produce
el máximo cambio de contraste

figure(4), imshow(Ip), title('Imagen Bordes') %Se presenta el
resultado final

figure(5), imshow(Ib), title('Imagen Bordes') %Se presenta el
resultado final
```

La figura 2.35. muestra el ejemplo de una imagen a la que se aplicó el Gradiente Laplaciano, su histograma y su umbralización en bordes color blanco o negro.

(a) (b)

Figura 2.35. Ejemplo de la aplicación del gradiente Laplaciano sobre una imagen, (a) Imagen original en escala de grises, (b) Resultado obtenido mediante el filtro Laplaciano.

(Fuente: elaboración propia)

Como se puede observar en la figura 2.35. (b), al aplicar el filtro Laplaciano se obtienen los bordes de la imagen, pero no se pueden apreciar con suficiente claridad. Es por eso, que es necesario el proceso de umbralización que en este caso se obtiene a partir del histograma de frecuencias mostrado en la figura 2.36.

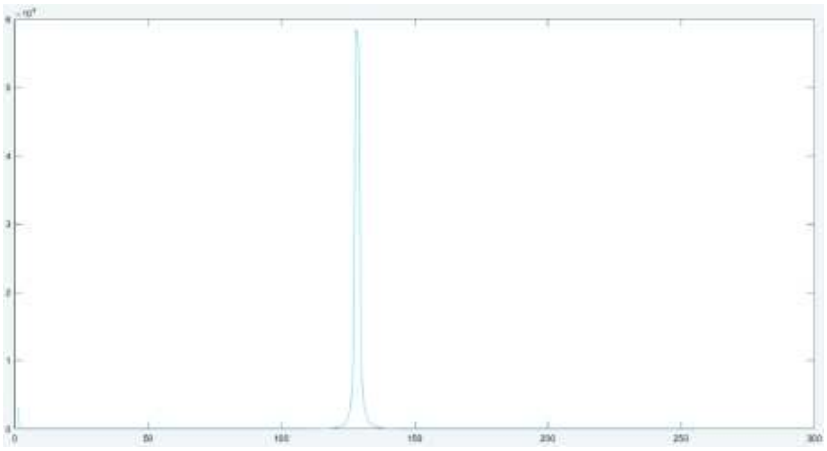
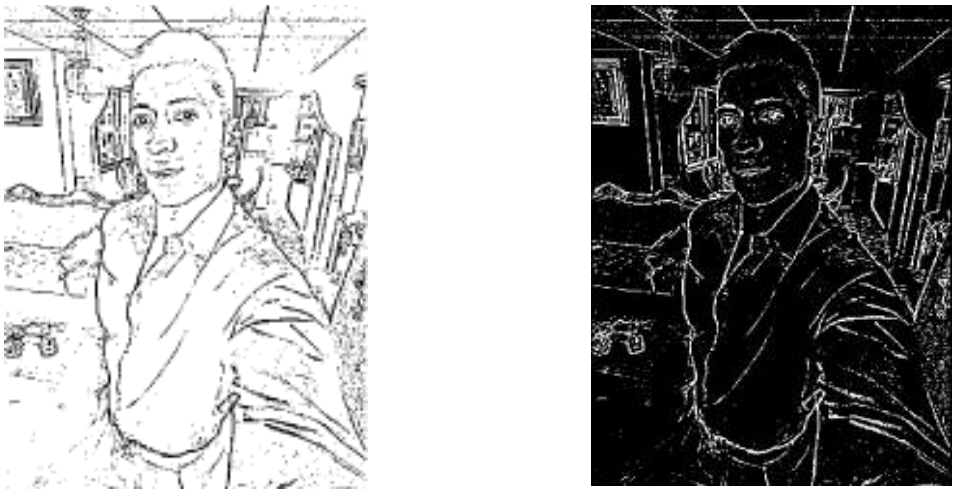


Figura 2.36. Histograma de frecuencias de la imagen ejemplo para el gradiente Laplaciano.
(Fuente: elaboración propia)

Como se puede visualizar en la figura 2.37., el gradiente Laplaciano funciona muy bien para la segmentación de bordes, pero se comprueba que el proceso de detección de bordes al provenir de la detección de discontinuidades provoca que el ruido también sea detectado como un falso positivo. De aquí, se concluye la importancia de la implementación de métodos de suavizado como puede ser un filtro Median, Gaussiano, entre otros.



(a) (b)

Figura 2.37. Resultados obtenidos para el ejemplo de detección de bordes empleando el gradiente espacial Laplaciano, (a) Bordes en negro (ceros), (b) Bordes en blanco (unos).
(Fuente: elaboración propia)

2.4.6.2. Caso 2: Gradiente de Scharr

En el caso 2 del switch lógico desarrollado para la segmentación de las imágenes, se propone emplear una convolución de primer orden utilizando un filtro diferencial de primer orden. De entre los gradientes de primer orden más comúnmente empleados se tienen: Prewitt, Sobel, Scharr entre otros, entre los que se eligió el gradiente de Scharr al considerarse uno de los más eficientes.

La diferencia principal en la aplicación de este filtro diferencial de primer orden radica en la posibilidad de emplearlo en diferentes direcciones, teniéndose de esta manera, una amplia variedad de Kernels que se pueden emplear dependiendo si la detección de bordes se desea aplicar en la dirección x, y o incluso con un ángulo de inclinación específico.

De esta manera, no se puede decir que represente una ventaja o desventaja respecto a la utilización de un Kernel Laplaciano, debido a que son filtros de gradientes de diferente naturaleza, mientras el filtro diferencial de primer orden de Scharr permite detectar bordes en una dirección específica, el filtro espacial Laplaciano detecta en todas las direcciones. Es así, que la decisión en la utilización de uno u otro método depende de la aplicación hacia donde está aplicada la detección de bordes.

El filtro diferencial de primer orden de Scharr se basa en la ecuación diferencial 2.145.

$$\nabla f = \frac{\partial f}{\partial x}\hat{x} + \frac{\partial f}{\partial y}\hat{y} \qquad \text{Ec. [2.145]}$$

$$\theta = atan2\left(\frac{\partial f}{\partial y}, \frac{\partial f}{\partial x}\right) \qquad \text{Ec. [2.146]}$$

Como se mencionó, la principal diferencia entre el gradiente de Scharr y el Laplaciano radica en su naturaleza de primer orden lo que le convierte en un filtro direccionado en el plano. Las máscaras que se pueden usar para sus distintas orientaciones del tipo 3x3 son las que se muestran en las ecuaciones 2.147 – 2.149.

$$Scharr_x = \frac{1}{16}\begin{bmatrix} -3 & 0 & 3 \\ -10 & 0 & 10 \\ -3 & 0 & 3 \end{bmatrix} \qquad \text{Ec. [2.147]}$$

$$Scharr_y = \frac{1}{16}\begin{bmatrix} -3 & -10 & -3 \\ 0 & 0 & 0 \\ 3 & 10 & 3 \end{bmatrix} \qquad \text{Ec. [2.148]}$$

$$Scharr_{45^o} = \frac{1}{16}\begin{bmatrix} -10 & -3 & 0 \\ -3 & 0 & 3 \\ 0 & 3 & 10 \end{bmatrix} \quad \text{Ec. [2.149]}$$

Si adicionalmente se desea emplear un Kernel de orden 5x5 se pueden utilizar los ejemplos propuestos en las ecuaciones 2.150 – 2.153, aunque se debe considerar que si bien su utilización mejora los resultados reduciendo falsos negativos, puede generar mayores falsos positivos, y aumentar la presencia de ruido, además, de incrementar el costo computacional.

$$Scharr\ 5 \times 5_{anisotrópico} = \frac{1}{84 + W^2}\begin{bmatrix} 1 & 2 & 3 & 2 & 1 \\ 2 & 4 & 12 & 4 & 2 \\ 3 & 12 & W^2 & 12 & 3 \\ 2 & 4 & 12 & 4 & 2 \\ 1 & 2 & 3 & 2 & 1 \end{bmatrix} \quad \text{Ec. [2.150]}$$

$$Scharr\ 5 \times 5_x = \frac{1}{60}\begin{bmatrix} -1 & -1 & 0 & 1 & 1 \\ -2 & -2 & 0 & 2 & 2 \\ -3 & -6 & 0 & 6 & 3 \\ -2 & -2 & 0 & 2 & 2 \\ -1 & -1 & 0 & 1 & 1 \end{bmatrix} \quad \text{Ec. [2.151]}$$

$$Scharr\ 5 \times 5_y = \frac{1}{60}\begin{bmatrix} -1 & -2 & -3 & -2 & -1 \\ -1 & -2 & -6 & -2 & -1 \\ 0 & 0 & 0 & 0 & 0 \\ +1 & +2 & +6 & +2 & +1 \\ +1 & +2 & +3 & +2 & +1 \end{bmatrix} \quad \text{Ec. [2.152]}$$

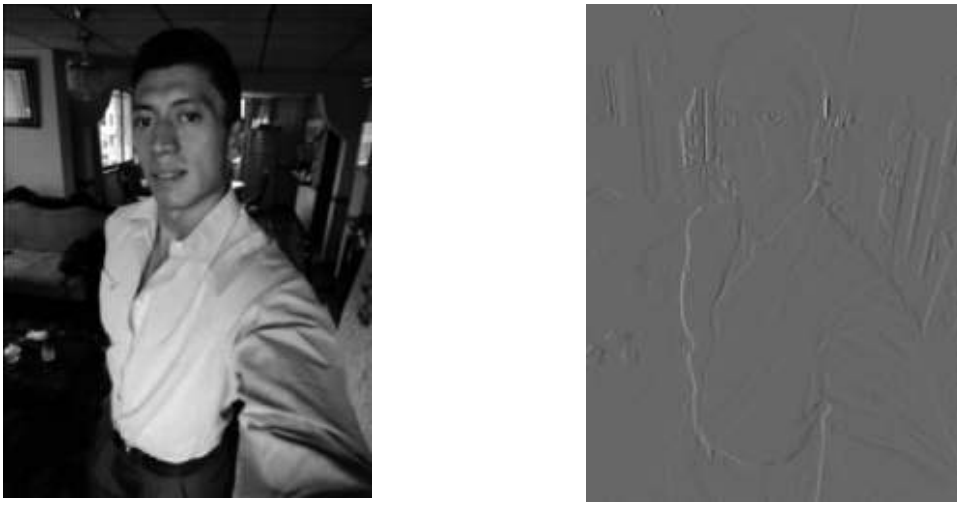
$$Scharr\ 5 \times 5_{45^0} = \frac{1}{60}\begin{bmatrix} -6 & -2 & -1 & -1 & 0 \\ -2 & -3 & -2 & 0 & 1 \\ -1 & -2 & 0 & 2 & 1 \\ -1 & 0 & 2 & 3 & 2 \\ 0 & 1 & 1 & 2 & 6 \end{bmatrix} \quad \text{Ec. [2.153]}$$

El código empleado para la detección de bordes mediante el filtro diferencial de primer orden de Scharr se presenta a continuación:

```
case 2 %Gradiente de Scharr

    kernel45scharr=[-10 -3 0;-3 0 3;0 3 10]/16; %Se genera la
    matriz del kernel a ser empleado

    for i=2:s(1)-1 %Se inicia un lazo for para las columnas de la
    imagen

        for j=2:s(2)-1 %Se inicia un lazo for para las filas de la
        imagen

          ventana=ar(i-1:i+1, j-1:j+1); %Se aísla la matriz que de
          la imagen 3x3 que será afectada por el kernel

          prod=ventana.*kernel45scharr; %Se multiplica la ventana
          por el laplaciano
```

```
            pix=sum(sum(prod));   %Se suman todos los valores
            obtenidos de la matriz

            arc(i,j)=(pix+255)/2; %Se sustituye cada pixel de la
            imagen por este nuevo valor

        end

    end
```

Los resultados que se obtienen a partir de la aplicación de un Kernel en dirección x empleando una máscara tipo Scharr de 5x5 para la imagen de la figura 2.38. se muestran en la figura 2.40., así como el histograma indicado en la figura 2.39.



(a) (b)

Figura 2.38. Ejemplo de la aplicación del gradiente Scharr 5x5 en dirección x sobre una imagen, (a) Imagen original en escala de grises, (b) Resultado obtenido mediante el filtro Scharr.

(Fuente: elaboración propia)

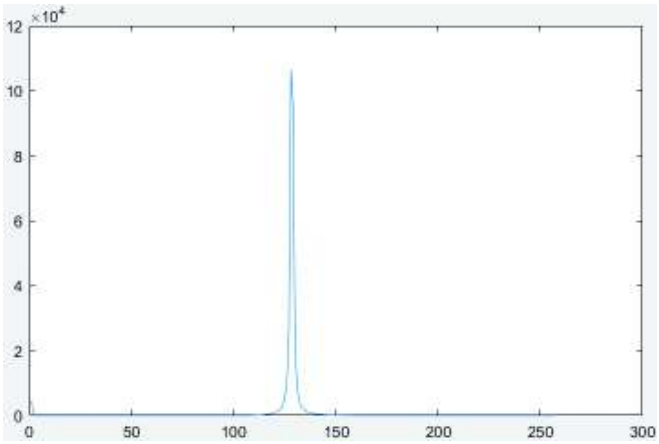
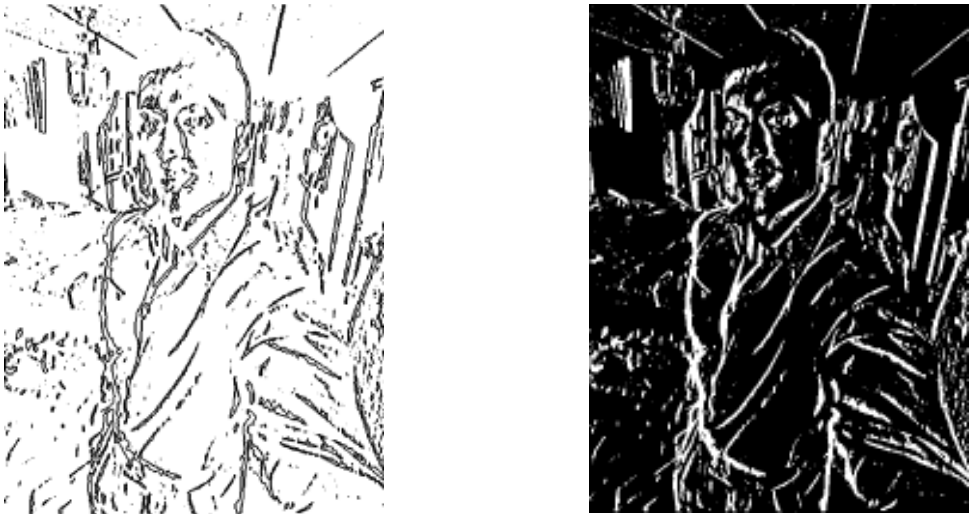


Figura 2.39. Histograma de frecuencias de la imagen ejemplo para el gradiente Scharr 5x5 en dirección x.

(Fuente: elaboración propia)

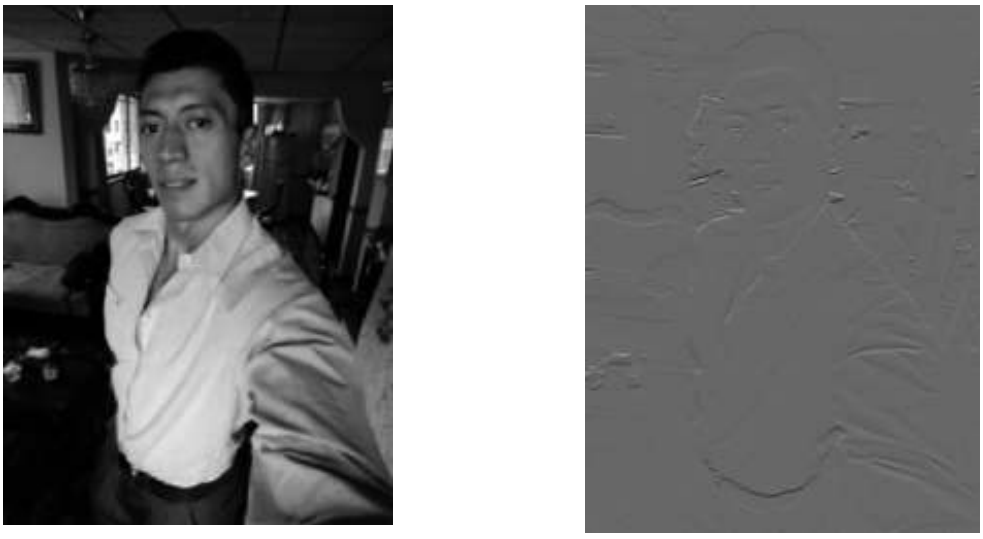


(a) (b)

Figura 2.40. Resultados obtenidos para el ejemplo de detección de bordes empleando el gradiente Scharr 5x5 en dirección x, (a) Bordes en negro (ceros), (b) Bordes en blanco (unos).

(Fuente: elaboración propia)

Los resultados que se obtienen a partir de la aplicación de un Kernel en dirección y empleando una máscara tipo Scharr de 5x5 para la imagen de la figura 2.41. se muestran en la figura 2.43., así como el histograma indicado en la figura 2.42.



(a) (b)

Figura 2.41. Ejemplo de la aplicación del gradiente Scharr 5x5 en dirección y sobre una imagen, (a) Imagen original en escala de grises, (b) Resultado obtenido mediante el filtro Scharr.

(Fuente: elaboración propia)

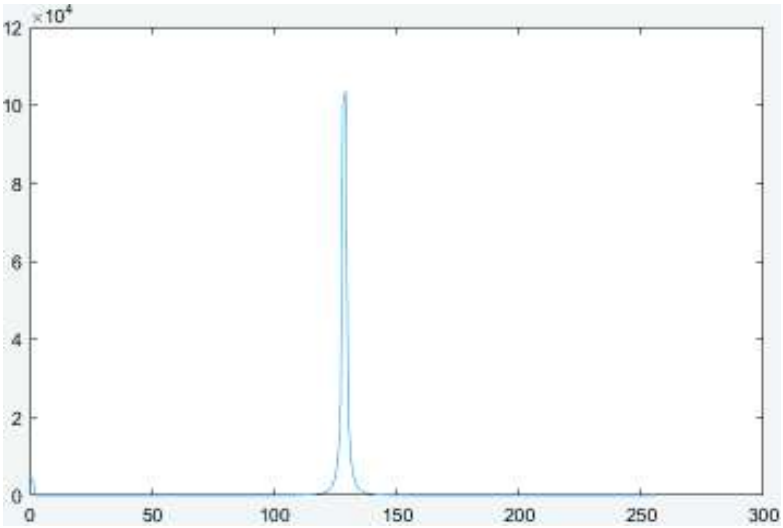
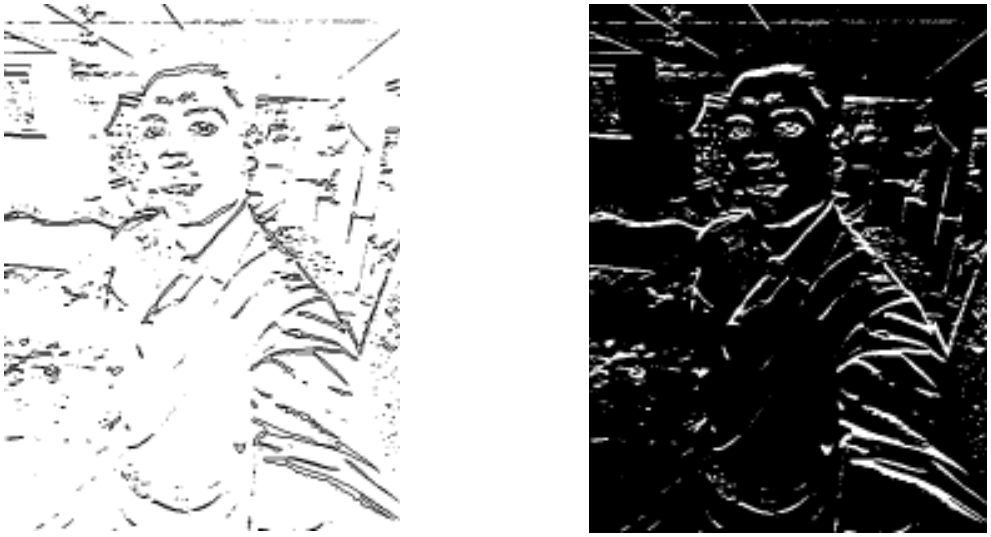


Figura 2.42. Histograma de frecuencias de la imagen ejemplo para el gradiente Scharr 5x5 en dirección y.

(Fuente: elaboración propia)

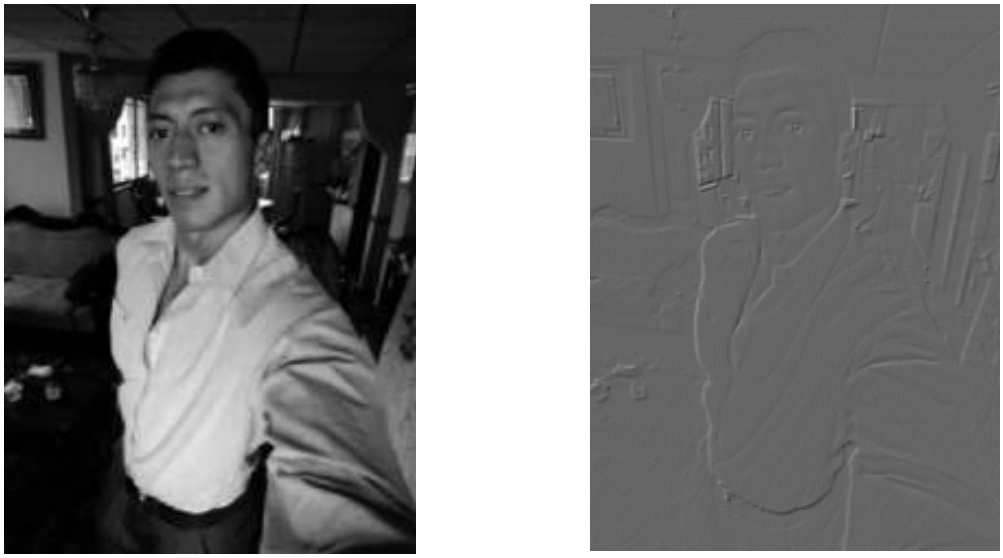


(a) (b)

Figura 2.43. Resultados obtenidos para el ejemplo de detección de bordes empleando el gradiente Scharr 5x5 en dirección y, (a) Bordes en negro (ceros), (b) Bordes en blanco (unos).

(Fuente: elaboración propia)

Los resultados que se obtienen a partir de la aplicación de un Kernel en dirección diagonal con un ángulo de 45° empleando una máscara tipo Scharr de 5x5 para la imagen de la figura 2.44. se muestran en la figura 2.46., así como el histograma indicado en la figura 2.45.



(a) (b)

Figura 2.44. Ejemplo de la aplicación del gradiente Scharr 5x5 en dirección diagonal 45º sobre una imagen, (a) Imagen original en escala de grises, (b) Resultado obtenido mediante el filtro Scharr.

(Fuente: elaboración propia)

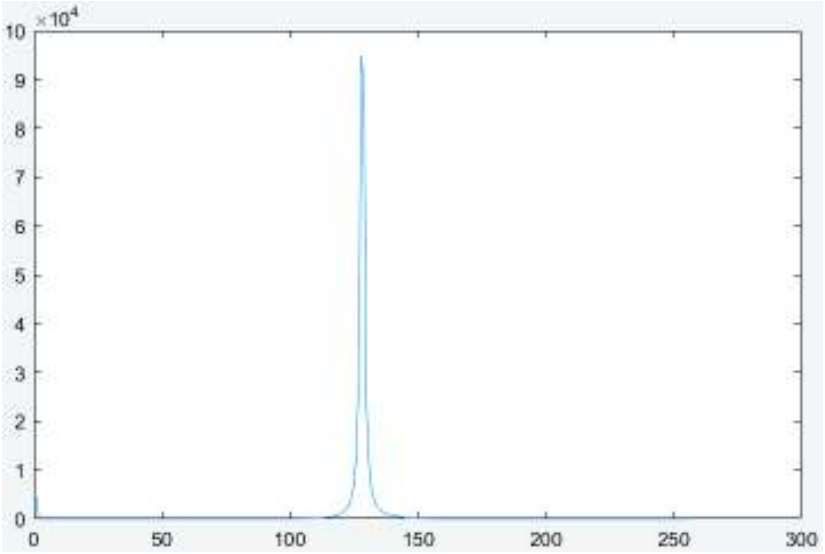


Figura 2.45. Histograma de frecuencias de la imagen ejemplo para el gradiente Scharr 5x5 en dirección diagonal 45º.

(Fuente: elaboración propia)

(a) (b)

Figura 2.46. Resultados obtenidos para el ejemplo de detección de bordes empleando el gradiente Scharr 5x5 en dirección diagonal 45º, (a) Bordes en negro (ceros), (b) Bordes en blanco (unos).
(Fuente: elaboración propia)

En las figuras 2.38. – 2.46. presentadas anteriormente se puede verificar el correcto funcionamiento del detector de bordes por convolución Scharr, y especialmente se puede corroborar los resultados previstos. Así que, para un filtro Scharr de orientación en x de la figura 2.40. se esperaba y se verificó que la detección se realiza de manera satisfactoria en dirección x, mientras que en la dirección y la detección no es buena. De la misma manera, para un filtro Scharr de orientación en y de la figura 2.43. se esperaba y se verificó que la detección se realiza de manera satisfactoria en sentido vertical, mientras que en la dirección horizontal la detección no es buena. Finalmente, se puede concluir gracias a los ejemplos anteriores, que en la aplicación particular de detección de rostros humanos, como en estos no predominan ni los componentes verticales ni horizontales, el filtro que entregó mejores resultados es el Kernel de Scharr en dirección diagonal a 45º mostrado en la figura 2.46., es por esto que, para el código de procesamiento de imágenes se empleó este Kernel en particular.

2.4.6.3. Caso 3: Gradiente Laplaciano con filtro Gaussiano

En todos los casos anteriores se puede visualizar la presencia de ruido, especialmente de tipo sal y pimienta, esto es, existe presencia dispersa de pixeles blancos y negros en la imagen . Esto se hace muy notorio en las figuras 2.37., 2.40., 2.43. y 2.46., lo que puede atribuirse en primera instancia a la calidad de imagen que se recibe a partir de la cámara,

especialmente en condiciones de escasa iluminación. Pero, se debe considerar que su efecto se incrementa en gran medida debido a la aplicación de los gradientes que permiten detectar los bordes de las imágenes, pero al ser diferenciales permiten identificar las discontinuidades en la tonalidad de los pixeles.

Las discontinuidades detectadas mediante los gradientes corresponden a los bordes de la imagen, que son los elementos deseados, pero también el ruido presente en los píxeles será detectado a manera de discontinuidad, de ahí su presencia en la imagen binarizada, e incluso es el motivo por el que se los puede hallar con mayor intensidad en la imagen resultante.

Es de especial interés eliminar estos falsos positivos de la imagen final, debido a que la presencia de este ruido en la imagen binarizada podría desembocar en una detección errónea de rostros, ya que estos falsos positivos podrían ser extraídos como características y alimentar al clasificador en el proceso de aprendizaje o permitir un matching erróneo.

Es por esto que, de ser necesario, y de visualizarse una gran cantidad de ruido en la imagen resultante presentada en la GUIDE desarrollada, esta interfaz brinda al usuario la alternativa de aplicar un método de suavizado, que para este caso, se seleccionó el filtro Gaussiano debido a su excelente desempeño apegándose a la metodología de Marr-Hildreth.

EL filtro Gaussiano es un mecanismo de convolución bidimensional para cada píxe que compone la imagen donde una función Kernel es aplicada sobre cada fragmento posible de la imagen de igual dimensión de acuerdo con la expresión de la ecuación 2.154.

$$f(x,y) * g(x,y) = \int_{\tau_1=-\infty}^{\infty} \int_{\tau_2=-\infty}^{\infty} f(\tau_1,\tau_2) \cdot g(x-\tau_1, y-\tau_2) d\tau_1 d\tau_2 \qquad \text{Ec. [2.154]}$$

$$f[x,y] * g[x,y] = \sum_{n_1} \sum_{n_2} f[n_1,n_2] \cdot g[x-n_1, y-n_2] \qquad \text{Ec. [2.155]}$$

Donde la ecuación 2.154 representa el proceso aplicado de manera discreta. Es así, que la función Gausiana bidimensional se encargará de suavizar el efecto de cambio de intensidad de los píxeles, teniendo especial efecto en los píxeles que tienen un cambio muy brusco en un espacio muy reducido (ruido sal y pimienta), distribuyendo este cambio de manera uniforme sobre los píxeles aldeanos mediante una función de distribución Gaussiana mostrada en la ecuación 2.156. En la figura 2.47. se pueden observar algunos ejemplos de funciones Gaussianas bidimensionales usadas para suavizar el efecto de cambio en los pixeles.

$$g(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \qquad \text{Ec. [2.156]}$$

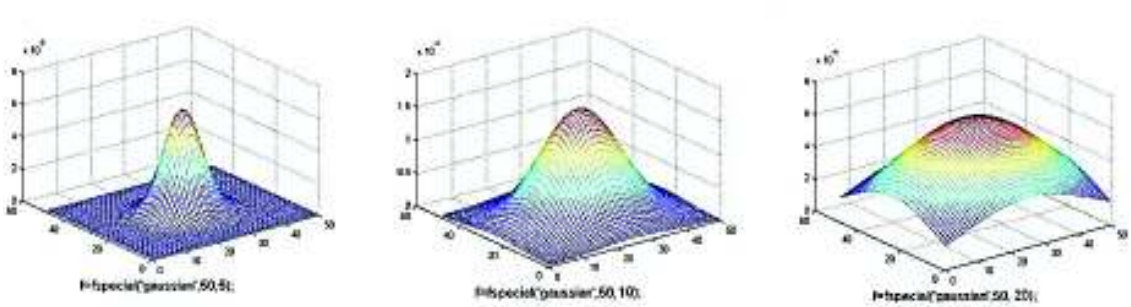


Figura 2.47. Ejemplos de funciones Gaussianas bidimensionales.

(Fuente: https://bit.ly/2GvqKSZ)

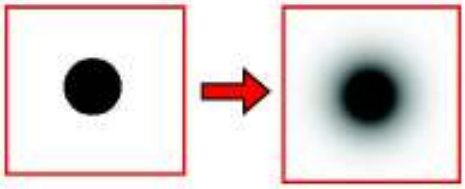


Figura 2.48. Efecto del suavizado por filtro Gaussiano.

(Fuente: elaboración propia)

De esta manera, se puede visualizar en la figura 2.48., que el Kernel Gaussiano al ser multiplicado por cada píxel contenido en una región del mismo tamaño, es suavizado. Los Kernels empleados para el desarrollo del código de procesamiento de imágenes se muestran en las ecuaciones 2.157 y 2.158.

Para una máscara de 5x5 de filtro Gaussiano con $\sigma = 1.0$:

$$KernelGauss_{5\times5} = \frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix} \qquad \text{Ec. [2.157]}$$

Para una máscara de 3x3 se tiene por ejemplo:

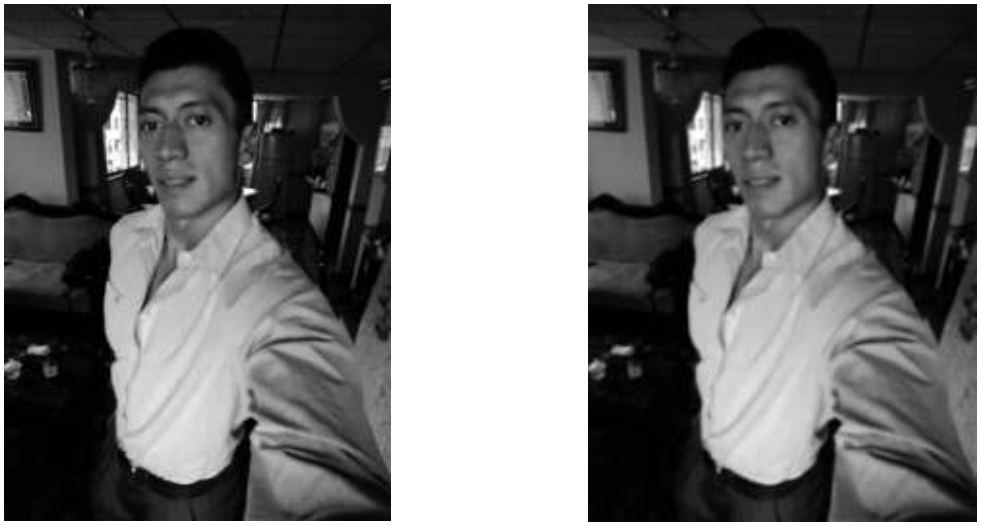
$$KernelGauss_{3\times3} = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \qquad \text{Ec. [2.158]}$$

Es así que, para efectuar la convolución mediante el filtro Gaussiano de 5x5 se tiene el siguiente código:

```
%Extracción de dimensiones

x=imread('erick2.png'); %Se lee el archivo de la imagen
```

```
y=imresize(x,0.5);

a=rgb2gray(y); %Se transforma la imagen a escala de grises

ar=double(a); %Se extiende la valor de cada pixel a 8 bytes

s=size(a); %Se obtiene las dimensiones de la matriz

arc=ar*0; %Se genera una matriz de ceros de las dimensiones de la
imagen

%Convolución Gaussiana

kernel=[1 4 7 4 1;4 16 26 16 4;7 26 41 26 7;4 16 26 16 4;1 4 7 4
1]/273; %Se genera el filtro Gaussiano y se divide para el total para
que obtenga un valor promedio;

for i=3:s(1)-2 %Se inicia un lazo for para las columnas de la imagen

    for j=3:s(2)-2 %Se inicia un lazo for para las filas de la imagen

      ventana=ar(i-2:i+2, j-2:j+2); %Se aísla la matriz que da la
      imagen 3x3 que será afectada por el kernel

      prod=ventana.*kernel; %Se multiplica la ventana por el kernel

      pix=sum(sum(prod)); %Se suman todos los valores obtenidos de
      la matriz

      arc(i,j)=pix; %Se sustituye cada pixel de la imagen por este
      nuevo valor

    end

end

figure(1),    subplot(3,2,1),    imshow(uint8(ar)),    title('Imagen
original')  %Se  presentan  los  resultados  de  la  imagen  original
respecto a la suavizada

figure(1), subplot(3,2,2), imshow(uint8(arc)), title('Imagen Filtro
Gausiano')
```

La aplicación de un filtrado Gaussiano en resumen permite suavizar el valor de cada píxel en función de los valores de los píxeles de los píxeles aldeanos como se puede apreciar en la figura 2.49.



(a) (b)

Figura 2.49. Ejemplo de aplicación de un suavizado por convolución Gaussiana empleando un Kernel de 5x5, (a) Imagen original, (b) Imagen suavizada.

(Fuente: elaboración propia)

Al visualizar la figura 2.49., se puede notar claramente el efecto que tiene la convolución Gaussiana sobre la imagen original, suavizando cada píxel de la imagen. Esto podría llevar a pensar que no es de ayuda en el proceso de detección de bordes debido a que se puede notar un efecto de disminución en la nitidez de la imagen. Sin embargo, si bien el proceso de suavizado puede desembocar en la cantidad de elementos detectados mediante el gradiente, los elementos más atenuados por el suavizado serán los correspondientes al ruido en la imagen, contribuyendo de esta manera a una detección más eficaz, disminuyendo en gran medida los falsos positivos.

Además, las bondades que brinda la aplicación de esta convolución son mucho más notorias en la imagen binarizada final. Es por esto que, al aplicar el case 3 de la GUIDE, se procederá a aplicar un suavizado Gaussiano en serie con un gradiente Laplaciano. Para poder llevar a cabo esta tarea se empleó el código que se detalla a continuación:

```
case 3 %Filtro Gausiano + Gradiente Laplaciano
      kv=[1;2;1]; %Se genera la componente vertical del kernel
      kh=[1 2 1]; %Se genera la componente horizontal del kernel
      kernel=kv*kh/16; %Se genera el filtro Gaussiano y se divide
      para el total para que obtenga un valor promedio;
      for i=2:s(1)-1 %Se inicia un lazo for para las columnas de la
      imagen
            for j=2:s(2)-1 %Se inicia un lazo for para las filas de
            la imagen
```

```
            ventana=ar(i-1:i+1, j-1:j+1); %Se aísla la matriz de
            la imagen 3x3 que será afectada por el kernel

            prod=ventana.*kernel; %Se multiplica la ventana por el
            kernel

            pix=sum(sum(prod)); %Se suman todos los valores
            obtenidos de la matriz

            arc(i,j)=pix; %Se sustituye cada pixel de la imagen
            por este nuevo valor

        end

    end

    arcc=zeros(s(1) , s(2)); %Se genera una matriz de ceros de las
    dimensiones de la imagen

    %Laplaciano

    lapl1=[1 1 1;1 -8 1;1 1 1]/8; % e genera el kernel Laplaciano

    for i=2:s(1)-1 %Se inicia un lazo for para las columnas de la
    imagen

          for j=2:s(2)-1 %Se inicia un lazo for para las filas de
          la imagen

            ventana=arc(i-1:i+1, j-1:j+1); %Se aísla la matriz de
            la imagen obtenida luego de aplicar el filtro Gaussiano
            3x3 que será afectada por el kernel

            prod=ventana.*lapl1; %Se multiplica la ventana por el
            laplaciano

            pix=sum(sum(prod)); %Se suman todos los valores
            obtenidos de la matriz

            arcc(i,j)=(pix+255)/2; %Se sustituye cada pixel de la
            imagen por este nuevo valor

        end

    end
```

Como se detalla en el código anterior, se aplica en secuencia el filtro Gaussiano, y una vez obtenida la imagen $arc(i,j)$ se ensambla píxel por píxel, a continuación para la segmentación Laplaciana se leen ventanas de dimensión 3x3 a partir de esta imagen, para luego de la convolución Laplaciana almacenarlas en una nueva imagen $arcc(i,j)$, misma que finalmente, mediante el histograma de frecuencias será umbralizada para obtener su resultado binarizado.

En este resultado para la imagen binarizada final, es donde se puede visualizar en mayor medida el efecto de eliminación de falsos positivos gracias al suavizado como se aprecia en la figura 2.50.

(a) (b)

Figura 2.50. Ejemplo de comparación de la detección de bordes por gradiente Laplaciano con o sin suavizado (a) solo gradiente Laplaciano, (b) suavizado Gaussiano con gradiente Laplaciano.

(Fuente: elaboración propia)

En la figura 2.50 se puede visualizar con claridad el efecto del filtro Gaussiano que elimina en gran medida el ruido presente en la imagen en comparación con una detección sin suavizado. Cabe destacar que así como se eliminan falsos positivos, también se generan falsos negativos de bordes que no fueron detectados con este método. Es precisamente por esto, que mediante la GUIDE de procesamiento de imágenes se deja al usuario seleccionar la configuración de convolución, ya que dependiendo de factores como la iluminación y el tipo de cámara que disponga el dron será o no necesario emplear un suavizado en la imagen.

2.4.6.4. Caso 4: Gradiente de Scharr con filtrado Gaussiano

Para este caso se propone realizar la convolución de la imagen ejecutando primero un filtrado Gaussiano que permita suavizar la imagen y eliminar en lo posible el ruido, para posteriormente obtener el gradiente de la imagen mediante un filtro diferencial de Scharr, y finalmente umbralizar el resultado, obteniendo así la imagen binarizada que contiene los bordes detectados de la imagen.

Para lograr este objetivo se desarrolló el siguiente código, que puede ser ejecutado mediante el case 4 a través de la listbox 1 de la GUIDE desarrollada. En este caso se

presenta la secuencia de código completa que permite visualizar el proceso de detección de bordes:

```
case 4 %Filtro Gaussiano + Gradiente Scharr
y=im; %Se lee el valor de cada píxel

a=rgb2gray(y); %Se transforma la imagen a escala de grises

ar=double(a); %Se extiende la valor de cada pixel a 8 bytes

s=size(a); %Se obtiene las dimensiones de la matriz

arc=ar*0; %Se genera una matriz de ceros de las dimensiones de la
imagen

%Convolución Gaussiana (suavizado)

kv=[1;2;1]; %Se genera el componente vertical del Kernel

kh=[1 2 1]; %Se genera el componente horizontal del Kernel

kernel=kv*kh/16; %Se genera el filtro Gaussiano y se divide para el
total para que obtenga un valor promedio;

for i=2:s(1)-1 %Se inicia un lazo for para el alto de la imagen

    for j=2:s(2)-1 %Se inicia un lazo for para el ancho de la imagen

        ventana=ar(i-1:i+1, j-1:j+1); %Se aísla la matriz que de la
        imagen 3x3 que será afectada por el Kernel

        prod=ventana.*kernel; %Se multiplica la ventana por el Kernel

        pix=sum(sum(prod)); %Se suman todos los valores obtenidos de
        la matriz

        arc(i,j)=pix; %Se sustituye cada pixel de la imagen por este
        nuevo valor

    end

end


%% Detector de bordes Scharr

arcc=zeros(s(1)  ,  s(2)); %Se genera una matriz de ceros de
lasdimensiones de la imagen

%Scharr
kernel45scharr5x5=[-6 -2 -1 -1 0;-2 -3 -2 0 1;-1 -2 0 2 1;-1 0 2 3
2;0 1 1 2 6]/60; %Se genera el kernel de Scharr

for i=3:s(1)-2 %Se inicia un lazo for para el alto de la imagen

    for j=3:s(2)-2 %Se inicia un lazo for para el ancho de la imagen

        ventana=arc(i-2:i+2, j-2:j+2); %Se aísla la matriz que de la
```

```
        imagen 3x3 a partir de la imagen suavizada por el filtro
        Gaussiano que será afectada por el kernel

        prod=ventana.*kernel45scharr5x5; %Se multiplica la ventana
        por el Laplaciano

        pix=sum(sum(prod)); %Se suman todos los valores obtenidos de
        la matriz

        arcc(i,j)=(pix+255)/2; %Se sustituye cada pixel de la imagen
        por este nuevo valor

    end

end


%% Obtención de histograma

arrcc=uint8(arcc); %Se transforma el resultado obtenido a enteros
histR=zeros(1,256); %Se genera una matriz de ceros en una fila para
acumular las frecuencias del histograma

for f = 1:s(1)

    for c = 1:s(2)

        ngR=arrcc(f,c); %Se obtiene el nivel de gris de cada pixel
        en la matriz

        histR(ngR+1)=histR(ngR+1)+1; %Se acumula un valor de 1 en el
        histograma si el pixel coincide

    end
end
figure(3), plot(histR), title('Histograma') %Se presenta el
resultado del histograma

Prob=histR/(s(1)*s(2)); %Se transforma los valores del histograma a
probabilísticos (0 - 1)

A=Prob; %Se define una copia del histograma Probabilístico

T=find(A==max(max((Prob)))); %Se obtiene la posición del valor de
luminosidad donde se encuentra la saturación para los bordes

Tprob=(T+1)/256; %Se calcula el threshold en porcentaje decimal

I=uint8(arcc); %Se transforma la imagen detectada los bordes en
valores enteros

Ib=im2bw(I,Tprob); %Se binariza la imagen empleando el trhreshold
calculado mediante el histograma

EE=strel('disk',3); %Se crea un elemento disco de grosor 2 para dar
el grosor deseado a los bordes

Ie=imerode(Ib,EE); %Se expande el grosor de los bordes mediante la
resta de la imagen original menos la erosionada
```

```
Ip=abs(Ib-Ie-1); %Se genera el gradiente morfológico porque en el
borde se produce el máximo cambio de contraste

imshow(Ip); %Se muestra el resultado
```

En el código anterior, se puede visualizar la secuencia completa de operaciones a realizar para la detección de bordes que comprende: la adquisición del frame, el suavizado, la obtención del gradiente de la imagen y la umbralización del resultado, para poder obtener como finalmente la imagen binarizada.

En la figura 2.51. se puede observar el resultado obtenido y su comparación con el método de Scharr con o sin suavizado. Como se puede visualizar, la aplicación de un suavizado ratifica la eliminación en gran mediad de los falsos positivos generados a partir del ruido en la imagen, como era de esperarse.

(a) (b)

Figura 2.51. Comparación entre los métodos de filtrado aplicando el Kernel Scharr con o sin suavizado, (a) solo gradiente de Scharr, (b) gradiente de Scharr con suavizado Gaussiano.

(Fuente: elaboración propia)

De los ejemplos anteriores se puede inferir que la aplicación de un suavizado resulta conveniente, sin embargo, se debe considerar que el uso de una convolución adicional sobre la imagen implica la adición de una enorme cantidad de operaciones píxel a píxel que el ordenador debe realizar. Por lo tanto, la aplicación del suavizado Gaussiano se debe evitar en lo posible, y solo efectuarse donde sea necesario, ya que su uso implica un gran incremento en el costo computacional del procesamiento de la imagen.

2.4.7. Detección de bordes mediante el algoritmo de Canny

Matlab mediante su toolbox de procesamiento de imágenes ofrece la alternativa de extraer los bordes de la imagen empleando el algoritmo de Canny mediante la función “edge”. Este algoritmo funciona de manera similar al método antes descrito, pero con la diferencia que aplica un filtro Gaussiano y la obtención de los gradientes lo realiza en varias etapas y direcciones horizontal y vertical mediante el uso de las ecuaciones 2.159 – 2.161.

$$KernelGauss_{5\times5} = \frac{1}{159}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix} \qquad \text{Ec. [2.159]}$$

$$G = \sqrt{{G_x}^2 + {G_y}^2} \qquad \text{Ec. [2.160]}$$

$$\Theta = \arctan\left(\frac{G_x}{G_y}\right) \qquad \text{Ec. [2.161]}$$

Este método se puede implementar mediante un código muy sencillo en la interfaz de Matlab, pero no fue implementado como alternativa relevante en el presente proyecto, ya que la cantidad de falsos negativos que presenta es demasiado elevada respecto a los métodos antes citados, y su costo computacional es similar, ya que aunque no se visualiza directamente, se ejecuta en segundo plano. El código se muestra a continuación:

```
%Detector de bordes Canny

    axes(handles.axes6); %Se destina la proyección de los resultados al
    axes 6

    Im1=rgb2gray(im1); %Se convierte la imagen en binaria

    Im2=edge(Im1,'Canny'); %Se aplica el detector de bordes Canny

    imshow(Im2); %Se muestra el resultado
```

El resultado de la extracción de bordes se muestra en la figura 2.52.

Figura 2.52. Detección de bordes mediante el algoritmo de Canny.

(Fuente: elaboración propia)

2.4.8. Visualizador de Matching Points

En esta sección se presenta el código empleado en la programación de la GUIDE, que muestra en tiempo real el proceso de extracción de características a partir de la posición de la característica extraída de la imagen y ploteado sobre ella a manera de puntos.

Esta función al igual que la próxima, tienen por objetivo mostrar gráficamente al usuario que el correcto seteo del threshold permite una extracción de características más enfocada al rostro, que es lo que se desea encontrar en la imagen, y además, muestra al usuario como este valor umbral permite reducir el área de aprendizaje y extracción de información a una mucho más particularizada y pequeña. Esto permite reducir el costo computacional y mejorar en general el resultado del aprendizaje y la detección.

El código desarrollado para la visualización del Matching, es el siguiente:

```
%Visualizador de matching points

% Imagen de la base de datos

I=imread('C:\Users\Erick
Herrera\Documents\MATLAB\data\base\00001.jpg'); %Se carga la foto que
se tomó y almacenó en la base de datos

im5=imsubtract(I(:,:,1),rgb2gray(I)); %Se extrae el componente rojo de
la matriz RGB

im5=medfilt2(im5,[3,3]); %Se usa filtro median para reducir ruido
```

```
im5=im2bw(im5,gtr); %Se convierte la imagen de la matriz en escala de
grises a binaria

im5=bwareaopen(im5,300); %Se remueven los pixeles cuya frecuencia sea
menor o igual a 300

I1=im5; %Se almacena la imagen en una nueva variable

% Frame del video

diff_im=imsubtract(im(:,:,1),Im1); %Se extrae el componente rojo de la
matriz RGB de la imagen actual capturada

diff_im=medfilt2(diff_im,[3,3]); %Se usa un filtro median para reducir
ruido

diff_im=im2bw(diff_im,gtr); %Se convierte la imagen de la matriz en
escala de grises a binaria empleando el threshold seteado previamente

diff_im=bwareaopen(diff_im,157); %Se remueven los pixeles cuya
frecuencia sea menor o igual a 300

I2=diff_im; %Se le da un nuevo nombre a la variable

points1=detectSURFFeatures(I1); %Se detecta las características
principales de la foto inicial tomada

points2=detectSURFFeatures(I2); %Se detectan las características
principales del frame extraido en tiempo real

[features1,vpts1]=extractFeatures(I1,points1); %Se extraen las
características de la foto de la base de datos

[features2,vpts2]=extractFeatures(I2,points2); %Se extraen las
características del frame capturado video

indexPairs=matchFeatures(features1,features2); %Se comparan e
identifican las coincidencias entre las dos imágenes

matchedPoints1=points1(indexPairs(:,1),:); %Se ubican las
coincidencias de la primera imagen a manera de puntos

matchedPoints2=points2(indexPairs(:,2),:); %Se ubican las
coincidencias de la segunda imagen (video) a manera de puntos

axes(handles.axes5); %Se destina el axes 5 para presentar los
resultados

imshow(im); %Se muestra el video

hold on %Se mantiene el frame para montar sobre el los puntos que
coinciden

plot(matchedPoints2) %Se plotean los puntos sobre la imagen
```

Los resultados obtenidos gracias al proceso de matching en tiempo real se visualizan en el axes 5 como se muestra en la figura 2.53.

(a) (b)

Figura 2.53. Ejemplo de visualización del proceso de matching y extracción de características mediante la GUIDE, (a) Imagen binarizada aplicado un umbral de binarización, (b) Representación gráfica de las posiciones de donde se extraen las características para el aprendizaje (matching points).

(Fuente: elaboración propia)

2.4.9. Visualizador de resultados

Finalizado el proceso de procesamiento de las imágenes, cabe mencionar que el motivo por el que se realizó todo este procesamiento es reducir el área de análisis de la imagen que se obtiene a partir de la cámara del dron a un área mucho menor a partir de donde se extraerán las características para el aprendizaje y la posterior detección del objetivo. Además, teniendo en cuenta que el algoritmo de detección tiene un elevado costo computacional, y sumado a esto el algoritmo de control, implican una tarea muy compleja para el ordenador.

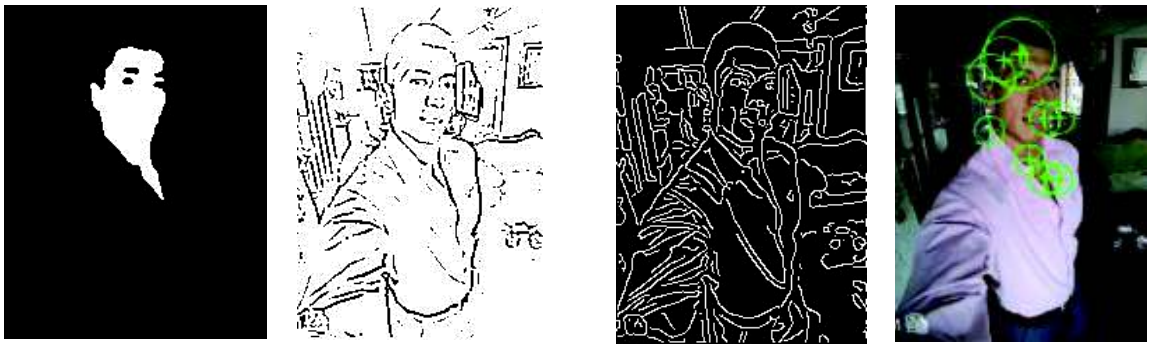
Es así, que gracias al procesamiento adecuado, mediante una correcta configuración del valor umbral del cual se extrae la imagen binarizada, y un procesamiento conveniente que permite una correcta extracción de bordes se logra delimitar un área de análisis menor y gracias a ella el costo computacional de la detección del objetivo será mucho menor y más efectivo.

Entonces, finalmente se ofrece al usuario una interfaz gráfica donde podrá visualizar en tiempo real el resultado obtenido del área sobre la cual se está realizando el procesamiento y la extracción de características. Para lograr esto se desarrolló el siguiente código que ejecutará el resultado en el elemento axes 7 de la GUIDE.

```
%Visualizador de resultados

axes(handles.axes7); %Se destina el axes 7 como elemento donde se
visualizará el resultado

imshow(im); %Se proyecta cada frame original del video
```

```
hold on; %Se mantiene esta imagen para plotear el resultado sobre
ella

bw=bwlabel(I2,8); %Se crean regiones rectangulares alrededor del
resultado del matching

stats=regionprops(logical(bw),'BoundingBox','Centroid');        %Se
analiza cada característica (feature) extraída y se obtiene su
centroide y propiedades

for object=1:length(stats); %Se inicia un lazo for para crear los
recuadros para cada una de las características extraídas

      bb=stats(object).BoundingBox; %Se llama a la región encontrada
      de la característica y se guarda en bb (origen, ancho, largo)
      del rectángulo

      bc=stats(object).Centroid; %Se guarda el valor del centroide
      en bc

      rectangle('Position',bb,'EdgeColor','r','LineWidth',2)     %Se
      dibujan los rectángulos con los datos de las variables bb y bc

      plot(bc(1),bc(2),'-m+')  %Se plotean los rectángulos en la
      salida

      pers=get(handles.text9,'String'); %Se obtiene el nombre de la
      persona ingresado en edittext 1 para la base de datos

      text(bc(1),bc(2),pers,'Color','r','FontSize',14);  %Se plotea
      el nombre de  la persona identificada sobre cada boundingbox

end
```

Los resultados obtenidos para reducir el área de análisis del rostro a partir del procesamiento adecuado de la imagen mediante el código realizado se puede visualizar en el ejemplo de la figura 2.54.



(a) (b) (c) (d)

Figura 2.54. Ejemplo de resultados obtenidos para el proceso de reducción de la imagen mediante el código de procesamiento de imágenes, (a) Configuración del umbral, (b) Detección de bordes gradiente Laplaciano, (c) Detección de bordes algoritmo de Canny, (d) Matching de características.

(Fuente: elaboración propia)

En la figura 2.55. se puede visualizar la importancia del procesamiento de imágenes ya que el área de extracción de características se ve considerablemente reducida y orientada a la detección facial, produciendo excelentes resultados para el presente proyecto.

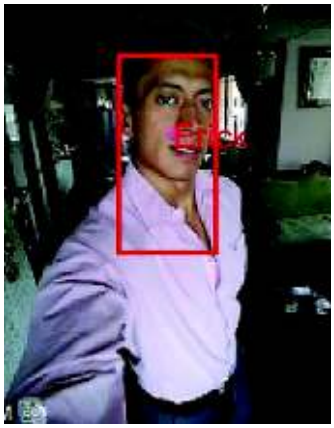


Figura 2.55. Resultado final del área delimitada mediante el código de procesamiento, BoundingBox y nombre para la base de datos.
(Fuente: elaboración propia)

2.4.10. Simulador de vuelo

En la GUIDE desarrollada adicionalmente se proporcionó el elemento puschbutton12, que está etiquetado como “Simulador de vuelo”, que permite al usuario visualizar la correcta adquisición de la información de los sensores de altitud, giroscopio, magnetómetro y acelerómetro que dispone el dron a través de la dirección TCP mediante la cual se comunica.
Entonces, al presionar este pushbutton se genera la orden de ejecutar una nueva GUIDE de Matlab, que presenta un único axes donde se ejecutará la función “realtimeAltitudeViewer”.

```
addpath('./ARDroneMatlabAPI');
realtimeAltitudeViewer(10);
```

Finalmente, el código que ejecuta el simulador de vuelo se obtuvo mediante el menú de Add-Ons de Matlab, buscando el modelo del dron Parrot. La función se obtiene mediante el nombre “AR Drone ARDrone_Matlab_V1” donde se incluyen varias librerías mediante las cuales se puede obtener los valores de los sensores, y a su vez permiten cargar controladores al procesador del dron, por ejemplo. El código para el simulador de vuelo se detalla a continuación:

```
% function realtimeAltitudeViewer (Duration_in_second)
%
% This function opens the UDP ports to the drone, inquires drone
states, and then plots the Drone attitude in a 3-D model.
% Users can use the function to test connection between drone and
local computer.
%
%
% ************************************
% *  Authors:
%    Kun Zhang (dabiezu@gmail.edu)
%    Pieter J. Mosterman (pmosterman@yahoo.com) *
% ************************************


function realtimeAltitudeViewer (Duration_in_second)

if Duration_in_second <=0

    return

end

controlChannel = udp('192.168.1.1', 5556, 'LocalPort', 5556);

stateChannel = udp('192.168.1.1', 5554, 'LocalPort', 5554);

try

    fopen(controlChannel);

    fopen(stateChannel);

catch excp

    disp('failed to open udp channels.');

    disp(excp.message)

    return

end

try

    SequenceNumber = tic;

    t_ = 0;

    t_0 = clock;

    while(t_<Duration_in_second);
        [~, ~, SequenceNumber] = Ask4DroneState (SequenceNumber,
        controlChannel, stateChannel, 1);
        t_ = etime(clock,t_0);

    end


catch excp

    disp(excp.message)
```

```
        fclose(controlChannel);
        fclose(stateChannel);
    end
    fclose(controlChannel);
    fclose(stateChannel);
```

EL resultado obtenido es una gráfica tridimensional que muestra en tiempo real la altitud y orientación del dron a manera de simulación como se puede visualizar en la figura 2.56.

Figura 2.56. Simulador de vuelo del dron, aplicando la función “realtimeAltitudeViewer” para el Parrot AR. 2.0.

(Fuente: elaboración propia)

2.5. Desarrollo del Algoritmo de reconocimiento del objetivo

En esta sección cabe destacar que mediante el procesamiento de imágenes previo ya se tiene una excelente detección del objetivo, por lo que para aplicaciones más simples que el objeto del estudio de este proyecto, se podría emplear la metodología previa y de esta manera reducir el costo computacional.

Pero tomando en cuenta que el dron es un sistema complejo de control, y que necesita un rango elevado de precisión para las acciones de control que se tomarán sobre el mismo se propone aplicar un último filtro en la detección del objetivo, ya que hasta el momento se puede tener aún varios falsos positivos, ya que la detección previa se basó en colores (tonalidad del rostro humano) y en bordes. Estos elementos podrían también estar presentes en objetos del entorno donde se efectúa la detección, generando de esta manera falsos positivos, que podrían desestabilizar el vuelo del dron.

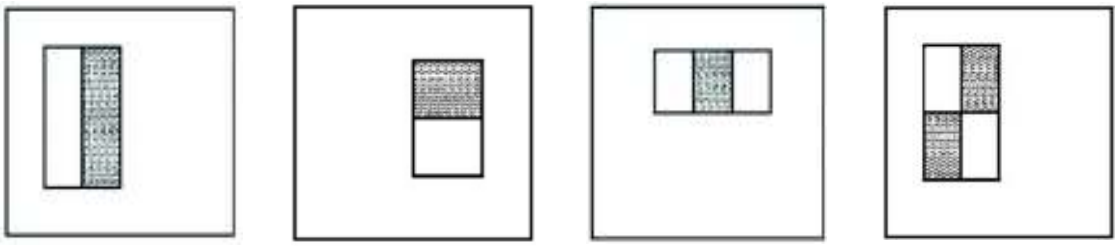
Es así, que se empleó un último elemento de detección, que es el "vision.CascadeObjectDetector" del toolbox Computer Vision System de Matlab, mismo que está basado en el algoritmo de Viola Jones para detección facial. Este mecanismo se empleó por encima de otras alternativas existentes, como lo son Eigenfaces, Fisherfaces, entre otros, debido a que es el método que realiza la operación de detección facial de manera más veloz y con resultados igualmente satisfactorios debido a que emplea clasificadores simples que no requieren mucho costo computacional, que si bien no son muy efectivos por si solos al aplicarse en cascada logran obtener un excelente resultado. Este algoritmo se basa en tres ideas principales que son:

- La representación de la imagen de una nueva manera conocida como imagen integral.
- La aplicación de clasificadores simples pero eficientes empleando el algoritmo de aprendizaje AdaBoost.
- La utilización de un método de aplicación combinada de los clasificadores en cascada que permite eliminar rápidamente las regiones del fondo de la imagen para conservar solo las correspondientes a un rostro.

Las características que emplea este algoritmo para el aprendizaje son funciones básicas tipo Haar, que emplean las operaciones de tipo:

- Característica de dos rectángulos, diferencia entre la suma de los valores de los píxeles de dos regiones rectangulares.
- Característica de tres rectángulos, suma entre dos rectángulos externos restados de la suma del rectángulo central.
- Característica de cuatro rectángulos, diferencia entre parejas de rectángulos en diagonal

Las funciones de tipo Haar mencionadas anteriormente se muestran en la figura 2.35.



(a) (b) (c) (d)

Figura 2.35. Funciones tipo Haar empleadas en el algoritmo Viola Jones, (a) Función de dos rectángulos horizontal, (b) Función de dos rectángulos vertical, (c) Función de tres rectángulos, (d) Función de cuatro rectángulos.

(Fuente: https://bit.ly/2IqXEoC)

La obtención de características mediante este método, da lugar a tener un elevado número de características, ya que por ejemplo para una imagen de tan solo 384 x 288 píxeles, teniendo en cuenta que la resolución de este detector es en regiones de 24 x 24 píxeles, se tiene un total de 160000 características, es por esto, que se requiere de un método eficiente para procesarlas y un buen método para clasificarlas que en este caso es AdaBoost.

2.5.1. Imagen integral

La obtención de la imagen integral compete a una metodología de cómputo de la imagen para facilitar un procesamiento veloz de las características. La imagen integral en una localización (x, y) contiene la suma de los píxeles de ella y los que se encuentran a la izquierda y por debajo del área definida. Esto obedece a la función y a la pareja de recurrencias que se detallan en las ecuaciones 2.162 – 2.164.

$$ii(x,y) = \sum_{x' \leq x, y' \leq y} i(x', y') \qquad \text{Ec. [2.162]}$$

$$s(x,y) = s(x, y-1) + i(x,y) \qquad \text{Ec. [2.163]}$$

$$ii(x,y) = ii(x-1, y) + s(x,y) \qquad \text{Ec. [2.164]}$$

Donde $s(x, y)$ es la suma acumulativa de las filas. Es así, que la imagen integral puede ser procesada al pasar sobre la imagen original, como se muestra en la figuras 2.57. y 2.58.

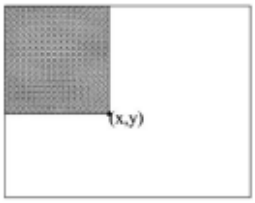


Figura 2.57. Obtención de la imagen integral en una ubicación (x,y).
(Fuente: https://bit.ly/2IqXEoC)

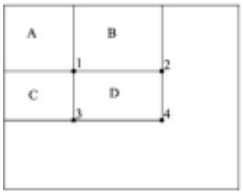


Figura 2.58. Elementos que participan en la suma de píxeles para obtener la imagen integral.
(Fuente: https://bit.ly/2IqXEoC)

Por ejemplo para la obtención de la nueva suma para el elemento D de la imagen integral mostrada en la figura 2.58. se obtienen los valores:

- 1 que es la suma de los píxeles del rectángulo A.
- 2 que es la suma de los píxeles de los rectángulos A + B.
- 3 que es la suma de las regiones A + C.
- 4 que es la suma de los píxeles A + B + C + D.

De esta manera la suma para D se puede obtener como: $4 + 1 - (2 + 3)$.
Para formar los clasificadores se combinan estas características, pero muy pocas de estas características pueden ser combinadas para formar un clasificador efectivo. Es así, que se procede a encontrar estas características.

2.5.2. AdaBoost

AdaBoost es un algoritmo de impulso desarrollado para la clasificación binaria a partir de una serie de clasificadores débiles, por lo que es considerado un clasificador fuerte. Esto lo logra construyendo un modelo a partir de datos de entrenamiento, para luego crear un segundo modelo que corregirá los errores del primer modelo. Estos modelos se siguen creando hasta que un modelo sea capaz de predecir perfectamente los datos de salida.

Por ejemplo las primeras dos características seleccionadas por el algoritmo AdaBoost para detección facial son obtenidas como se muestra en la figura 2.59.

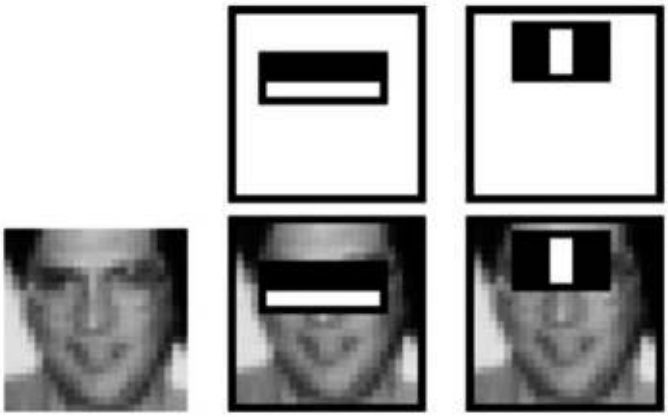


Figura 2.59. Obtención de las primeras dos características para el clasificador algoritmo AdaBoost.
(Fuente: https://bit.ly/2IqXEoC)

La primera característica obtenida mediante dos rectángulos muestra que la región de los ojos es más oscura que la región de las mejillas y nariz. Mientras la segunda característica obtenida a partir de la función Haar de tres rectángulos, muestra que los ojos son más oscuros que el puente de la nariz.

A continuación, el algoritmo Viola Jones aplica los clasificadores en cascada, de manera que los clasificadores más simples son los que se usan primero y se encargan de eliminar la mayoría de subregiones de la imagen, por ejemplo, el primer clasificador podría encargarse de eliminar el 50% de los elementos que no son rostros. Mientras los clasificadores más complejos se emplearán posteriormente con el objetivo de reducir falsos positivos. Este algoritmo logra tener gran eficacia, ya que está armado por capas, y no necesariamente necesita entrar a las capas complejas para obtener buenos resultados. Este principio se muestra en la figura 2.60.

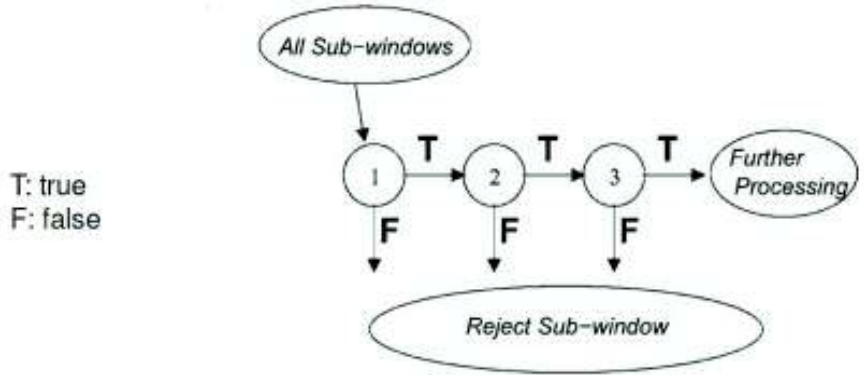


Figura 2.60. Mecanismo de aplicación de los clasificadores en cascada.
(Fuente: https://bit.ly/2IqXEoC)

2.5.3. Algoritmo de entrenamiento del detector de rostros

El motivo principal por el que se desarrolló la GUIDE de procesamiento de imágenes es debido a que previamente al entrenamiento del clasificador en cascada mediante la función "vision.CascadeObjectDetector" se necesitan cuatro elementos que son:

- Una base de datos de las imágenes que contienen los rostros de la persona a ser detectada.
- Un archivo de variable ".mat" que contenga las bounding boxes donde se encuentra localizado el rostro de la persona en cada imagen.

- Una base de datos de imágenes que no contengan el rostro a ser detectado para que el clasificador pueda aprender a partir de la no generación de falsos positivos con elementos que no contengan rostros.
- Un archivo en formato ".xml" que contenga el método que va a combinar los clasificadores en cascada, para que los más simples vayan primero, y los más complejos después como se mencionó en la sección anterior.

Es así, que el toolbox de visión artificial de Matlab presenta herramientas como la "Image Labeler app" que brindan una interfaz donde el usuario puede ir seleccionando una por una en cada imagen el área donde se encuentra un rostro mediante un elemento BoundingBox e irlas guardando. Pero como en la aplicación que se desarrolla para el presente proyecto, se dispone de una enorme cantidad de imágenes que constituyen cada frame, que se adquieren en velocidades de hasta 30 frames por segundo, de esta manera se propone agilizar este proceso al máximo volviéndolo automático.

2.5.4. Creación de la base de datos

Para generar la base de datos, las imágenes procesadas en la GUIDE de ejecución y las BoundingBox que se generaron para mostrar la ubicación del rostro del objetivo serán guardadas de manera automática en la carpeta Rostros, y el archivo "RostrosDetectados.mat" respectivamente. Para poder llevar a cabo esta tarea se desarrolló el siguiente código que mediante el pushbutton "Crear base de datos" permite crear una base de 100 imágenes y 100 BoundingBox donde se encuentra el rostro de manera automática.

Primero en el visualizador de resultados se define como global al elemento bb (BoundingBox), para que pueda ser llamado mediante la callout del pushbutton que genera la base de datos, así:

```
%Visualizador de resultados

global bb im %Primero se definen los elementos BoundingBox e im
(imagen procesada) como global para que puedan ser llamados en el
callout que genera la base de datos

axes(handles.axes7); %Se destina el axes 7 como elemento donde se
visualizará el resultado

imshow(im); %Se proyecta cada frame original del video

hold on; % e mantiene esta imagen para plotear el resultado sobre
ella
```

```
bw=bwlabel(I2,8); %Se crean regiones rectangulares alrededor del
resultado del matching

stats=regionprops(logical(bw),'BoundingBox','Centroid');        %Se
analiza cada característica (feature) extraída y se obtiene su
centroide y propiedades

for object=1:length(stats); %Se inicia un lazo for para crear los
recuadros para cada una de las características extraídas

      bb=stats(object).BoundingBox; %Se llama a la región encontrada
      de la característica y se guarda en bb (origen, ancho, largo)
      del rectángulo

      bc=stats(object).Centroid; %Se guarda el valor del centroide
      en y bc

      rectangle('Position',bb,'EdgeColor','r','LineWidth',2)    %Se
      dibujan los rectángulos con lo datos de las variables bb y bc

      plot(bc(1),bc(2),'-m+')  %Se plotean los rectángulos en la
      salida

      pers=get(handles.text9,'String'); %Se obtiene el nombre de la
      persona ingresado en edittext 1, para la base de datos

      text(bc(1),bc(2),pers,'Color','r','FontSize',14);  %Se plotea
      el nombre de  la persona identificada sobre cada boundingbox

end
```

A continuación, se procede a crear la base de datos de las 100 imágenes que contienen los rostros en la carpeta Rostros, y el archivo tipo array que contiene las 100 BpundingBox que delimitan la posición exacta de cada rostro dentro cada una de las cien imágenes.

```
global k bb im  %Se define k como global para que pueda ser activada
desde la GUIDE para  k=1 desde el pushbutton1, se llama a las
variables bb e im para emplearlas en este calluot

k=0; %Si k es igual a cero no crea la base de datos mientras el
procesamiento no esté configurado

n=1; %Se define un valor inicial de 1 para que pueda iniciar el
almacenado de las imágenes y BoundingBox

if k==1  %Se emplea el swuitch lógico k para que se genere la base
de datos solo al activarse el pushbutton respectivo

      if n<101 %Se define que el almacenado de la base de datos se
      efectúe hasta tener 100 elementos, se podría aumentar a bases
      de datos más numerosas en esta sección

            rostroprocesado(n,1:2)=bw; %Se almacena la bounding box
            del rostro detectado a partir de la GUI anterior en cada
            fila del array rostroprocesado

            imwrite(im,strcat('C:\Users\Erick
            Herrera\Documents\MATLAB\data\Rostros\',num2str(n),'.jp
```

```
                g')); % Seguarda cada imagen mediante la que  se obtuvo
                el BoundingBox en la carpeta rostros con el mismo nombre
                de la BoundingBox

            end
            n=n+1; %Se agrega el valor más uno para almacenar el siguiente
            elemento de la base de datos

        if  n==100  %Se  crea  un  switch  lógico  para  que  la  variable
        RostrosDetectados  se  guarde  solo  cuando  el  array  tenga  100
        elementos

            save('RostrosDetectados.mat','rostroprocesado'); %Se guarda
            la variable tipo array que contiene las localización de los
            100 rostros por medio de las 100 BoundingBox

        end

    end
```

El resultado obtenido mediante este código es una base de datos creada de 100 imágenes en la carpeta Rostros, y un archivo ".mat" que contiene las 100 BoundingBox de la ubicación de los rostros en las respectivas imágenes. Un ejemplo de los resultados obtenidos se muestra en la figura 2.61.

Figura 2.61. Ejemplo de base de datos para las imágenes creadas automáticamente para el entrenamiento del clasificador mediante el algoritmo desarrollado.

(Fuente: elaboración propia)

Adicionalmente, ya que el entrenamiento del detector en cascada llamado "vision.CascadeObjectDetector" requiere de un archivo de las BoundingBox, la base de datos de imágenes que contienen el rostro, y una base de datos que no contenga a los rostros para poder comparar con ella. Para generar este último elemento, se lo puede realizar introduciendo manualmente imágenes cualesquiera de las mismas dimensiones y

resolución que las imágenes de la base de datos Rostros, o generándolas automáticamente empleando la cámara del dron empleando un código de programación.
Cabe mencionar que estas imágenes estarán destinadas al aprendizaje del clasificador, ya que una vez entrenado mediante las imágenes de la base de datos “Rostros” podrá verificar los falsos negativos (rostros que no fueron detectados), mientras que mediante las imágenes de esta nueva base de datos “NoRostros”, el clasificador podrá determinar la aparición de falsos negativos (elementos que no eran rostros y fueron detectados). Además, se debe considerar que para que este proceso sea efectivo durante la creación de la base de datos NoRostros, no debe estar la persona frente a la cámara, y también se debe considerar que el tiempo de creación de esta base de datos será muy corto, de un poco más de 3 segundos si la cámara adquiere imágenes a 30 FPS.
El código que se emplea para la creación de la base de datos NoRostros, se detalla a continuación:

```
vid=videoinput('winvideo', 1, 'RGB24_320x240'); %Se ingresa el video adquirido mediante la cámara del dron como si se tratara de una cámara USB

set(vid,'ReturnedColorspace','rgb'); %Se define el espacio de color como RGB

set(vid,'TriggerRepeat',Inf); %Se inicia un trigger (captura)

set(vid,'FramesPerTrigger',1); %Se define cuantos frames obtendrá en cada captura

vid.FrameGrabInterval=5; %Se define cada cuantos frames se presentará una imagen

triggerconfig(vid,'manual'); %Se define la configuración del trigger como manual

start(vid); %Se inicia el objeto de video

tic

trigger(vid);while islogging(vid)==1,

end

i=1;

j=1;

while i<101 %Se define un lazo while y un valor r mediante el cual se podrá ejecutar o detener la adquisición de frames

    %Adquisición de cada Frame

    data=flip(getdata(vid,1),2); %Se obtiene cada frame del video para el intervalo definido, y se voltea dos veces debido a que la imagen que se recibe se visualiza como un espejo de la original
```

```
        trigger(vid); %Se captura a partir del video

        imwrite(data,strcat('C:\Users\Erick
        Herrera\Documents\MATLAB\data\NoRostros\',num2str(i),'.jpg'));
        % Se guarda cada frame adquirido de manera automática en la
        carpeta NoRostros para generar la base de datos

        %imshow(data) % Se podría emplear para visualizar cada frame el
        momento que se almacena

        i=i+1;

    end

    stop(vid); %Se detiene la cámara

    flushdata(vid); %Se resetea la memoria del adaptador de adquisición
    de video

    clc; clear;

    close all; objects = imaqfind %Se obtiene todos los objetos de
    adquisición de video que se están ejecutando

    delete(objects) %Se borra la entrada de video generada
```

Mediante el código anterior se crea una base de datos de 100 imágenes que no contienen rostros para que permita entrenar al clasificador. Un ejemplo de los resultados obtenidos mediante este código se muestra en la figura 2.62.

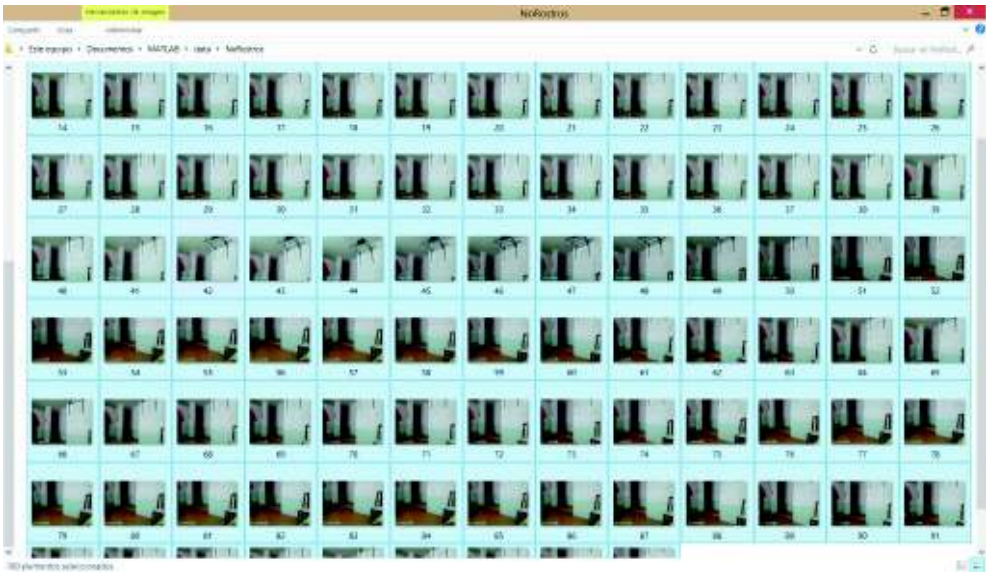


Figura 2.62. Ejemplo de base de datos NoRostros generada para el entrenamiento del clasificador.

(Fuente: elaboración propia)

2.5.5. Entrenamiento del clasificador

Finalmente, al disponer de todos los elementos necesarios para el entrenamiento del clasificador “vision.CascadeObjectDetector”, que se entrena mediante el algoritmo de Viola Jones, se procedió a crear el código de entrenamiento del mismo. Cabe mencionar que esta sección de código se ubica inmediatamente a continuación de los algoritmos de creación de la base de datos en serie dentro del callout del pushbutton “Crear base de datos” de la GUIDE de procesamiento de imágenes.

El código empleado para el entrenamiento del clasificador se detalla a continuación:

```
load('RostrosDetectados.mat'); %Se carga la variable .mat que
contiene las 100 BoundingBox de los rostros

positiveInstances = RostrosDetectados(:,1:2); %Se define a estas
BoundingBox como elementos positivos para la detección

imDir =
fullfile('C:\Users\ErickHerrera\Documents\MATLAB\data\Rostros\');
%Se añade la dirección de la carpeta donde se encuentra la base de
datos de los rostros

addpath(imDir); %Se añade la carpeta de los rostros como path para
que Matlab pueda acceder a ella

negativeFolder =
'C:\Users\ErickHerrera\Documents\MATLAB\data\Rostros\';
% Se añade la dirección de la carpeta de las 100 imágenes que no
contienen rostros para que el clasificador pueda comparar

negativeImages = imageDatastore(negativeFolder); %Se define los
elementos que no contienen rostros

trainCascadeObjectDetector('DetectorRostros.xml',positiveInstances,
negativeFolder,'FalseAlarmRate',0.1,'NumCascadeStages',5);      %Se
ejecuta la función de entrenamiento que emplea la base de datos como
entrada y entregará como resultado el archivo "DetectorRostros.xml"
que contendrá el método de clasificación de las características que
representa el resultado del aprendizaje, donde “FalseAlarmRate”
indica que el proceso continuará hasta tener un máximo de 10% de
falsos positivos, y  'NumCascadeStages' indica el número de etapas
de entrenamiento
```

Como se puede notar en el código anterior, el entrenamiento se realiza por medio de la función “trainCascadeObjectDetector” que tiene como elementos de entrada la carpeta de base de datos que contiene rostros, la carpeta de la base de datos que no contiene rostros y el array de BoundingBox que indica en que área de cada imagen se encuentra cada rostro. El resultado que se obtiene mediante ella es el archivo “DetectorRostros.xml” que contiene el método de clasificación, es decir, que clasificadores de tipo Haar se van a

utilizar y en que configuración en la secuencia de cascada, seleccionando cuál o cuáles de ellos irán en cada capa para ir discriminando las características.

Una vez entrenado el clasificador de detector de rostros se lo puede llamar mediante la función “vision.CascadeObjectDetector” de la siguiente manera por ejemplo:

```
DetectorConfigurado =
vision.CascadeObjectDetector('DetectorRostros.xml');
```

Cabe mencionar que el proceso de entrenamiento puede tomar varios minutos dependiendo de las prestaciones del ordenador que se emplee, del rango de alarma que indica que el aprendizaje continúa hasta tener un porcentaje máximo de falsos positivos, y del número de etapas de aprendizaje que se emplea, el cual indica cuantas veces se repetirá el proceso de aprendizaje. En las pruebas desarrolladas con la configuración antes mostrada se tuvieron tiempos de aprendizaje menores a cinco minutos por ejemplo.

Finalmente, los resultados obtenidos para la detección de rostros mediante el detector de cascada entrenado, fueron satisfactorios como se observa en la figura 2.63.

Figura 2.63. Ejemplo de resultados de detección obtenidos mediante el algoritmo de detección entrenado.

(Fuente: elaboración propia)

Cabe mencionar que la gran ventaja de disponer de un clasificador de cascada para la detección final a diferencia del procesamiento antes mencionado, radica en que gracias al detector de cascada ya no se presenta ninguna situación donde más de un rostro o más de una BoundingBox pueda ser detectada en la misma imagen, lo que si podía suceder en el código de procesamiento ya que solo se tenía matching de características y colores, ya

que estos elementos podrían estar presentes en objetos que no sean rostros, mientras en el clasificador solo se pueden detectar rostros.

2.6. Desarrollo del Algoritmo de seguimiento del objetivo

Una vez desarrollado el algoritmo de detección y verificado que la detección del rostro por visión artificial empleando un clasificador en cascada entrenado entrega excelentes resultados, se procedió a desarrollar un algoritmo de seguimiento que permita emplear la información obtenida a partir del objetivo identificado en la imagen para tomar acciones de control que se efectuarán aplicando distintas configuraciones de velocidad angular en los cuatro motores mediante el programa de control a bordo del dron. Permitiendo de esta manera que el dron pueda volar de manera autónoma siguiendo al objetivo para el que fue entrenado el clasificador.

Para esto se desarrolló una nueva GUIDE de Matlab con el propósito de que permita al usuario visualizar los resultados de la detección que se efectúan en la secuencia de frames en tiempo real, y a su vez permita visualizar las acciones de control que se están efectuando para el vuelo del dron cuando se active el controlador de vuelo mediante un pushbutton.

2.6.1. GUIDE del controlador

La GUIDE desarrollada para llevar a cabo esta tarea se muestra en la figura 2.64.

Figura 2.64. Elementos de la GUIDE de detección y seguimiento del objetivo.

(Fuente: elaboración propia)

Como se puede visualizar en la figura 2.65, la GUIDE de detección y seguimiento del objetivo está diseñada para permitir al usuario visualizar el objetivo detectado, y ejecutar el controlador de vuelo y las acciones de control correspondientes mediante los pushbutton “Ejecutar Controlador” y “Detener Controlador”.

Inicialmente al ejecutar llamar a la GUIDE mediante el código de procesamiento de imágenes luego de haber entrenado el clasificador lo primero que se podrá visualizar en ella es la secuencia de frames en tiempo real, que están siendo adquiridas mediante el código de procesamiento a manera de video proyectado en el elemento axes2. De esta manera no se verán ejecutadas las acciones de control aún, ya que se deja esta tarea para el momento en el que el dron está visualizando al objetivo. Entonces, cuando el dron tenga el objetivo en la mira se llamará a ejecutar el controlador. Es así, que el código inicial permite que se ejecuta inmediatamente llamando a la GUIDE y presentando directamente el objetivo detectado rodeado por una BoundingBox, y mostrando un label indicando la palabra objetivo y las coordenadas del centroide de la BoundingBox del área del rostro detectado. El código empleado para llevar a cabo esta tarea y el detalle del funcionamiento de cada línea de programación se presenta a continuación:

```
global m cont %Se definen las variables m y cont como globales para
poder llamarlas desde cualquier función del código de la GUIDE
m=0; %Se inicia una variable m con el valor cero que permitirá iniciar
el lazo while de detección
cont=0; %Se inicia una variable cont que permitirá ejecutar el
controlador cuando tome el valor de uno, inicialmente con el valor
de cero está desactivada hasta que el usuario la active

etiqueta='Objetivo'; %Se define la etiqueta que llevará el objetivo
detectado

vid=videoinput('winvideo', 1, 'RGB24_1920x1440'); %Se genera una
entrada de video en el puerto 1 con una resolución de 1920x1440

set(vid,'ReturnedColorspace','rgb'); %Se define el espacio de
colores para la imagen de tipo RGB

set(vid,'TriggerRepeat',Inf); %Se inicia un obturador que capturará
los frames del video

set(vid,'FramesPerTrigger',1); %Se define cuantos frames se
capturarán por cada obturación

vid.FrameGrabInterval=5; %Se define un intervalo de cada cuantos
frames del video se capturará la imagen, en este ejemplo cada 5

triggerconfig(vid,'manual'); %Se define el método de captura como
manual para que la captura se realice mediante el lazo while

start(vid); %Se inicia el objeto vid que corresponde al video
obtenido mediante la cámara
```

```
tic %Se genera una alerta para cada captura

trigger(vid);while islogging(vid)==1, end %Se define que se obture
siempre mientras el objeto vid esté en ejecución

DetectorConfigurado =
vision.CascadeObjectDetector('DetectorRostros.xml');
%Se inicializa el detector de cascada entrenado mediante la GUIDE de
procesamiento a través del archivo 'DetectorRostros.xml'

%Inicializar motor java

      import java.awt.*; %Se inicializa el motor java que ejecutará
      las acciones del controlador

      import java.awt.event.*; %Se genera el evento que simula la
      pulsación de los comandos del controlador

      rob=Robot; %Se genera el elemento Robot que simula la
      interacción de un usuario con la PC

while (m==0) %La detección del rostro del objetivo seguirá
ejecutándose mientras m sea igual a cero

      %Adquisición de cada Frame

      data=flip(getdata(vid,1),2); %Se adquiere cada frame del video
      de la cámara del dron, como la imagen del dron está reflejada
      se aplican dos flip para ponerla en posición normal

      trigger(vid); %Se ejecuta la obturación sobre el video que se
      está ejecutando

      im=data; %Se almacena la imagen obtenida en una variable nueva

      EntradaDetector = uint8(im); %Se aproximan los píxeles a
      valores enteros de 8 bits

      Sim=size(EntradaDetector); %Se obtienen las dimensiones de las
      matrices de las imágenes de entrada del detector

      centrox=960; %Se define el centro de la matriz de la imagen en
      x

      centroy=720; %Se define el centro de la matriz de la imagen en
      y

      BB = step(DetectorConfigurado,EntradaDetector); %Se obtienen
      las BoundingBox del rostro detectado mediante el clasificador
      entrenado

      NumeroDeRostrosADetectar=size(BB,1); %Se define una variable
      que indica el número de rostros a detectar

      for i = 1:NumeroDeRostrosADetectar %Se inicia un lazo for para
      detectar los rostros en las imágenes
            axes(handles.axes2); %Se define el axes2 como elemento
            donde se van a visualizar los resultados
```

```
imshow(EntradaDetector); %Se proyecta la imagen de cada
frame adquirido mediante la cámara en el axes2

bound=BB(i,:); %Se define el elemento bound que es la
BoundingBox de cada rostro detectado

centroidex=bound(1)+bound(3)/2; %Se obtiene la posición
del centroide en x de cada rostro

centroidey=bound(2)+bound(4)/2; %Se obtiene la posición
del centroide en y de cada rostro

xlin=[centrox centroidex]; %Se genera un vector de
coordenadas en x de un elemento gráfico línea que une el
centro de la imagen con el centro de cada rostro para
que el usuario pueda visualizar a que distancia se
encuentra

ylin=[centroy centroidey]; %Se genera un vector de
coordenadas en y de un elemento gráfico línea que une el
centro de la imagen con el centro de cada rostro para
que el usuario pueda visualizar a que distancia se
encuentra

line(xlin,ylin,'Color','green','LineStyle','--')     %Se
dibuja una línea en cada imagen que una el centroide de
cada rostro con el centro de la imagen

errorx=centroidex-centrox; %Se obtiene el valor del error
en x que representa la distancia en x que el rostro está
desplazado respecto al dron para que el controlador luego
pueda corregirla

errory=centroy-centroidey; %Se obtiene el valor del error
en y que representa la distancia en y que el rostro está
desplazado respecto al dron para que el controlador luego
pueda corregirla

area=bound(3)*bound(4); %Se obtiene el área de cada
rostro detectado para que mediante ella el control de
proximidad pueda determinar si acercarse o alejarse

posicion=strcat('(         ',num2str(errorx),'         ;
',num2str(errorx),'  )'); %Se genera la etiqueta que
llevará cada centroide ploteado sobre la imagen con las
coordenadas del rostro respecto al centro de la imagen
en píxeles

rectangle('Position',BB(i,:),'LineWidth',5,'LineStyle',
'-.','EdgeColor','b'); %Se plotea la BoundingBox sobre
el rostro detectado para que el usuario pueda
visualizarla

hold on %Se mantiene los elementos ploteados para
proyectar sobre ellos los demás elementos

plot(centroidex,centroidey,'r*'); %Se plotea un punto
rojo que simboliza el centroide del rostro detectado
```

```
                text(centroidex,centroidey-
                10,etiqueta,'color','white','FontSize',12);
                %Se plotea la etiqueta del objetivo

                text(centroidex,centroidey+10,posicion,'color','white',
                'FontSize',12); %Se plotean las coordenadas del centroide
                sobre la imagen

                pause(0.01); %Se realiza una pausa de una centésima de
                segundo en cada iteración para que estos elementos
                gráficos puedan ser visualizados

                hold off %Se detiene el hold que mantenía los elementos
                gráficos visibles
            end
        end
```

De esta manera se ejecuta la detección de los rostros y se presentan al usuario para que pueda visualizarlos en tiempo real. Además, dentro de este mismo código se obtienen los elementos “errorx”, “errory” y “area” que son los parámetros de entrada para el controlador de vuelo. Un ejemplo de los resultados de detección obtenidos y su presentación al usuario mediante la GUIDE se muestra en la figura 2.65.

(a) (b) (c)

Figura 2.65. Detección de rostros en diferentes posiciones y presentación de resultados mediante la GUIDE del controlador.

(Fuente: elaboración propia)

2.6.2. Algoritmo de seguimiento

Una vez extraída la información del rostro detectado, y disponiendo del controlador del vuelo a bordo del dron es relativamente sencillo realizar la tarea de seguimiento, ya que en términos generales la imagen extraída, el rostro y sus propiedades se podrían describir como los nuevos sensores de vuelo del dron mediante los cuales se ejecutarán las señales de control. Un ejemplo de las variables extraídas del rostro detectado se muestra en la figura 2.66.

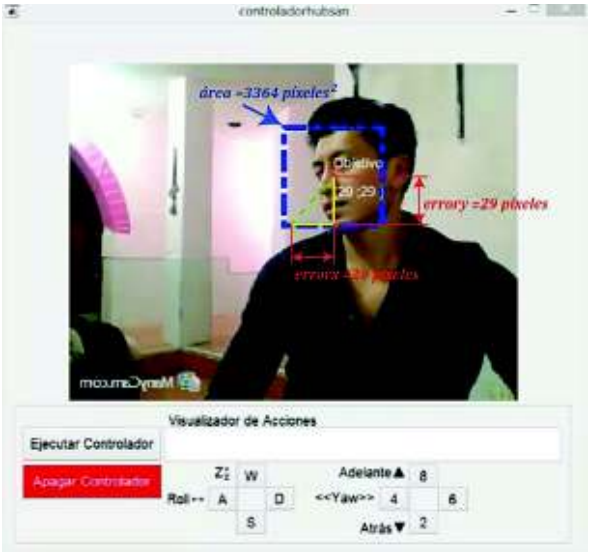


Figura 2.66. Variables extraídas del rostro detectado.

(Fuente: elaboración propia)

Donde la variable “errorx” almacena la información de que tan lejos se encuentra el rostro respecto al centro de la imagen, de manera que el dron mediante el controlador buscará reducir este error a cero. Un error positivo en x representa que el rostro se encuentra a la derecha del dron por lo tanto la acción de control será que el dron se mueva hacia la derecha, mientras un error negativo en x indica que el dron está a la izquierda por lo que el dron deberá moverse hacia la izquierda. Estas acciones de control se ejecutan mediante un momento aplicado en roll variando las velocidades angulares como se mostró en la sección 2.1. El error en x se obtiene mediante la expresión mostrada en la ecuación 2.165.

$$errorx = centroidex_{BoundingBox} - centrox_{imagen} \qquad \text{Ec. [2.165]}$$

La variable “errory” permite realizar el control de altitud del dron. Es así, que un error positivo en y representa que el objetivo se encuentra por encima del dron, por lo que tendrá que aplicarse un empuje vertical que permita que el dron suba, mientras que si el error es negativo en y implica que el objetivo está por debajo del dron, por lo que la acción de control será disminuir el empuje vertical. El error en y se obtiene mediante la expresión mostrada en la ecuación 2.166.

$$errory = centroy_{imagen} - centroidey_{BoundingBox} \qquad \text{Ec. [2.166]}$$

Además, la variable “area” se obtiene multiplicando las dimensiones del largo y el ancho del BoundingBox que rodea el rostro “boun(3)” y “bound(4)” respectivamente en la matriz

de la variable bound, y esta permitirá realizar un control de proximidad del objetivo mediante un momento en dirección roll que cambiará la orientación de propulsión del dron permitiéndole desplazarse hacia adelante o hacia atrás. Es así, que se define un valor de área límite que dependiendo de la óptica de la cámara permitirá al dron mantenerse a una distancia fija, en este ejemplo para el dron Hubsan H507a se tiene que un área de $2500\ píxeles^2$ equivale a una distancia de tres metros aproximadamente. Si el área del BoundingBox es menor que los $2500\ píxeles^2$ el dron tendrá que acercarse, mientras si esta área es mayor, el dron tendrá que alejarse (retroceder).

Finalmente, el código de seguimiento y el detalle de cada una de sus líneas de programación se detallan a continuación:

```
%Parámetros controladorhubsan

%Control roll

if (cont==1) %Se emplea la variable cont que se definió previamente como
global para activar a manera de switch el algoritmo que generará las señales
de control

      if errorx < 0 %Para el control de posición si el error en horizontal
      es negativo, el objetivo está a la izquierda

            rob.keyPress(KeyEvent.VK_A); %Se solicita al motor java que
            simule una pulsación de la tecla A del controlador de vuelo

            pause(0.1); %Se genera una pausa de una décima de segundo para
            que se lleve a cabo la acción de control

            rob.keyRelease(KeyEvent.VK_A); %Se solicita que se libere la
            pulsación de la tecla

            set(handles.pushbutton4,'BackgroundColor',[1   0   0]);   %Se
            solicita que el elemento gráfico de la GUIDE cambie a color
            rojo para poder visualizar esta pulsación

            set(handles.pushbutton5,'BackgroundColor',[0.94  0.94  0.94]);
            %Se solicita que el elemento de la interfaz gráfica de la
            derecha vuelva a tono gris

      else %Correscpondería al caso de que el objetivo esté a la derecha

            rob.keyPress(KeyEvent.VK_D); %Se solicita al motor java que
            simule una pulsación de la tecla D del controlador de vuelo

            pause(0.1); %Se genera una pausa de una décima de segundo para
            que se lleve a cabo la acción de control

            rob.keyRelease(KeyEvent.VK_D); %Se solicita que se libere la
            pulsación de la tecla
```

```
            set(handles.pushbutton5,'BackgroundColor',[1   0   0]);   %Se
            solicita que el elemento gráfico de la GUIDE cambie a color
            rojo para poder visualizar esta pulsación

            set(handles.pushbutton4,'BackgroundColor',[0.94  0.94  0.94]);
            %Se solicita que el elemento de la interfaz gráfica de la
            izquierda vuelva a tono gris
      end


%Control altitud

      if errory < 0 %Para el control de altitud si errory es negativo el
      objetivo está debajo del dron por lo que la acción de control será
      bajar
            rob.keyPress(KeyEvent.VK_S); %Se solicita al motor java que
            simule una pulsación de la tecla S del controlador de vuelo

            pause(0.1); %Se genera una pausa de una décima de segundo para
            que se lleve a cabo la acción de control

            rob.keyRelease(KeyEvent.VK_S); %Se solicita que se libere la
            pulsación de la tecla

            set(handles.pushbutton6,'BackgroundColor',[1   0   0]);   %Se
            solicita que el elemento gráfico de la GUIDE cambie a color
            rojo para poder visualizar esta pulsación

            set(handles.pushbutton3,'BackgroundColor',[0.94  0.94  0.94]);
            %Se solicita que el elemento de la interfaz gráfica W vuelva a
            tono gris

      else %para el caso de que errory sea positivo, el objetivo se
      encuentra arriba del dron, por lo que la acción de control será subir

            rob.keyPress(KeyEvent.VK_W); %Se solicita al motor java que
            simule una pulsación de la tecla  W del controlador de vuelo

            pause(0.1); %Se genera una pausa de una décima de segundo para
            que se lleve a cabo la acción de control

            rob.keyRelease(KeyEvent.VK_W); %Se solicita que se libere la
            pulsación de la tecla

            set(handles.pushbutton3,'BackgroundColor',[1   0   0]);   %Se
            solicita que el elemento gráfico de la GUIDE cambie a color
            rojo para poder visualizar esta pulsación

            set(handles.pushbutton6,'BackgroundColor',[0.94  0.94  0.94]);
            %Se solicita que el elemento de la interfaz gráfica S vuelva a
            tono gris
      end

%Control proximidad

      if area > 2500 %Se define un área máxima de 2500 píxeles cuadrados
      que para el Hubsan H507a equivalente a aproximadamente tres metros,
      si el área del rostro es menor, la acción de control será que el dron
      se acerque
```

```
            rob.keyPress(KeyEvent.VK_UP); %Se solicita al motor java que
            simule una pulsación de la tecla  UP (equivalente a forward)
            del controlador de vuelo

            pause(0.1); %Se genera una pausa de una décima de segundo para
            que se lleve a cabo la acción de control

            rob.keyRelease(KeyEvent.VK_UP); %Se solicita que se libere la
            pulsación de la tecla

            set(handles.pushbutton9,'BackgroundColor',[1   0   0]);   %Se
            solicita que el elemento gráfico de la GUIDE cambie a color
            rojo para poder visualizar esta pulsación

            set(handles.pushbutton8,'BackgroundColor',[0.94  0.94  0.94]);
            %Se solicita que el elemento de la interfaz gráfica 2 vuelva a
            tono gris

      else %Para el caso que el área del rostro detectado sea mayor que
      2500 píxeles cuadrados, implica que el dron está muy cerca, por lo
      que la acción de control será alejarse
            rob.keyPress(KeyEvent.VK_DOWN); %Se solicita al motor java que
            simule una pulsación de la tecla  DOWN (equivalente a backward)
            del controlador de vuelo

            pause(0.1); %Se genera una pausa de una décima de segundo para
            que se lleve a cabo la acción de control

            rob.keyRelease(KeyEvent.VK_DOWN); %Se solicita que se libere
            la pulsación de la tecla

            set(handles.pushbutton8,'BackgroundColor',[1   0   0]);   %Se
            solicita que el elemento gráfico de la GUIDE cambie a color
            rojo para poder visualizar esta pulsación

            set(handles.pushbutton9,'BackgroundColor',[0.94  0.94  0.94]);
            %Se solicita que el elemento de la interfaz gráfica 8 vuelva a
            tono gris
      end



%Control de orientación 25% 75%

      if centroidex < 480 %Para el control de orientación yaw se define
      que si el objetivo se encuentra en una región menor que el 25% de
      los píxeles en horizontal, el objetivo está saliendo del rango de la
      imagen por la izquierda por lo que la acción de control es rotar a
      la izquierda

            rob.keyPress(KeyEvent.VK_LEFT); %Se solicita al motor java que
            simule una pulsación de la tecla LEFT equivalente a un giro
            hacia la izquierda del controlador de vuelo

            pause(0.1); %Se genera una pausa de una décima de segundo para
            que se lleve a cabo la acción de control

            rob.keyRelease(KeyEvent.VK_LEFT); %Se solicita que se libere
            la pulsación de la tecla
```

```
            set(handles.pushbutton10,'BackgroundColor',[1   0   0]);   %Se
            solicita que el elemento gráfico de la GUIDE cambie a color
            rojo para poder visualizar esta pulsación

            set(handles.pushbutton11,'BackgroundColor',[0.94 0.94 0.94]);
            %Se solicita que el elemento de la interfaz gráfica 6 vuelva a
            tono gris

        elseif centroidex > 1440 %Para el control de orientación yaw se
        define que si el objetivo se encuentra en una región mayor que el
        75% de los píxeles en horizontal, el objetivo está saliendo del rango
        de la imagen por la izquierda por lo que la acción de control es
        rotar a la izquierda

            rob.keyPress(KeyEvent.VK_RIGHT); %Se solicita al motor java
            que simule una pulsación de la tecla RIGHT equivalente a un
            giro hacia la izquierda del controlador de vuelo

            pause(0.1); %Se genera una pausa de una décima de segundo para
            que se lleve a cabo la acción de control

            rob.keyRelease(KeyEvent.VK_RIGHT); %Se solicita que se libere
            la pulsación de la tecla

            set(handles.pushbutton11,'BackgroundColor',[1   0   0]);   %Se
            solicita que el elemento gráfico de la GUIDE cambie a color
            rojo para poder visualizar esta pulsación

            set(handles.pushbutton10,'BackgroundColor',[0.94 0.94 0.94]);
            %Se solicita que el elemento de la interfaz gráfica 4 vuelva a
            tono gris
        else
            set(handles.pushbutton10,'BackgroundColor',[0.94 0.94 0.94]);
            set(handles.pushbutton11,'BackgroundColor',[0.94 0.94 0.94]);

          end

    end
```

Además, el código desarrollado dispone de un algoritmo de control de orientación que se crea dividiendo la imagen en cuatro elementos de manera horizontal. Es así, que si el objetivo se encuentra en una posición inferior al 25% del ancho de la imagen, el dron deberá rotar en yaw hasta tener a la imagen comprendida en un intervalo del 25% al 75%, es decir enfocada. Adicionalmente, si el objetivo se encuentra en una posición de más del 75% del ancho de la imagen, el dron deberá girar en yaw hacia la derecha hasta tener al objetivo en el rango aceptable del 25% al 75%.

Finalmente, un ejemplo de la ejecución del algoritmo de seguimiento a partir del rostro detectado se muestra en la figura 2.67.

(a) (b) (c)

Figura 2.67. Detección de rostros y acciones de control para distintas posiciones y orientaciones.

(Fuente: elaboración propia)

3. RESULTADOS Y DISCUSIÓN

En el presente capítulo se detallan los resultados obtenidos gracias a la implementación del sistema de seguimiento mediante la visión artificial por computadora para un cuadricóptero. Para el desarrollo del sistema en una fase inicial se tenía previsto implementarlo en un cuadricóptero comercial muy popular aplicado en investigaciones que es el Parrot AR. 2.0 Drone, debido a que existe una gran cantidad de información desarrollada para este dron en particular, ya que este presenta la principal ventaja de poder ser controlado con facilidad desde un ordenador mediante programas como el AR. 2 PC Flight, AR Free Flight, o eDrone Project, para el caso de Windows. Además, existen toolbox y funciones desarrolladas por la comunidad y disponibles como Add-Ons en las librerías de Matlab, por lo que la utilización de este dron en particular para el desarrollo de trabajos de investigación resulta muy práctica.

A diferencia de este, los drones comerciales en su gran mayoría no permiten su pilotaje desde la PC, y no disponen de softwares particulares para cada uno de ellos de los cuales se pueda sacar ventaja para la implementación de un sistema de visión artificial y teleoperación desde el ordenador, más bien estos están orientados para ser piloteados y controlados desde smartphones.

Sin embargo, considerando que el año de lanzamiento del Parrot AR. 2.0 Drone fue en el 2010, y actualmente existe una gran variedad de drones comerciales de bajo precio y excelentes prestaciones, en este proyecto se realizó la implementación adicional del sistema de visión artificial en un dron comercial de bajo costo, como lo es el Hubsan H507a.

3.1. Implementación

La implementación del sistema para los dos casos mencionados se desarrolló empleando dos metodologías de adquisición de imágenes diferentes, como se mostró en la sección 2.4.1 y 2.4.2. Para el caso del Parrot AR. 2.0 Drone se contó con un programa ejecutable desde el ordenador mediante el cual se pudo adquirir la información de la cámara a bordo del dron, así como, la información de sus sensores, y mediante ellos realizar el procesamiento de las imágenes, el seguimiento de objetivos y generar las acciones de control desde el ordenador.

Mientras que en el caso del Hubsan H507a se empleó una máquina virtual con el sistema operativo Android, encargada de adquirir la información del dron y su cámara, misma que se puede transferir a Matlab en el ordenador, mediante el cual se procesa las imágenes, se realiza el seguimiento de objetivos y se generan las acciones de control.

Posteriormente en la fase de implementación de los algoritmos de procesamiento de imágenes, entrenamiento del clasificador, detección del objetivo y seguimiento del mismo, los códigos empleados en ambos casos fueron casi los mismos obteniéndose resultados similares en ambos casos. Donde la única diferencia radica en el procesamiento de las imágenes adquiridas a partir de la secuencia de frames, provenientes de la cámara del dron, ya que el procesamiento varía dependiendo de la resolución que presenta la cámara a bordo del dron.

Es así, que el único factor a modificarse en el código de adquisición y procesamiento de imágenes es el ajuste del tamaño de la imagen que se va a procesar. Esto se logró en particular mediante la función “imresize” que se muestra en la siguiente línea de código:

```
im=imresize(data,0.5);
```

Donde data, representa la variable donde se almacenó la imagen capturada a partir del video de la cámara del dron, y el factor 0.5 indica el porcentaje de escalado que se le va a aplicar a la imagen, siendo en este caso de un 50% del tamaño real de la imagen adquirida. Dado que la cámara a bordo del Parrot AR. 2.0 Drone cuenta con una resolución de 1080 x 720 píxeles, el factor de escalado empleado fue mayor que el del Hubsan H507a, con un valor de 0.5, quedando en este caso una imagen de 540 x 360 píxeles. Mientras que, en el Hubsan H507a se realizaron pruebas sin factor de escalado (imresize=1) debido a que gracias al software ManyCam la resolución puede ser ajustada directamente en un valor de 320 x 240 píxeles, obteniéndose buenos resultados.

El redimensionamiento de la imagen mediante “imresize” resulta de gran importancia, debido a que el costo computacional de la detección de rostros y control del dron en tiempo real es muy elevada, y se determinó que implementando un redimensionamiento de la imagen las operaciones se pueden alcanzar de manera muy fluida, sin sacrificar eficacia en la detección, teniendo en general mejores resultados.

A continuación, la implementación en el caso más genérico del dron Hubsan, del sistema desarrollado se puede decir que es aplicable para la mayoría de drones comerciales, por lo que su implementación se da de la siguiente manera.

Primero se instala una máquina virtual Android que permita emplear el software de pilotaje que provee el fabricante del dron, en el caso del Hubsan H507a para las pruebas efectuadas se emplearon las consolas virtuales Nox player y Bluestacks, de las que se puede mencionar que destaca Nox player ya que no requiere computadoras de grandes prestaciones para funcionar de manera fluida. Por su parte Bluestacks presenta una

imagen de mayor resolución al ser adquirida mediante la cámara, factor que no es crucial dado que como se mencionó antes la mayor resolución de la imagen genera un costo computacional elevado y presenta resultados muy similares a una de baja resolución para la detección de rostros. Además, se dispone de máquinas virtuales más sofisticadas que se ejecutan mediante Virtual Box, VM Ware, entre otras, pero las dos antes mencionadas funcionan de muy buena manera y no requieren de muchos recursos.

Es así, que una vez disponiendo de una máquina virtual Android emulada en Windows se procede a instalar el software de pilotaje suministrado por el fabricante, como se puede apreciar en la figura 3.1.

Figura 3.1. Aplicación de pilotaje suministrada por el fabricante, emulada en Nox Player.
(Fuente: elaboración propia)

Posteriormente dentro de la interfaz Android, en la aplicación de pilotaje se tiene que para empezar su funcionamiento, la misma requiere de comandos táctiles para su funcionamiento. Es así, que tanto Nox Player como Bluestacks presentan emuladores de acciones táctites, que se ejecutan mediante el teclado para el usuario en Windows. Estos emuladores fueron diseñados para permitir que los usuarios de las máquinas virtuales puedan interactuar con las diversas aplicaciones y juegos mediante el teclado. Entonces, el siguiente paso en la implementación corresponde a configurar los botones que se generaron en el controlador de vuelo, para que estas acciones de control puedan ser interpretadas como acciones táctiles para el vuelo del dron. Es así, que la distribución de acciones táctiles para el control del dron mediante la aplicación del fabricante, se muestra en la figura 3.2.

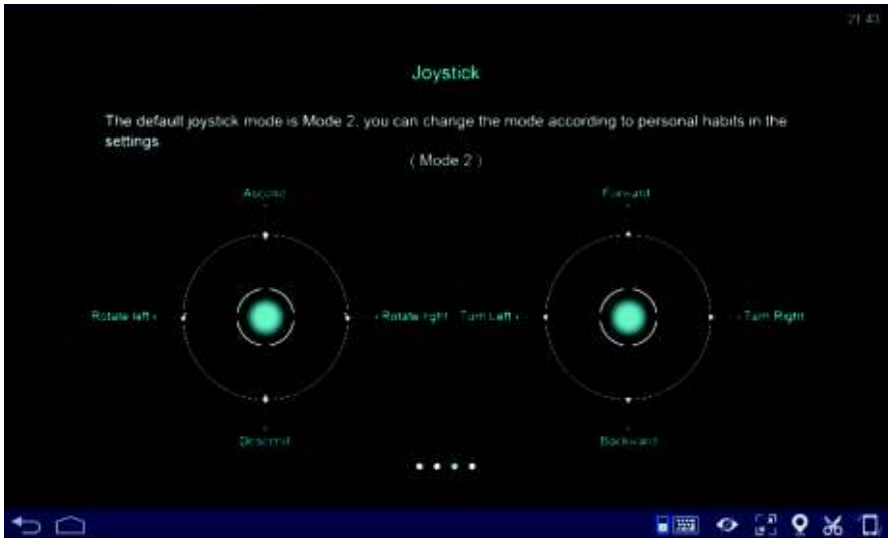


Figura 3.2. Distribución de controles táctiles en la aplicación del fabricante, para el Hubsan H507a.

(Fuente: elaboración propia)

Entonces de acuerdo con la figura 3.2. el software de pilotaje está diseñado mediante dos Joysticks táctiles donde en el primero (izquierda) se darán las órdenes de control correspondientes al empuje vertical para el ascenso y descenso, además, del momento en Yaw que le permitirá al dron rotar sobre su eje z. Por su parte el segundo Joystick (derecha) se encargará de ejecutar las órdenes de avanzar y retroceder mediante un momento en pitch, y trasladarse de izquierda a derecha mediante un momento en roll.

El programa Nox Player presenta una herramienta de emulación touch, por lo que de acuerdo con los comandos diseñados y empleados en la GUIDE de Matlab, su configuración se puede visualizar en la figura 3.3.

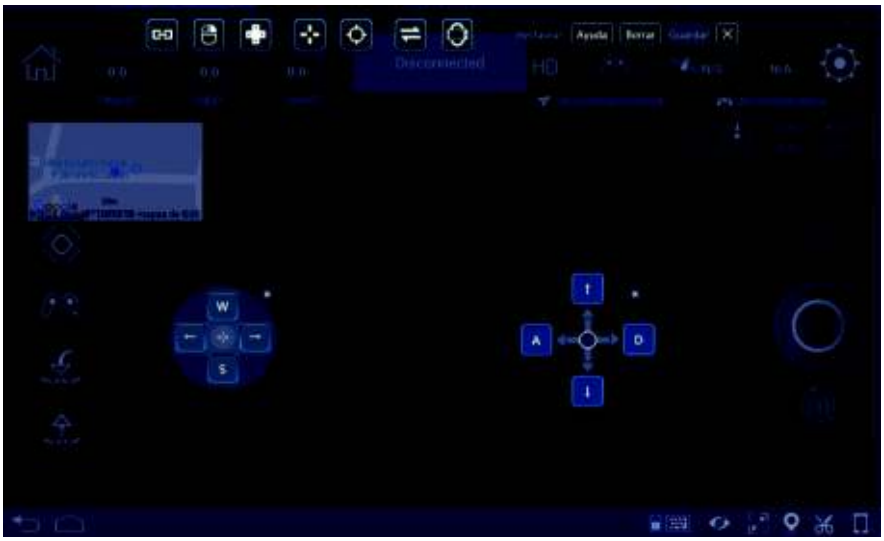


Figura 3.3. Disposición y configuración de los mandos táctiles para el Hubsan 507a.

(Fuente: elaboración propia)

Luego para dar por concluida la configuración del software de pilotaje, se tiene que el dron en su interfaz presenta muchas opciones de configuración, para las diversas funciones y aplicaciones que este dron posee. La distribución de los elementos de configuración de vuelo del dron se puede visualizar en la figura 3.4.

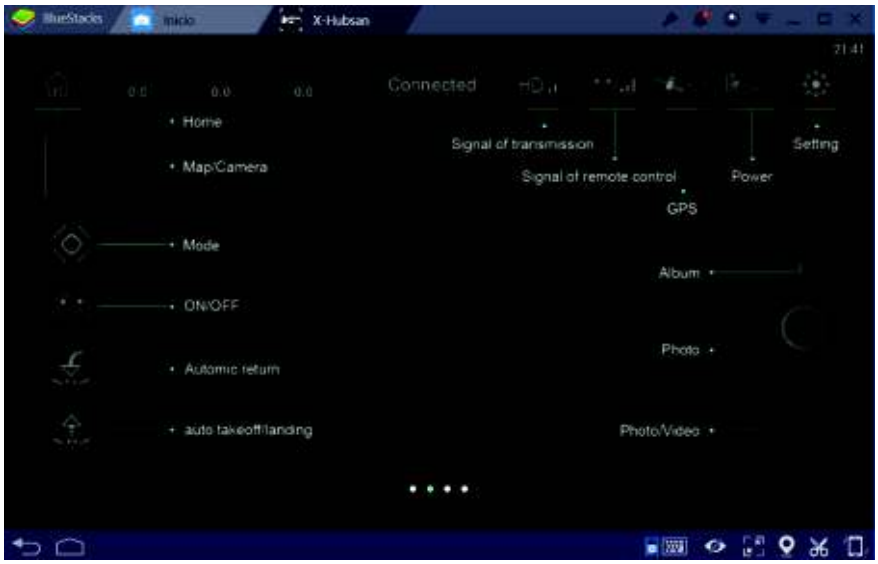


Figura 3.4. Elementos de pilotaje y configuración del dron

(Fuente: elaboración propia)

Posteriormente para poder tener resultados óptimos de seguimiento durante las pruebas, se consideraron tres factores adicionales que mejoran significativamente la ejecución del algoritmo de seguimiento.

La primera es considerar que durante la detección por medio de la cámara a bordo del dron, se reciben aproximadamente 30 frames por segundo hacia la interfaz de Matlab, valor que puede variar dependiendo de las condiciones de transmisión de los datos por protocolo UDP que puede estar sujeta a interferencias o ruido. De esta manera los 30 frames por segundo podrían generar 30 detecciones de rostros por segundo en el mejor de los casos, en computadoras de altas prestaciones, pero en el código se agregaron elementos que permiten modificar este valor para mejorar la velocidad de procesamiento, a cambio de generar menos detecciones procesadas por segundo.

El primer elemento consiste en la adición de la función FrameGrabInterval. Esta permite definir un intervalo mediante el cual se procesará uno de cada n píxeles que se hayan definido mediante esta función. La línea de código que permite llevar a cabo esta tarea se muestra a continuación:

```
vid.FrameGrabInterval=3;
```

En el ejemplo mostrado se ha definido que se capture un frame y detecte un rostro en uno de cada tres frames de la secuencia de video. El efecto que esto tiene es que el video presentado en la GUIDE seguirá siendo el mismo, pero el rostro se detectará en uno de cada tres frames, de esta manera, la velocidad de detección se reducirá a 10 rostros por segundo, lo que es considerablemente bueno ya que se tendrán 10 acciones de control por segundo. Cabe destacar que si se desea mejorar la velocidad de detección se puede lograr mediante este comando ingresando un valor menor al antes mencionado.

Como segundo elemento que permite mejorar la ejecución del algoritmo de seguimiento, se tiene el hecho de la presencia de falsos negativos en la detección, lo que representa un inconveniente en la ejecución del visualizador de la GUIDE por medio del elemento axes, debido a que el código no continuará su ejecución a menos que tenga un rostro identificado. Es así, que este hecho puede generar errores durante la ejecución del algoritmo y desembocar en un fallo, por esto se implementó un lazo "if" que permite que el código de procesamiento siga ejecutándose a pesar de no tener un rostro identificado, y se mantenga dispuesto a realizar una nueva detección el momento en que el objetivo esté presente de nuevo en un frame.

El código modificado y empleado para esta tarea se presenta a continuación, donde se hace uso de la función "isempty(BB)", lo que permite que el código muestre cada nuevo frame a pesar de que el array BoundingBox esté vacío (no exista un rostro detectado), y en caso de que exista un rostro detectado procede a graficar su ubicación, calcular las entradas del controlador y ejecutar el seguimiento.

```
if isempty(BB)

      imshow(EntradaDetector); %Se proyecta la imagen de cada frame
      adquirido mediante la cámara en el axes2

      pause(0.01); %Se realiza una pausa de una centésima de segundo
      en cada iteración para que estos elementos gráficos puedan ser
      visualizados

else

      for i = 1:NumeroDeRostrosADetectar %Se inicia un lazo for para
      detectar los rostros en las imágenes

            k=k+1;

            axes(handles.axes2); %Se define el axes2 como elemento
            donde se van a visualizar los resultados

            imshow(EntradaDetector); %Se proyecta la imagen de cada
            frame adquirido mediante la cámara en el axes2

            s=size(im);
```

```
bound=BB(i,:); %Se define el elemento bound que es la
BoundingBox de cada rostro detectado

centroidex=bound(1)+bound(3)/2; %Se obtiene la posición
del centroide en x de cada rostro

centroidey=bound(2)+bound(4)/2; %Se obtiene la posición
del centroide en y de cada rostro

xlin=[centrox centroidex]; %Se genera un vector de
coordenadas en x de un elemento gráfico línea que une el
centro de la imagen con el centro de cada rostro para
que el usuario pueda visualizar a que distancia se
encuentra

ylin=[centroy centroidey]; %Se genera un vector de
coordenadas en y de un elemento gráfico línea que une el
centro de la imagen con el centro de cada rostro para
que el usuario pueda visualizar a que distancia se
encuentra

line(xlin,ylin,'Color','green','LineStyle','--')    %Se
dibuja una linea en cada imagen que una el centroide de
cada rostro con el centro de la imagen

errorx=centroidex-centrox; %Se obtiene el valor del
error en x que representa la distancia en x que el rostro
está desplazado respecto al dron para que el controlador
luego pueda corregirla

errory=centroy-centroidey; %Se obtiene el valor del
error en y que representa la distancia en y que el rostro
está desplazado respecto al dron para que el controlador
luego pueda corregirla

area=bound(3)*bound(4); %Se obtiene el área de cada
rostro detectado para que mediante ella el control de
proximidad pueda determinar si acercarse o alejarse

posicion=strcat('(         ',num2str(errorx),'         ;
',num2str(errorx),'  )'); %Se genera la etiqueta que
llevará cada centroide ploteado sobre la imagen con las
coordenadas del rostro respecto al centro de la imagen
en píxeles


rectangle('Position',BB(i,:),'LineWidth',5,'LineStyle',
'-.','EdgeColor','b'); %Se plotea la BoundingBox sobre
el rostro detectado para que el usuario pueda
visualizarla

hold on %Se mantiene los elementos ploteados para
proyectar sobre ellos los demás elementos

plot(centroidex,centroidey,'r*'); %Se plotea un punto
rojo que simboliza el centroide del rostro detectado

text(centroidex,centroidey-
10,etiqueta,'color','white','FontSize',12); %Se plotea
la etiqueta del objetivo
```

```
text(centroidex,centroidey+10,posicion,'color','white',
'FontSize',12); %Se plotean las coordenadas del centroide
sobre la imagen

pause(0.01); %Se realiza una pausa de una centésima de
segundo en cada iteración para que estos elementos
gráficos puedan ser visualizados

hold off %Se detiene el hold que mantenía los elementos
gráficos visibles
```

Adicionalmente cabe mencionar que un elemento de mucha utilidad que permite una visualización fluida de la imagen en el axes de la GUIDE es la adición de una línea de código que permite pausar la presentación de una imagen, previo a la presentación de la siguiente para la siguiente iteración. El código empleado es:

```
pause(0.01);
```

Este comando permite pausar la presentación de una imagen y mantenerla durante una centésima de segundo, lo que no afecta considerablemente en la velocidad de procesamiento y seguimiento ya que este elemento se empleó dos veces a lo largo de todo el código, ya que agrega un retraso total de dos centésimas de segundo, pero mejora considerablemente la velocidad de presentación de resultados en la GUIDE ya que se debe considerar que el mecanismo de presentación del video en la interfaz gráfica, se da mediante un ploteo sucesivo de cada frame encima del anterior, lo que genera problemas en la visualización. Es así, que al añadir el elemento pause se limita el tiempo de presentación de cada frame, lo que facilita el cerrado y visualización del siguiente.

Finalmente el tercer elemento empleado para la mejora de la detección, implementación y presentación de resultados durante las pruebas efectuadas constituye el desafío de permitir que el controlador desarrollado pueda enviar de manera correcta los comandos generados mediante un keypress y touch virtual, hacia el software de pilotaje en la máquina virtual Android. Para esto se tomó en cuenta inicialmente que la ejecución de la GUIDE involucra un ploteo sucesivo de 30 a 10 imágenes por segundo lo que provoca que la interfaz gráfica se sobreponga encima de cualquier ventana que se esté ejecutando en Windows.

Es así, que al generar las acciones de control y enviarlas mediante una pulsación virtual de teclas empleando el evento keypress mediante el objeto robot de java, las acciones llegan a la interfaz Android pero ante el ploteo sucesivo de la GUIDE, dejan de llegar ya que el ploteo lleva a salir de la ejecución de una ventana a otra. Para esto la solución que se generó fue contener al código del controlador dentro de un evento MousePress que simula la pulsación simultánea del mouse y una tecla que es la acción de control para de

esta manera emular satisfactoriamente la interacción en Android por medio de un comando táctil. Las líneas de comando que permiten inicializar y librear la pulsación por click derecho del mouse y emular un touch son:

```
rob.mousePress(InputEvent.BUTTON3_MASK);
rob.mouseRelease(InputEvent.BUTTON3_MASK);
```

Entonces el código del controlador del Hubsan implementado para el pilotaje mediante la máquina virtual Android se detalla a continuación:

```
%Parámetros controladorhubsan
%Control roll

k=k+1;

if (cont==1) %Se emplea la variable cont que se definió previamente
como global para activar a manera de switch el algoritmo que generará
las señales de control

      if k>5

            if errorx < 0 %Para el control de posición si el error
            en horizontal es negativo, el objetivo está a la
            izquierda

                  rob.mousePress(InputEvent.BUTTON3_MASK);

                  rob.keyPress(KeyEvent.VK_A); %Se solicita al motor
                  java que simule una pulsación de la tecla  A del
                  controlador de vuelo

                  rob.keyRelease(KeyEvent.VK_A); %Se solicita que se
                  libere la pulsación de la tecla

                  rob.mouseRelease(InputEvent.BUTTON3_MASK);
                  set(handles.pushbutton4,'BackgroundColor',[1     0
                  0]); %Se solicita que el elemento gráfico de la
                  GUIDE cambie a color rojo para poder visualizar
                  esta pulsación

                  set(handles.pushbutton5,'BackgroundColor',[0.94
                  0.94  0.94]); %Se solicita que el elemento de la
                  interfaz gráfica de la derecha vuelva a tono gris

            else %Correspondería al caso de que el objetivo esté a
            la derecha

                  rob.mousePress(InputEvent.BUTTON3_MASK);

                  rob.keyPress(KeyEvent.VK_D); %Se solicita al motor
                  java que simule una pulsación de la tecla  D del
                  controlador de vuelo
```

```
                rob.keyRelease(KeyEvent.VK_D); %Se solicita que se
                libere la pulsación de la tecla

                rob.mouseRelease(InputEvent.BUTTON3_MASK);

                set(handles.pushbutton5,'BackgroundColor',[1     0
                0]); %Se solicita que el elemento gráfico de la
                GUIDE cambie a color rojo para poder visualizar
                esta pulsación

                set(handles.pushbutton4,'BackgroundColor',[0.94
                0.94  0.94]); %Se solicita que el elemento de la
                interfaz gráfica de la izquierda vuelva a tono gris

        end

%Control altitud

        if errory < 0 %Para el control de altitud si errory es
        negativo el objetivo está debajo del dron por lo que la
        acción de control será bajar

                rob.mousePress(InputEvent.BUTTON3_MASK);

                rob.keyPress(KeyEvent.VK_S); %Se solicita al motor
                java que simule una pulsación de la tecla  S del
                controlador de vuelo

                rob.keyRelease(KeyEvent.VK_S); %Se solicita que se
                libere la pulsación de la tecla

                rob.mouseRelease(InputEvent.BUTTON3_MASK);

                set(handles.pushbutton6,'BackgroundColor',[1     0
                0]); %Se solicita que el elemento gráfico de la
                GUIDE cambie a color rojo para poder visualizar
                esta pulsación

                set(handles.pushbutton3,'BackgroundColor',[0.94
                0.94  0.94]); %Se solicita que el elemento de la
                interfaz gráfica W vuelva a tono gris

        else %Para el caso de que errory sea positivo, el objetivo
        se encuentra arriba del dron, por lo que la acción de
        control será subir

                rob.mousePress(InputEvent.BUTTON3_MASK);

                rob.keyPress(KeyEvent.VK_W); %Se solicita al motor
                java que simule una pulsación de la tecla  W del
                controlador de vuelo

                rob.keyRelease(KeyEvent.VK_W); %Se solicita que se
                libere la pulsación de la tecla

                rob.mouseRelease(InputEvent.BUTTON3_MASK);

                set(handles.pushbutton3,'BackgroundColor',[1     0
                0]); %Se solicita que el elemento gráfico de la
```

```
            GUIDE cambie a color rojo para poder visualizar
            esta pulsación

            set(handles.pushbutton6,'BackgroundColor',[0.94
            0.94 0.94]); %Se solicita que el elemento de la
            interfaz gráfica S vuelva a tono gris

        end

    %Control proximidad

        if area > 2500 % Se define un área máxima de 2500 píxeles
        cuadrados que para el Hubsan equivale aproximadamente a
        tres metros, si el área del rostro es menor, la acción
        de control será que el dron se acerque

            rob.mousePress(InputEvent.BUTTON3_MASK);

            rob.keyPress(KeyEvent.VK_UP);  %Se  solicita  al
            motor java que simule una pulsación de la tecla UP
            (equivalente a forward) del controlador de vuelo

            rob.keyRelease(KeyEvent.VK_UP); %Se solicita que
            se libere la pulsación de la tecla

            rob.mouseRelease(InputEvent.BUTTON3_MASK);

            set(handles.pushbutton9,'BackgroundColor',[1    0
            0]); %Se solicita que el elemento gráfico de la
            GUIDE cambie a color rojo para poder visualizar
            esta pulsación

            set(handles.pushbutton8,'BackgroundColor',[0.94
            0.94 0.94]); %Se solicita que el elemento de la
            interfaz gráfica 2 vuelva a tono gris

        else %Para el caso que el área del rostro detectado sea
        mayor que 2500 píxeles cuadrados, implica que el dron
        está muy cerca, por lo que la acción de control será
        alejarse

            rob.mousePress(InputEvent.BUTTON3_MASK);
            rob.keyPress(KeyEvent.VK_DOWN);  %Se  solicita  al
            motor java que simule una pulsación de la tecla
            DOWN (equivalente a backward) del controlador de
            vuelo

            rob.keyRelease(KeyEvent.VK_DOWN); %Se solicita que
            se libere la pulsación de la tecla

            rob.mouseRelease(InputEvent.BUTTON3_MASK);

            set(handles.pushbutton8,'BackgroundColor',[1    0
            0]); %Se solicita que el elemento gráfico de la
            GUIDE cambie a color rojo para poder visualizar
            esta pulsación

            set(handles.pushbutton9,'BackgroundColor',[0.94
            0.94 0.94]); %Se solicita que el elemento de la
            interfaz gráfica 8 vuelva a tono gris
```

```
        end

    %Control de orientación 25% 75%

        if centroidex < (s(2)*0.25) %Para el control de
        orientación yaw se define que si el objetivo se encuentra
        en una región menor que el 25% de los píxeles en
        horizontal, el objetivo está saliendo del rango de la
        imagen por la izquierda por lo que la acción de control
        es rotar a la izquierda

            rob.mousePress(InputEvent.BUTTON3_MASK);

            rob.keyPress(KeyEvent.VK_LEFT); %Se solicita al
            motor java que simule una pulsación de la tecla
            LEFT equivalente a un giro hacia la izquierda del
            controlador de vuelo

            rob.keyRelease(KeyEvent.VK_LEFT); %Se solicita que
            se libere la pulsación de la tecla

            rob.mouseRelease(InputEvent.BUTTON3_MASK);

            set(handles.pushbutton10,'BackgroundColor',[1    0
            0]); %Se solicita que el elemento gráfico de la
            GUIDE cambie a color rojo para poder visualizar
            esta pulsación

            set(handles.pushbutton11,'BackgroundColor',[0.94
            0.94  0.94]); %Se solicita que el elemento de la
            interfaz gráfica 6 vuelva a tono gris

        elseif centroidex > (s(2)*0.75) %Para el control de
        orientación yaw se define que si el objetivo se encuentra
        en una región mayor que el 75% de los píxeles en
        horizontal, el objetivo está saliendo del rango de la
        imagen por la izquierda por lo que la acción de control
        es rotar a la izquierda

            rob.mousePress(InputEvent.BUTTON3_MASK);

            rob.keyPress(KeyEvent.VK_RIGHT); %Se solicita al
            motor java que simule una pulsación de la tecla
            RIGHT equivalente a un giro hacia la izquierda del
            controlador de vuelo

            rob.keyRelease(KeyEvent.VK_RIGHT); %Se solicita
            que se libere la pulsación de la tecla

            rob.mouseRelease(InputEvent.BUTTON3_MASK);

            set(handles.pushbutton11,'BackgroundColor',[1    0
            0]); %Se solicita que el elemento gráfico de la
            GUIDE cambie a color rojo para poder visualizar
            esta pulsación

            set(handles.pushbutton10,'BackgroundColor',[0.94
            0.94  0.94]); %Se solicita que el elemento de la
            interfaz gráfica 4 vuelva a tono gris
```

```
        else

              set(handles.pushbutton10,'BackgroundColor',[0.94
              0.94 0.94]);

              set(handles.pushbutton11,'BackgroundColor',[0.94
              0.94 0.94]);

        end
        k=0;

end
```

Cabe recalcar que la implementación del evento MousePress se realizó solo en el código del dron Hubsan H507a debido a que en ese caso la consola virtual se ejecuta de manera paralela como ventana, en cambio para el Parrot AR. 2.0 no fue necesario ya que el software de pilotaje AR. 2 PC Flight mediante el software fdshow funciona de manera similar a la GUIDE ejecutando un frame sobre el otro, es así que las dos ventanas pueden permanecer abiertas simultáneamente.

Finalmente el elemento k empleado en un lazo if al inicio del lazo del controlador se implementó con la intención de ejecutar un determinado número de acciones de control por cada cierto rostro detectado, como se muestra a continuación:

```
k=k+1;

if (cont==1)

      if k>5

            …
      end
```

El efecto que tiene la implementación de este lazo adicional es permitir que se ejecuten las acciones de control cada cierto número de frames, lo cual se realizó debido a que ejecutar demasiados comandos que no puedan ser acatados con la suficiente velocidad por el dron, podría generar un fallo en el vuelo. Es así, que mediante k, si el procesamiento se dio a 30 frames por segundo, el dron ejecutará 6 lasos de control en un segundo, pudiéndose generar 24 acciones de control en un segundo para las variables de control principales.

3.2. Pruebas y Resultados

Las pruebas realizadas empleando los códigos antes descritos para los dos drones seleccionados, se pueden separar en dos etapas fundamentales que se refieren a las

pruebas del algoritmo de detección y su vinculación con el software de pilotaje, y posteriormente las pruebas de vuelo.

Para las pruebas de funcionamiento del algoritmo se emplearon las consolas virtuales Nox Player y Bluestacks, siendo Nox Player la que mejores resultados presentó ya que su ejecución en Windows es más fluida, y requiere menor costo computacional. De esta manera, considerando que los algoritmos de procesamiento de imágenes y detección tienen un elevado costo computacional, en varias de las pruebas realizadas Bluestacks se vio forzado a cerrarse, en cambio mediante Nox Player no se tuvieron inconvenientes en ninguno de los ordenadores empleados (Toshiba Satellite S55-C5161 y MSI MS-1799).

Durante las pruebas de funcionamiento del algoritmo en la PC y sincronización se encontró que empleando la PC Toshiba Satellite S55-C5161 se puede realizar la detección de 10 a 15 rostros de manera fluída para el dron Hubsan H507a lo que desemboca en un rango de acciones de control de hasta 60 acciones de control por segundo para las cuatro variables de control (z, roll, pitch y yaw).

Mientras para el caso de la PC MSI MS-1799 mediante las pruebas de funcionamiento del algoritmo se determinó que es posible realizar la detección de 10 a 30 rostros por segundo de manera fluida, lo que representa un total de aproximadamente 40 a 120 acciones de control sobre las cuatro variables de control de vuelo del dron. Lo que determina una detección y seguimiento exitoso sobre el objetivo a detectar definido mediante el entrenamiento del clasificador. La figura 3.5. muestra un ejemplo de las pruebas realizadas para el algoritmo con el dron aterrizado.

Figura 3.5. Resultados de detección y pilotaje del dron obtenidos en las pruebas del Algoritmo.
(Fuente: elaboración propia)

De esta manera el protocolo de pruebas realizado comprendió principalmente en la sincronización de los controles del dron con la respuesta del mismo y la velocidad de reconocimiento facial del detector de visión artificial.

Inicialmente se configuran los parámetros del controlador mediante las variables $vid.FrameGrabInterval$, k y p, correspondientes al intervalo de cada cuantos frames se reconoce un rostro, el contador k que permite configurar cada cuantos rostros se toma una acción de control de vuelo, y la pausa p que permite que el evento java robot mantenga pulsada la tecla simulando un touch sostenido.

En una primera fase se realizó las pruebas de detección y acción del controlador sobre la interfaz Android empleando solo la cámara del dron sin que este despegue del piso, de esta manera empleando la PC Toshiba Satellite S55-C5161 se obtuvo los resultados detallados en la tabla 3.1.

Tabla 3.1. Pruebas de detección y acción de control sobre la interfaz Android mediante la cámara del dron sin vuelo empleando la PC Toshiba Satellite S55-C5161.

Prueba número	Video Grab Interval	k	p	Tiempo de detección (s)	Tiempo de reacción del controlador (s)	Tiempo total (s)	Uso CPU (%)	Memoria (%)
1	2	3	0,20	0,06	0,20	0,46	15	67
2			0,40	0,06	0,20	1,20	16	68
3			0,60	0,06	0,20	0.86	17	70
4		5	0,20	0,06	0,33	0,59	14	67
5			0,40	0,06	0,33	0,79	15	70
6			0,60	0,06	0,33	0,99	15	71
7		10	0,20	0,06	0,66	0,92	12	66
8			0,40	0,06	0,66	1,12	13	68
9			0,60	0,06	0,66	1,32	13	69
10	3	3	0,20	0,10	0,30	0,60	13	68
11			0,40	0,10	0,30	0,80	14	68
12			0,60	0,10	0,30	1,00	15	68
13		5	0,20	0,10	0,50	0,80	16	68
14			0,40	0,10	0,50	1,00	16	69
15			0,60	0,10	0,50	1,20	17	69
16		10	0,20	0,10	1,00	1,30	15	67
17			0,40	0,10	1,00	1,50	15	67
18			0,60	0,10	1,00	1,70	15	68
19	5	3	0,20	0,16	0,50	0,86	13	68
20			0,40	0,16	0,50	1,06	12	66
21			0,60	0,16	0,50	1,26	13	67
22		5	0,20	0,16	0,83	1,19	12	68
23			0,40	0,16	0,83	1,39	11	67
24			0,60	0,16	0,83	1,59	10	69

Continuación de la tabla 3.1.

25			0,20	0,16	1,66	2,02	12	68
26		10	0,40	0,16	1,66	2,22	10	69
27			0,60	0,16	1,66	2,42	13	69

(Fuente: elaboración propia)

La figura 3.6. muestra los porcentajes de uso de la CPU y la RAM para las diferentes pruebas que se indican en la tabla 3.1.

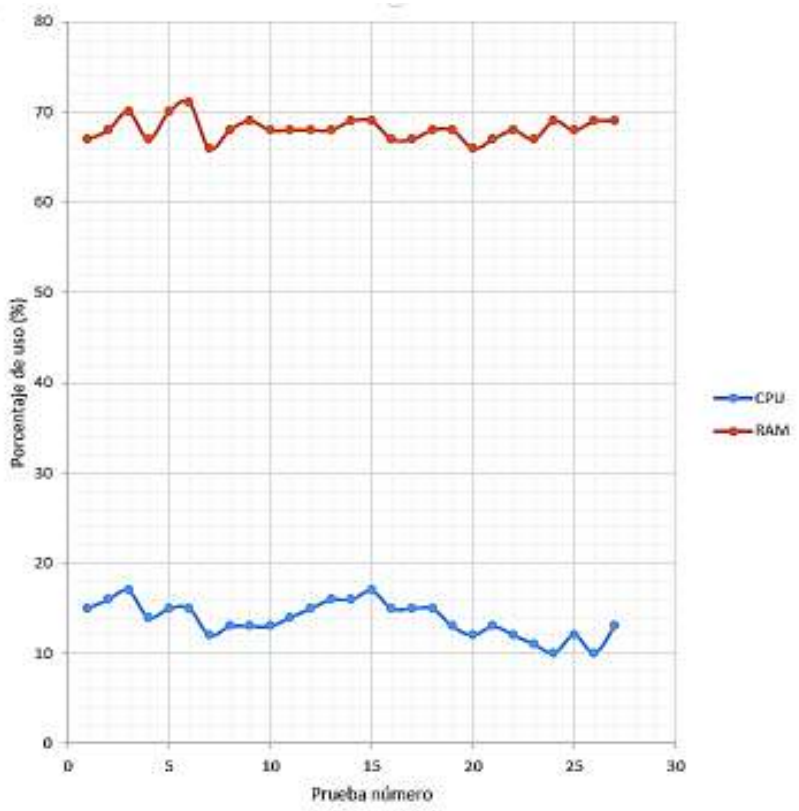


Figura 3.6. Porcentajes de uso de la CPU y la RAM para la PC Toshiba Satellite S55-C5161 con el dron aterrizado.

(Fuente: elaboración propia)

Posteriormente se realizaron las pruebas de detección y acción del controlador sobre la interfaz Android ejecutándose simultáneamente el programa de pilotaje con el dron en vuelo, de esta manera, empleando la PC Toshiba Satellite S55-C5161 se obtuvieron los resultados detallados en la tabla 3.2.

Tabla 3.2. Pruebas de detección y acción de control sobre la interfaz Android mediante la cámara del dron en vuelo empleando la PC Toshiba Satellite S55-C5161.

Prueba número	Video Grab Interval	k	p	Tiempo de detección (s)	Tiempo de reacción del controlador (s)	Tiempo total (s)	Uso CPU (%)	Memoria (%)
1	2	3	0,20	0,06	0,20	0,46	19	82
2			0,40	0,06	0,20	1,20	21	84
3			0,60	0,06	0,20	0.86	22	81
4		5	0,20	0,06	0,33	0,59	22	82
5			0,40	0,06	0,33	0,79	18	81
6			0,60	0,06	0,33	0,99	21	82
7		10	0,20	0,06	0,66	0,92	25	84
8			0,40	0,06	0,66	1,12	25	83
9			0,60	0,06	0,66	1,32	25	82
10	3	3	0,20	0,10	0,30	0,60	14	84
11			0,40	0,10	0,30	0,80	15	84
12			0,60	0,10	0,30	1,00	13	82
13		5	0,20	0,10	0,50	0,80	13	84
14			0,40	0,10	0,50	1,00	12	81
15			0,60	0,10	0,50	1,20	14	83
16		10	0,20	0,10	1,00	1,30	15	83
17			0,40	0,10	1,00	1,50	16	81
18			0,60	0,10	1,00	1,70	15	80
19	5	3	0,20	0,16	0,50	0,86	14	84
20			0,40	0,16	0,50	1,06	12	83
21			0,60	0,16	0,50	1,26	16	84
22		5	0,20	0,16	0,83	1,19	15	83
23			0,40	0,16	0,83	1,39	13	83
24			0,60	0,16	0,83	1,59	14	82
25		10	0,20	0,16	1,66	2,02	13	82
26			0,40	0,16	1,66	2,22	11	82
27			0,60	0,16	1,66	2,42	12	81

(Fuente: elaboración propia)

La figura 3.7. muestra los porcentajes de uso de la CPU y la RAM en porcentaje para las diferentes pruebas que se indican en la tabla 3.2.

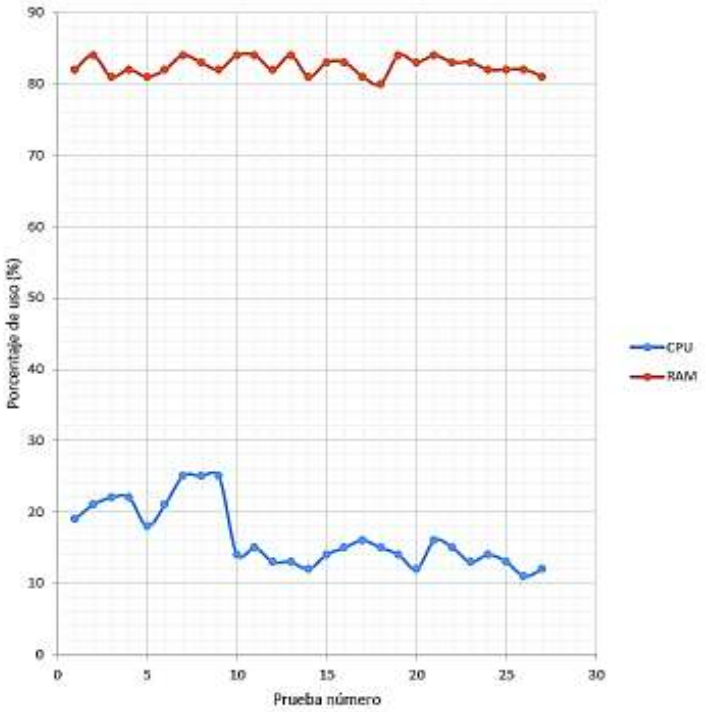


Figura 3.7. Porcentajes de uso de la CPU y la RAM para la PC Toshiba Satellite S55-C5161 con el dron en vuelo.

(Fuente: elaboración propia)

En la segunda fase se realizó las pruebas de detección y acción del controlador sobre la interfaz Android empleando solo la cámara del dron sin que este despegue del piso, de esta manera empleando la PC MSI MS-1799 se obtuvieron los resultados detallados en la tabla 3.3.

Tabla 3.3. Pruebas de detección y acción de control sobre la interfaz Android mediante la cámara del dron sin vuelo empleando la PC MSI MS-1799.

Prueba número	Video Grab Interval	k	p	Tiempo de detección (s)	Tiempo de reacción del controlador (s)	Tiempo total (s)	Uso CPU (%)	Memoria (%)
1	2	3	0,20	0,06	0,20	0,46	36	26
2			0,40	0,06	0,20	1,20	35	26
3			0,60	0,06	0,20	0.86	36	26
4		5	0,20	0,06	0,33	0,59	38	26
5			0,40	0,06	0,33	0,79	37	25
6			0,60	0,06	0,33	0,99	37	24
7		10	0,20	0,06	0,66	0,92	35	27

Continuación de la tabla 3.3.

8			0,40	0,06	0,66	1,12	33	27
9			0,60	0,06	0,66	1,32	34	28
10	3	3	0,20	0,10	0,30	0,60	29	27
11			0,40	0,10	0,30	0,80	27	27
12			0,60	0,10	0,30	1,00	28	26
13		5	0,20	0,10	0,50	0,80	25	26
14			0,40	0,10	0,50	1,00	25	25
15			0,60	0,10	0,50	1,20	25	26
16		10	0,20	0,10	1,00	1,30	28	26
17			0,40	0,10	1,00	1,50	28	24
18			0,60	0,10	1,00	1,70	27	25
19	5	3	0,20	0,16	0,50	0,86	19	26
20			0,40	0,16	0,50	1,06	19	26
21			0,60	0,16	0,50	1,26	18	27
22		5	0,20	0,16	0,83	1,19	18	26
23			0,40	0,16	0,83	1,39	19	26
24			0,60	0,16	0,83	1,59	17	26
25		10	0,20	0,16	1,66	2,02	16	26
26			0,40	0,16	1,66	2,22	15	25
27			0,60	0,16	1,66	2,42	17	27

(Fuente: elaboración propia)

La figura 3.8. muestra los porcentajes de uso de la CPU y la RAM en porcentaje para las diferentes pruebas que se indican en la tabla 3.3.

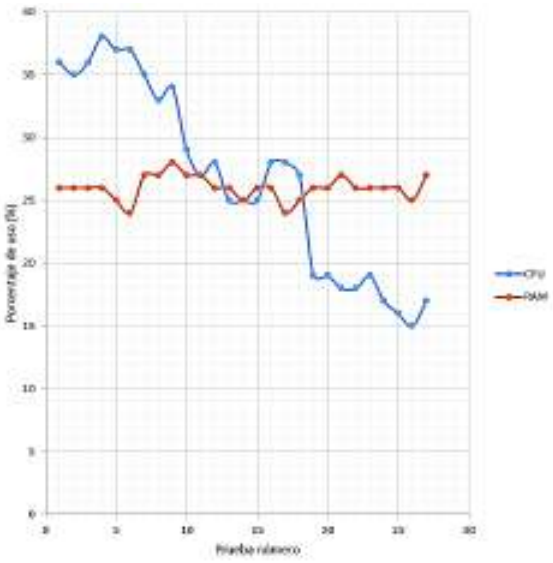


Figura 3.8. Porcentajes de uso de la CPU y la RAM para la MSI MS-1799 con el dron aterrizado.

(Fuente: elaboración propia)

Posteriormente se realizaron las pruebas de detección y acción del controlador sobre la interfaz Android ejecutándose simultáneamente el programa de pilotaje con el dron en vuelo, de esta manera empleando la PC MSI MS-1799 se obtuvieron los resultados detallados en la tabla 3.4.

Tabla 3.4. Pruebas de detección y acción de control sobre la interfaz Android mediante la cámara del dron en vuelo empleando la PC MSI MS-1799.

Prueba número	Video Grab Interval	k	p	Tiempo de detección (s)	Tiempo de reacción del controlador (s)	Tiempo total (s)	Uso CPU (%)	Memoria (%)
1	2	3	0,20	0,06	0,20	0,46	59	38
2			0,40	0,06	0,20	1,20	59	37
3			0,60	0,06	0,20	0.86	60	39
4		5	0,20	0,06	0,33	0,59	36	38
5			0,40	0,06	0,33	0,79	35	38
6			0,60	0,06	0,33	0,99	36	37
7		10	0,20	0,06	0,66	0,92	38	38
8			0,40	0,06	0,66	1,12	38	37
9			0,60	0,06	0,66	1,32	39	36
10	3	3	0,20	0,10	0,30	0,60	34	37
11			0,40	0,10	0,30	0,80	31	37
12			0,60	0,10	0,30	1,00	33	38
13		5	0,20	0,10	0,50	0,80	32	38
14			0,40	0,10	0,50	1,00	32	37
15			0,60	0,10	0,50	1,20	33	38
16		10	0,20	0,10	1,00	1,30	31	38
17			0,40	0,10	1,00	1,50	31	36
18			0,60	0,10	1,00	1,70	30	39
19	5	3	0,20	0,16	0,50	0,86	21	37
20			0,40	0,16	0,50	1,06	21	39
21			0,60	0,16	0,50	1,26	19	38
22		5	0,20	0,16	0,83	1,19	22	38
23			0,40	0,16	0,83	1,39	23	37
24			0,60	0,16	0,83	1,59	21	38
25		10	0,20	0,16	1,66	2,02	20	36
26			0,40	0,16	1,66	2,22	19	36
27			0,60	0,16	1,66	2,42	20	35

(Fuente: elaboración propia)

La figura 3.9. muestra los porcentajes de uso de la CPU y la RAM en porcentaje para las diferentes pruebas que se indican en la tabla 3.4.

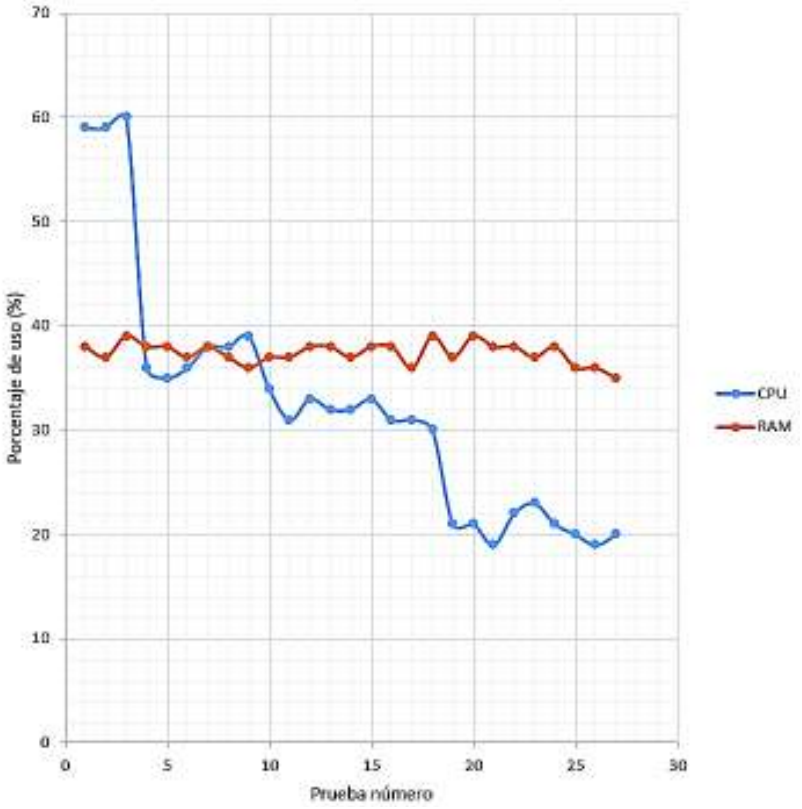


Figura 3.9. Porcentajes de uso de la CPU y la RAM para la PC MSI MS-1799 con el dron en vuelo.

(Fuente: elaboración propia)

De las pruebas realizadas con el dron aterrizado y en vuelo, se evidencia que el uso paralelo del software de pilotaje involucra un incremento considerable del costo computacional.

Además, al emplear las herramientas $vid.FrameGrabInterval$, k y p, incluídas en el código del sistema implementado se tiene una considerable disminución del costo computacional y mejora en la fluidez del algoritmo de detección.

Finalmente, se efectuaron pruebas de vuelo que entregaron excelentes resultados para las pruebas de seguimiento, detección y ejecución del controlador. En la figura 3.10. se visualiza la realización de las pruebas de vuelo efectuadas para el dron HubsanH507a.

Figura 3.10. Resultados obtenidos durante las pruebas de vuelo efectuadas.

(Fuente: elaboración propia)

3.3. Comparativa de costos del sistema implementado

En el presente proyecto se desarrolló e implementó un sistema de visión artificial en dos drones comercial de bajo costo, esto es el AR. Drone 2.0 de Parrot y el Hubsan H507a. Pese a que el software fue implementado tanto en el AR. Drone 2.0 como en el Hubsan H507a, cada uno presenta características propias, es así, que el primero de ellos cuenta con una cámara con mejor resolución, mientras que el segundo dron tiene una excelente estabilidad. Además, de la ventaja antes mencionada, el dron Hubsan H507a es mucho más económico respecto al de la marca Parrot, el precio en el mercado es de aproximadamente USD 80 para el primero y alrededor de USD 200 para el segundo, por lo que económicamente resulta más viable la implementación del sistema de visión artificial en el Hubsan H507a, aunque cabe recalcar que el sistema de visión artificial para seguimiento de objetivos humanos puede ser implementado en cualquier tipo de dron comercial.

En el mercado existen drones con la función “follow me” que son capaces de dar seguimiento a un objetivo humano, sin embargo para que cumplan con la función de seguimiento es imprescindible que la persona lleve consigo el dispositivo al cual va a dar seguimiento el dron así como su mando de control, contrario al sistema desarrollado en el presente proyecto donde no se requiere de un dispositivo adicional para que el dron realice

el seguimiento del objetivo. Por otro lado, el costo de los drones que cuentan con esta función bordean los USD 1000 como muestra la tabla 3.5.

Si se tiene en cuenta que la inversión inicial es el costo del dron de USD 80 más un costo de desarrollo del sistema de visión artificial de USD 6400 en dos meses para 2 ingenieros especialistas en el área de investigación, por lo tanto se estima que el costo total de desarrollo del proyecto es aproximadamente USD 6480. Hay que puntualizar que el costo de desarrollo se puede implementar a los mismos drones teniendo un margen de utilidad significativo. Una empresa puede incrementar el costo de cada dron en un 10% del valor inicial implementando este sistema, costo que no sería para nada elevado comparado con los beneficios que podría tener al adquirir un dron con capacidad de reconocimiento de objetivos humanos y sus muchas ventajas, sean para seguridad o a su vez para seguir desarrollando investigaciones en el campo de la visión artificial enfocado en el procesamiento de imágenes para el reconocimiento de objetivos.

Tabla 3.5. Tabla comparativa de drones que cuentan con la función “follow me”.

Drones	Autonomía de vuelo	Auto-Follow me	Precio
Staaker	30 min	sí	1.800$, 1.150$ (preorder)
3D Robotics Solo Drone	20 min	sí	799$ (sin la GoPro)
Ghost Drone	30 min sin el gimbal ni cámara/20 min con ellos	sí	769$(por la web del fabricante)
3D Robotics Iris + Quadcopter	17 a 22 min	sí	784 dólares canadienses (604$) (Amazon Canadá)
Hexo+	10 a 15 min	sí	613$ (por Amazon UK)
Airdog – Follow me Drone	10 a 20 min ,dependiendo de su velocidad	sí	1.599$ (por Amazon
The ONAGOfly Drone	15 min sin cámara, 12 min si está filmando	sí	129$ (De momento por ebay)
Lily Camera Drone	20 min	sí	244$(De momento por ebay)
PlexiDrone	15 a 35 min	sí	1672$ (por videpan, con todo incluido, cámara y otros accesorios)
Yuneec Typhoon H Drone	25 min	sí	1.128$ (por Amazon)

(Fuente: https://bit.ly/2GQsypZ)

Es así, que el dron Hubsan H507a con el sistema implementado de visión artificial para el

seguimiento de objetivos humanos sin la necesidad de un dispositivo adicional al cual dé seguimiento el dron podría venderse en USD 88 con el margen de utilidad planteado del 10 %, siendo aún mucho más barato que los drones comerciales presentados en la tabla 3.5.

Por lo que la empresa encargada de la fabricación y venta de los drones podría recuperar la inversión inicial con la venta de tan solo 810 drones que resulta de la inversión inicial dividido para el porcentaje de ganancia (USD 8), y de esta manera empezar a generar grandes utilidades por la implementación del software de visión artificial en sus drones, en un mercado en gran auge y mucho más exigente. Se puede asegurar rotundamente que este dron con el sistema implementado puede tener gran acogida porque ningún dron disponible en el mercado cuenta con el sistema desarrollado en este presente proyecto, además, se conoce que en el año 2017 se tuvieron ventas de aproximadamente 3 millones de drones comerciales a nivel mundial, por lo que se garantiza que el mercado meta tiene gran acogida entre los consumidores de esta tecnología.

4. CONCLUSIONES

4.1. Conclusiones

- Se logró cumplir con el objetivo fundamental del proyecto al implementar un sistema de visión artificial y seguimiento de objetivos humanos en dos tipos de drones comerciales de bajo costo: el AR. Drone 2.0 de Parrot y el Hubsan H507a, los cuales lograron realizar el reconocimiento de rostros humanos mediante la ejecución de algoritmos de procesamiento de imágenes digitales que se desarrollaron en el software Matlab.
- El desarrollo del algoritmo para el seguimiento y detección de objetivos humanos pudo procesar imágenes con una capacidad de 10 a 30 rostros por segundo, y esto permitió que se ejecuten de 24 hasta 120 acciones de control por cada segundo, concluyendo que el sistema brinda una amplia gama de posibilidades de control para los dispositivos en los que se implementa el sistema de visión artificial.
- La implementación del controlador de vuelo para el dron se pudo realizar a través de técnicas de modelado por espacio de estados, ya que el AR. Drone 2.0 de Parrot cuenta con un procesador a bordo. En el caso de drones comerciales que no cuentan con procesadores como el Hubsan H507a se pudo implementar el controlador a través de estructuras boleanas sincronizadas con el software de pilotaje del fabricante, cuyo código fue desarrollado en Matlab.
- De las pruebas de funcionamiento se pudo determinar que la velocidad de captura de los dispositivos de video que se disponen en la actualidad están en el orden de 30 a 60 FPS. Por lo tanto, se puede concluir que la cámara a bordo del dron funciona como un sensor que obtiene información hasta 60 veces por segundo, lo que representa una gran ventaja para la toma de decisiones al realizar acciones de control sobre cualquier dispositivo. Es así que, en los drones empleados en el presente proyecto se logró generar de 24 a 120 acciones de control que permitieron obtener un vuelo estable y un correcto seguimiento del rostro humano.
- En cuanto a Matlab, que fue el software utilizado para el desarrollo del sistema de visión artificial se puede concluir que es eficiente para el desarrollo de algoritmos de visión artificial para el reconocimiento de objetivos ya que presenta muchas herramientas incorporadas como el toolbox de adquisición de imágenes, el vision object detector, entre otros, que están prestos para ser entrenados e implementados, entregando muy bueno resultados. En contra parte un pequeño problema que presenta el software es que se ejecuta como una aplicación paralela

al sistema operativo lo que implica un costo computacional mayor para la PC, por lo que sería ideal desarrollar en futuros proyectos, un software que se ejecute sin necesidad de un sistema operativo.

- Mediante el desarrollo del presente proyecto se pudo concluir que el sistema de visión artificial para el reconocimiento de objetivos humanos puede ser implementado en drones comerciales de bajo costo, pudiendo tener sistemas de reconocimiento eficaces a costos asequibles para aplicaciones de investigación e industriales.

4.2. Recomendaciones

- Se recomienda que en futuros proyectos, cuando los Smartphones cuenten con la capacidad de procesamiento necesaria, se trabaje directamente en las plataformas Android, lo que sería ideal debido a que los softwares de pilotaje proporcionados por los fabricantes están desarrollados para esta plataforma. En el presente proyecto no se pudo implementar el sistema de visión artificial directamente en un Smartphone ya que el procesamiento de imágenes requiere un considerable costo computacional, y los Smartphones aún no son capaces de ejecutar esta tarea.
- Para futuros trabajos de implementación de sistemas de visión artificial se recomienda emplear cámaras térmicas para que la detección de objetivos no dependan del nivel de luminosidad, pudiendo trabajarse en aplicaciones cuyo uso sea en entornos poco iluminados como por ejemplo en la noche.
- Se recomienda que en el desarrollo de futuros trabajos se implemente métodos de adquisición de imágenes propios de Matlab, ya que fue una limitante para el presente proyecto al no existir toolboxes que trabajen directamente con protocolos de comunicación UDP o TCP, así como tampoco se cuenta con algoritmos de adquisición de imagen y video en formatos M-JPEG o H.264.
- Por otro lado el desarrollo del presente proyecto puede servir de base para el desarrollo de aplicaciones industriales donde se requiera una rápida ejecución de las acciones de control, dependiendo del proceso en el que se esté implementando el sistema.
- En cuanto a Matlab se recomienda optimizar el número de líneas de código para que el software no se vuelva muy pesado computacionalmente hablando, ya que la mayoría de funciones empleadas para el procesamiento de las imágenes, implican la ejecución paralela de todo un código en segundo plano.

Referencias Bibliográficas

1] Ahmed E. (2010). *Computer Vision 3D Model-based recognition.* Obtenido de https://www.cs.rutgers.edu/~elgammal/classes/cs534/lectures/3D_modelbasedvision.pdf

2] Banda H. (2014). *Inteligencia Artificial: Principios y Aplicaciones.* Quito: Escuela Politécnica Nacional.

3] Banggood. (2018). *Tienda de artículos tecnológicos.* Obtenido de https://www.banggood.com/es/Hubsan-X4-STAR-H507A-App-Compatible-Wifi-FPV-With-1080P-HD-Camera-GPS-RC-Quadcopter-RTF-p-1115752.html?ID=520278&cur_warehouse=CN

4] Calderón M., M. D. (2014). *Control por visión de un cuadricóptero utilizando ROS (Tesis de pregrado).* Quito: Escuela Politécnica Nacional.

5] Díaz H. (2014). *Diseño e implementación de un sistema de teleoperación y evasión de obstáculos en un robot móvil mediante el uso del entorno ROS (Tesis de pregrado).* Quito: Escuela Politécnica Nacional.

6] *Drones que te siguen: La revolución de los mini drones.* (2016). Obtenido de http://www.minidrons.com/mini-drones-drones-te-siguen/

7] *Especificaciones Técnicas, AR. Drone 2.0.* (2018). Obtenido de http://ardrone-2.es/especificaciones-ar-drone-2/

8] *Features and Image Matching.* (s.f.). Obtenido de https://courses.cs.washington.edu/courses/cse455/09wi/Lects/lect6.pdf

9] García-Carrillo, L.R., Dzul-López, A.E, Lozano, R., (2013). Modeling the quad-rotor minirotorcraft. *Quad Rotorcraft Control, Advances in Industrial Control* (págs. 23-24). Londres: Springer.

10] Matlab. (1999). *Image Processing Toolbox for Using with Matlab.* The Math Works Inc.

11] Moeslund, T. B. (2012). *Introduction to Video and Image Processing: Building Real Systems and Applications.* London: Springer.

12] Ottado, G. (2015). *Reconocimiento de caras: Eigenfaces y Fisherfaces.*

13] Parrot SA. (2018). *Sitio Web Oficial de Parrot.* Obtenido de https://www.parrot.com/es/drones/parrot-ardrone-20-elite-edition

14] Passino, K. M. (2015). *Intelligent Control: An overview of techniques.* Columbus: The Ohio State University.

15] PC Componentes y multimedia SLU. (s.f.). *Tienda de Portátiles.* Obtenido de 1] https://www.pccomponentes.com/msi-gp72-7rd-081es-leopard-intel-core-i7-7700hq-16gb-1tb-256ssd-gtx1050-173

16] Peña A. (2010). *Módulo de visión artificial del robot humanoide HOAP3. Aplicación al seguimiento de objetivos móviles (Tesis de pregrado).* Madrid: Universidad Carlos III de Madrid.

17] Peña M., V. E. (2010). *Modelamiento dinámico y control LQR de un Quadrotor.* AVANCES Investigación en Ingeniería.

18] Ponce, P. (2010). *Inteligencia artificial con aplicaciones a la ingeniería.* México: Alfaomega Grupo Editor, S.A. de C.V.

19] Raffo G. (2007). *Modelamiento y Control de un Helicóptero Quadrotor.* Sevilla: Universidad de Sevilla (Tesis de Máster).

20] Rossius, S. (2013). *Reconocimiento de objetos mediante WebCam en tiempo real.* Obtenido de https://riunet.upv.es/bitstream/handle/10251/29009/Reconocimiento%20de%20objetos%20en%20tiempo%20real%20mediante%20WebCam.pdf?sequence=1

21] Santiaguillo J., R. M. (2017). *Observer-based Time-varying Backstepping Control for Parrot's AR. Drone 2.0.* International Federation of Automatic Control.

22] Santiaguillo-Salinas, J., Rosaldo-Serrano, M., & Aranda-Bricaire, E. (20117). Observed-based Time-varying Backstepping Control for Parrot's AR. Drone 2.0. *ScienceDirect* (pág. 2). México: ELSEVIER.

23] Sobrado, E. (2003). *Sistema de visión artificial para el reconocimiento y manipulación de objetos utilizando un brazo robot.* Lima: Pontificia Universidad Católica del Perú.

24] Universidad de Valladolid. (s.f.). *Visión Artificial Industrial.* Obtenido de http://slideplayer.es/slide/1737320/Filtrado

25] Viola P., J. M. (2001). *Rapid Object Detection using a Boosted Cascade of Simple.* Cambridge: Cambridge University.

26] Viola P., M. J. (2004). *The Viola Jones algorithm for face detection.* Obtenido de https://www2.units.it/carrato/didatt/EI_web/slides/ti/72_ViolaJones.pdf

27] ZonaTecno. (2011). *Tienda de portátiles.* Obtenido de http://www.zonatecno.com.uy/ProductDetail/90487/Notebook%20Toshiba%20Satellite%20S55-C5161%20i7-6700HQ%202-6GHz%20512GB

ANEXOS

Anexo A1. Especificaciones técnicas del AR. Drone 2.0 de Parrot

El AR. Drone 2.0 de Parrot que se muestra en la figura A.1. fue el cuadricóptero en el que se realizaron las pruebas del algoritmo de procesamiento de imágenes digitales del presente proyecto.

Figura A.1. AR. Drone 2.0 de Parrot.
(Fuente: https://bit.ly/2HCBnVs)

Especificaciones técnicas:

- Procesador de 1 GHz 32 bits ARM Cortex A8, video 800 MHz
- Memoria RAM de 1 GB DDR2 a 200 MHz
- Puerto USB 2.0 de alta velocidad
- Cámara HD 720px, 30 FOS
- Giroscopio de 3 ejes con precisión de 2000 grados por cada segundo
- Acelerómetro de 3 ejes con una precisión de ± 50 mg
- Magnetómetro de 3 ejes con una precisión de 6 grados
- Sensor de presión con una precisión de ± 10 Pa
- Altímetros
- Cámara vertical QVGA 60 FPS
- Formato de grabación JPEG
- Conexión WiFi
- 4 motores sin escobilla de 14,5 vatios y 28500 RPM
- Peso aproximado de 380 gramos

Anexo A2. Especificaciones Técnicas del Hubsan H507a

El drone Hubsan H507a que se muestra en la figura A.2. fue el cuadricóptero en el que se realizaron las pruebas del algoritmo de control para sistema de procesamiento de imágenes digitales del presente proyecto ya que cuenta con mayor estabilidad que el AR. Drone 2.0 de Parrot.

Figura A.2. AR. Hubsan H507a.
(Fuente: https://bit.ly/2ERxby6)

Especificaciones técnicas:

- Frecuencia de 2.4 GHz
- Canal 4CH
- Peso aproximado de 162 gramos
- Batería 7.4 voltios, 550 mAh con una corriente máxima de 6 amperios
- Motor sin núcleo
- Cámara 720p HD, FOV de 85°
- Conexión WiFi de un máximo de radio de 100 m
- Sensor de gravedad
- Sistema de posicionamiento GPS

Anexo A3. Especificaciones de la PC Toshiba Satellite S55-C5161

La computadora Toshiba Satellite S55-C5161 de la figura A3. cuyas principales características se presentan a continuación fue en la que se desarrolló los algoritmos de procesamiento de imágenes.

Figura A.3. Laptop Toshiba Satellite S55-C5161.
(Fuente: https://bit.ly/2IWqnla)

Especificaciones:

- Procesador Intel Core i7 6700HQ 2.6 GHz
- Disco duro en estado sólido SSD de 512 GB
- Memoria RAM de 8 GB DDR3
- Tarjeta de video NVIDIA GeForce GT 950M de 4096 MB
- Sistema Operativo Windows 10
- Resolución de la pantalla 1920 x 1080 Full HD
- Conexión WiFi y Bluetooht
- Puertos USB, HDMI y lector de tarjeta multimedia
- Batería 4 celdas Litio-Ion

Anexo A4. Especificaciones de la PC MSI GP72 7RD Leopard 3168NGW

La computadora MSI GP72 7RD Leopard 3158NGW de la figura A.4. cuyas principales características se presentan a continuación fue en la que se realizaron las pruebas del controlador para el seguimiento de objetivos del presente proyecto.

Figura A.4. Laptop Toshiba Satellite S55-C5161.
(Fuente: https://bit.ly/2H3qEq8)

Especificaciones:

- Procesador Intel Core i7-7700HQ (2.8 Ghz, 4 núcleos), séptima generación
- Memoria RAM 16 GB DDR4 SODIMM (2 x 8 GB)
- Disco duro 1 TB (7200 rpm S-ATA), adicional 256 GB SATA M.2
- Tarjeta gráfica GeForce GTX 1050 2GB GDDR5
- Resolución gráfica 1920 x 1080 Full HD
- Conectividad LAN 10/100/1000 y Bluetooth V4.0 de alta velocidad
- Puerto USD reversible Type-C
- Batería 6 celdas Ion-Litio