SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA
SUBSECRETARÍA DE EDUCACIÓN SUPERIOR
TECNOLÓGICO NACIONAL DE MÉXICO

TECNM
TECNOLÓGICO NACIONAL DE MÉXICO

DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

"Análisis de imágenes en extensiones terrestres utilizando un UAV"

POR

José Manuel Busquets González

TESIS

PRESENTADA COMO REQUISITO PARCIAL PARA OBTENER EL GRADO DE MAESTRO EN CIENCIAS DE LA INGENIERÍA ELÉCTRICA

DIRECTORES DE TESIS

Dr. Alejandro Enrique Dzul López
Dr. Arturo Tadeo Espinoza Fraire

ISSN 0188-9060

TECNOLÓGICO DE LA LAGUNA

RIITEC:(15)-TMCIE-2019

Torreón, Coahuila, México,
Agosto 2019

El comité tutorial de este trabajo está conformado por:

Asesor: Dr. Alejandro Enrique Dzul López
Secretario: Dr. Arturo Tadeo Espinoza Fraire
Vocal: Dr. Héctor Ríos Barajas
Vocal suplente: Dr. Jaime González Sierra

Dedicatoria

Dedicado a mi esposa Maite, a mi madre Angélica y a mis hermanos Angélica y Enrique.

Agradecimientos

A mi esposa *Maite*, por tu amor incondicional, tu sonrisa en los momentos de estrés y tu comprensión. Gracias por todos los momentos que hemos vivido y por los que aún nos faltan por vivir.

A mi madre, *Angélica María* por el amor y apoyo que me ha brindado durante toda mi vida, por sus enseñanzas, su guía, su ejemplo y su esfuerzo para convertirnos a mi y a mis hermanos en las personas que somos.

A mis hermanos *Angélica* y *Enrique* por su apoyo durante todos estos años, su ejemplo y su amistad.

A mis compañeros de maestría, *Ing. Felipe Guerra, Ing. Edgar Martínez, Ing. Andrés Ramírez, M.C. Ricardo Ovalle, M.C. Roberto Franco, M.C. Oscar González, M.C. Omar Martínez, M.C. Romeo Falcón, Ing. Amaury Meza, Ing. Ariana Gutiérrez, Ing. Ulises Vázquez, Ing. Daniel Flores, Ing. Eduardo de la Cruz, Ing. Pablo Rochel*, por su apoyo y su amistad.

Gracias al *Dr. Alejandro Dzul*, por permitirme trabajar con él, por sus consejos durante mis estudios de maestría y la ayuda brindada para la elaboración de este trabajo.

Agradezco a todos los doctores que conforman el posgrado del Instituto, gracias a ellos por compartir su experiencia y sus enseñanzas que me ayudaron a obtener conocimiento para la realización de este documento.

Al Instituto Tecnológico de la Laguna, al Tecnológico Nacional de México y al CONACYT (beca 860663) por el apoyo económico brindado durante la realización de esta tesis.

"Análisis de imágenes en extensiones terrestres utilizando un UAV"

José Manuel Busquets González

Resumen

En este documento se estudian técnicas de procesamiento digital de imágenes para el análisis de las mismas a través de un sistema de visión a bordo de un vehículo aéreo no tripulado tipo multi-rotor. Un algoritmo de detección de objetos en conjunto con un algoritmo de seguimiento de objetos es propuesto, basado en el lenguaje de programación Python y librerías de OpenCV, en las cuales se utiliza una red neuronal para la identificación de objetivos. Este algoritmo es capaz de identificar personas, vacas, caballos y ovejas y, a tráves de la detección de su centroide, realizar un seguimiento y un conteo, tratando de asegurar que un objetivo no sea contado más de una vez. Se propone además el diseño de dos controladores: uno basado en la metodología PID y otro a través de la técnica de Backstepping. Finalmente, los resultados experimentales muestran que el algoritmo aplicado en un Quad-Rotor, permite la detección y conteo de los objetivos a través de vuelos no tripulados.

Keywords: Procesamiento Digital de Imágenes, Visión, OpenCV, UAV, Quad-Rotor, Control de vehículos aéreos.

"Image analysis in terrestrial extensions using a UAV"

José Manuel Busquets González

Abstract

In this document we study digital image processing techniques for the analysis of images through onboard vision systems for multi-rotor unmanned aerial vehicles. An object detection algorithm and an object tracking algorithm are proposed; both algorithms are based on Python and OpenCV libraries in which a neural network is used to identify different types of targets. The algorithm is able to identify people, cows, horses and sheeps and, through the detection of the centroid, track and count, trying to ensure that an objective is not counted more than once. This work also proposes two controllers: one based on the PID method, and another based on the Backstepping control technique. Finally, experimental results show that the algorithm allows the detection and counting of the targets when it is running on a Quad-Rotor flight.

Keywords: Digital Image Processing, Vision, OpenCV, UAV, Quad-Rotor, Aerial Vehicles Control.

Índice general

Índice de figuras

Índice de tablas

Lista de algoritmos

Capítulo 1

Introducción

El avance en la robótica ha impulsado la aplicación de vehículos autónomos para realizar tareas repetitivas o peligrosas [1]. En las últimas décadas, el desarrollo de los vehículos autónomos no tripulados ha sido de gran interés, por lo que diferentes tipos de vehículos han sido estudiados y desarrollados alrededor del mundo. Los UAV´s (del inglés *Unmanned Aerial Vehicles*) tienen un gran número de aplicaciones tales como: búsqueda y rescate en situaciones de emergencia, análisis de terrenos, análisis de infraestructura, mantenimiento de estructuras, sondeo de cultivos, análisis de vegetación, entrega de suministros, entre muchas otras [2].

I. Motivación

La agricultura y la ganadería representan la base de la economía de los países del mundo. Su importancia radica en la capacidad de generar elementos alimenticios de primera necesidad así como materia prima. Es por ello que la inversión en el desarrollo de nuevas tecnologías y técnicas para este sector va cada día en aumento. Los robots juegan un papel importante en el desarrollo de estas tecnologías, pues gracias a su flexibilidad y versatilidad permiten la disminución de costos y tiempo en los procesos agropecuarios. En la actualidad, los vehículos aéreos en conjunto con sistemas de visión se utilizan de forma constante para estas actividades. Una de las de mayor importancia en la ganadería es el conteo de cabezas de ganado. Esta actividad normalmente resulta complicada y en ocasiones ineficiente, debido a que no siempre se logra un conteo adecuado, lo que implica pérdida de tiempo y recursos. Es por ello que este trabajo está enfocado en el desarrollo de algoritmos de visión para la detección y conteo de ganado a través del uso de vehículos aéreos no tripulados, en conjunto con sistemas de visión embedidos.

II. Estado del arte

Hoy en día, los vehículos aéreos no tripulados son considerados como una parte importante en el campo de investigación del control automático debido a su gran cantidad de aplicaciones, tanto comunes como especializadas. En conjunto con sistemas de visión embedidos, estos pueden realizar actividades tales como: análisis de cultivos, fumigación, conteo de ganado, mapeo de terrenos, vigilancia, búsqueda y rescate, inspección de áreas, inspección de estructuras, monitoreo en tiempo real, entre muchas otras. Por otro lado, el estudio de las ecuaciones matemáticas que describen a los vehículos aéreos ha sido ampliamente analizado para encontrar el modelo más simple y cercano a la realidad con el fin de diseñar controladores más eficientes y con mejores respuestas durante la experimentación. Por ejemplo, en [3] se realiza una comparación entre dos modelos dinámicos. En el primero, la dinámica traslacional es obtenida a través del método de Euler-Lagrange y la dinámica rotacional es obtenida a través del método de Newton-Euler. En el segundo modelo, éste se obtiene utilizando los mismos formalismos, con una diferencia en la obtención de la dinámica del movimiento traslacional al cambiar el orden de las matrices de rotación. Los resultados muestran que ambos modelos se comportan de la misma manera para el movimiento angular y la elevación del Quad-Rotor. Sin embargo, el movimiento traslacional muestra un comportamiento opuesto en las posiciones en el eje x y el eje y y sus velocidades. En [4], se presenta el modelado matemático del Quad-Rotor basado en el método Hamiltoniano. Los autores comentan que este método presenta ventajas sobre los métodos de Newton-Euler y Euler-Lagrange tales como estructura más compacta, más simple y una mayor facilidad para diseñar ciertos tipos de controladores; el modelado está basado en una ecuación diferencial de primer orden cuyos estados incluyen la posición y momentos generalizados que contienen información correspondiente a velocidad y masa. Además, se propone un controlador no lineal basado en el modelo Hamiltoniano con dos sub-sistemas, uno para el control de la orientación y otro para el control de posición. En [5], un controlador no lineal de orientación es propuesto, basado en el modelo no lineal del Quad-Rotor representado a través de quaterniones, el cual está sujeto a incertidumbres. Este controlador, determina los polos deseados del sistema de control en lazo cerrado y un compensador robusto atenúa el efecto de las perturbaciones. En [6], se presenta el modelado dinámico y un control para un Quad-Rotor que cuenta con un brazo manipulador de 2 grados de libertad (GDL). Para poder resolver el problema de control de este vehículo, las restricciones no holónomas son utilizadas en el modelo. Para el diseño del controlador robusto, los autores utilizaron el método de control adaptable más un observador para las perturbaciones. En [7], se presenta un esquema de control no lineal basado en el modelo dinámico obtenido a través del método de Euler-Lagrange. Este control, utiliza un método de linealización parecido al de realimentación del backstepping. En [8], un esquema de control robusto basado en backstepping en conjunto con un control por modos deslizantes para el seguimiento de trayectorias sujeto a perturbaciones y con incertidumbres paramé-

tricas en un Quad-Rotor es propuesto. Además, para mejorar la robustez, se propone un observador de perturbaciones no lineal. En [9], se diseña un control por modos deslizantes para el seguimiento de trayectorias basado en un modelo simplificado del Quad-Rotor, el cual es obtenido a través del método de Euler-Lagrange. Este modelo se simplifica a través de aproximaciones considerando variaciones angulares pequeñas para reducir la complejidad de las ecuaciones en el diseño del control. En [10], se propone un control de orden fraccional que impone la convergencia en tiempo finito de la variedad deslizante y garantiza seguimiento local y exponencial sin necesidad de conocer el modelo del vehículo. En [11], se aborda el diseño de un controlador robusto para que un Quad-Rotor realice la maniobra de aterrizaje deslizándose sobre una rampa hasta frenar completamente durante la cual, las dinámicas del vehículo cambian con el tipo de contacto con el suelo. Gran parte de las aplicaciones mencionadas al inicio de este capítulo se basan en controladores apoyados de un sistema de visión, normalmente embedido. Por ejemplo, en [12] se presenta un control basado en visión monocular diseñado para evadir obstáculos en un ambiente forestal. El algoritmo es capaz de estimar distancias con el obstáculo más cercano como varias franjas de la imagen de entrada utilizando una combinación ponderada de características de textura. Además, utiliza una técnica básica de regresión lineal para aprendizaje supervisado. En [13], se propone un algoritmo de visión para la detección y seguimiento de caminos en exteriores con el cual se le proporciona la trayectoria al Quad-Rotor; la técnica de visión está basada en segmentación de superpíxeles. De igual manera, se propone un algoritmo de detección de objetos como piedras, personas o carros basado en redes neuronales. En [14], se realiza el control de un nano Quad-Rotor en ambientes sin señal de GPS y utiliza un sistema de visión para detectar la posición del vehículo. El algoritmo de visión utiliza una técnica modificada de localización y modelado simultáneos (*SLAM*, por sus siglas en inglés) para utilizar la información de una cámara monocular a bordo del vehículo. En [15], se realiza un control adaptable por modos deslizantes basado en una técnica de backstepping para el seguimiento de trayectorias, en conjunto con un algoritmo de visión para obtener la información de posición y de movimiento basado en la técnica de odometría visual monocular semi directa (*SVO*, por sus siglas en inglés). En [16], se utiliza un vehículo aéreo en conjunto un sistema de visión embedido para la estimación de la densidad de plantas de trigo utilizando imágenes RGB. El algoritmo de visión separa los pixeles verdes del fondo para después extraer filas. Luego, el grupo de pixeles detectados que están conectados en cada fila permiten estimar el número de plantas que existen usando un conjunto de algoritmos de aprendizaje supervisado. En [17], se propone un sistema para el sondeo de hierba mala dentro de cultivos de caña de azúcar utilizando un vehículo aéreo en conjunto con un algoritmo de visión con *Machine Learning* y un clasificador llamado Weka RF Classifier para la creación del modelo para entrenar una red neuronal. Para crear el modelo, se utilizaron las técnicas de promedio, desviación promedio, desviación estándar, varianza, cutorsis, oblicuidad y valores máximos y mínimos aplicados a sub-grupos de la imagen con el fin de detectar ciertos rasgos de la hierba. En [18], se propone un algoritmo de visión para el conteo de árboles en zonas urbanas a través de un

sistema de visión embedido en un Quad-Rotor. El algoritmo utiliza una técnica de visión basada en segmentación de imágenes para extraer objetos del fondo y dividir la imagen en varias regiones de interés. Luego, la imagen se satura para obtener una mejor detección del color verde y se realiza la segmentación basada en color utilizando k-medias, para después separar los arboles del jardín en las imágenes a través del análisis de texturas. Por último, se utiliza la técnica de visión llamada aproximación de ajuste de círculo, la cual trata de ajustar la vista superior de los árboles en círculos y posteriormente contarlos.

III. Contribución

En este documento un algoritmo de visión para la detección de y conteo de objetivos es propuesto, basado en redes neuronales del tipo SSD (*Single Shot Multiplebox Detector* por sus siglas en inglés) con la capacidad de detectar vacas, caballos, ovejas y personas. Este algoritmo de visión es capaz de adaptarse para detectar cualquier objeto que se requiera al simplemente implementar una red neuronal distinta a la ya entrenada. Además, un algoritmo de detección de centroides es propuesto y adaptado en el algoritmo de visión para realizar el seguimiento de los objetos detectados, consiguiendo evitar un conteo erróneo. El algoritmo de visión está implementado en la plataforma Odroid-XU4 y fue adaptado para poder realizar conteo de objetivos en tiempo real.

IV. Estructura de la tesis

El Capítulo 2 presenta los preliminares de este trabajo, donde se encuentra el modelado dinámico de un Quad-Rotor, conceptos básicos de procesamiento digital de imágenes y las herramientas utilizadas para la elaboración de los algoritmos de visión. En el Capítulo 3, se presenta el diseño de un controlador PID y un controlador por Backstepping para el Quad-Rotor. En el Capítulo 4, se presenta el diseño del algoritmo de visión así como el diseño del algoritmo de detección de centroides. En el Capítulo 5, se muestran los resultados de simulación de los controladores, así como los resultados experimentales del algoritmo de visión. En el Capítulo 6, se presentan las conclusiones obtenidas de este trabajo así como propuestas de trabajo futuro.

Capítulo 2

Preliminares

En este capítulo se presentan los conceptos básicos utilizados para este trabajo.

I. Vehículos aéreos.

Un robot móvil es un dispositivo dotado de un sistema de locomoción capaz de navegar a través de un ambiente dado así como también de un cierto nivel de autonomía para su desplazamiento [19]. Estos se clasifican en tres tipos: Vehículos aéreos o UAV´s, vehículos terrestres o UGV´s, vehículos acuáticos o UUV´s [20], (Figura 2.1). Para fines de este trabajo, sólo serán tratados los UAV´s.

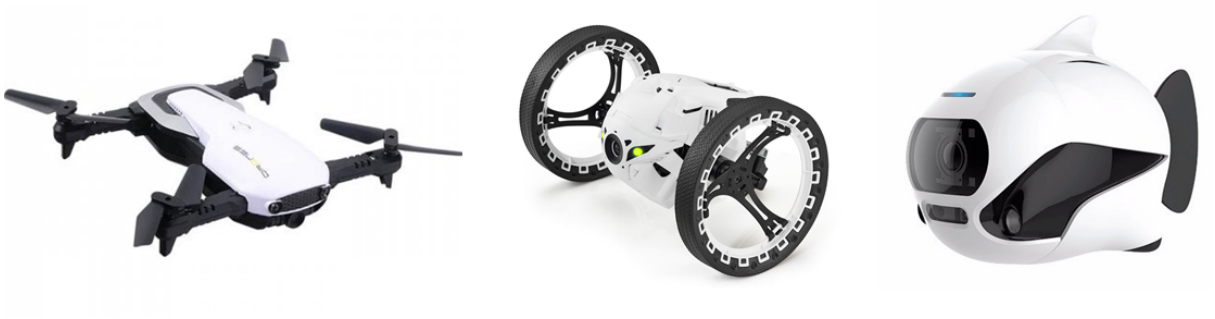

Figura 2.1: Tipos de robots móviles.

Un UAV o vehículo aéreo no tripulado es una aeronave que no cuenta con un piloto a bordo. Este tipo de vehículo puede subdividirse en:

UAV´s de ala fija: Son aeronaves que poseen un perfil alar que permite que el vehículo pueda moverse a través del aire y sea capaz de generar fuerzas sustentadoras para mantenerse en el aire [21]. La principal característica de este tipo de UAV es la gran autonomía que ofrecen ya que pueden volar por periodos de tiempo largos debido a su eficiencia aerodinámica. Este tipo de UAV no es capaz de realizar vuelos estacionarios como los de ala rotatoria así como tampoco pueden despegar y aterrizar de forma vertical. (véase Figura 2.2)

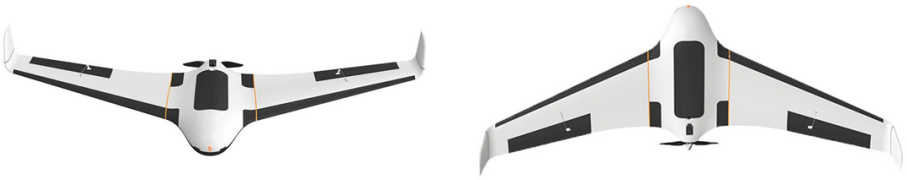

Figura 2.2: UAV del tipo ala fija.

UAV´s de ala rotatoria o hélice: Mejor conocidos como multirrotores, cuentan con múltiples hélices rotatorias, y a diferencia de los de ala fija, estos generan la sustentación a través de las fuerzas generadas por los rotores. Según el número de rotores, existen helicópteros (1-2 rotores), tricópteros (3 rotores), cuadricópteros (4 rotores), hexacópteros (6 rotores) y octacópteros (8 rotores). Algunas de estas configuraciones se muestran en la Figura 2.3

Figura 2.3: UAV del tipo ala rotatoria o multirotor.

Este tipo de UAV se caracteriza por su capacidad de despegue y aterrizaje vertical, sobre todo la capacidad de mantenerse suspendido en el aire (llamado también como vuelo en *hovering*). Una de las desventajas de este tipo de aeronave consiste en el consumo energético. A diferencia de los vehículos de ala fija que pueden mantenerse en vuelo gracias a la sustentación aerodinámica, los multicópteros requieren que sus rotores permanezcan encendidos constantemente.

II. Modelo dinámico de un Quad-Rotor

A continuación, se presenta el modelo dinámico de un Quad-Rotor, el cual es obtenido utilizando las ecuaciones de Euler-Lagrange y a través de una representación de la aeronave como un cuerpo rígido en un espacio tridimensional, el cual está sujeto a un empuje total dado por los cuatro motores y tres pares aplicados sobre los ejes $X - Y - Z$, [2].

Defínase las coordenadas generalizadas del Quad-Rotor de la siguiente manera:

$$q = \begin{bmatrix} \xi \\ \eta \end{bmatrix}, \qquad \xi = \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \qquad \eta = \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix}, \tag{2.1}$$

donde $\xi \in \mathbb{R}^3$ corresponde al vector de posición del centro de masa del vehículo con respecto al marco inercial, mientras que la orientación del vehículo, representada por los ángulos de Euler, es expresada como $\eta \in \mathbb{R}^3$, siendo ψ el ángulo de guiñada (Yaw) sobre el eje Z, θ es el ángulo de balanceo (Pitch) sobre el eje Y, y ϕ es el ángulo de cabeceo (Roll) sobre el eje X. En la Figura 2.4 se muestra una ilustración del vehículo y los marcos de referencia.

Utilizando la ecuación del Lagrangiano:

$$L(q, \dot{q}) = T_{Trans} + T_{Rot} - U, \tag{2.2}$$

donde $T_{Trans} = \frac{m}{2}\dot{\xi}^T\xi$ es la energía cinética traslacional, $T_{Rot} = \frac{1}{2}\Omega^T I\Omega$ es la energía cinética rotacional, $U = mgz$ es la energía potencial del vehículo, m es la masa del quad-rotor, $\Omega = \begin{bmatrix} p & q & r \end{bmatrix}^T \in \mathbb{R}^3$ es el vector de velocidades angulares, $I \in \mathbb{R}^{3\times 3}$ es la matriz de inercia, y $g \in \mathbb{R}$ es la aceleración dada por la gravedad. El modelado de la dinámica de la aeronave es obtenido a partir de las ecuaciones Euler-Lagrange con fuerzas generalizadas

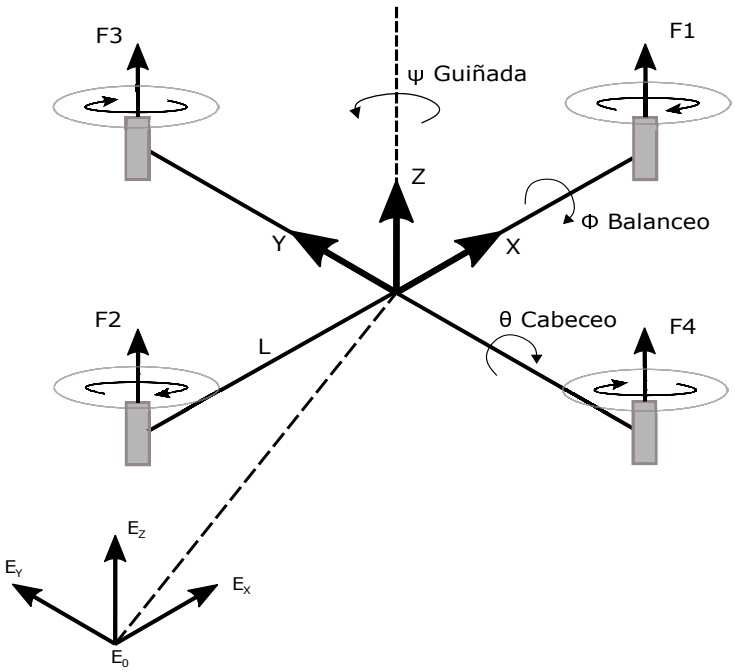

Figura 2.4: Marco inercial y marco móvil del UAV.

externas:

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}}\right) - \frac{\partial \mathcal{L}}{\partial q} = \begin{bmatrix} F_\xi \\ \tau \end{bmatrix}, \tag{2.3}$$

donde $F_\xi = \mathcal{R}\hat{F} - F_d$ es la fuerza traslacional aplicada al vehículo por el empuje, τ representa los momentos angulares de cabeceo, balanceo y guiñada; $\mathcal{R}$ representa la matriz de rotación y F_d =diag$\{kx \quad ky \quad kz\}\dot{\xi}$ es el vector de fuerzas de arrastre. La matriz de orientación $\mathcal{R}(\psi, \theta, \phi)$ de la aeronave, con respecto al marco inercial y girando en el siguiente orden:

$$\mathcal{R}(\psi, \theta, \phi) = \mathcal{R}(\psi)\mathcal{R}(\theta)\mathcal{R}(\phi), \tag{2.4}$$

donde

$$\mathcal{R}_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & C_\phi & S_\phi \\ 0 & -S_\phi & C_\phi \end{bmatrix}, \quad \mathcal{R}_y(\theta) = \begin{bmatrix} C_\theta & 0 & -S_\theta \\ 0 & 1 & 0 \\ S_\theta & 0 & C_\theta \end{bmatrix}, \quad \mathcal{R}_z(\psi) = \begin{bmatrix} C_\psi & S_\psi & 0 \\ -S_\psi & C_\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Con, $S_i := \sin(i)$ y $C_i := \cos(i)$. Resolviendo 2.4, nos queda

$$\mathcal{R}(\psi, \theta, \phi) = \begin{bmatrix} C_\theta C_\psi & S_\phi S\theta C_\psi - C_\phi S_\psi & C_\phi S_\theta C_\psi + S_\phi S_\psi \\ C_\theta S_\psi & S_\phi S_\theta S_\psi + C_\phi C_\psi & C_\phi S_\theta S_\psi - S_\phi C_\psi \\ -S_\theta & S_\phi C_\theta & C_\phi C_\theta \end{bmatrix}. \tag{2.5}$$

De la Figura 2.4, se tiene que:

$$\hat{F} = \begin{bmatrix} 0 \\ 0 \\ u \end{bmatrix},$$

donde u es el empuje total sobre el eje z (generado por los cuatro motores), y puede ser expresado de la siguiente manera:

$$u = \sum_{i=1}^{4} F_i, \tag{2.6}$$

donde F_i es la fuerza producida por el motor M_i. En la literatura, puede encontrarse una simplificación de la relación de dicha fuerza como $F_i = k_i \omega_i^2$, donde $k_i \in \mathbb{R}_+$ y ω_i es la velocidad angular del i-ésimo motor. De esta manera, se tiene que el vector de pares generalizados es

$$\tau = \begin{bmatrix} \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} = \begin{bmatrix} L(F_1 - F_2) \\ L(F_3 - F_4) \\ -F_1 + F_2 - F_3 + F_4 \end{bmatrix}, \tag{2.7}$$

donde L es la distancia entre los motores y el centro de gravedad.

Las velocidades angulares, en el marco móvil, están relacionadas a las velocidades generalizadas de $\dot{\eta}$ por medio de

$$\Omega = W_\eta \dot{\eta}, \tag{2.8}$$

donde W_η se obtiene al resolver las velocidades de los ángulos de Euler en el marco móvil:

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix} + \mathcal{R}_x(\phi)^{-1} \begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} + \mathcal{R}_x(\phi)^{-1} \mathcal{R}_y(\theta)^{-1} \begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix} = W_\eta \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix}. \tag{2.9}$$

Por lo tanto,

$$W_\eta = \begin{bmatrix} 1 & 0 & -S_\theta \\ 0 & C_\phi & S_\phi C_\theta \\ 0 & -S_\phi & C_\phi C_\theta \end{bmatrix}, \tag{2.10}$$

y las velocidades angulares resultan en

$$\Omega = \begin{bmatrix} \dot{\phi} - \dot{\psi} S_\theta \\ \dot{\theta} C_\phi + \dot{\psi} S_\phi C_\theta \\ \dot{\theta} S_\phi + \dot{\psi} C_\phi C_\theta \end{bmatrix}. \tag{2.11}$$

La energía cinética rotacional T_{Rot} puede ser expresada como

$$T_{Rot} = \frac{1}{2}\Omega^T I \Omega. \tag{2.12}$$

Sustituyendo (2.8) en (2.12):

$$T_{Rot} = \frac{1}{2}(W_\eta \dot{\eta})^T I (W_\eta \dot{\eta}) = \frac{1}{2}\dot{\eta}^T W_\eta^T I W_\eta \dot{\eta},$$

donde se toma

$$\mathbb{J} = \mathbb{J}(\eta) = W_\eta^T I W_\eta, \tag{2.13}$$

entonces se tiene que

$$T_{Rot} = \frac{1}{2}\dot{\eta}^T \mathbb{J} \dot{\eta}. \tag{2.14}$$

Se sabe también que la matriz de inercias está dada por

$$I = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix}, \tag{2.15}$$

A partir de (2.13), se obtiene

$$\mathbb{J} = \begin{bmatrix} I_{xx} & 0 & -I_{xx}S_\theta \\ 0 & I_{yy}C_\phi^2 + I_{zz}S_\phi^2 & S_\phi C_\phi C_\theta (I_{yy} - I_{zz}) \\ -IxxS_\theta & S_\phi C_\phi C_\theta (I_{yy} - I_{zz}) & I_{xx}S_\theta^2 + I_{yy}S_\phi^2 C_\theta^2 + I_{zz}C_\phi^2 C_\theta^2 \end{bmatrix}.$$

Como el Lagrangiano no contiene términos cruzados entre la energía cinética $\dot{\xi}$ con $\dot{\eta}$, la ecuación de Euler-Lagrange puede ser dividida en dinámicas para las coordenadas de ξ y las coordenadas de η. Entonces, la ecuación de Euler-Lagrange para el movimiento traslacional utiliza el Lagrangiano $L_{Trans} = T_{Trans} - U$, y puede ser escrita como

$$\frac{d}{dt}\left(\frac{\partial L_{Trans}}{\partial \dot{\xi}}\right) - \frac{\partial L_{Trans}}{\partial \xi} = F_\xi, \tag{2.16}$$

por lo tanto,

$$m\ddot{\xi} + mgF_z = F_\xi. \tag{2.17}$$

Recordando que $F_\xi = \mathcal{R}\hat{F} - F_d$, y reacomodando (2.17), nos queda que

$$m\ddot{\xi} = \mathcal{R}\hat{F} - F_d - mgF_z. \tag{2.18}$$

Entonces

$$m\begin{bmatrix}\ddot{x}\\ \ddot{y}\\ \ddot{z}\end{bmatrix} = u\begin{bmatrix}C_\phi C_\psi S_\theta + S_\phi S_\psi\\ C_\phi S_\theta S_\psi - C_\psi S_\phi\\ C_\theta C_\phi\end{bmatrix} - \begin{bmatrix}k_x\dot{x}\\ k_y\dot{y}\\ k_z\dot{z}\end{bmatrix} - \begin{bmatrix}0\\ 0\\ mg\end{bmatrix}. \tag{2.19}$$

La ecuación de Euler-Lagrange para la orientación, en términos para las coordenadas de η, se representa como

$$\frac{d}{dt}\left(\frac{\partial L_{Rot}}{\partial \dot{\eta}}\right) - \frac{\partial L_{Rot}}{\partial \eta} = \tau, \tag{2.20}$$

sabiendo que $L_{Rot} = T_{Rot}$, la ecuación 2.20 se reescribe como

$$\frac{d}{dt}\left(\dot{\eta}^T \mathbb{J}\frac{\partial \dot{\eta}}{\partial \dot{\eta}}\right) - \frac{1}{2}\frac{\partial}{\partial \eta}(\dot{\eta}^T \mathbb{J}\dot{\eta}) = \tau. \tag{2.21}$$

Entonces, se obtiene que

$$\mathbb{J}\ddot{\eta} + \dot{\mathbb{J}}\dot{\eta} - \frac{1}{2}\frac{\partial}{\partial \eta}(\dot{\eta}^T \mathbb{J}\dot{\eta}). \tag{2.22}$$

Defínase el vector de Coriolis como

$$\begin{aligned}\overline{V}(\eta,\dot{\eta}) &= \dot{\mathbb{J}}\dot{\eta} - \frac{1}{2}\frac{\partial}{\partial \eta}(\dot{\eta}^T \mathbb{J}\dot{\eta}),\\ \overline{V}(\eta,\dot{\eta}) &= C(\eta,\dot{\eta})\dot{\eta} = \left(\dot{\mathbb{J}} - \frac{1}{2}\frac{\partial}{\partial \eta}(\dot{\eta}^T \mathbb{J})\right)\dot{\eta},\end{aligned} \tag{2.23}$$

donde $C(\eta,\dot{\eta})$ es el término de Coriolis. La derivada de la matriz $\mathbb{J}$ está dada por

$$\dot{\mathbb{J}} = W_\eta^T I \dot{W}_\eta + \dot{W}_\eta^T I W_\eta. \tag{2.24}$$

Sustituyendo las ecuaciones (2.10) y (2.15) en (2.24), se obtiene

$$\dot{\mathbb{J}} = \begin{bmatrix}0 & 0 & -\dot{\theta}C_\theta I_{xx}\\ 0 & 2\dot{\phi}C_\phi S_\phi(I_{zz} - I_{yy}) & \alpha\\ -\dot{\theta}C_\theta I_{xx} & \alpha & \beta\end{bmatrix}, \tag{2.25}$$

con

$$\begin{aligned}\alpha &= \dot{\phi}C_\theta(I_{yy} - I_{zz})(C_\phi^2 - S_\phi^2) + \dot{\theta}S_\theta S_\phi C_\phi(I_{zz} - I_{yy}),\\ \beta &= 2\phi S_\phi C_\phi C_\theta^2(I_{yy} - I_{zz}) + \dot{\theta}C_\theta S_\theta(I_{xx} - I_{yy}S_\phi^2 - I_{zz}C_\phi^2).\end{aligned}$$

Para el término $\frac{1}{2}\frac{\partial}{\partial \eta}(\dot{\eta}^T \mathbb{J} \dot{\eta})$, se tiene que

$$\dot{\eta}^T \mathbb{J} = \begin{bmatrix} \dot{\phi} I_{xx} - \dot{\psi} I_{xx} S_\theta \\ \dot{\theta}(I_{yy} C_\phi^2 + I_{zz} S_\phi^2) + \dot{\psi} S_\phi C_\phi C_\theta (I_{yy} - I_{zz}) \\ -\dot{\phi} I_{xx} S_\theta + \dot{\theta} S_\phi C_\phi C_\theta (I_{yy} - I_{zz}) + \dot{\psi}(I_{xx} S_\theta^2 + I_{yy} S_\phi^2 C_\theta^2 + I_{zz} C_\phi^2 C_\theta^2) \end{bmatrix}^T . \quad (2.26)$$

Por lo que

$$\frac{1}{2}\frac{\partial}{\partial \eta}(\dot{\eta}^T \mathbb{J} \dot{\eta}) = \begin{bmatrix} 0 & -\frac{1}{2}\dot{\psi} C_\theta I_{xx} & 0 \\ \dot{\theta} S_\phi C_\phi (I_{zz} - I_{yy}) + \frac{1}{2}\dot{\psi} C_\theta (I_{yy} - I_{zz})(C_\phi^2 - S_\phi^2) & -\frac{1}{2}\dot{\psi} S_\theta S_\phi C_\phi (I_{yy} - I_{zz}) & 0 \\ \frac{1}{2}\dot{\theta} C_\theta (I_{yy} - I_{zz})(C_\phi^2 - S_\phi^2) + \dot{\psi} S_\phi C_\phi C_\theta^2 (I_{yy} - I_{zz}) & \gamma & 0 \end{bmatrix},$$

donde

$$\gamma = -\frac{1}{2}\dot{\phi} C_\theta I_{xx} + \frac{1}{2}\dot{\theta} S_\phi C_\phi S_\theta (I_{zz} - I_{yy}) + \dot{\psi} S_\theta C_\theta (I_{xx} - S_\phi^2 I_{yy} - C_\phi^2 I_{zz}).$$

Entonces, la matriz $C(\eta, \dot{\eta})$ puede escribirse como

$$C(\eta, \dot{\eta}) = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{bmatrix}, \quad (2.27)$$

con

$$\begin{aligned}
c_{11} &= 0, \\
c_{12} &= (I_{yy} - I_{zz})(\dot{\theta} C_\phi S_\phi + \dot{\psi} S_\phi^2 C_\theta) + (I_{zz} - I_{yy})\dot{\psi} C_\phi^2 C_\theta - I_{xx}\dot{\psi} C_\theta, \\
c_{13} &= (I_{zz} - I_{yy})\dot{\psi} C_\phi S_\phi C_\theta^2, \\
c_{21} &= (I_{zz} - I_{yy})(\dot{\theta} C_\phi S_\phi + \dot{\psi} S_\phi^2 C_\theta) + (I_{yy} - I_{zz})\dot{\psi} C_\phi^2 C_\theta - I_{xx}\dot{\psi} C_\theta, \\
c_{22} &= (I_{zz} - I_{yy})\dot{\phi} C_\phi S_\phi, \\
c_{23} &= -I_{xx}\dot{\psi} S_\theta C_\theta + I_{yy}\dot{\psi} S_\phi^2 S_\theta C_\theta + I_{zz}\dot{\psi} C_\phi^2 S_\theta C_\theta, \\
c_{31} &= (I_{yy} - I_{zz})\dot{\psi} C_\theta^2 S_\phi C_\phi - I_{xx}\dot{\theta} C_\theta, \\
c_{32} &= (I_{zz} - I_{yy})(\dot{\theta} C_\phi S_\phi S_\theta + \dot{\phi} S_\phi^2 C_\theta) + (I_{yy} - I_{zz})\dot{\phi} C_\phi^2 C_\theta + I_{xx}\dot{\phi} S_\theta C_\theta - I_{yy}\dot{\phi} S_\phi^2 S_\theta C_\theta - I_{zz}\dot{\phi} C_\phi^2 S_\theta C_\theta, \\
c_{33} &= (I_{yy} - I_{zz})\dot{\phi} S_\phi C_\phi C_\theta^2 - I_{yy}\dot{\theta} S_\phi^2 C_\theta S_\theta - I_{zz}\dot{\theta} S_\phi^2 C_\theta S_\theta + I_{xx}\dot{\theta} C_\theta S_\theta.
\end{aligned}$$

De la ecuación (2.22), se tiene que

$$\ddot{\eta} = \mathbb{J}^{-1}\left[\tau - C(\eta, \dot{\eta})\dot{\eta}\right], \quad (2.28)$$

con

$$\mathbb{J}^{-1} = \begin{bmatrix} \frac{1}{I_{xx}} + \frac{C_\phi^2 S_\phi^2}{I_{zz} C_\theta^2} + \frac{S_\phi^2 S_\theta^2}{I_{yy} C_\theta^2} & \frac{(I_{zz}-I_{yy}) C_\phi S_\phi S_\theta}{I_{yy} I_{zz} C_\theta^2} & \frac{(I_{yy} C_\phi^2 + I_{zz} S_\phi^2) S_\theta}{I_{yy} I_{zz} C_\theta^2} \\ \frac{(I_{zz}-I_{yy}) C_\phi S_\phi S_\theta}{I_{yy} I_{zz} C_\theta^2} & \frac{C_\phi^2}{I_{yy} + \frac{S_\phi^2}{I_{zz}}} & \frac{(I_{zz}-I_{yy}) C_\phi S_\phi}{I_{yy} I_{zz} C_\theta} \\ \frac{(I_{yy} C_\phi^2 + I_{zz} S_\phi^2) S_\theta}{I_{yy} I_{zz} C_\theta^2} & \frac{(I_{zz}-I_{yy}) C_\phi S_\phi}{I_{yy} I_{zz} C_\theta} & \frac{I_{yy} C_\phi^2 + I_{zz} S_\phi^2}{I_{yy} I_{zz} C_\theta^2} \end{bmatrix}. \tag{2.29}$$

Finalmente, desarrollando (2.28) nos queda que la dinámica para la orientación es

$$\begin{bmatrix} \ddot{\phi} \\ \ddot{\theta} \\ \ddot{\psi} \end{bmatrix} = \mathbb{J}^{-1} \left[\begin{bmatrix} \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} - C(\eta, \dot{\eta}) \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \right]. \tag{2.30}$$

Considerando a $u = U_z$,$\tau_\phi = U_\phi$, $\tau_\theta = U_\theta$ y $\tau_\psi = U_\psi$, desarrollando la ecuación (2.30) y sabiendo que lo obtenido contiene términos que se pueden considerar despreciables, además, retomando las ecuaciones que se obtienen de (2.19), entonces, el modelo simplificado del Quad-Rotor está dado por

$$\begin{aligned} m\ddot{x} &= U_z(S_\phi S_\psi + C_\phi C_\psi S_\theta) - k_x \dot{x}, \\ m\ddot{y} &= U_z(C_\phi S_\theta S_\psi - C_\psi S_\phi) - k_y \dot{y}, \\ m\ddot{z} &= U_z C_\theta C_\phi - g - k_z \dot{z}, \\ I_{xx}\ddot{\phi} &= U_\phi + \dot{\phi}\dot{\psi}(I_{yy} - I_{zz}), \\ I_{yy}\ddot{\theta} &= U_\theta + \dot{\phi}\dot{\psi}(I_{zz} - I_{xx}), \\ I_{zz}\ddot{\psi} &= U_\psi + \dot{\phi}\dot{\theta}(I_{xx} - I_{yy}). \end{aligned} \tag{2.31}$$

III. Control PID

Control PID es un nombre comúnmente dado a un control que posee tres términos y hace referencia a las siglas de los cada uno de los términos: P para la parte proporcional, I para la parte integral y D para la parte derivativa.

El control por PID tiene tres diferentes representaciones. La primera (Figura 2.5(a)), donde cada uno de los términos puede ser seleccionado para alcanzar diferentes acciones de control. La segunda, es la forma en el dominio del tiempo (Figura 2.5(b)); y por último, la tercera en el dominio de Laplace (Figura 2.5(c)), la cual permite un enlace entre el dominio del tiempo y el dominio de la frecuencia.

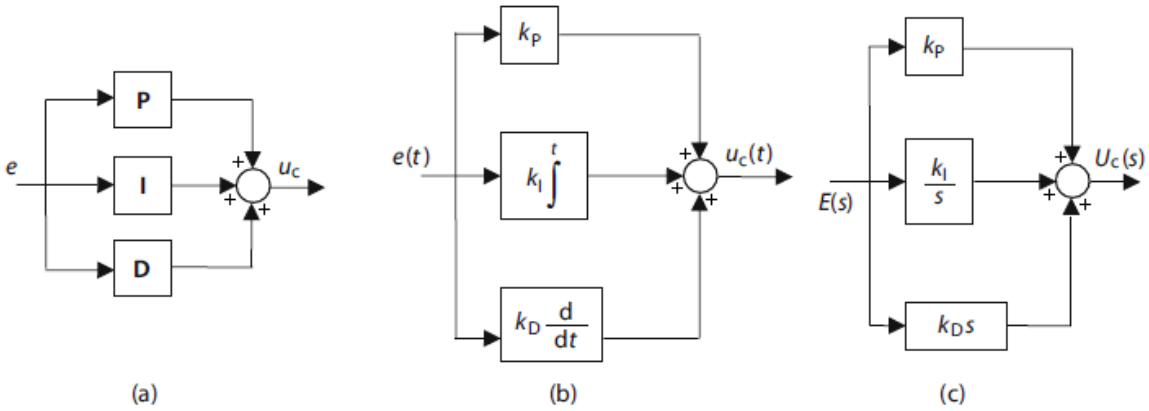

Figura 2.5: Representaciones del PID.

Control Proporcional: Denotado por el término P, es la acción que consiste en la multiplicación de la señal del error $e(t) = r(t) - y_m(t)$, con la ganancia proporcional K_p, donde $r(t)$ es la señal de referencia y $y_m(t)$ es la señal de salida del sistema, de forma que se logre que el error en estado estacionario tienda a cero. Su representación está dada por

$$u(t) = K_p e(t).$$

En otras palabras, el control proporcional es, en esencia, un amplificador con una ganancia ajustable [22].

Control Integral: Denotado por el término I, esta acción causa que el valor de la salida $u(t)$ del controlador se cambie a una razón proporcional a la señal del error $e(t)$. Es decir,

$$\frac{du(t)}{dt} = K_i e(t),$$

o bien

$$u(t) = K_i \int_0^t e(t)dt.$$

donde K_i es una constante [22].

Control Derivativo: Denotado por el término D, utiliza la razón de cambio de la señal del error. Esta clase de control por si solo no puede ser aplicado debido a posible amplificación de ruido. Su representación está dada por

$$u(t) = K_d \frac{de(t)}{dt}.$$

donde K_d es una ganancia constante ajustable [23].

Entonces, la combinación de estas tres acciones tiene la ventaja de cada una de ellas, quedando finalmente la ecuación como

$$u(t) = K_p e(t) + K_i \int_0^t e(t)dt + K_d \frac{de(t)}{dt}.$$

IV. Control por Backstepping

La metodología de control por Backstepping ha sido muy utilizada en los últimos 20 años. Esta metodología está basada en la idea de realimentación pasiva, descomponiendo el sistema original en n subsistemas, siempre y cuando el sistema original presente una estructura triangular inferior [24]. La metodología va proponiendo controles de cada subsistema, para que al final, se pueda generar la entrada de control real de todo el sistema, garantizando, a través de la teoría de Lyapunov, la estabilización del sistema. Considere el sistema

$$\dot{\eta} = f(\eta) + g(\eta)\xi, \tag{2.32}$$

$$\dot{\xi} = u. \tag{2.33}$$

donde $[\eta^T, \xi]^T \in \mathbb{R}^{n+1}$ es el estado y $u \in \mathbb{R}$ es la entrada de control. Las funciones $f : D \to \mathbb{R}^n$ y $g : D \to \mathbb{R}^n$ son funciones suaves en el dominio $D \subset \mathbb{R}^n$ que contiene a $\eta = 0$ y $f(0) = 0$. El objetivo consiste en diseñar una ley de control de realimentación de estados para estabilizar el origen ($\eta = 0, \quad \xi = 0$). Se asume que las funciones f y g son conocidas. El sistema puede ser visto como una conexión de dos componentes en cascada. Suponga que el componente (2.32) se puede estabilizar con una ley de control por realimentación de estados suave $\xi = \phi(\eta)$, con $\phi(0) = 0$, es decir, el origen de

$$\dot{\eta} = f(\eta) + g(\eta)\phi(\eta),$$

es asintóticamente estable. Suponga además que se conoce una función de Lyapunov $V(\eta)$ que satisface la desigualdad

$$\frac{\partial V}{\partial \eta}[f(\eta) + g(\eta)\phi(\eta)] \leq -W(\eta), \qquad \forall \eta \in D \tag{2.34}$$

donde $W(\eta)$ es definida positiva. Sumando y restando $g(\eta)\phi(\eta)$ en el lado derecho de (2.32), se obtiene la representación equivalente

$$\dot{\eta} = [f(\eta) + g(\eta)\phi(\eta)] + g(\eta)[\xi - \phi(\eta)],$$
$$\dot{\xi} = u.$$

Realizando un cambio de variables como

$$z = \xi - \phi(\eta),$$

resulta el sistema

$$\begin{aligned}\dot{\eta} &= [f(\eta) + g(\eta)\phi(\eta)] + g(\eta)z,\\ \dot{\xi} &= u - \dot{\phi}.\end{aligned}$$

Siendo f, g y ϕ conocidas, la derivada $\dot{\phi}$ se puede calcular con la expresión

$$\dot{\phi} = \frac{\partial \phi}{\partial \eta}[f(\eta) + g(\eta)\xi].$$

Tomando $v = u - \dot{\phi}$, se reduce el sistema a la conexión en cascada

$$\begin{aligned}\dot{\eta} &= [f(\eta) + g(\eta)\phi(\eta)] + g(\eta)z,\\ \dot{z} &= v.\end{aligned}$$

El resultado es similar al sistema original, excepto que ahora el primer componente tiene un origen asintóticamente estable cuando la entrada es igual a cero. Esta característica se utilizará en el diseño de v para estabilizar todo el sistema. Usando

$$V(\eta, \xi) = V(\eta) + \frac{1}{2}z^2,$$

como la función candidata de Lyapunov, se obtiene

$$\dot{V} = \frac{\partial V}{\partial \eta}[f(\eta) + g(\eta)\phi(\eta)] + \frac{\partial V}{\partial \eta}g(\eta)z + zv \leq -W(\eta) + \frac{\partial V}{\partial \eta}g(\eta)z + zv.$$

Seleccionando

$$v = -\frac{\partial V}{\partial \eta}g(\eta) - kz, \qquad k > 0$$

se obtiene

$$\dot{V} \leq -W(\eta) - kz^2.$$

Lo que demuestra que el origen ($\eta = 0, \quad z = 0$) es asintóticamente estable. Siendo $\phi(0) = 0$ se concluye que el origen ($\eta = 0, \quad \xi = 0$) es asintóticamente estable. Sustituyendo v, z y $\dot{\phi}$, se obtiene la ley de control por realimentación de estados:

$$u = \frac{\partial \phi}{\partial \eta}[f(\eta) + g(\eta)\xi] - \frac{\partial V}{\partial \eta}g(\eta) - k[\xi - \phi(\eta)]. \tag{2.35}$$

Si todas las suposiciones se mantienen globalmente, y $V(\eta)$ es radialmente desacotada, entonces se concluye que el origen es globalmente, asintóticamente estable. Entonces, el procedimiento anterior puede ser descrito como sigue:

Lema 1 [25] *Considere el sistema (2.32)-(2.33). Sea $\phi(\eta)$ una ley de control por realimentación de estados para 2.32 con $\phi(0) = 0$, y $V(\eta)$ una función de Lyapunov que satisface 2.34 con alguna función definida positiva $W(\eta)$. Entonces, la ley de control por realimentación de estados 2.35 estabiliza el origen de 2.32-2.33, con $V(\eta) + [\xi - \phi(\eta)]^2/2$ como función de Lyapunov. Además, si todas las suposiciones se cumplen globalmente y $V(\eta)$ es radialmente desacotada, el origen será globalmente, asintóticamente estable.*

V. Visión artificial

La Visión Artificial es el campo de la Inteligencia Artificial que permite la adquisición, procesamiento, análisis y comprensión de la información obtenida a través de imágenes digitales, y tiene la finalidad de reproducir el sentido de la vista para sistemas digitales [26]. Se compone principalmente de un conjunto de procesos destinados al análisis de imágenes. Dentro de estos procesos, se encuentran: la adquisición de imágenes, la cuál se logra a través de sensores capaces de transformar señales de luminosidad en información numérica o simbólica, comúnmente conocidos como cámaras; la memorización de la información; el procesamiento de la misma, y la interpretación de los resultados siguiendo ese orden.

Como se observa en la Figura 2.6, en el primer paso, se trata de conseguir que la imagen sea lo más adecuada posible para que pueda continuarse con las siguientes etapas. Una correcta adquisición de la imagen supone un paso muy importante para que el proceso de reconocimiento tenga éxito. Toda imagen que se adquiere por medios electrónicos sufre en cierta medida los efectos de la degradación que se manifiesta en forma de ruido o pérdida de definición. La degradación es provocada por el ruido de los sensores de captura, imprecisiones en el enfoque de la cámara, movimiento de la misma o perturbaciones aleatorias. Los mecanismos que tratan de contrarrestar estos efectos se incluyen dentro de la etapa de pre-procesado.

Generalmente, el pre-procesado tiene la finalidad de reparar en la imagen desperfectos producidos por lo anteriormente mencionado. Los algoritmos de pre-procesado permiten modificar la imagen para eliminar ruido, transformarla geométricamente, mejorar la intensidad o contraste. Una parte importante de pre-procesado es la realización de un filtrado que nos ayude a eliminar el ruido existente. La mayoría de las implementaciones de filtros se realizan en dos dominios: dominio espacial y dominio frecuencial.

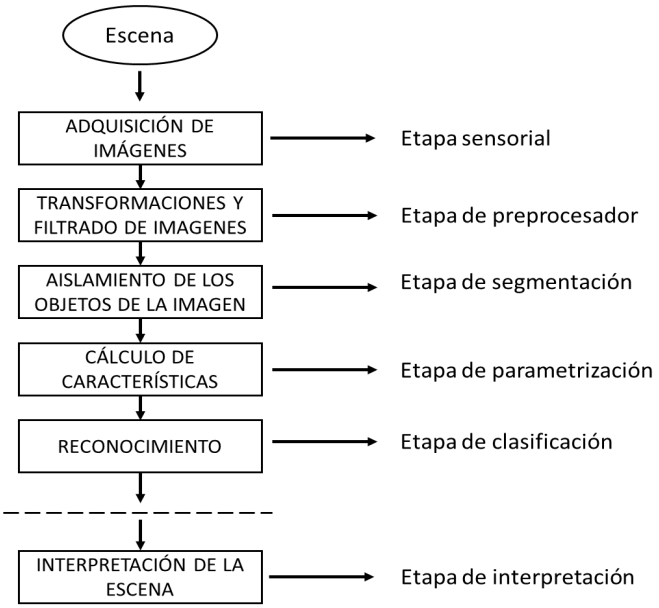

Figura 2.6: Diagrama del proceso básico de Visión Artificial

La etapa de segmentación consiste en diferenciar los objetos que componen la señal y determinar su posición con respecto al fondo de la imagen. El objetivo de la segmentación es conocer los objetos que existen en la imagen y extraer las características propias de cada uno. Además, cada pixel de la imagen tiene que tener una etiqueta que los defina, de forma que simplemente por agrupación de puntos con la misma etiqueta, conectados espacialmente, se pueda determinar la lista de objetos.

Una vez realizado el proceso de segmentación, se continua con la etapa de parametrización, la cual consiste en obtener parámetros que definan las características de cada objeto: forma, textura, color, orientación, entre otras. De entre estos parámetros, habrá que seleccionar aquellos que tengan las siguientes características:

- *Discriminación*: Esto significa que sean capaces de diferenciar de la mejor manera los objetos de una clase de otras.
- *Independencia*: Los descriptores que definan cada objeto no deben estar relacionados de forma que si uno de ellos varía, los demás permanezcan invariantes.
- *Suficiencia*: Esta característica debe delimitar de forma suficiente la pertenencia de un objeto a una clase determinada.

Por último, se realiza la clasificación de los objetos basada en la parametrización y los

descriptores para posteriormente interpretar los resultados y realizar acciones dependiendo de ellos.

Con la visión artificial es posible automatizar procesos repetitivos, realizar controles de calidad en productos, realizar inspecciones de objetos sin contacto físico, y sus aplicaciones son variadas y extensas. Por ejemplo, en la robótica, la visión artificial se utiliza para detectar e identificar objetos, inspeccionar estructuras, determinar posición y distancia de los objetos en el espacio, guía de robots, determinación de coordenadas de objetos o del mismo robot, mediciones tridimensionales, sondeo de objetivos, entre muchas otras.

VI. Procesamiento digital de imágenes

El término procesamiento digital de imágenes generalmente se refiere al procesamiento de una imagen bidimensional realizado por una computadora digital. Este tiene una gran cantidad de aplicaciones tales como, detección de objetos, seguridad, análisis de crecimiento poblacional, análisis de trafico, reconocimiento facial, reconocimiento de huellas dactilares, análisis satelitales, etc. [27].

Una imagen digital es un arreglo de números, ya sean reales o complejos, representados por un número finito de bits, que es el resultado de la proyección de una escena tridimensional en el sensor de la cámara. Para poder realizar su procesamiento, es necesario realizar dos operaciones principales. La primer operación se conoce como *segmentación*, la cual, es el proceso que subdivide una imagen en un número de regiones uniformemente homogéneas. Cada región homogénea es una parte constitutiva u objeto en toda la escena. En otras palabras, la segmentación de una imagen está definida por un conjunto de regiones que están conectadas y no sobrepuestas, de forma que cada pixel en un segmento en la imagen adquiere una etiqueta de región única que indica la región a la pertenece. La segmentación es uno de los elementos más importantes en el análisis de imágenes, principalmente debido a que en esta operación es se extraen los elementos de interés de la imagen para después procesarla. La segunda operación es la *interpretación*, la cual, se encarga de la extracción de las características obtenidas en la segmentación.

Existe una herramienta llamada OpenCV (*Open Source Computer Visión Library*, por sus siglas en inglés) la cual es un conjunto de librerías de acceso abierto basadas en los lenguajes de programación *C* y *C++* que incluyen cientos de algoritmos especialmente diseñados para visión artificial y aprendizaje automático; esta herramienta fue creada para proveer una infraestructura común en aplicaciones de visión por computadora con el fin de acelerar su desarrollo en la industria y en la investigación [28]. Desde su creación, las

librerías de OpenCV han incrementado sus capacidades gracias a miles de usuarios que contribuyen alrededor del mundo así como empresas como Google, Intel, Microsoft, entre otras, con nuevos algoritmos para el procesamiento de imágenes.

OpenCV tiene una estructura modular, lo que significa que el paquete incluye varias librerías compartidas o estáticas. Las principales librerías estáticas se presentan a continuación:

- Funcionalidad central (*Core*): Es un módulo compacto que define estructuras de datos básicas y las funciones básicas utilizadas por los demás módulos.
- Procesamiento de imágenes (*imgproc*): Como su nombre lo dice, es un módulo de procesamiento de imágenes que incluye filtrado de imágenes lineales y no lineales, transformaciones de imágenes geométricas (cambio de tamaño, distorsión afín y perspectiva, reasignación genérica basada en tablas), conversión de espacio de color, histogramas, entre otras transformaciones.
- Análisis de vídeo (*video*): Es un módulo de análisis de vídeo que incluye la estimación de movimiento, la resta de fondo y los algoritmos de seguimiento de objetos.
- Calibración de cámara y reconstrucción 3D (*calib3d*): son algoritmos básicos de geometría de vista múltiple, calibración de cámara única y estéreo, estimación de posición de objeto, algoritmos de correspondencia estéreo y elementos de reconstrucción 3D.
- Marco de funciones 2D (*features2d*): detectores de características salientes, descriptores y correlacionadores de descriptores.
- Detección de objetos (*objdetect*): detección de objetos e instancias de las clases predefinidas (por ejemplo: caras, ojos, tazas, personas, automóviles, etc.).
- GUI de alto nivel (*highgui*): una interfaz fácil de usar para capacidades de interfaz de usuario simples.
- Vídeo I/O (*videoio*): una interfaz fácil de usar para la captura de vídeo y los codecs de vídeo.

Entre las aplicaciones de OpenCV se encuentran: Detección y reconocimiento de caras, identificación de objetos, clasificación de acciones humanas, seguimiento de movimiento de la cámara, seguimiento de objetos en movimiento, extracción de modelos 3D de objetos, producción de nubes de puntos 3D de cámaras de tipo estéreo, unión de imágenes para producir una imagen de alta resolución de una escena completa, encontrar imágenes simi-

lares en una base de datos, remover ruido de las imágenes, seguimiento del movimiento de ojos en personas o animales, reconocimiento de paisajes y establecimiento de marcadores para superponerlos con realidad aumentada, entre otras [29]. Para fines de este trabajo, se utilizaron las librerías de OpenCV version 3.4.0. A continuación, se presentan los principales módulos utilizados.

VI.1. Cambio entre espacio de colores

Esta función transforma una imagen de entrada de un espacio de colores a otro. Para el procesamiento de imágenes, estas transformaciones entre espacios de colores son de vital importancia debido a que, el color es un descriptor que en la mayoría de los casos facilita la identificación y extracción de los objetos en una escena. La función es llamada como sigue

$$\textit{void cv2.cvtColor(scr, dst, code, dstCn=0)};$$

donde *scr* es el arreglo de entrada, *dst* es el arreglo de salida, *code* es el código clave de la conversión y *dstCn* es el número de canales en la imagen resultante, normalmente se utiliza el 0 para seleccionar los canales automáticamente.

Existen más de 150 métodos de conversión entre espacios de colores dentro de OpenCV. Sin embargo, los más utilizados son $BGR \rightarrow$ *Escala de grises*. Esta transformación es realizada mediante el comando $cv.COLOR\_BGR2GRAY$, y se basa en la siguiente operación

$$BGR \quad to \quad Gray : Y \leftarrow 0.114 \cdot B + 0.587 \cdot G + 0.299 \cdot R$$

donde B representa al canal del color azul de la imagen, G el canal del color verde y R el canal del color rojo. Y representa el arreglo final en escala de grises.

Otra transformación muy utilizada es $BGR \rightarrow HSV$, la cual es realizada con el comando $cv.COLOR\_BGR2HSV$. En el caso de imágenes de 8 bits y de 16 bits, R, G y B se convierten al formato de punto flotante y se escalan para ajustarse al rango de 0 a 1.

$$V \leftarrow \max(R, G, B)$$

$$S \leftarrow \begin{cases} \frac{V - \min(R,G,B)}{V} & , \quad \text{si} \quad V \neq 0 \\ 0 & , \quad \text{de lo contrario} \end{cases}$$

$$H \leftarrow \begin{cases} 60(G-B)/(V-\min(R,G,B)) & , \text{ si } V_{\max} = R \\ 120+60(B-R)/(V-\min(R,G,B)) & , \text{ si } V_{\max} = G \\ 240+60(R-G)/(V-\min(R,G,B)) & , \text{ si } V_{\max} = B \end{cases}$$

donde V toma el máximo de los valores de los espacios R, G y B, y representa la amplitud de la luz que define el color. S toma diferente valor dependiendo de V y del valor mínimo de los espacio R, G y B, y representa la saturación, que es la intensidad de un tono en específico y se basa en la pureza del color. H toma diferente valor dependiendo de V, y representa el tono (conocido como *Hue* en inglés), que se define como el grado en el cual un estímulo puede ser descrito como similar o diferente de los estímulos como rojo, amarillo y azul, y se refiere a la propiedad en los aspectos cualitativamente diferentes de la experiencia de color que tienen relación con diferencias de longitudes de onda o con mezclas de diferentes longitudes de onda. En palabras simples, el estado puro del color, sin mezcla de blanco o negro, [30].

VI.2. Funciones de dibujo

Estas funciones permiten dibujar sobre la imagen líneas, círculos, rectángulos, elipses y polígonos, así como también agregar texto a la imagen. Las figuras más utilizadas son

cv2.Line(img, pt1, pt2, color, thickness, lineType, shift);
cv2.Circle(img, center, radius, color, thickness, lineType, shift);
cv2.Rectangle(img, pt1, pt2, color, thickness, lineType, shift);

donde *img* es la imagen en la que se dibujará, $pt1$ y $pt2$ son los vértices de la figura, *color* es el color o brillo de la imagen, *thickness* es el grosor del rectángulo, *lineType* es el tipo de línea que se usará (punteada, continua, etc.) y $shift$ es el número de bits fraccionarios en las coordenadas del punto. La primera función dibuja un segmento de línea desde $pt1$ hasta $pt2$. Para líneas no suavizadas se utiliza el algoritmo de Bresenham, y las líneas suavizadas a través de filtro Gaussiano. La segunda función, dibuja un círculo según el centro y radio proporcionado. Por último, la tercera función dibuja un rectángulo cuyas esquinas opuestas son $pt1$ y $pt2$. Ahora, para agregar texto a una imagen con OpenCV, se utiliza la función

cv2.PutText(img, text, org, font, color);

donde *img* es la imagen en la que se colocará el texto, *text* es la cadena a escribir en la imagen, *org* son las coordenadas origen de la cadena a dibujar, $font$ es el tipo de letra de la cadena y *color* es el color del texto.

VI.3. Módulo de redes neuronales

Este módulo de OpenCV contiene las funciones necesarias para pasar la información de una imagen a través de una red neuronal. Es importante mencionar que este módulo no está diseñado para el entrenamiento de redes neuronales, sin embargo, permite crear nuevas capas para las redes neuronales, cargar modelos de redes serializadas desde diferentes marcos de referencia de aprendizaje automático como Caffe, Darknet, Tensorflow, ONNX, entre otras [31]. Dentro de las funciones más importantes, se encuentran:

readNet(): Se utiliza para leer redes neuronales en alguno de los formatos compatibles. La función es llamada con el comando

cv2.dnn.readNet(model, config, framework);

donde *model* es el archivo binario que contiene los pesos entrenados de la red, $config$ es el archivo de texto que contiene la configuración de la red,y, por último, $framework$ es la etiqueta explícita del nombre del marco para determinar su formato. Esta función detecta automáticamente el origen de la referencia de un modelo entrenado, sin embargo, existen funciones explícitas para cada una de las referencias, por ejemplo, *readNetFromCaffe*.

blobFromImage(): Esta función crea una mancha de 4 dimensiones. De forma opcional, corta la imagen, sustrae valores medios, escala los valores e intercambia los canales de los colores rojo y azul. El comando para llamar a esta función está dado por

cv2.dnn.blobFromImage(img, sf, sz, mean, swapRB, crop);

donde *img* es la imagen de entrada con 1, 3 o 4 canales, sf es el factor de escalamiento, sz es el tamaño de la imagen de salida, *mean* es el escalar con los valores medios a sustraer de los canales, *swapRB* es la bandera que indica si se intercambiarán los canales rojo y azul, y por último *crop* es la bandera que indica a la función si la imagen será recortada o no.

VII. Python

Para el procesamiento de imágenes, Matlab presenta una gran ventaja pues trabaja con matrices de manera muy eficiente. Sin embargo, la velocidad de procesamiento es lenta. Python, al ser un lenguaje similar a Matlab, también es una herramienta muy útil para esta tarea y presenta una mayor velocidad de procesamiento, además de ser de acceso libre.

Python es un lenguaje de programación creado por Guido van Rossum en el año 1991 en los Países Bajos. Es un lenguaje interpretado, es decir, es capaz de analizar y ejecutar otros programas sin la necesidad de realizar el paso extra de la compilación [32]. De igual manera, es un lenguaje escrito dinámicamente lo que significa que no se requiere declarar variables y reservar memoria antes de usarlas. Se trata de un lenguaje multiparadigma pues soporta orientación a objetos, programación imperativa y programación funcional. Otra característica importante de Python es la resolución dinámica de nombres, es decir, lo que enlaza un método y un nombre de variable durante la ejecución del programa (también conocido como enlace dinámico). Python permite dividir el programa en módulos reutilizables. Contiene una gran colección de módulos estándar que se pueden utilizar como base de los programas.

Este lenguaje fue diseñado para ser leído con facilidad. Una de sus características es el uso de palabras en donde otros lenguajes utilizarían símbolos. (por ejemplo, los operadores lógicos || y && en Python se escriben or y and, respectivamente). En vez de delimitar los bloques de código con el uso de llaves {} Python utiliza la identación, como se muestra en la Figura 2.7, haciendo que esta se vuelva obligatoria, ayudando a la claridad y consistencia del código, [33].

```
int factorial(int x)
{
 if (x == 0)
   return 1;
 else
   return x*factorial(x - 1);
}
```

```
def factorial(x):
    if not x:
        return 1
    else:
        return x * factorial(x-1)
```

Figura 2.7: Indentación de código en Python.

Python soporta implícitamente una gran variedad de tipos de datos, los principales son:

- *str*: Esta variable convierte un valor determinado en una cadena. Este valor no puede cambiar durante la ejecución del programa.
- *unicode*: Este es una versión de una cadena en una codificación estándar para facilitar el tratamiento de información.
- *list*: Es un arreglo que contiene información de diferentes tipos que se encuentran ordenados y pueden cambiar durante la ejecución del programa y permite valores duplicados.
- *tuple*: Este tipo de dato es similar a *list*, la diferencia radica en que este arreglo no permite cambios durante la ejecución del programa.
- *set*: Es un arreglo de información que no está ordenada ni indexada y no permite valores repetidos.
- *dict*: Esta arreglo es una recolección de datos que no está ordenada, que permite cambios y está indexada.

Además de poder utilizar los tipos de datos comunes como *int*, *long*, *float* y *bool*. De igual manera, como en otros lenguajes de programación, es posible utilizar condicionales, ciclos, bucles *while* y casos.

Existen muchas propiedades que pueden ser agregadas al lenguaje a través de la importación de módulos, los cuales, son pequeños códigos que llaman a recursos del sistema. Estos módulos se agregan utilizando la palabra reservada *import* seguida del nombre del módulo que se requiera utilizar.

Capítulo 3

Diseño de controladores

Considere la dinámica del Quad-Rotor mostrada en la ecuación (2.31) vista en el Capítulo 2, donde $x, y, z \in \mathbb{R}$ definen la posición del Quad-Rotor, $\phi, \theta, \psi \in \mathbb{R}$ son los ángulos de cabeceo, balance y guiñada; las entradas de control $U_\phi, U_\theta, U_\psi, U_z \in \mathbb{R}$, $m \in \mathbb{R} > 0$ es la masa del Quad-Rotor, $g \in \mathbb{R}$ es la aceleración de la gravedad, $I_{xx}, I_{yy}, I_{zz} \in \mathbb{R}$ representan los momentos de inercia respecto a sus ejes y $k_x, k_y, k_z \in \mathbb{R}$ representan los coeficientes de arrastre. Este se representa en el espacio de estados $\dot{W} = f(W, U)$ por el vector de estados $W \in \mathbb{R}^{12}$ de la siguiente manera

$$W^T = [w_1 \quad w_2 \quad w_3 \quad ... \quad w_{12}] = \begin{bmatrix} \phi & \dot{\phi} & \theta & \dot{\theta} & \psi & \dot{\psi} & z & \dot{z} & x & \dot{x} & y & \dot{y} \end{bmatrix}^T. \quad (3.1)$$

Entonces, se puede reescribir

$$\begin{aligned}
\dot{w}_1 &= w_2 \\
\dot{w}_2 &= \ddot{\phi} = w_4 w_6 a_1 + b_1 U_\phi, \\
\dot{w}_3 &= w_4, \\
\dot{w}_4 &= \ddot{\theta} = w_2 w_6 a_2 + b_2 U_\theta, \\
\dot{w}_5 &= w_6, \\
\dot{w}_6 &= \ddot{\psi} = w_2 w_4 a_3 + b_3 U_\psi. \\
\dot{w}_7 &= w_8 \\
\dot{w}_8 &= \ddot{z} = b_4 C_\phi C_\theta U_z - g, \\
\dot{w}_9 &= w_{10} \\
\dot{w}_{10} &= \ddot{x} = b_4 (C_\phi S_\theta C_\psi + S_\phi S_\psi) U_z, \\
\dot{w}_{11} &= w_{12} \\
\dot{w}_{12} &= \ddot{y} = b_4 (C_\phi S_\theta S_\psi - S_\phi C_\psi) U_z.
\end{aligned} \quad (3.2)$$

Por simplicidad, defínase

$$a_1 = \frac{I_{yy} - I_{zz}}{I_{xx}}, \quad a_2 = \frac{I_{zz} - I_{xx}}{I_{yy}}, \quad a_3 = \frac{I_{xx} - I_{yy}}{I_{zz}}, \quad b_1 = \frac{1}{I_{xx}}, \quad b_2 = \frac{1}{I_{yy}}, \quad b_3 = \frac{1}{I_{zz}} \quad b_4 = \frac{1}{m}.$$

I. PID

Se definen los errores para cada grado de libertad del Quad-Rotor como

$$\begin{aligned} e_x &= w_{9d} - w_9, \quad e_\phi = w_{1d} - w_1, \\ e_y &= w_{11d} - w_{11}, \quad e_\theta = w_{3d} - w_3, \\ e_z &= w_{7d} - w_7, \quad e_\psi = w_{5d} - w_5. \end{aligned} \tag{3.3}$$

Considere el esquema de control PID para las entradas de control $U_z, U_\phi, U_\theta, U_\psi$ de la siguiente manera

$$U_z = k_{pz} e_z(t) + k_{iz} \int_0^t e_z(\tau) d\tau + k_{dz} e_z(t), \tag{3.4}$$

$$U_\phi = k_{p\phi} e_\phi(t) + k_{i\phi} \int_0^t e_\phi(\tau) d\tau + k_{d\phi} e_\phi(t), \tag{3.5}$$

$$U_\theta = k_{p\theta} e_\theta(t) + k_{i\theta} \int_0^t e_\theta(\tau) d\tau + k_{d\theta} e_\theta(t), \tag{3.6}$$

$$U_\psi = k_{p\psi} e_\psi(t) + k_{i\psi} \int_0^t e_\psi(\tau) d\tau + k_{d\psi} e_\psi(t). \tag{3.7}$$

donde $k_p\sigma, k_i\sigma, k_d\sigma \in \mathbb{R}$ son constantes de diseño, para $\sigma = z, \phi, \theta, \psi$. Las posiciones x y y no pueden desacoplarse y no pueden ser controladas directamente con una entrada de control (sistema subactuado), además, están relacionadas a los ángulos de balanceo ϕ y cabeceo θ. Utilizando la dinámica de las aceleraciones $\ddot{x}$ y $\ddot{y}$, se obtiene que

$$w_{1d} = \frac{m}{U_z}(U_x \cos(\psi) + U_y \sin(\psi)), \tag{3.8}$$

$$w_{3d} = \frac{m}{U_z}(U_x \sin(\psi) + U_y \cos(\psi)). \tag{3.9}$$

Para la obtención de U_x y U_y, se utiliza el mismo esquema PID, por lo tanto

$$U_x = k_{px} e_x(t) + k_{ix} \int_0^t e_x(\tau) d\tau + k_{dx} e_x(t), \tag{3.10}$$

$$U_y = k_{py} e_y(t) + k_{iy} \int_0^t e_y(\tau) d\tau + k_{dy} e_y(t), \tag{3.11}$$

donde $k_{p\delta}, k_{i\delta}, k_{d\delta} \in \mathbb{R}$ son valores constantes, con $\delta = x, y$. Estos valores representan las ganancias de diseño de este controlador.

La prueba de estabilidad del controlador propuesto se presenta en [34] y en [35], donde se concluye convergencia asintótica a una región cercana al origen.

II. Control por Backstepping

Primero, considérese z_1 como el error para el ángulo de alabeo:

$$z_1 = w_{1d} - w_1, \tag{3.12}$$

con w_{1d} como el ángulo de alabeo deseado. La derivada de z_1 es

$$\begin{aligned} \dot{z}_1 &= \dot{w}_{1d} - \dot{w}_1, \\ &= \dot{w}_{1d} - w_2. \end{aligned} \tag{3.13}$$

En esta primera etapa, el objetivo consiste en diseñar un control virtual w_2^* que haga que $\lim_{t\to\infty} z_1 \to 0$. Para ello, se propone la siguiente función candidata de Lyapunov

$$V_1 = \frac{1}{2} z_1^2 > 0, \tag{3.14}$$

cuya derivada es

$$\dot{V}_1 = z_1 \dot{z}_1 = z_1(\dot{w}_{1d} - w_2). \tag{3.15}$$

Se propone un control virtual dado por

$$w_2^* = \dot{w}_{1d} + k_1 z_1, \tag{3.16}$$

donde el término $k_1 z_1$ es agregado para estabilizar z_1. Ahora, se define

$$z_2 = w_2^* - w_2. \tag{3.17}$$

Sustituyendo (3.16) en (3.17), se tiene que

$$z_2 = \dot{w}_{1d} + k_1 z_1 - w_2, \tag{3.18}$$

por lo que

$$w_2 = \dot{w}_{1d} + k_1 z_1 - z_2. \tag{3.19}$$

Sustituyendo (3.19) en (3.15), se obtiene

$$\dot{V}_1 = -k_1 z_1^2 + z_1 z_2, \tag{3.20}$$

donde ya aparece la variable z_1 cuadrática y negativa, además de un término cruzado que se eliminará en la siguiente etapa. Para la etapa 2, se obtiene primero la derivada de (3.19), despejando $\dot{z}_2$ da

$$\dot{z}_2 = \ddot{w}_{1d} + k_1 \dot{z}_1 - \dot{w}_2. \tag{3.21}$$

Sustituyendo $\dot{w}_2$ en la ecuación anterior, se tiene

$$\dot{z}_2 = \ddot{w}_{1d} + k_1 \dot{z}_1 - a_1 w_4 w_6 - b_1 U_\phi. \tag{3.22}$$

Se propone una nueva función candidata de Lyapunov:

$$V_2 = V_1 + \frac{1}{2} z_2^2, \tag{3.23}$$

cuya derivada es

$$\dot{V}_2 = \dot{V}_1 + z_2 \dot{z}_2. \tag{3.24}$$

Sustituyendo (3.20) y (3.22) en (3.24), se obtiene

$$\dot{V}_2 = -k_1 z_1^2 + z_1 z_2 + z_2(\ddot{w}_{1d} + k_1 \dot{z}_1 - a_1 w_4 w_6 - b_1 U_\phi). \tag{3.25}$$

Por lo tanto, como ya aparece el control real del subsistema de cabeceo, se propone U_ϕ:

$$U_\phi = \frac{1}{b_1} \left(z_1 + k_1 \dot{z}_1 + k_2 z_2 + \ddot{w}_{1d} - a_1 w_4 w_6 \right), \tag{3.26}$$

que al ser sustituida en (3.25), resulta en

$$\dot{V}_2 = -k_1 z_1^2 - k_2 z_2^2. \tag{3.27}$$

El procedimiento anterior se aplica igualmente para U_θ, U_ψ y U_z, obteniéndose lo siguiente:

$$U_\theta = \frac{1}{b_2} \left(z_3 + k_3 \dot{z}_3 + k_4 z_4 + \ddot{w}_{3d} - a_2 w_2 w_6 \right), \tag{3.28}$$

$$U_\psi = \frac{1}{b_3} \left(z_5 + k_5 \dot{z}_5 + k_6 z_6 + \ddot{w}_{5d} - a_3 w_2 w_4 \right), \tag{3.29}$$

$$U_z = \frac{m}{C_\phi C_\theta} \left(z_7 + k_7 \dot{z}_7 + k_8 z_8 + \ddot{w}_{7d} + g \right). \tag{3.30}$$

Para el control de las coordenadas x y y, se sigue el mismo procedimiento que con los controles anteriores. Defínase entonces

$$z_9 = w_{9d} - w_9, \tag{3.31}$$

cuya derivada es

$$\begin{aligned} \dot{z}_9 &= \dot{w}_{9d} - \dot{w}_9, \\ &= \dot{w}_{9d} - w_{10}. \end{aligned} \tag{3.32}$$

Proponiendo un control virtual como

$$w_{10}^* = \dot{w}_{9d} + k_9 z_9. \tag{3.33}$$

Considérese además

$$\begin{aligned} z_{10} &= w_{10}^* - w_{10}, \\ &= \dot{w}_{9d} + k_9 z_9 - w_{10}. \end{aligned} \tag{3.34}$$

Se propone ahora la siguiente función candidata de Lyapunov

$$V_9 = \frac{1}{2} z_9^2, \tag{3.35}$$

cuya derivada es

$$\dot{V}_9 = z_9 \dot{z}_9. \tag{3.36}$$

Sustituyendo (3.32) en (3.36), se tiene que

$$\dot{V}_9 = -k_9 z_9^2 + z_9 z_{10}. \tag{3.37}$$

Derivando (3.34), se obtiene

$$\dot{z}_{10} = \ddot{w}_{9d} + k_9 \dot{z}_9 - \frac{1}{m}(C_\phi S_\theta C_\psi + S_\phi S_\psi) U_z. \tag{3.38}$$

Propóngase también la siguiente función de Lyapunov

$$V_{10} = V_9 + \frac{1}{2} z_{10}^2, \tag{3.39}$$

cuya derivada es

$$\dot{V}_{10} = \dot{V}_9 + z_{10} \dot{z}_{10}. \tag{3.40}$$

Sustituyendo, se tiene que

$$\dot{V}_{10} = -k_9 z_9^2 + z_9 z_1 0 + z_1 0 \left(\ddot{w}_{9d} + k_9 \dot{z}_9 - \frac{1}{m}(C_\phi S_\theta C_\psi + S_\phi S_\psi) U_z \right), \tag{3.41}$$

si se define

$$U_x = C_\phi S_\theta C_\psi + S_\phi S_\psi, \tag{3.42}$$

entonces 3.41 se reescribe como

$$\dot{V}_{10} = -k_9 z_9^2 + z_9 z_1 0 + z_1 0 \left(\ddot{w}_{9d} + k_9 \dot{z}_9 - \frac{1}{m} U_x U_z \right), \tag{3.43}$$

por lo que es posible proponer el siguiente control

$$U_x^* = \frac{m}{U_z} \left(z_9 + k_9 \dot{z}_9 + k_{10} z_{10} + \ddot{w}_{9d} \right). \tag{3.44}$$

Es posible encontrar la trayectoria deseada para la coordenada de cabeceo despejando de la ecuación 3.42 de la manera siguiente

$$\theta_d = w_{3d} = \sin^{-1} \left(\frac{U_x^* - S_\phi S_\psi}{C_\phi C_\psi} \right). \tag{3.45}$$

Siguiendo la misma metodología, se encuentra que

$$U_y^* = \frac{m}{U_z} \left(z_{11} + k_{11} \dot{z}_{11} + k_{12} z_{12} + \ddot{w}_{11d} \right), \tag{3.46}$$

con $U_y = C_\phi S_\theta S_\psi - S_\phi C_\psi$, y sustituyendo θ por θ_d, se puede encontrar la trayectoria deseada para el ángulo alabeo de la manera siguiente

$$\phi_d = w_{1d} = \sin^{-1} \left(U_x^* S_\psi - U_y^* C_\psi \right). \tag{3.47}$$

Definición ([25])Se dice que el sistema (3.2) es BIBO (*"bounded input bounded output"*, por sus siglas en inglés o bien entrada acotada salida acotada) es estable si para una entrada acotada $u(x)$, el sistema produce una salida acotada $y(x)$ para $0 < x < \infty$, cuya demostración se muestra en [36].

Dado lo anterior, sabiendo que el Quad-Rotor es un sistema mecánico y que el desarrollo del controlador por Backstepping fue basado en el modelo del mismo, además, sabiendo que U_x, U_y, U_z, U_ϕ, U_θ, U_ψ son entradas acotadas, entonces el sistema en lazo cerrado será estable, como se demuestra en [37] .

Capítulo 4

Algoritmos de visión

En este Capítulo, se muestra el desarrollo de dos algoritmos de visión para la detección de objetos en tiempo real. Para ello se utilizó el sistema operativo Ubuntu Mate en la plataforma Odroid-XU4. A este sistema operativo se le instalaron las librerías de Python versión 2.7, así como las librerías de *pyueye* para utilización de la cámara UI-1241LE-C-HQ, y las librerías de OpenCV versión 3.4 para realizar el procesamiento de imágenes y el uso de las herramientas de redes neuronales en Python. La elaboración de los códigos se realizó en el editor de textos *GNU nano*. Para acceder a este editor, en una terminal de Ubuntu se introduce el comando *nano*. Este editor, permite escribir código y grabarlo con la extensión deseada que, en este caso, es *.py*.

I. Diseño de algoritmos

I.1. Algoritmo de detección y conteo de objetivos

```
from Centroide.Seguidor_de_centroides import Seguidor_de_centroide

import pyueye
import numpy
import cv2
import sys
import imutils
import argparse
```

Figura 4.1: Librerías a utilizar en el algoritmo.

```
pyueye.is_InitCamera(hCam, None)
pyueye.is_GetCameraInfo(hCam, cInfo)
pyueye.is_GetSensorInfo(hCam, sInfo)
pyueye.is_SetAutoParameter(hCam, pyueye.IS_SET_ENABLE_AUTO_SHUTTER, pval1, pval2)
pyueye.is_SetDisplayMode(hCam, pyueye.IS_SET_DM_DIB)
pyueye.is_AOI(hCam, pyueye.IS_AOI_IMAGE_GET_AOI, rectAOI, pyueye.sizeof(rectAOI))
pyueye.is_AllocImageMem(hCam, width, height, nBitsPerPixel, pcImageMemory, MemID)
pyueye.is_SetImageMem(hCam, pcImageMemory, MemID)
pyueye.is_SetColorMode(hCam, m_nColorMode)

m_nColorMode = pyueye.IS_CM_BGRA8_PACKED
nBitsPerPixel = pyueye.INT(32)
bytes_per_pixel = int(nBitsPerPixel / 8)

pyueye.is_CaptureVideo(hCam, ueye.IS_DONT_WAIT)
```

Figura 4.2: Líneas de configuración de la cámara fabricada por Imaging Developng Systems.

El algoritmo de visión para la detección de objetos y conteo de objetivos inicia importando las librerías necesarias para utilizar las herramientas de OpenCV, como se muestra en la Figura 4.1. Se utiliza la función *from* para importar las variables requeridas que están definidas en el algoritmo de seguimiento de centroides. La librería de *pyueye* es utilizada para la configuración de la cámara a usar. La librería *numpy* es utilizada para la creación de arreglos para el procesamiento de la información de las imágenes tomadas. La librería *cv2* es necesaria para utilizar la herramienta OpenCV. La librería *imutils* es utilizada para realizar algunas operaciones básicas durante el procesamiento de las imágenes, por ejemplo, redimensionar. Por último, la librería *argsparse* se utiliza para la creación de argumentos de líneas de comando con el fin de llamar al programa desde la consola de mando.

Una vez declaradas las variables a utilizar, se realiza la configuración de la cámara a través de las funciones de la librería *pyueye* tal y como se muestra en la Figura 4.2. El segmento de código comienza inicializando la cámara, adquiriendo la información de la misma así como la información de su sensor. Luego, se habilita la función *auto shutter* para que el sensor de la cámara se ajuste automáticamente a los cambios de iluminación (cabe destacar que el modelo de la cámara UI-1241LE-C-HQ no cuenta con auto enfoque). Después, se define el modo de muestreo así como el área de interés y la memoria para alojar la información obtenida. El siguiente paso, es definir el modo de color, el cual, debido al modelo de la cámara utilizada, se define como *BGRA8_PACKED*. Por último, se inicializa el vídeo en vivo.

Otra parte importante del diseño del algoritmo es la definición de los argumentos, los cuales son mostrados en la Figura 4.3. El argumento *–p* permite al algoritmo acceder a la dirección del archivo que contiene el modelo de Caffe pre-entrenado para la red

```
ap = argparse.ArgumentParser()
ap.add_argument("-p", help="direccion al archivo de Caffe")
ap.add_argument("-m", help="direccion al modelo pre-entrenado de Caffe")
ap.add_argument("-c", type=float, default=0.3, help="Probabilidad de deteccion")
ap.add_argument("-o", help="direccion al archivo de video")
ap.add_argument("-t", type=int, default=-1, help="Grabar video")
ap.add_argument("-f", type=int, default=3, help="FPS del video")
ap.add_argument("-e", type=str,default="MJPG", help="codec del video")
args = vars(ap.parse_args())
```

Figura 4.3: Argumentos de línea de comando utilizados.

```
CLASES = ["persona", "ave", "gato", "perro", "vaca", "caballo", "oveja"]

IGNORAR = set(["ave", "gato", "perro"])
```

Figura 4.4: Arreglos de objetos a detectar e ignorar por la red nueronal.

neuronal. El segundo argumento *–m*, permite al algoritmo utilizar el modelo que contiene el archivo. El argumento *–c*, define la probabilidad que tiene la red neuronal para realizar una detección y sus valores deben variar entre 0.1 y 1. Estos valores indican la posibilidad de que la detección sea debil, es decir, que la red neuronal sea capaz de detectar con mayor precisión los objetos del modelo de Caffe. Los últimos cuatro argumentos, definen la capacidad del algoritmo para grabar el vídeo de la tarea realizada por el vehículo.

Dado que el algoritmo que se diseñó utiliza modelos de redes neuronales pre-entrenados, es posible encontrarse con objetos que no se desean detectar. La solución a este problema está dada en la definición de las clases que la red neuronal puede detectar, así como la definición de aquellas que debe ignorar, ambas mostradas en la Figura 4.4. La principal función de nuestro algoritmo es la detección y conteo de objetivos. La red neuronal utilizada es del tipo *SDDs* encontrada en *Caffe Model Zoo*. Dicha red está entrenada para detectar personas, aves, gatos, perros, vacas, caballos y ovejas. Dada la aplicación de este trabajo, las clases *ave, gato* y *perro* no son necesarias.

Posteriormente, se utiliza la herramienta *dnn* de las librerías de OpenCV (descrita en VI.3) para inicializar la red neuronal utilizando los modelos de los argumentos *–p* y *–m*, como se muestra en la Figura 4.5.

Dentro del ciclo principal del algoritmo se realiza el procesamiento de imágenes. El proceso inicia con la captura de la imagen, como se muestra en la Figura 4.6. Luego, se crea un arreglo con la información obtenida del sensor de la cámara por medio de la función *array* de la librería *numpy*. Después, se redimensiona la información. Debido a que la cámara proporciona un grupo de datos bastante extenso, se utiliza un cambio de

```
net=cv2.dnn.readNetFromCaffe(args["--p"], args["--m"])
```

Figura 4.5: Comando de inicialización de la red neuronal utilizando OpenCV.

```
array = ueye.get_data(pcImageMemory, width, height, nBitsPerPixel, pitch, copy=False)
frame = np.reshape(array,(height.value, width.value, bytes_per_pixel))
frame = cv2.resize(frame,(0,0),fx=0.5, fy=0.5)
hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
bgr = cv2.cvtColor(hsv, cv2.COLOR_HSV2BGR)
frame2 = imutils.resize(bgr, width=400)
```

Figura 4.6: Adquisición y acondicionamiento de la imagen.

```
blob = cv2.dnn.blobFromImage(cv2.resize(frame2, (300, 300)), (300, 300))
net.setInput(blob)
detecciones = net.forward()
```

Figura 4.7: Detección de grupos de pixeles y alimentación de la red neuronal.

espacio de color con las herramientas de OpenCV vistas en VI.1 como filtro, eliminando la información no deseada. Por último, se redimensiona la imagen con la función *resize* de la librería de *imutils*.

Ahora, la Figura 4.7 muestra el código para la detección de los grupos de pixeles conectados de la imagen que comparten ciertas características, para posteriormente alimentar la red neuronal con los grupos para su análisis.

En la Figura 4.8, se muestra el ciclo principal de la detección de objetos. Este se inicia extrayendo la probabilidad de una detección fuerte, es decir, la probabilidad de que la detección del objeto no cometa errores al distinguir entre las clases. El siguiente paso es verificar que esta probabilidad sea mayor al valor definido por el argumento $-c$, en cuyo caso, se generan los índices de las clases. Estos índices son filtrados para verificar si la red neuronal debe ignorarlos. Por último, se define un arreglo con las dimensiones de nuestra imagen y se calculan las coordenadas para dibujar la detección.

Por último, en la Figura 4.9, se presenta el segmento de código que manda llamar al algoritmo de detección de centroides, el cual nos da el valor de identificación de cada objeto para posteriormente mostrarlo en la imagen e incrementar el conteo de objetos, así mismo permite al algoritmo de detección seguir cada objeto y no contarlo más de una vez. Posteriormente, se muestra la imagen y se graba un archivo *.avi* para poder observar los resultados del algoritmo.

```
for i in np.arange(0, detecciones.shape[2]):
        confianza = detecciones[0, 0, i, 2]
        if confianza > args["--c"]:
           indice = int(detecciones[0, 0, i, 1])
           if CLASES[indice] in IGNORAR:
              continue
           cuadro = detecciones[0, 0, i, 3:7] * np.array([fW, fH, fW, fH])
           rectangulos.append(cuadro.astype("int"))
           (Xi, Yi, Xf, Yf) = cuadro.astype("int")
           label = "{}".format(CLASES[indice])
           cv2.rectangle(frame2, (Xi, Yi), (Xf, Yf), (255,0,0), 2)
           y = Yi - 15 if Yi - 15 > 15 else Yi + 15
           cv2.putText(frame2, label, (Xi, y), cv2.FONT_HERSHEY_PLAIN, 0.5, (255,0,0), 2)
```

Figura 4.8: Cíclo principal para la detección.

```
for (ID_objeto, centroide) in objetos.items():
    Identificacion = "ID {}".format(ID_objeto)
    cv2.putText(frame2, Identificacion, (centroide[0] - 10, centroide[1] - 10), (255, 0, 0), 2)
    cv2.circle(frame2, (centroide[0], centroide[1]), (255,0, 0))
    c_persona=ID_objeto+1
```

Figura 4.9: Cíclo para la detección de centroides y seguimiento de objetos.

I.2. Algoritmo de detección de centroides

```
def __init__(self, maxDesaparecio=2):
        self.ID_sig_objeto = 0
        self.objetos = OrderedDict()
        self.desaparecio = OrderedDict()
        self.maxDesaparecio = maxDesaparecio
```

Figura 4.10: Definición del constructor.

Para el diseño de este algoritmo, se requirieron las librerías de *scipy* para utilizar las funciones espaciales de medición de distancias entre objetos, *collections* para la definición de diccionarios que funcionan como bases de datos, y *numpy* para la definición de arreglos. El algoritmo se inicia con la definición de la clase *Seguidor_de_centroide*. Esta clase, incluye un constructor, mostrado en la Figura 4.10 para la inicialización de los miembros de datos de la clase, es decir, asignar su valor y mantenerlo para su constante uso.

En este caso, nuestro constructor cuenta con cuatro variables:

- *maxDesaparecio*: Es el número de cuadros consecutivos de imagen en los que un objeto puede ser marcado como "desaparecido" hasta que se borre del registro.

- *ID_sig_objeto*: Es un contador usado para asignar identificaciones unicas para cada objeto. En el caso de que un objeto desaparezca de la imagen y no regrese por la cantidad definida en *maxDesaparecio*.
- *objetos*: Es un diccionario que utiliza la identificación del objeto como clave y las coordenadas $(x,\ y)$ como valor.
- *desaparecio*: Es un diccionario que mantiene el valor de los cuadros consecutivos de la imagen que una identificación de un objeto en particular se ha mantenido como "desaparecido".

En la Figura 4.11, se define el método de *registro*, responsable de agregar nuevos objetos a nuestro seguidor. Este método recibe la información del centroide y lo agrega al diccionario *objetos*, definido anteriormente, utilizando la siguiente identificación de objetos disponible. Se inicializa en 0 las veces que el objeto ha desaparecido del cuadro de la imagen en el diccionario *desaparecio*. Finalmente se incrementa el valor de la identificación del siguiente objeto.

De igual manera, en la Figura 4.12, se define el método para eliminar del registro objetos que ya no se encuentran en el cuadro de imagen. Este método simplemente borra de los diccionarios *objetos* y *desaparecio* aquellos objetos que desaparecieron.

Ahora, lo que le permite al algoritmo realizar su función es el método de *actualización*. Este método acepta un listado de recuadros que se definen en el algoritmo de detección de objetos. En caso de no existir algún recuadro, se realiza un ciclo para aumentar la cuenta de *desaparecio*. Se revisa si ya se alcanzó el número máximo de cuadros consecutivos de imagen para que un objeto sea marcado como "desaparecido" y en caso de que se cumpla, se elimina el registro del objeto y se regresa el conjunto de objetos rastreables (Figura 4.13).

```
def registro(self, centroide):
        self.objetos[self.ID_sig_objeto] = centroide
        self.desaparecio[self.ID_sig_objeto] = 0
        self.ID_sig_objeto += 1
```

Figura 4.11: Método de registro de objetos.

```
def desregistro(self, ID_objeto):
        del self.objetos[ID_objeto]
        del self.desaparecio[ID_objeto]
```

Figura 4.12: Método de eliminación de registro de objetos.

```
def actualizar(self, rectangulos):
        if len(rectangulos) == 0:
                for ID_objeto in self.desaparecio.keys():
                        self.desaparecio[ID_objeto] += 1
                        if self.desaparecio[ID_objeto] > self.maxDesaparecio:
                                self.desregistro(ID_objeto)
                return self.objetos
```

Figura 4.13: Revisión por objetos "perdidos".

```
in_Centroides = np.zeros((len(rectangulos), 2), dtype="int")

for (i, (Xi, Yi, Xf, Yf)) in enumerate(rectangulos):
        coor_X = int((Xi + Xf) / 2.0)
        coor_Y = int((Yi + Yf) / 2.0)
        in_Centroides[i] = (coor_X, coor_Y)

if len(self.objetos) == 0:
        for i in range(0, len(in_Centroides)):
                self.registro(in_Centroides[i])
```

Figura 4.14: Ciclo para cálculo de centroide para cada recuadro.

En el caso de que si existan recuadros de detección, se inicializa un arreglo con *numpy* para guardar los centroides. Se realiza un ciclo para calcular el centroide de cada recuadro y se almacena. Si no hay objetos que ya se estén siguiendo, se registran los nuevos objetos, (Figura 4.14). De otra manera, es necesario actualizar las coordenadas $(x,\ y)$ del objeto existente basado en la localización del centroide que minimice la distancia Euclidiana entre el centroide de entrada y el centroide del objeto (Figura 4.15).

La idea principal es seguir los objetos y mantener la identificación correcta para cada uno. Esto se logra calculando la distancia Euclidiana entre todos los pares de centroides para después asociar aquellos que minimizan esta distancia. Primero se toman los valores de los centroides de los objetos y de los centroides de entrada. Luego, se calculan las distancias y a su vez éstas se mapean en un arreglo de Numpy D. Posteriormente, se calculan los valores máximos y mínimos para cada fila y columna, y clasificando los valores basados en los mínimos. Una vez realizado esto, se obtiene un índice con los valores mínimos al inicio de la lista.

Se inicializan dos conjuntos para determinar cuales filas y columnas ya se han utilizado. Luego, se realiza un ciclo sobre las combinaciones (*fila, columna*) para actualizar los centroides. Si alguno de los dos valores ya se utilizó, se ignora. De lo contrario, se encontró el centroide de entrada que tiene la menor distancia Euclidiana con respecto al centroide del objeto y que no concuerda con algún otro objeto y por lo tanto se actualiza el centroide del objeto, se agregan las filas y columnas ya utilizadas a su respectivo conjunto, (Figura 4.16). Para aquellas no utilizadas se definen dos conjuntos (Figura 4.17).

```
else:
        IDs = list(self.objetos.keys())
        Centroides_objetos = list(self.objetos.values())
        D = dist.cdist(np.array(Centroides_objetos), in_Centroides)
        filas = D.min(axis=1).argsort()
        columnas = D.argmin(axis=1)[filas]
```

Figura 4.15: Registro de nuevos objetos.

```
filas_usadas = set()
columnas_usadas = set()

for (fila, columna) in zip(filas, columnas):
        if fila in filas_usadas or columna in columnas_usadas:
                continue
        ID_objeto = IDs[fila]
        self.objetos[ID_objeto] = in_Centroides[columna]
        self.desaparecio[ID_objeto] = 0
        filas_usadas.add(fila)
        columnas_usadas.add(columna)
```

Figura 4.16: Calculo de distancia entre cada par de centroides.

```
filas_sin_usar = set(range(0, D.shape[0])).difference(filas_usadas)
columnas_sin_usar = set(range(0, D.shape[1])).difference(columnas_usadas)
```

Figura 4.17: Mapeo de valores mínimos y máximos de distancias Euclidianas.

```
if D.shape[0] >= D.shape[1]:
        for fila in filas_sin_usar:
                ID_objeto = IDs[fila]
                self.desaparecio[ID_objeto] += 1

                if self.desaparecio[ID_objeto] > self.maxDesaparecio:
                        self.desregistro(ID_objeto)

else:
        for columna in columnas_sin_usar:
                self.registro(in_Centroides[columna])

return self.objetos
```

Figura 4.18: Verificación de objetos que han desaparecido durante el procesamiento de los centroides.

Para terminar, si el número de centroides de objetos es mayor o igual que los centroides de entrada, debe verificarse si alguno de los objetos desapareció de la imagen a través de un ciclo sobre los conjuntos de filas y columnas no utilizadas. Dentro de este ciclo, se

incrementa la cuenta *desaparecio* y se verifica si excede el valor de *maxDesaparecio*, en cuyo caso, se elimina el objeto del registro. En el caso de que el centroide de entrada sea mayor al de los objetos, se realiza un ciclo para registrar todos los nuevos centroides. Una vez realizados los procesos anteriores, la función regresa el conjunto de objetos rastreables para ser utilizados por el algoritmo de detección de objetos (Figura 4.18).

II. Síntesis de los algoritmos propuestos

Los algoritmos diseñados anteriormente quedan formalmente descritos a continuación.

Algoritmo 1: Detección de centroides

Entrada: Objetos detectados y rectángulos de la detección
Salida: Centroide e Identificación del objeto

Inicializar identificación de objetos: *IDsiguiente*;
Inicializar diccionarios: *objetos, desapararecio*;
Definir máximos *frames* en que desaparece el objeto: *maxDesaparecio*;
Definir función de registro de objetos;
Definir función borrador de objetos;
Definir función de actualización de objetos;
Inicializar arreglo de centroides: *inCentroide*;
para *i, (X_i, Y_i, X_f, Y_f) en rectangulos* **hacer**
    $coor_X = int((X_i + X_f)/2.0)$;
    $coor_Y = int((Y_i + Y_f)/2.0)$;
    $inCentroide[i] = (coor_X, coor_Y)$;
fin
si $len(objetos) == 0$ **entonces**
    para *i en* $range(0, len(inCentroide))$ **hacer**
        Llamar función de registro;
        Resetear *desaparecio*;
    fin
en otro caso
    Generar la identificación de los objetos: *IDs*;
    Calcular la distancia entre los centroides: D;
    Encontrar valores máximos y mínimos del arreglo para clasificar los objetos;
    para *(filas,columnas) en inCentroide* **hacer**
        Tomar la identificación del objeto y asignarla como el nuevo centroide: *IDobjeto*;
        Llamar función de actualización;
    fin
    Calcular índices no analizados;
    Revisar si algún objeto ya desapareció;
fin
Regresar valor de identificación de objetos y centroide;

Algoritmo 2: Detección de objetos y conteo de objetivos

Entrada: Imagen obtenida de la cámara
Salida: Objeto detectado y conteo de objetos detectados

```
Inicializar la cámara IDS;
Leer la información del sensor de la cámara;
Activar auto-shutter para ajuste de brillo automático;
Definir modo de visualización;
Seleccionar modo de color de la cámara;
Seleccionar tamaño y posición del área de AOI;
Selección de memoria para definir dimensiones y profundidad de color;
Activar video streaming: nRet;
Definir los argumentos a utilizar: ap, args;
Inicializar las clases que la red neuronal puede detectar;
Cargar modelo de Caffe: net;
mientras nRet == ueye.IS_SUCCESS hacer
    Extraer la imagen de la memoria: array;
    Crear un arreglo de numpy: frame;
    Redimensionar la imagen a la mitad: frame;
    Convertir la imagen de BGR a HSV: hsv;
    Convertir la imagen de HSV a BGR: bgr;
    Redimensionar la imagen: frame2;
    Medir la imagen y guardar las dimensiones: fH,fW;
    Detectar los pixeles que comparten propiedades: blob;
    Definir la entrada de la red neuronal;
    Alimentar la red neuronal con la entrada: detecciones;
    para i en np.arrange hacer
        Extraer la confianza de la detección: confianza;
        si confianza <args[confianza] entonces
            Obtener los índices de las clases: indice;
            Calcular las coordenadas para dibujar la detección:cuadro;
            Dibujar la detección y escribir el índice de la clase detectada:rectangulos;
        fin
    fin
    Definir objetos detectados: objetos;
    para ID_Objetos y centroide en objetos hacer
        Escribir el número de identifiación del objeto;
        Dibujar el centroide y la identificación;
        Incrementar conteo: c;
    fin
    Grabar video y mostrar imagen procesada;
fin
```

Capítulo 5

Resultados

En este Capítulo se muestran los resultados obtenidos durante el desarrollo de la tesis. Para ello, el capítulo se ha dividido en dos secciones con el fin de separar los resultados en simulación de los resultados experimentales. La primera sección está dedicada a los resultados obtenidos de los dos controladores diseñados en el capítulo 3, los cuales fueron probados con tres tipos de trayectorias, y los resultados de simulación de un Pixhawk. La segunda sección presenta los resultados experimentales del algoritmo de visión propuesto.

En el caso de los controladores, se muestran resultados en simulación dado que hasta el día de hoy no se logró tener éxito en la comprobación experimental del cambio del controlador de vuelo del Quad-Rotor 3DRobotics, pasando de un PixHawk versión 1 a la versión 2, la cual posee más memoria y por ende permite cargar leyes de control diseñadas desde el ambiente Matlab Simulink (más datos de la versión 2 de este controlador pueden consultarse en el Anexo C y en [38]).

I. Resultados de simulación

A continuación se presentan los resultados de las simulaciones de los controladores vistos en el Capítulo 3 para el Quad-Rotor de *3DRobotics* cuyos parámetros se muestran en la Tabla 5.1. Considerando que una de las ideas base de esta tesis es la detección de personas y/o ganado, se propusieron tres tipos de trayectorias pensando en las formas en las que se pudiesen realizar barridos de superficies terrestres. Las simulaciones fueron realizadas en MATLAB Simulink, se utilizó el método de integración de Euler con un paso de muestreo de 0.001 segundos.

Tabla 5.1: Parámetros que caracterizan al Quad-Rotor de 3DRobotics.

Parámetros	Valor
Masa	1 [Kg]
Inercias: I_{xx}, I_{yy}, I_{zz}	0.031; 0.03; 0.04 $[kg \cdot m^2]$
Medida de hélices	10x4.7 [in]
Longitud del brazo del robot	0.2 [m]

Para las trayectorias deseadas, se considera a z_d como:

$$z_d = h\omega(1 + \tanh(t - 5)) \tag{5.1}$$

donde $h = 4$[m] representa el valor de la altura deseada, $\omega = \pi/6$ [rad/s], t representa el tiempo de simulación, esto con el fin de generar una trayectoria de arranque en z que permita al robot disminuir el par de arranque.

I.1. Trayectoria circular

Las ecuaciones paramétricas que definen la trayectoria circular están dadas por:

$$x_d = a(\arctan(\varphi) + \arctan(t - \varphi))\cos(\omega t),$$
$$y_d = a(\arctan(\varphi) + \arctan(t - \varphi))\sin(\omega t)$$

donde x_d y y_d son los valores deseados para las posiciones de x y y, $\varphi = \pi/12$ [rad], $a = 2$[m] representa la amplitud que define el diámetro del círculo, $\omega = \pi/6$ [rad/s] representa la frecuencia, t representa el tiempo. Las condiciones iniciales para la posición son $[x(0) \quad y(0) \quad z(0)]^T = [0 \quad 0 \quad 0]^T$ y para la orientación son $[\phi(0) \quad \theta(0) \quad \psi(0)]^T = [0 \quad 0 \quad 0]^T$.

Control PID

Las ganancias utilizadas, sintonizadas heurísticamente, para esta trayectoria se muestran en la Tabla 5.2 En la Figura 5.1, se muestra la trayectoria en el plano $X - Y$ y en el espacio $X - Y - Z$ donde la línea negra representa la trayectoria deseada, mientras que la línea punteada en rojo representa la trayectoria realizada por el vehículo. Una vez que el robot ha alcanzado la referencia en Z, existe un transitorio para llegar a las referencias en X y en Y que se debe a las funciones arctan$(\cdot)$ de la trayectoria. Estas funciones se utilizan para que el sistema realice un seguimiento de una función más suave.

Tabla 5.2: Ganancias seleccionadas para la trayectoria circular con PID.

Ganancia	**Valor (x, y, z, ϕ, θ, ψ)**
K_p	25, 25, 95, 85, 25, 100
K_i	40, 40, 102, 6, 3, 0
K_d	10, 17, 20, 9, 8, 10

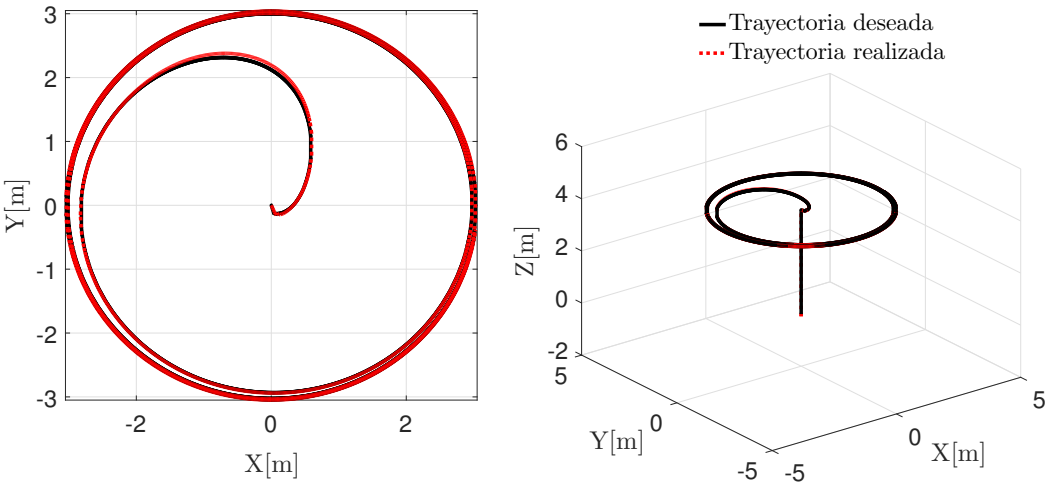

Figura 5.1: Trayectoria circular realizada por el robot en el plano y en el espacio con control PID.

En la Figura 5.2, se muestran las trayectorias en X, Y y Z respectivamente, donde la línea negra representa la trayectoria deseada y la línea punteada roja la trayectoria realizada. Durante los primeros segundos de la simulación el robot se mantiene inmóvil para posteriormente, realizar la trayectoria en Z, la cual se definió con la función suave $\tanh(\cdot)$. Cuando $t = 10$, se inicia el seguimiento de las trayectorias en X y Y de forma casi inmediata.

Los ángulos ϕ, θ y ψ se muestran en la Figura 5.3 donde la línea negra representa la referencia y la línea roja punteada representa el movimiento angular descrito por el robot. Durante los primeros 10 segundos de la simulación, permanecen en cero. Cuando $t = 10[s]$ se puede observar que los ángulos tienen un impulso inicial, que en el caso de ϕ alcanza el límite de $\pi/4$, esto se debe a que se inicia el seguimiento de la trayectoria deseada sobre los ejes X y Y. Posteriormente, cuando $t = 15[s]$ los ángulos oscilan entre valores de -10 y 10 grados. Cabe resaltar que los ángulos ϕ y θ no pueden permanecer en cero debido a que el control virtual es el que permite el movimiento $x - y$, esto quiere decir que se deben modificar estos ángulos para lograr el control de la posición.

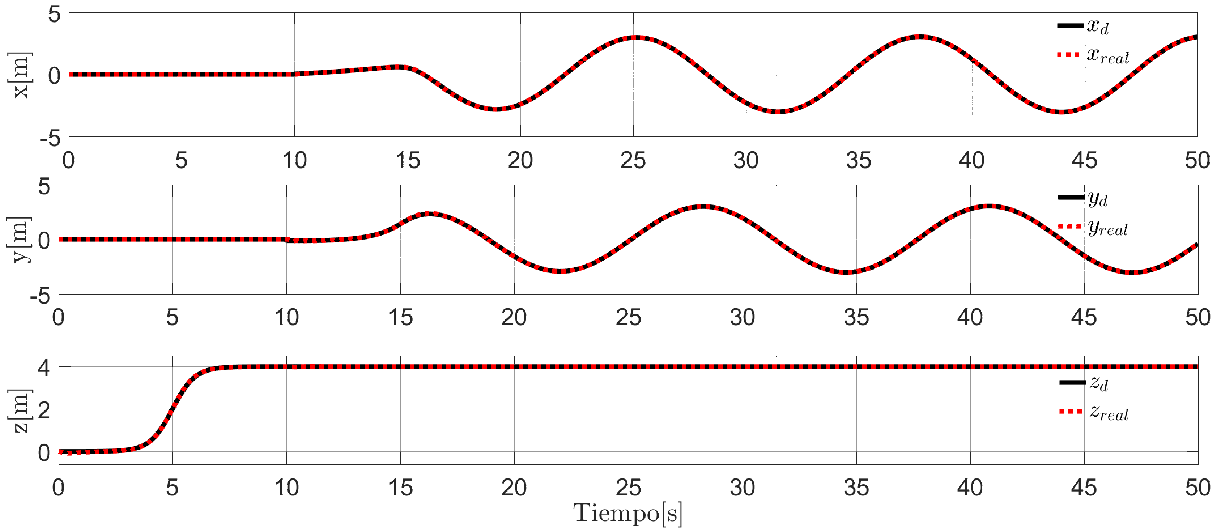

Figura 5.2: Posiciones del sistema al seguir la trayectoria circular con control PID.

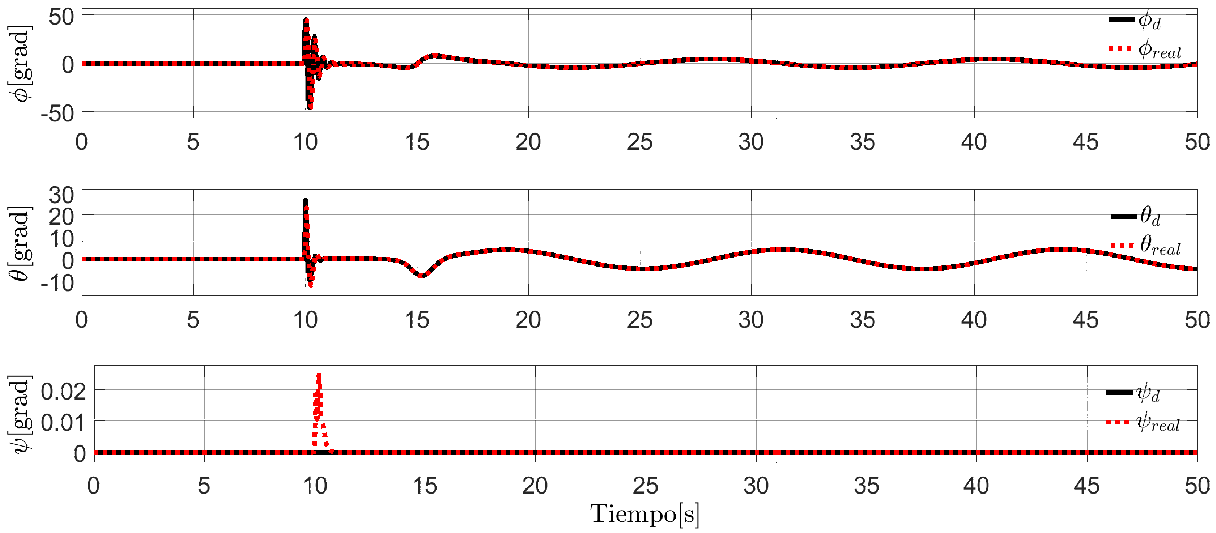

Figura 5.3: Ángulos del sistema al seguir la trayectoria circular con control PID.

Los errores de posición y orientación del robot se representan en la Figura 5.4, estos tienden a cero de forma asintótica, lo que significa que en realidad nunca son cero pero se mantienen en una región cercana a este.

Las señales de control se muestran en la Figura 5.5. U_ϕ, U_θ y U_ψ las cuales inician en cero y permanecen ahí mientras que U_z aumenta hasta tomar un valor constante que

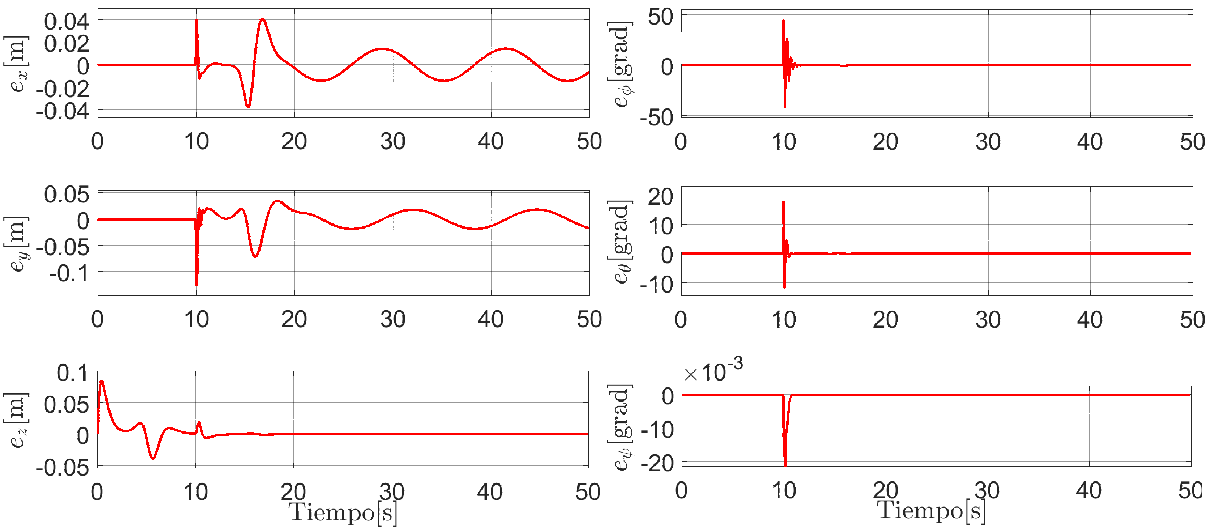

Figura 5.4: Errores de posición y orientación del sistema al seguir la trayectoria circular con control PID.

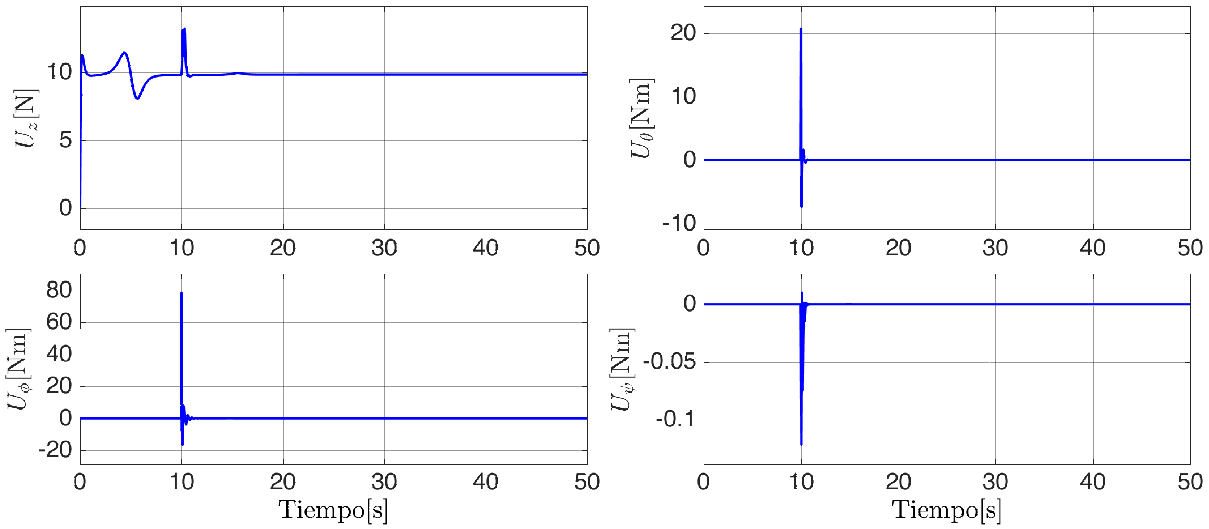

Figura 5.5: Entradas de control del sistema al seguir la trayectoria circular con control PID.

permite al robot mantenerse en la referencia sobre Z. Los pares generados tienden a cero conforme se alcanza la trayectoria. En cuanto al par U_z se mantendrá en un valor diferente a cero siempre para mantener al robot en la altura deseada.

Control por Backstepping

Tabla 5.3: Ganancias seleccionadas para la trayectoria circular con Backstepping.

Ganancia	Valor	Ganancia	Valor
K_1	100	K_7	2.5
K_2	40	K_8	2.5
K_3	90	K_9	1.5
K_4	40	K_{10}	0.3
K_5	100	K_{11}	3
K_6	100	K_{12}	0.2

Las ganancias seleccionadas para la trayectoria circular se muestran el la Tabla 5.3. En la figura 5.6, se muestra la trayectoria en el plano $X-Y$ y en el espacio $X-Y-Z$. La línea negra representa la trayectoria deseada mientras la línea punteada roja representa la trayectoria realizada por el robot. A diferencia del PID, después de alcanzar la referencia en Z, el transitorio para llegar a las referencias X y Y es más corto. Así mismo, en la Figura 5.7, se observan las trayectoria individuales para X, Y y Z, donde se observa que en realidad, debido a la respuesta asintótica del control, no son alcanzadas totalmente.

Los ángulos del sistema son mostrados en la Figura 5.8. Como se mencionó anteriormente, el comportamiento oscilante de los ángulos se debe al control virtual para poder realizar movimientos en X y Y. Para este controlador, los valores de los ángulos oscilan entre -5 y 5 grados, mostrando un mejor resultado en comparación con el PID.

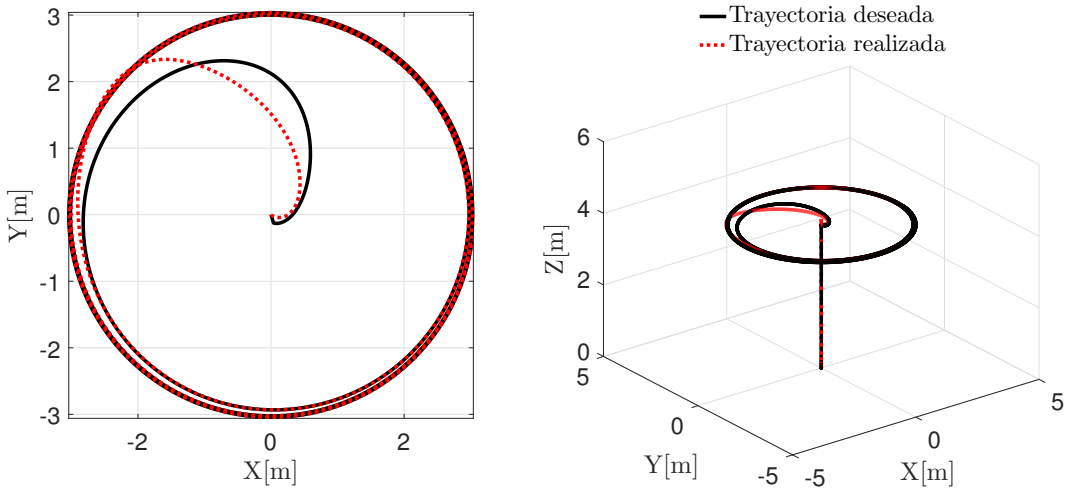

Figura 5.6: Trayectoria realizada en el plano y en el espacio por el robot con control Backstepping.

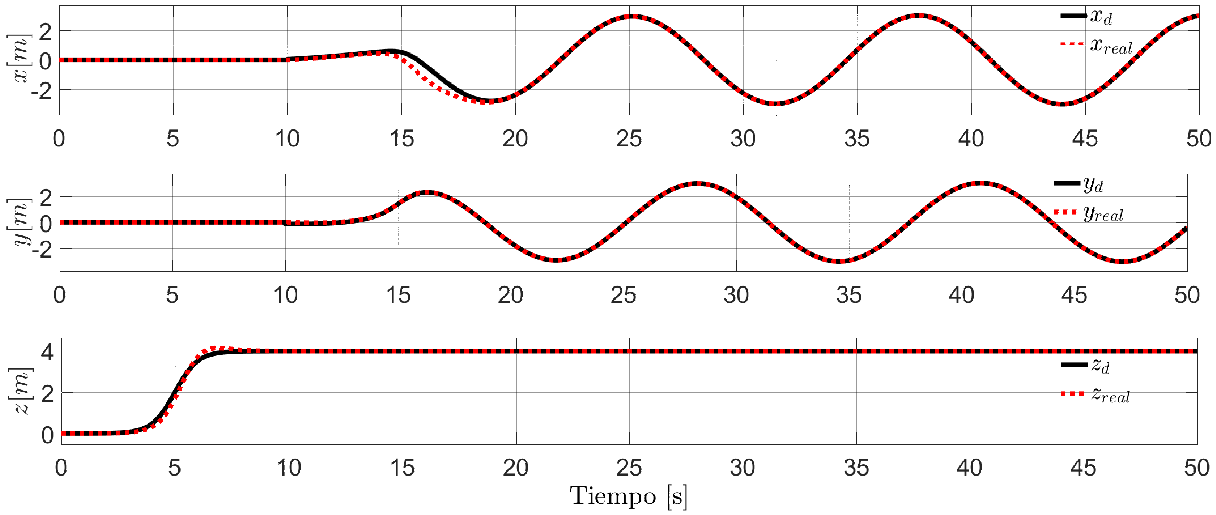

Figura 5.7: Posiciones del sistema al seguir la trayectoria circular con control Backstepping.

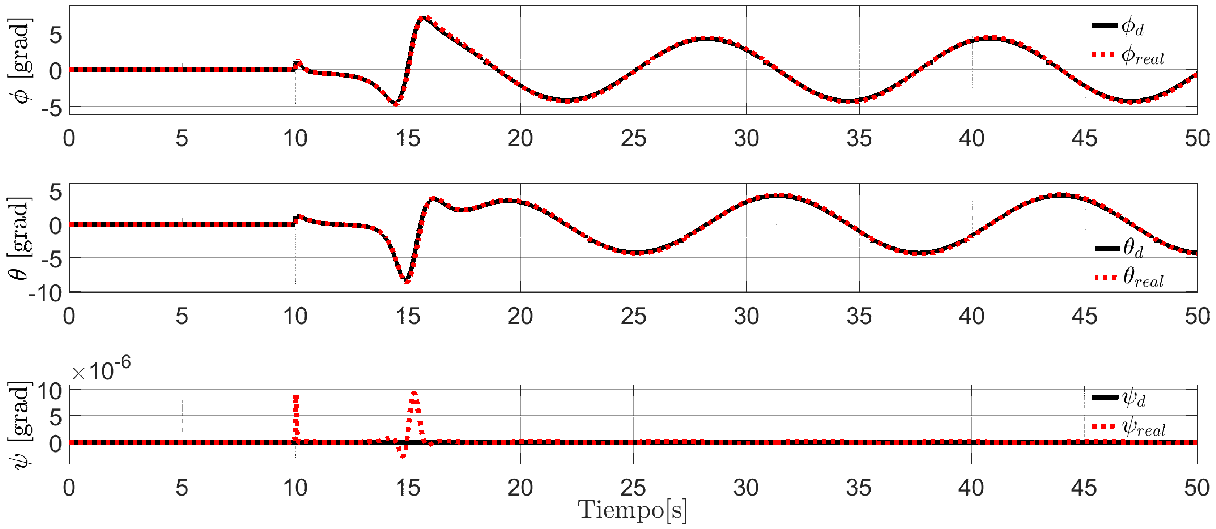

Figura 5.8: Ángulos del sistema al seguir la trayectoria circular con control Backstepping.

En la Figura 5.9, los errores de posición y orientación permiten ver que el control por Backstepping muestra un error menor en los ángulos al iniciar el seguimiento de la trayectoria, y se mantiene oscilando en valores pequeños cerca del cero, mientras que en el PID, el error inicial es elevado, llegando hasta los 45 grados. En la Figura 5.10, se muestran las señales de control generadas, las cuales, a excepción de U_z muestran valores muy pequeños en comparación con las señales generadas por el control PID.

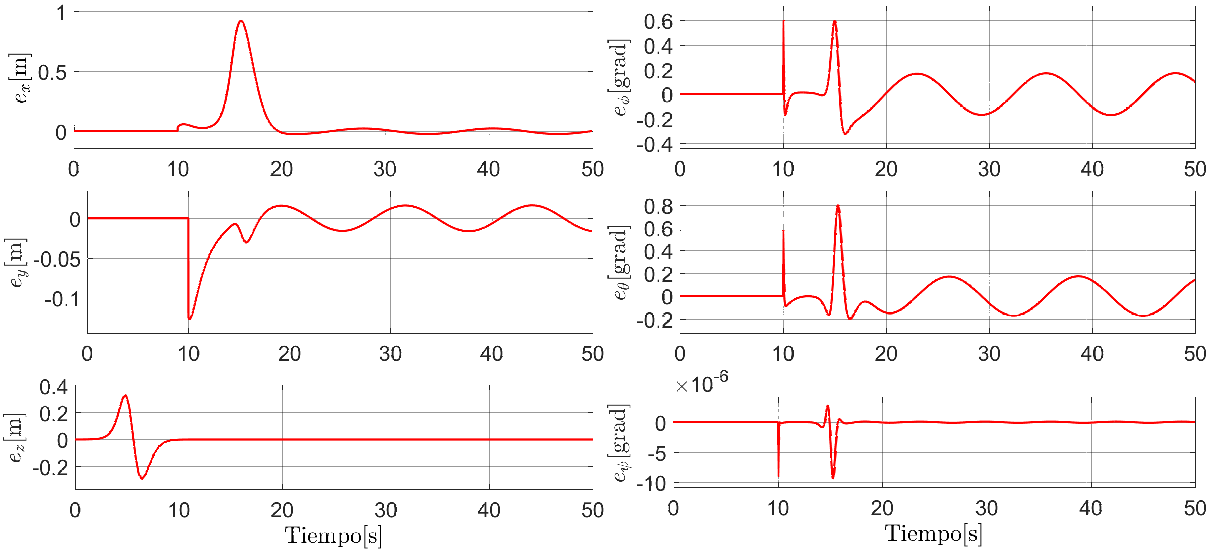

Figura 5.9: Errores de posición y orientación del sistema al seguir la trayectoria circular con control Backstepping.

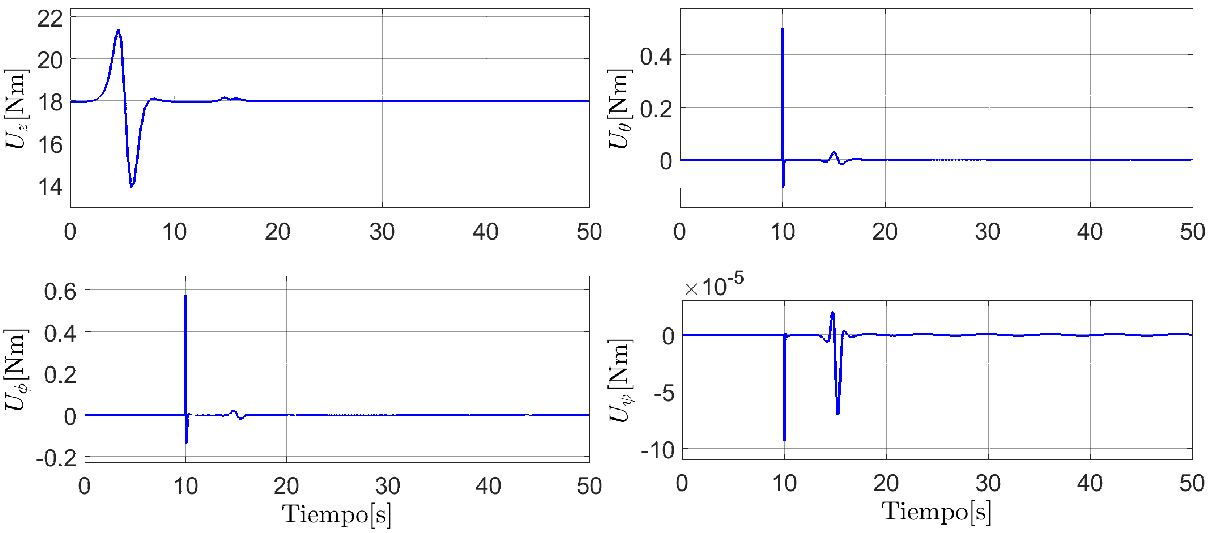

Figura 5.10: Entradas de control del sistema al seguir la trayectoria circular con control Backstepping.

I.2. Trayectoria Lemniscata

Las ecuaciones que definen la trayectoria de la Lemniscata utilizada están dadas por:

$$x_d = A(\arctan(\varphi) + \arctan(t-15))\cos(\omega t),$$
$$y_d = B(\arctan(\varphi) + \arctan(t-15))\sin(2\omega t)$$

donde x_d y y_d son los valores deseados para las posiciones de en x y y, $\varphi = \pi/12$ [rad], $A = 1$[m], $B = 2$[m] representan la amplitud que define la trayectoria, $\omega = \pi/6$ [rad/s] representa la frecuencia, t representa el tiempo de muestreo de la simulación. Las condiciones iniciales para la posición son $[x(0) \quad y(0) \quad z(0)]^T = [0 \quad 0 \quad 0]^T$ y para la orientación son $[\phi(0) \quad \theta(0) \quad \psi(0)]^T = [0 \quad 0 \quad 0]^T$.

Control PID

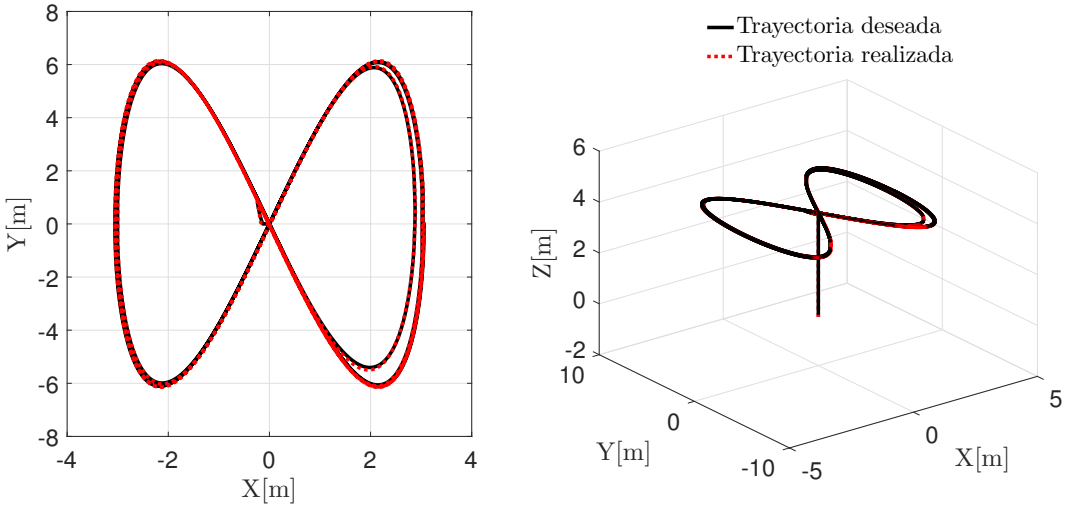

Figura 5.11: Trayectoria realizada en el plano y en el espacio por el robot con control PID.

Tabla 5.4: Ganancias seleccionadas para la trayectoria lemniscata con PID.

Ganancia	**Valor (x, y, z, ϕ, θ, ψ)**
K_p	45, 45, 105, 85, 25, 100
K_i	40, 40, 102, 6, 6, 0
K_d	10, 25, 20, 8, 8, 10

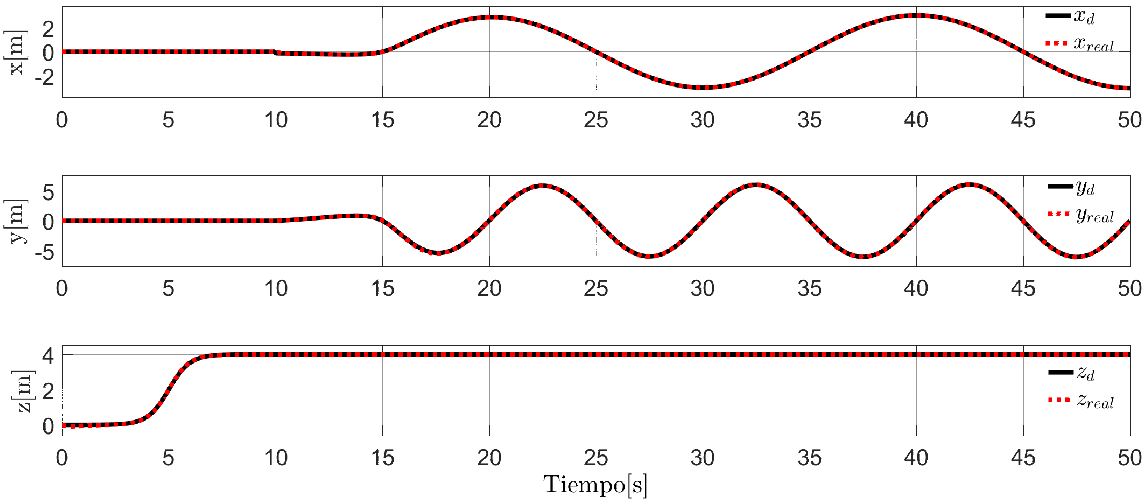

Figura 5.12: Posiciones del sistema al seguir la trayectoria lemniscata con control PID.

Las ganancias utilizadas para esta trayectoria se muestran el la Tabla 5.4. En la Figura 5.11, se muestran las trayectorias en el plano $X-Y$ y en el espacio $X-Y-Z$ donde la trayectoria deseada se representa con la línea negra y la trayectoria realizada por el robot se representa con la línea punteada roja. Se puede observar que la trayectoria es alcanzada con un transitorio corto. En la Figura 5.12, se pueden observar las señales para x, y y z por separado. Los ángulos del sistema mostrados en la Figura 5.13, de la respuesta al iniciar el seguimiento, no presenta un sobre impulso como en la trayectoria circular. Sin embargo, el sistema se mantiene con una oscilación que es mayor, al inicio del seguimiento, en el ángulo ϕ y un sobre impulso alto en el ángulo θ.

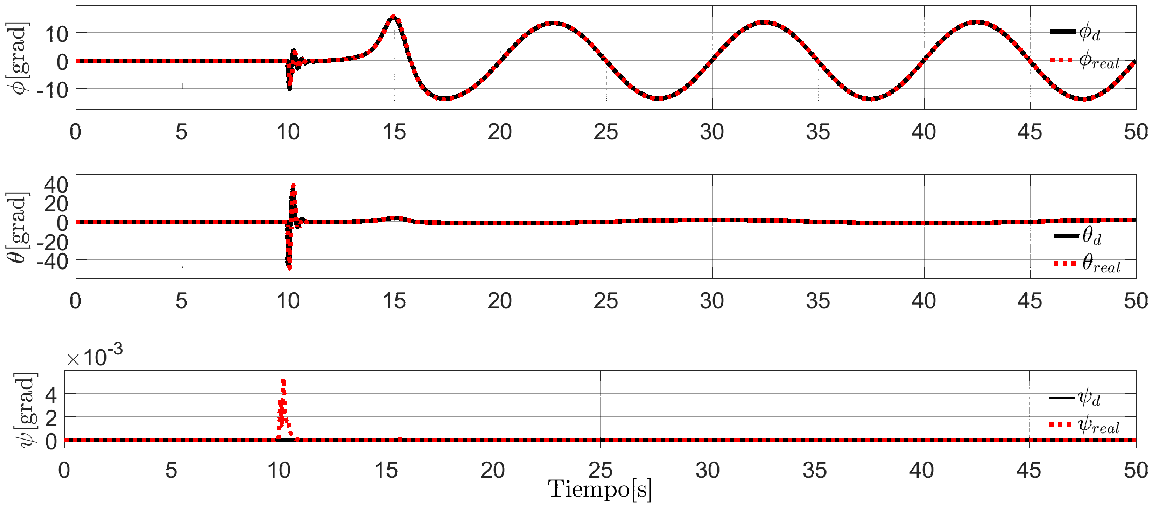

Figura 5.13: Ángulos del sistema al seguir la trayectoria lemniscata con control PID.

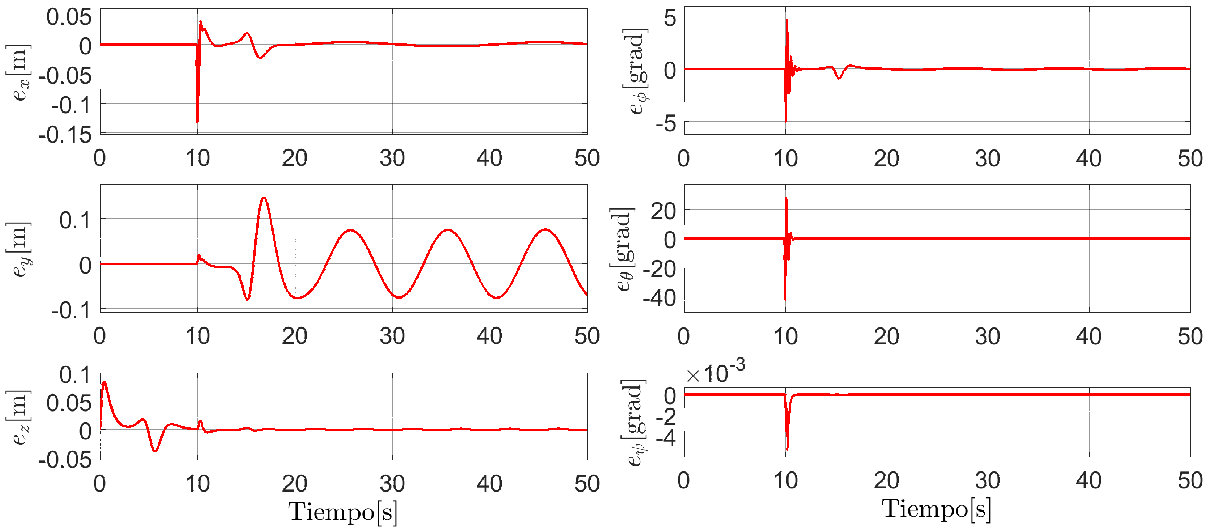

Figura 5.14: Errores de posición y orientación del sistema al seguir la trayectoria lemniscata con control PID.

Los errores de seguimiento mostrados en la Figura 5.14, tienden a una región cercana al cero y se mantienen ahí sin llegar al origen, tanto para la posición como para la orientación.

En la Figura 5.15, las entrada de control tienen su mayor valor al inicio del seguimiento y disminuyendo conforme la trayectoria es alcanzada.

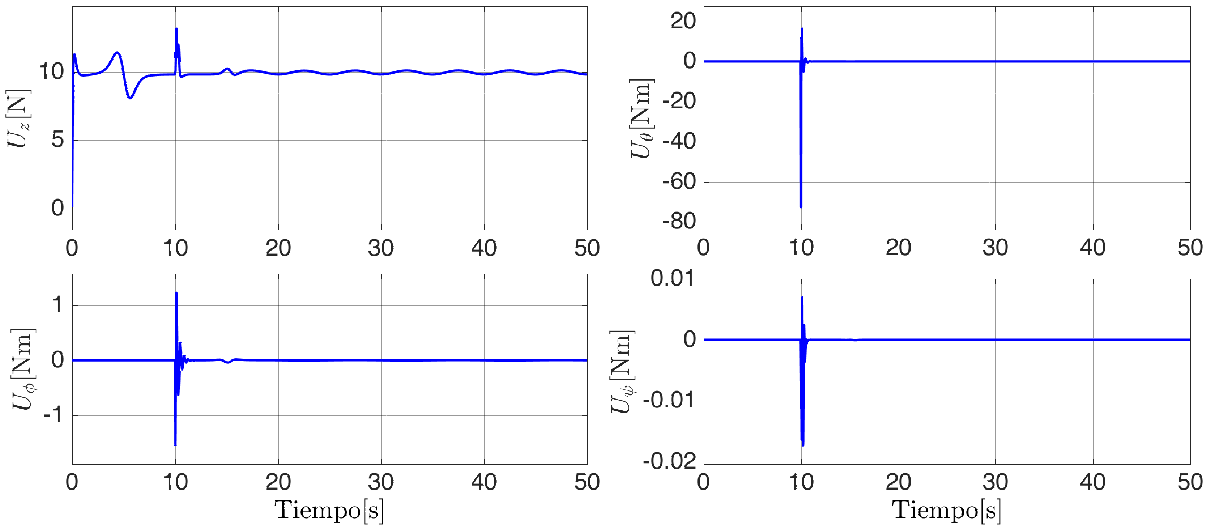

Figura 5.15: Entradas de control del sistema al seguir la trayectoria lemniscata con control PID.

Control por Backstepping

Tabla 5.5: Ganancias seleccionadas para la trayectoria lemniscata con Backstepping.

Ganancia	Valor	Ganancia	Valor
K_1	100	K_7	2.5
K_2	40	K_8	2.5
K_3	90	K_9	0.7
K_4	40	K_{10}	0.8
K_5	100	K_{11}	2
K_6	100	K_{12}	0.7

Las ganancias seleccionadas para la trayectoria lemniscata se muestran en la tabla 5.5. En la Figura 5.16, se muestran las trayectorias en el plano $X-Y$ y en el espacio $X-Y-Z$ donde la trayectoria deseada se representa con la línea negra y la trayectoria realizada por el robot se representa con la línea punteada roja. Se puede observar que la trayectoria es alcanzada con un transitorio similar al obtenido en el control PID. En la Figura 5.17, se pueden observar las señales para x, y y z por separado.

La respuesta del sistema en la orientación es mostrada en la Figura 5.18, donde se puede observar que los ángulos deseados son alcanzados casi de inmediato al igual que en el PID, sin embargo, el control por Backstepping muestra una mejor respuesta al inicio del seguimiento.

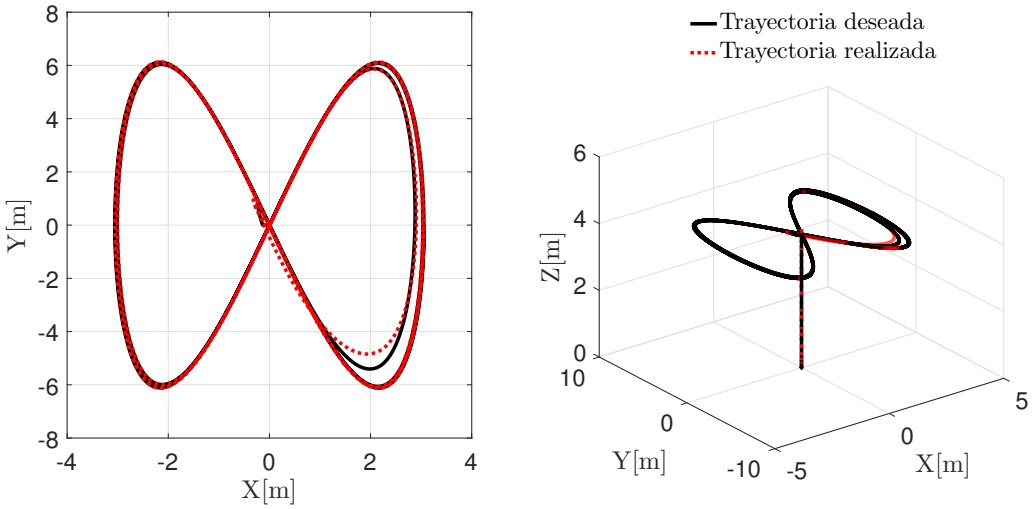

Figura 5.16: Trayectoria realizada en el plano y en el espacio por el robot con control Backstepping.

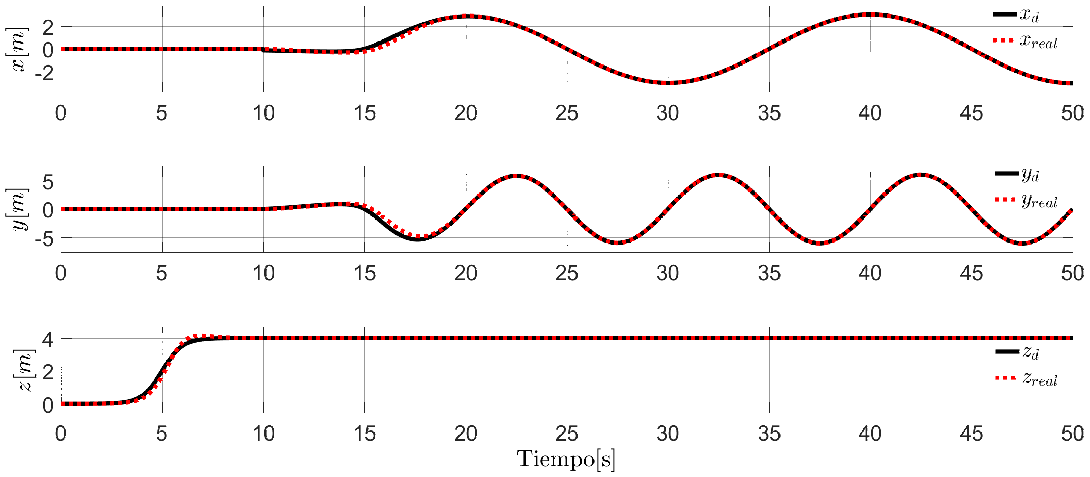

Figura 5.17: Posiciones del sistema al seguir la trayectoria lemniscata con control Backstepping.

Los errores de posición y orientación mostrados en la Figura 5.19 permanecen oscilando en una región cercana a cero, sin alcanzarlo. De la misma manera, se puede observar una mejor respuesta por parte del Backstepping al inicio del seguimiento de las trayectorias, por lo que los errores cuando $t = 10[s]$ muestran un mejor comportamiento que en el PID.

Las señales de control mostradas en la Figura 5.20 muestran nuevamente valores menores a los obtenidos con el PID, a excepción del control U_z.

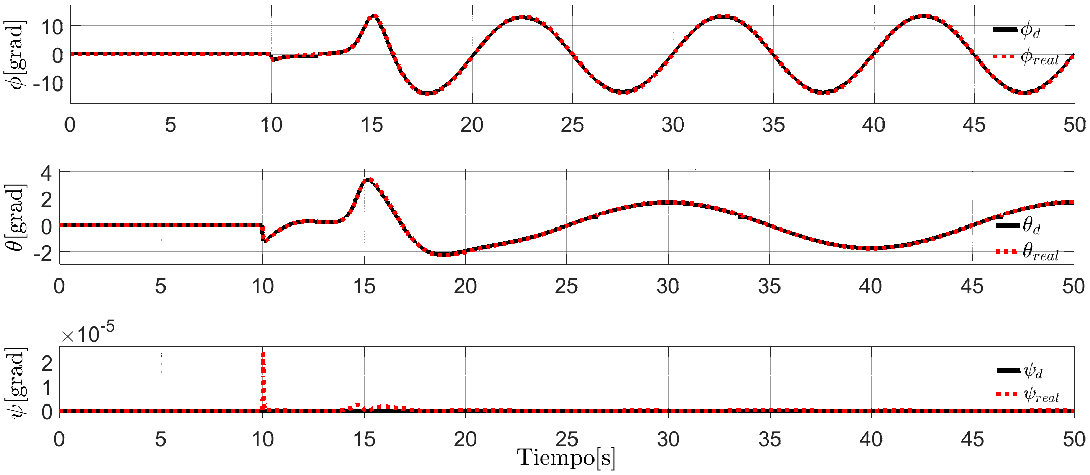

Figura 5.18: Ángulos del sistema al seguir la trayectoria lemniscata con control Backstepping.

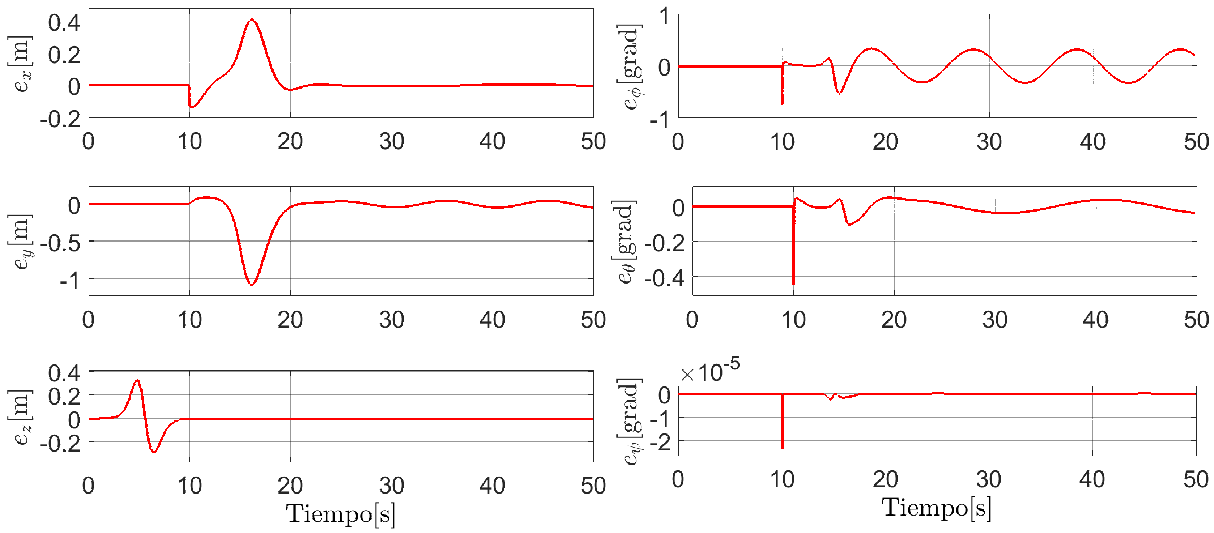

Figura 5.19: Errores de posición y orientación del sistema al seguir la trayectoria lemniscata con control Backstepping.

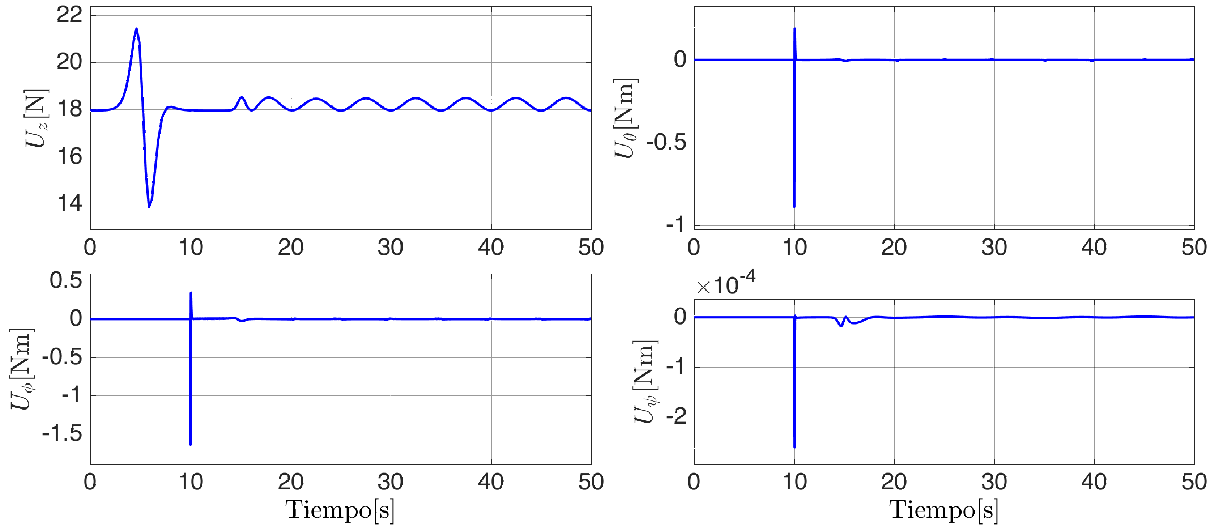

Figura 5.20: Entradas de control del sistema al seguir la trayectoria lemniscata con control Backstepping.

I.3. Trayectoria Lineal

Dada la aplicación que se quiere lograr en este trabajo, el seguimiento de trayectorias lineales resulta de gran interés. Para realizar el seguimiento de una línea en el espacio, es necesario establecer los valores de la posición inicial y los valores de la posición final y utilizar la ecuación de la recta siguiente:

$$x_d = x_1 + (x_2 - x_1)\lambda,$$
$$y_d = y_1 + (y_2 - y_1)\lambda$$

donde x_d y y_d son los valores deseados para x y y; x_1 y x_2 representan el valor inicial y el valor final de la recta en el eje X; y_1 y y_2 representan el valor inicial y el valor final de la recta en eje Y, y λ tiene un rango de $0 \leq \lambda \leq 1$, por lo que, para generar la velocidad de la trayectoria basta con definir la velocidad de incremento de λ. Por lo tanto, λ se define en función del tiempo como:

$$\lambda = \frac{1}{k}t$$

Definiendo a $k \geq 1$, es decir, si $k = 1$, la trayectoria lineal alcanzará el valor final deseado en 1 segundo. Por lo tanto, para disminuir la velocidad de avance de la trayectoria, se aumenta el valor de k.

Control PID

Para la simulación de la trayectoria se seleccionaron las ganancias mostradas en la Tabla 5.6. En la Figura 5.21, se muestran la trayectorias en el plano $X - Y$ así como en el espacio $X - Y - Z$. Se seleccionó una trayectoria que tuviera la forma en "S" debido a que, para fines de nuestro trabajo, resulta conveniente una trayectoria que permita al robot recorrer un campo para realizar búsquedas y conteos de objetivos. Al inicio de la simulación, el control es capaz de alcanzar la trayectoria, sin embargo, a partir del punto $(5, 0)$ el robot sigue la forma pero no alcanza por completo a realizar la tarea deseada.

Tabla 5.6: Ganancias seleccionadas para la trayectoria lemniscata con PID.

Ganancia	**Valor (x, y, z, ϕ, θ, ψ)**
K_p	8, 8, 95, 8, 8, 100
K_i	0.6, 0.6, 0.6, 0.6, 0.6, 0
K_d	6, 6, 6, 6, 6, 10

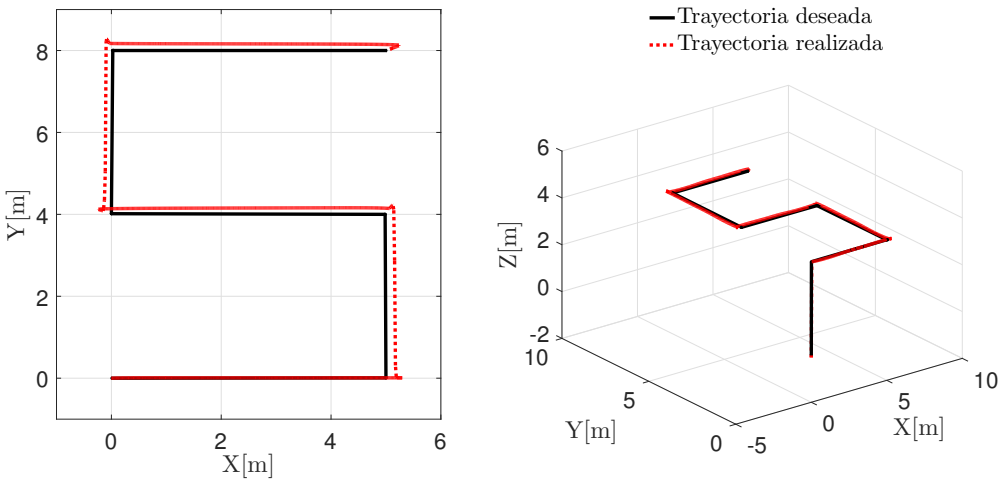

Figura 5.21: Trayectoria realizada en el plano y en el espacio por el robot.

En la Figura 5.22, se muestran la señales para x, y y z donde la trayectoria deseada es representada por la línea negra y la trayectoria realizada por el robot es representada por la línea punteada roja. Se puede observar que el robot alcanza la trayectoria y tiene un pequeño sobre impulso que no le permite permanecer sobre el valor deseado. Esto es debido a que la sintonización que se realizó no se lo permite. Es importante mencionar esto ya que, en la Figura 5.23, el ángulo θ, al momento de realizar un cambio en la trayectoria, llegan a los valores máximos de $\pi/4$ que permite el robot. Cambiando la sintonización estos valores exceden el máximo y se mantienen oscilando de forma que el robot correría el riesgo de caer.

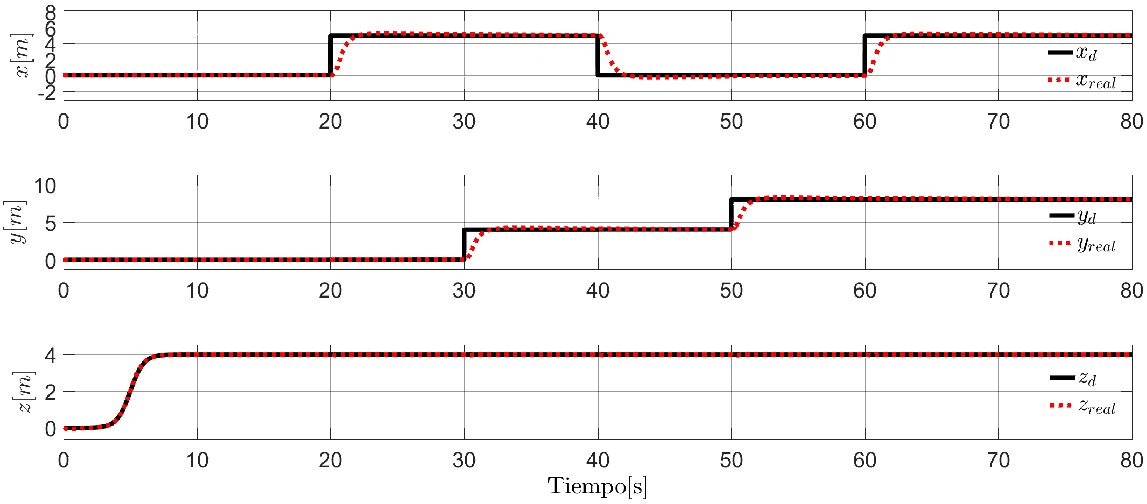

Figura 5.22: Posiciones del sistema al seguir la trayectoria lineal con control PID.

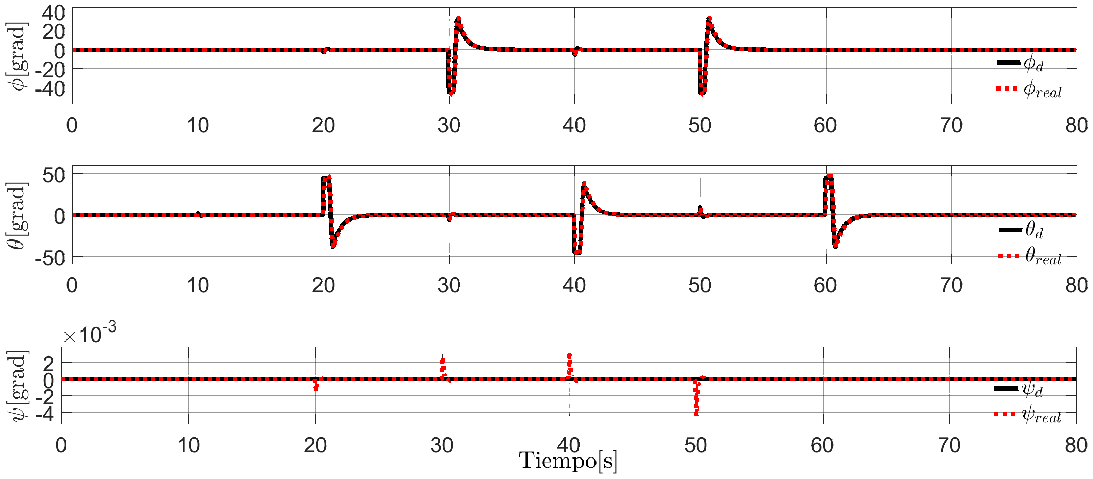

Figura 5.23: Ángulos del sistema al seguir la trayectoria lineal con control PID.

Los errores de posición y orientación mostrados en la Figura 5.24 muestran el comportamiento de las variables para cada cambio que existe en la trayectoria. En x y y se puede observar que cada cambio lleva un aumento del error, que tiende a cero conforme evoluciona la trayectoria. Claramente se puede observar que el error pasa por cero con un sobre impulso para posteriormente tratar de compensarlo sin lograr hacerlo lo suficientemente rápido. En cuanto al error en la orientación, cada cambio en la trayectoria representa un aumento considerable en el error, al grado de poder perder el control del vehículo aéreo. Las señales de control generadas, llegan a tener valores elevados en los instantes de cambio en la trayectoria lo cual representaría un problema al pasar al sistema físico.

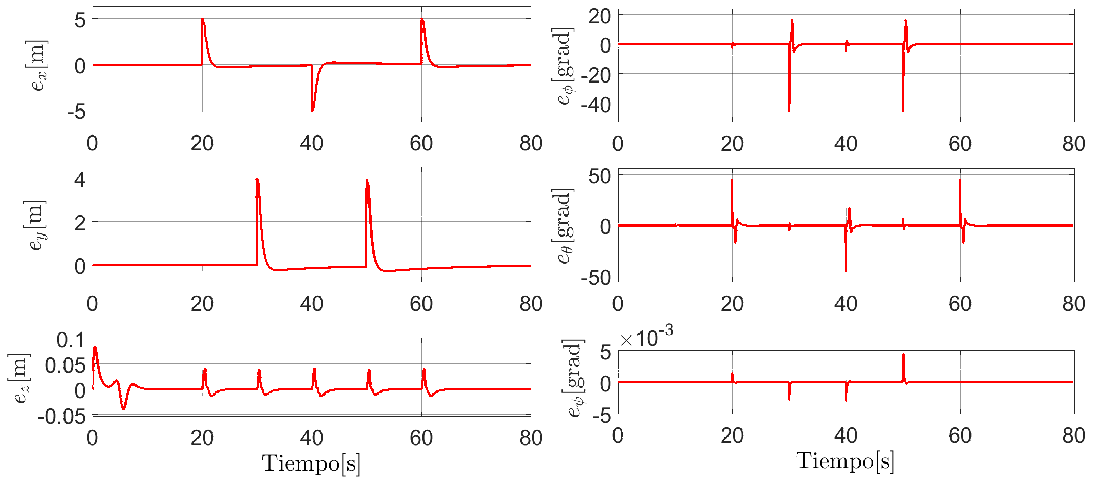

Figura 5.24: Errores de posición y orientación del sistema al seguir la trayectoria lineal con control PID.

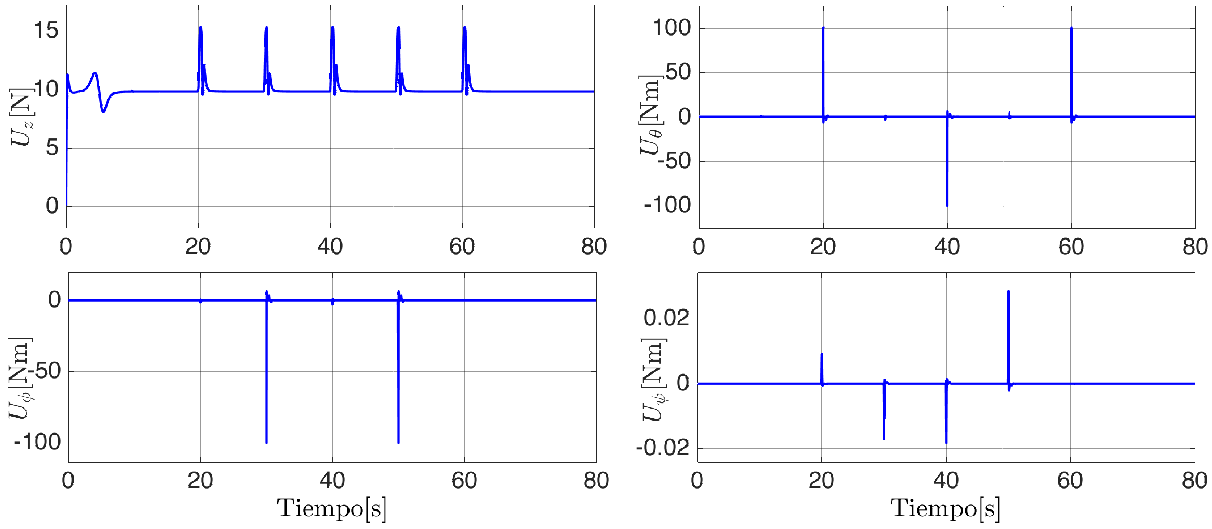

Figura 5.25: Entradas de control del sistema al seguir la trayectoria lineal con control PID.

Control por Backstepping

Las ganancias seleccionadas para el seguimiento de esta trayectoria se muestran en la Tabla 5.7. En la Figura 5.26, se muestran las trayectorias en el plano $X-Y$ y en el espacio $X-Y-Z$. Como se puede observar, a diferencia del control por PID, la trayectoria es alcanzada con un pequeño sobre impulso para las posiciones del eje X, mientras que en las posiciones del eje Y no se presenta ese sobre impulso. La trayectoria recorre 5 metros a lo largo del eje X mientras que en el eje Y se requiere únicamente que recorra 4 metros. Esto se debe a que el algoritmo de visión de este trabajo puede detectar objetos que estén como máximo a 4 metros de distancia. Después de esa distancia es posible encontrar detecciones falsas.

Tabla 5.7: Ganancias seleccionadas para la trayectoria lineal con Backstepping.

Ganancia	Valor	Ganancia	Valor
K_1	100	K_7	2.5
K_2	100	K_8	2.5
K_3	90	K_9	0.9
K_4	100	K_{10}	0.3
K_5	100	K_{11}	2
K_6	100	K_{12}	0.2

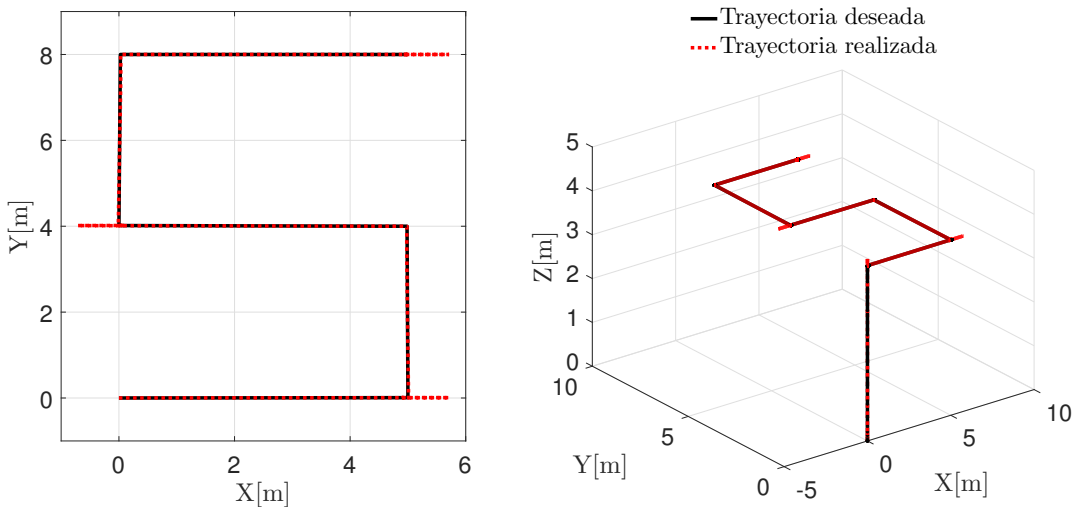

Figura 5.26: Trayectoria realizada en el plano y en el espacio por el robot.

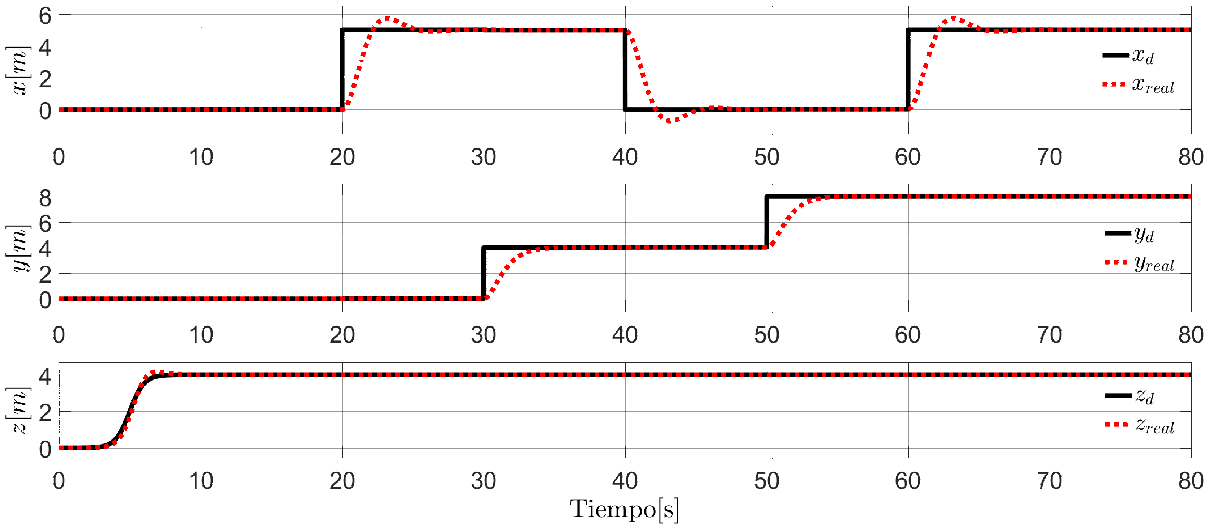

Figura 5.27: Posiciones del sistema al seguir la trayectoria lineal con control Backstepping.

En las Figuras 5.26 y 5.27 las trayectorias deseadas están representadas por una línea negra mientras las trayectorias realizadas por el Quad-Rotor están representadas por una línea punteada roja. Los comportamientos de los ángulos del sistema, mostrados en la Figura 5.28, permiten observar que no existe una saturación en ellos, permitiendo entonces al vehículo aéreo alcanzar las trayectorias deseadas.

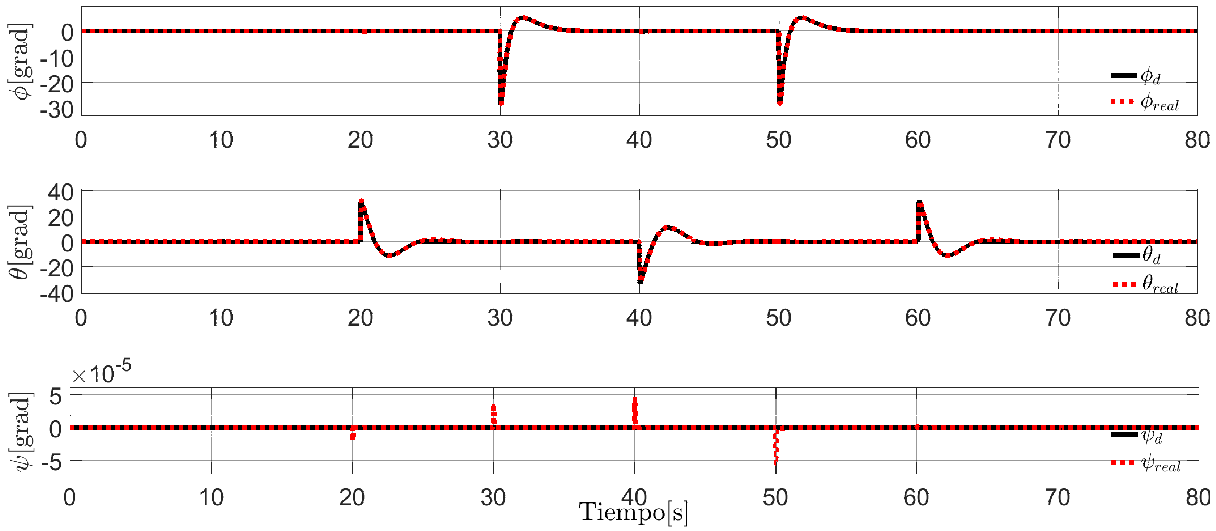

Figura 5.28: Ángulos del sistema al seguir la trayectoria lineal con control Backstepping.

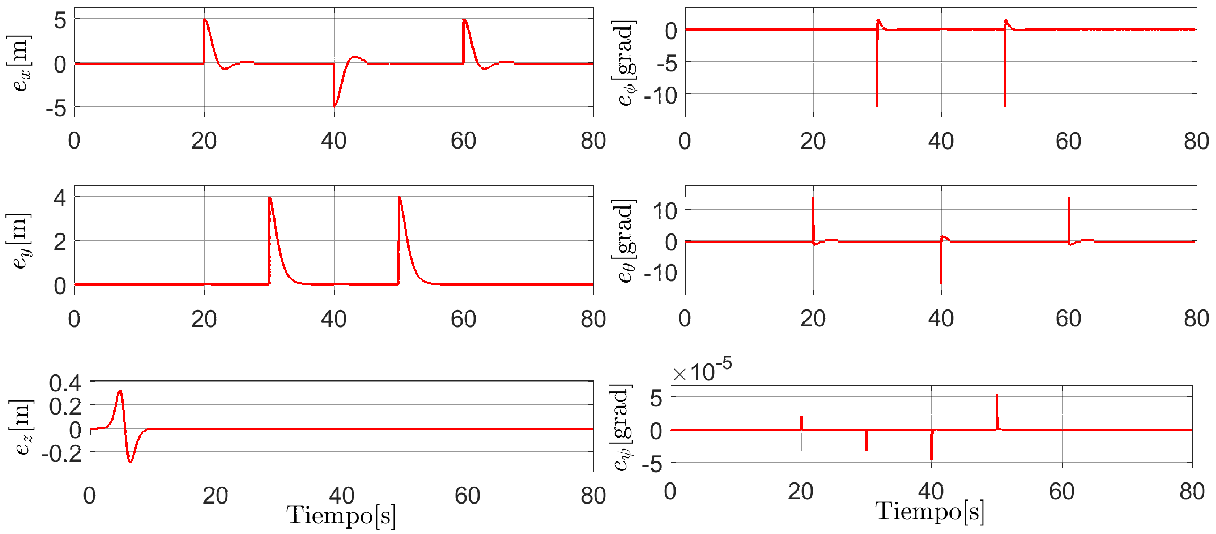

Figura 5.29: Errores de posición y orientación del sistema al seguir la trayectoria lineal con control Backstepping.

Los errores de posición y orientación de la Figura 5.29 tienden a una región cercana al cero debido al funcionamiento del controlador.

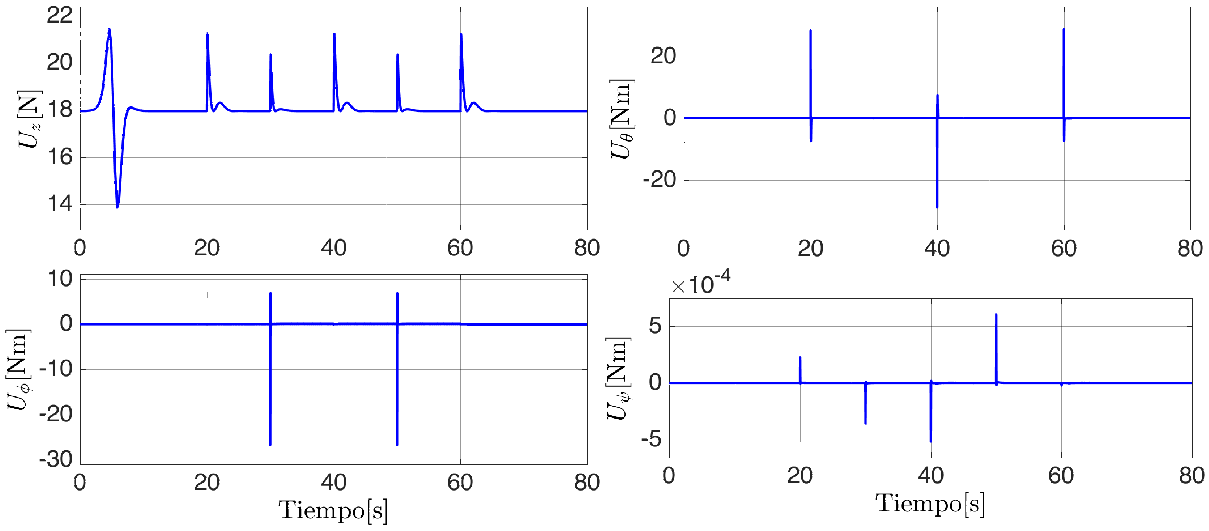

Figura 5.30: Entradas de control del sistema al seguir la trayectoria lineal con control Backstepping.

Las señales generadas de control muestran valores menores a los obtenidos por el control PID, permitiendo al robot realizar la trayectoria de forma más precisa y sin el riesgo de inyectar al sistema señales que puedan causar un daño en los actuadores o la pérdida del vehículo por colisión.

II. Resultados experimentales

En esta sección se presentan las pruebas experimentales del algoritmo de visión diseñado en el Capítulo 4.

II.1. Resultados prácticos del algoritmo propuesto de visión

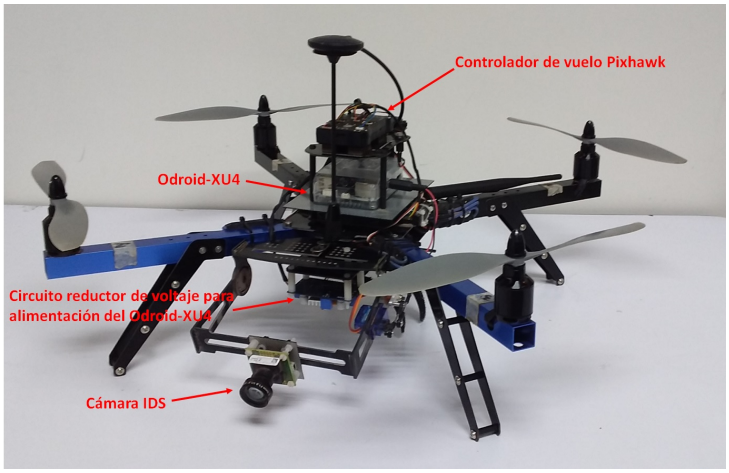

Figura 5.31: Odroid-XU4 y cámara IDS montados en el vehículo.

Para realizar la comprobación del algoritmo de visión, se utilizó una computadora de placa reducida Odroid-XU4, cuyas especificaciones se muestran en el Apéndice A, en conjunto con la cámara de IDS UI-1241LE-C-HQ, cuyas especificaciones se muestran en el Apéndice B, ambos dispositivos electrónicos están montados sobre el Quad-Rotor de 3DRobotics, tal y como se muestra en la Figura 5.31. Para los experimentos, ser realizaron vuelos en las instalaciones del Instituto Tecnológico de la Laguna a 4 metros de altura, cambiando el ángulo de guiñada del robot en intervalos de $\pi/6$ radianes, manteniendo las posiciones x y y constantes con la ayuda de la plataforma QGroundControl.

Una de las mayores ventajas que representa el uso de las librerías de OpenCV es que los módulos que se utilizan realizan las operaciones sobre las imágenes de una forma más eficiente y en menor tiempo, en comparación a si nosotros implementamos dichas funciones. Por ejemplo, en nuestra aplicación es necesario tomar la información obtenida de la cámara y transformarla a un paquete de datos con menor cantidad de elementos para poder realizar su procesamiento, recuérdese que la información obtenida por la cámara utilizada cuenta con una gran cantidad de elementos que no son útiles en nuestro caso.

Figura 5.32: Imagen convertida al espacio de color HSV.

Figura 5.33: Imagen convertida al espacio de color BGR.

Para poder filtrar la información, y mantener aquella que nos interesa, se utilizó el módulo de conversión de espacio de colores visto en VI.1 para pasar la imagen del espacio BGR obtenido de la cámara al espacio HSV (véase la Figura 5.32). Una vez realizada esta conversión, se realiza la misma operación para regresar la imágen de HSV a BGR (véase la Figura 5.33) con la diferencia de que la imagen ahora cuenta solamente con la información necesaria para hacer procesamiento de la misma. Después de filtrar la imagen, se obtienen

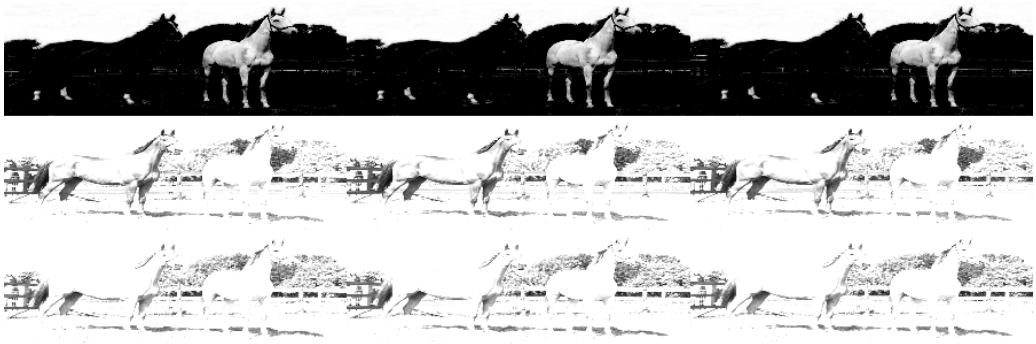

Figura 5.34: Imagen convertida en la "mancha" de 4 dimensiones.

los grupos de píxeles que comparten características con la función *blobFromImage*, vista en VI.3, creando una "mancha" de 4 dimensiones, la cual es nuestra imagen original después de haber realizado las siguientes operaciones: restar el valor medio, normalizar e intercambiar el orden de los canales BGR a RGB y por último cambiar del espacio de colores RGB a 3 canales diferentes de escala de grises, como se muestra en la Figura 5.34.

Posteriormente, este arreglo alimenta a la red neuronal a través del módulo visto en VI.3 para procesar las características del paso anterior y clasificar los objetos en base a la información de la red neuronal pre-entrenada. Dentro de la red, la información de la "mancha" pasa a través de cada capa de la red neuronal para dar como resultado la detección del objeto. Una vez detectado, se dibuja la detección con el módulo visto en VI.2, como se observa en la Figura 5.35.

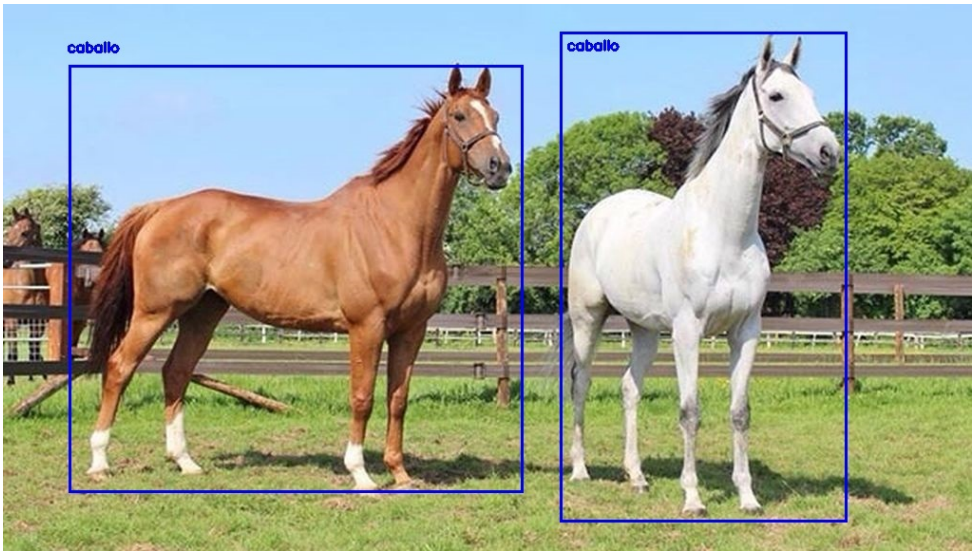

Figura 5.35: Detección del objeto a través de la red neuronal.

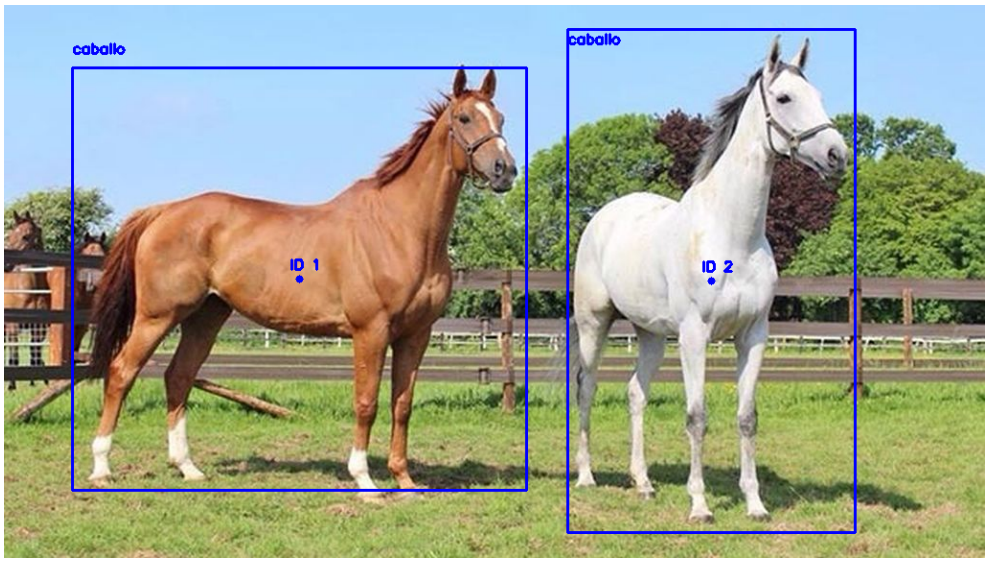

Figura 5.36: Localización del centroide de la detección objeto a través de la red neuronal.

Ahora que el programa ha detectado un objeto y que se ha dibujado la detección, se encuentra el centroide de la detección a través del uso de las coordenadas (x, y) y se dibuja el centroide de la imagen como se muestra en la Figura 5.36.

Para que el objeto detectado no sea contado más de una vez, se utiliza el centroide para seguir al objeto a través de la imagen. Esto se realiza al buscar la relación entre los centroides detectados inicialmente y los que se detectaron después. Aquellos cuya relación haga que la distancia Euclidiana entre ellos sea la menor representarán el punto siguiente del centroide del objeto que se está siguiendo.

Como ya se ha mencionado anteriormente, nuestra red neuronal pre-entrenada es capaz de detectar diferentes clases de objetos, entre ellos, caballos, ovejas, vacas y personas. Debido a que no se cuenta con el acceso a estos animales, se realizaron las pruebas de detección sobre imágenes. Los resultados son mostrados en las Figuras 5.37, 5.38, 5.39.

Se puede observar en la Figura 5.37 la detección de la clase "vaca" y que el algoritmo es capaz de distinguir los objetivos por separado dándole a cada uno su número de identificación, lo cual nos sirve para realizar nuestro conteo. También es importante mencionar que los objetos que se encuentran alejados no son detectados, esto se debe a que la red neuronal no puede observar correctamente sus características por lo que pasan desapercibidos. En la Figura 5.39, se puede observar la detección de la clase "oveja", con una

mayor cantidad de objetivos dentro de rango, los cuales, son detectados sin problema por la red neuronal. Por último, en la Figura 5.38, se observa la detección de la clase "caballo", es importante resaltar que en la imagen, los objetivos se encuentran cubiertos unos por otros de forma que parte de sus características no son visibles, sin embargo, el algoritmo diseñado es capaz de distinguir las suficientes características para poder observar a los cuatro objetivos y detectarlos como la clase que corresponde.

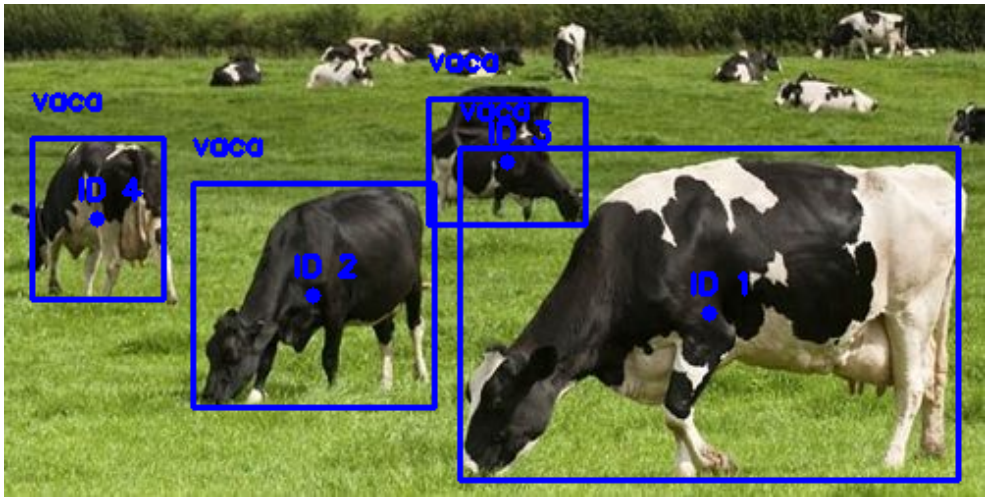

Figura 5.37: Detección de la clase "vacas".

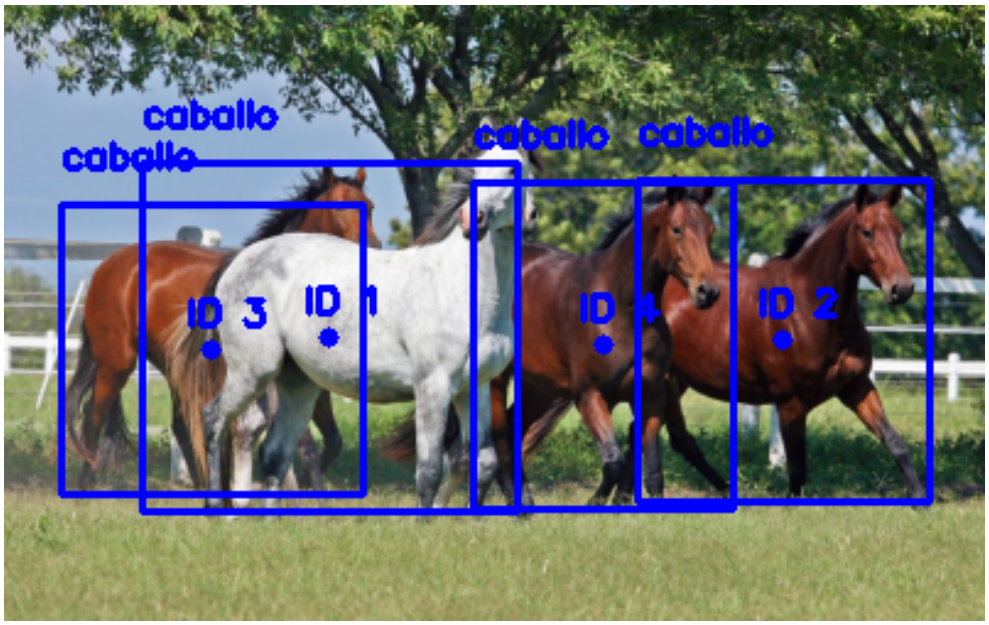

Figura 5.38: Detección de la clase "caballos".

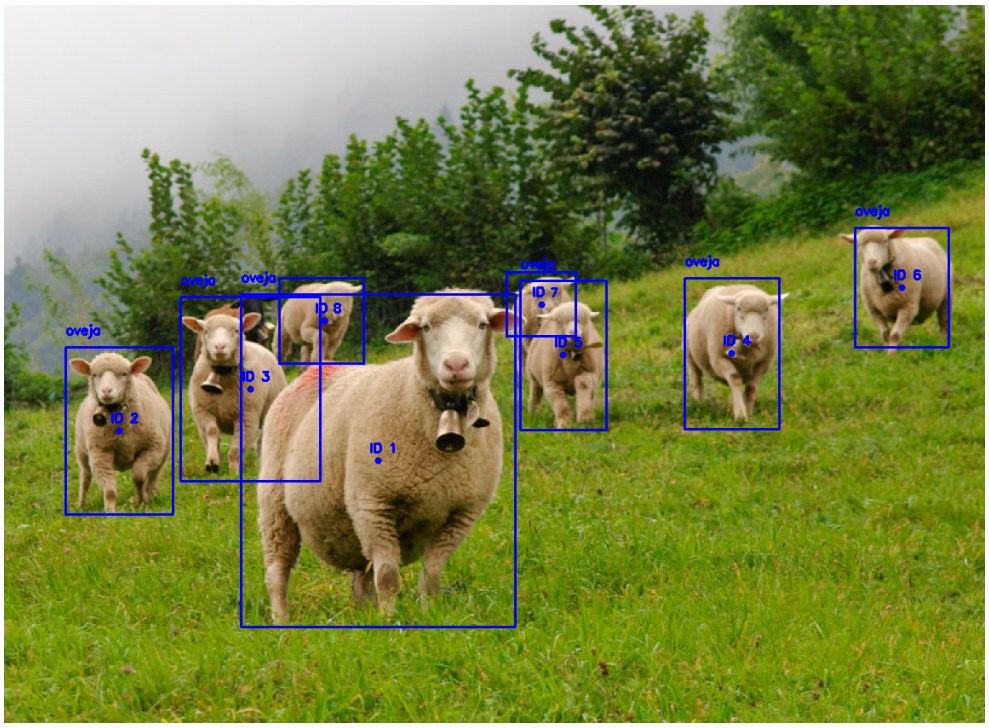

Figura 5.39: Detección de la clase “ovejas”.

Durante las pruebas en el Quad-Rotor, se encontraron cierto obstáculos para la detección de objetos. Por ejemplo, en un inicio se tuvo el problema de que la cámara de IDS no cuenta con un enfoque automático, ésta cuenta únicamente con un lente que se ajusta manualmente. En la Figura 5.40, se muestra que al momento que el vehículo realizaba su trayectoria, debido a las vibraciones, este lente cambiaba el enfoque, evitando que la red neuronal pudiera detectar a los objetivos.

Claramente se puede observar que existen objetivos para su posible detección, sin embargo, la red neuronal no fue capaz de detectarlos debido a que la imagen presentaba ruido. Una vez resuelto el problema del enfoque, se pudo observar que el algoritmo tiene un rango de visualización y detección limitado a 4-5 metros (por motivos del hardware y no del software). En la Figura 5.41, se muestra una de las pruebas realizadas con el Quad-Rotor donde un objetivo dentro del rango de 4 metros es detectado, mientras 4 objetivos que se encuentran a una mayor distancia no son considerados. Para ejemplificar mejor este rango, la Figura 5.42 muestra una toma realizada en un campo donde hay una cantidad numerosa de personas, aproximadamente a unos 15 metros de distancia del vehículo, resultando en una detección nula de objetivos.

Figura 5.40: Enfoque de la cámara desajustado.

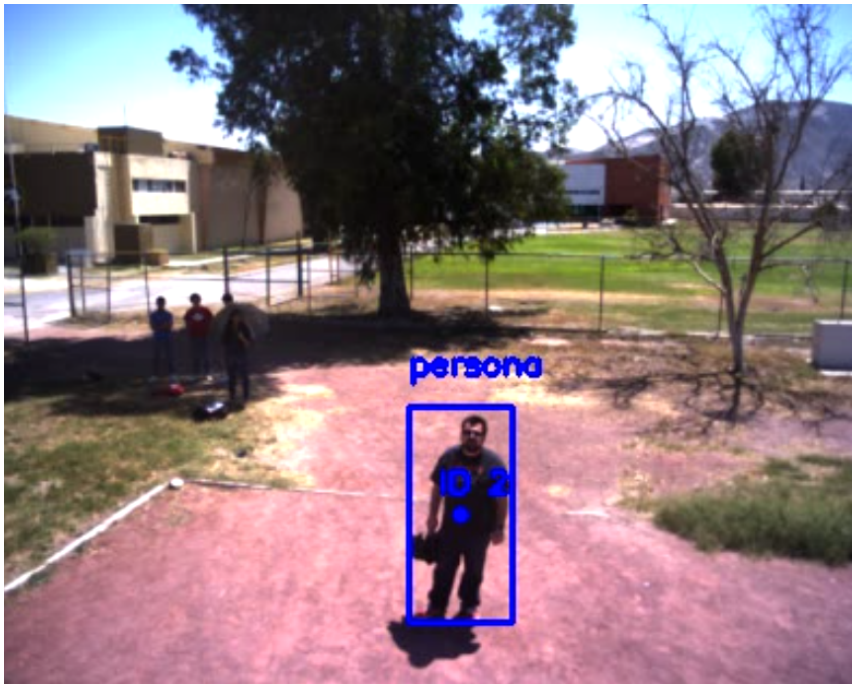

Figura 5.41: Detección de objetivos dentro y fuera de rango.

Figura 5.42: Objetivos en un rango aproximado de 15 metros de distancia.

En la Figura 5.43, se muestra la secuencia de imágenes de una prueba realizada con el vehículo, donde se ajustó a 30 grados en el ángulo de guiñada, manteniendo la posición. En el cuadro 1, el Quad-Rotor acaba de iniciar el vuelo y no existe algún objetivo que detectar. En el siguiente cuadro, existe en el campo visual de la cámara el primer objetivo al cual se le asigna el número de identificación 0. En el cuadro 3, existe un ligero movimiento en la posición del vehículo debido a las ráfagas de viento, lo que causa que el objetivo se mueva de posición en la imagen, lo que permite observar el seguimiento de centroides. En los cuadros 4 y 5, el objetivo sigue con su mismo número identificación gracias al seguimiento de los centroides. El cuadro 6, muestra que un nuevo objetivo se encuentra dentro del área visual de la cámara al cual se le asigna el número de identificación 1 sin dejar de detectar al objetivo anterior. en el cuadro 7, se muestra que el primer objetivo detectado desapareció del campo visual y el recuadro de detección ha desaparecido, mientras que el centroide y el número de identificación siguen presentes en la imagen. Esto se debe a que el algoritmo tiene la capacidad de permitir que un objetivo salga de la imagen por 2 fps y si éste reaparece, tal y como se muestra en el cuadro 8, seguir detectándolo como el mismo objeto que ya había registrado. Esta característica permite al algoritmo ser más preciso para el conteo de los objetivos pues no permite que se cuente dos veces a un mismo objeto.

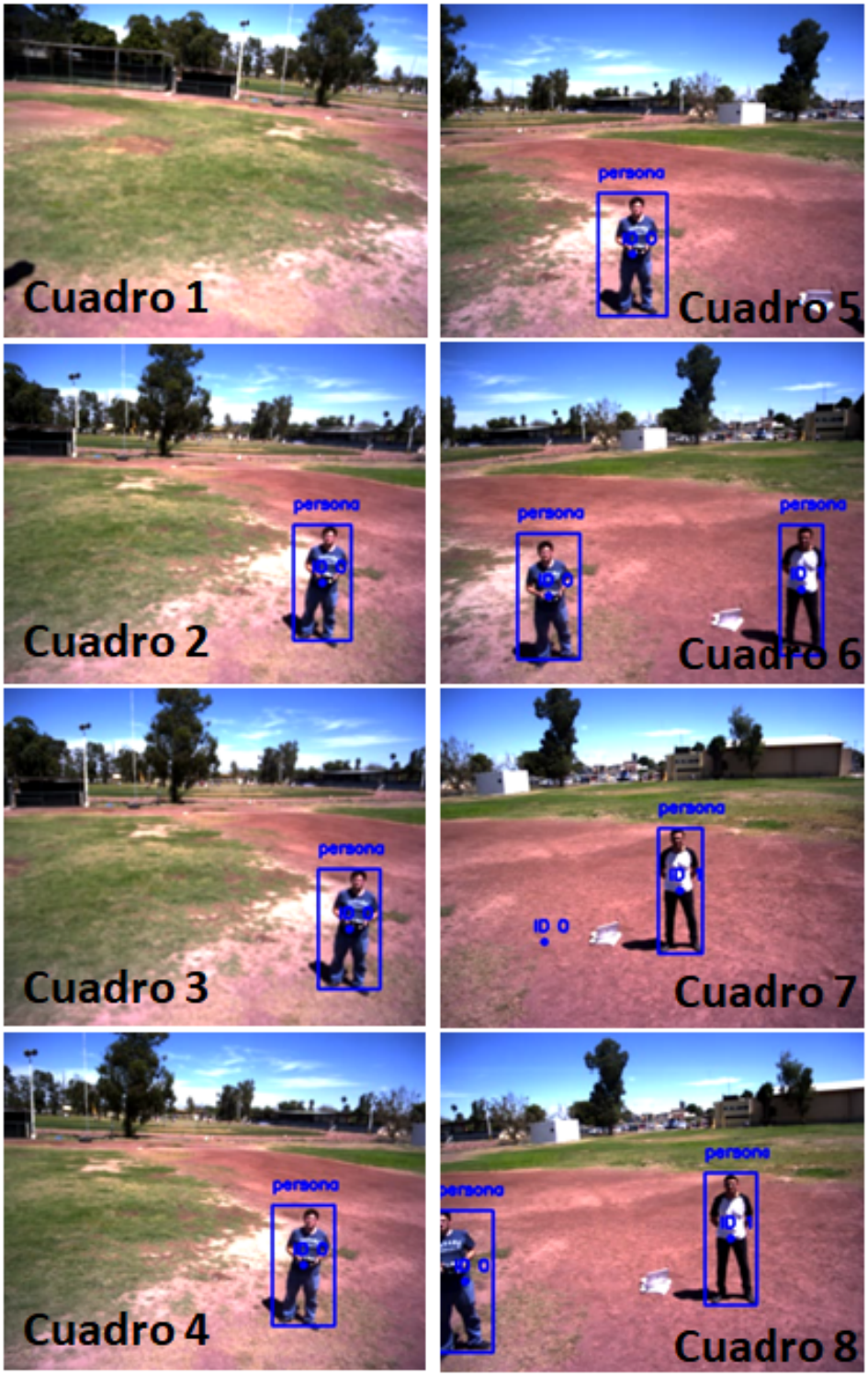

Figura 5.43: Detección de objetivos y seguimiento de centroides.

Capítulo 6

Conclusiones y trabajo futuro

I. Conclusiones

En este trabajo de tesis se abordó el problema de detección de objetos utilizando un vehículo aéreo no tripulado del tipo Quad-Rotor. Para ello, se realizó un estudio del modelado matemático que describe el comportamiento del vehículo aéreo, además de diseñar dos controladores, uno PID y otro a través de Backstepping. Los resultados de simulación se obtuvieron para tres diferentes tipos de trayectorias, mostrando un mejor comportamiento el algoritmo de Backstepping.

Por otro lado, para la detección de objetivos, se diseñó un algoritmo que detectara en tiempo real dichos objetos. Lo anterior se logra utilizando también otro algoritmo propuesto para la detección y seguimiento de centroides, basado en el uso de redes neuronales ya pre-entrenadas del tipo SSD y con aprendizaje automático. Todo esto fue posible gracias al uso de librerías de OpenCV. Para la validación del algoritmo, se modificó la plataforma del Quad-Rotor de 3DRobotics, instalándole una placa Odroid-XU4 en conjunto con una cámara IDS. El algoritmo de visión propuesto proporciona un conteo confiable y eficiente de los objetos detectados, así como la posibilidad de detectar diferentes tipos de objetivos.

Por último, se realizó con éxito un cambio de placa del controlador de vuelo, pasando del Pixhawk versión 1 a la versión 2, la cuál permitirá en un futuro corto, cargar los algoritmos de control, diseñados en Matlab Simulink, para su validación en tiempo real. Lamentablemente, por falta de tiempo, no se logró sintonizar un PID cargado desde Matlab Simulink.

II. Trabajo futuro

Como trabajo futuro, se propone lo siguiente:

- Validación de resultados del algoritmo en campos ganaderos.
- Comprobación de resultados del algoritmo utilizando diferentes modelos de redes neuronales.
- Generación de trayectorias en base a los algoritmos de visión propuestos.
- Realizar la comunicación entre la plataforma Odroid-XU4 y el controlador de vuelo Pixhawk para seguimiento de las trayectorias generadas en base a los algoritmos de visión.
- Validación de los controladores.

Apéndice A

Hardkernel Odroid-XU4

El Odroid XU4 de Hardkernel, mostrado en la Figura A.1, es una computadora de placa reducida (*Single Board Computer*, por sus siglas en inglés) que cuenta con un procesador de ocho núcleos Exynos 5422 big.LITTLE, un GPU (*Graphics Processing Unit*, por sus siglas en inglés) avanzado Mali, ethernet de Gigabit capaz de utilizar los sistemas operativos de Ubuntu, Android, Fedora, ARCHLinux, Debian y OpenELEC. El Odroid-XU4 es un dispositivo de arquitectura ARM, es decir, que su arquitectura está basada RISC (*Reduced Instruction Set Computer*, por sus siglas en inglés). En la Figura A.2 se muestran fotografías del Odroid-XU4 y una explicación de la ubicación de los puertos del mismo.

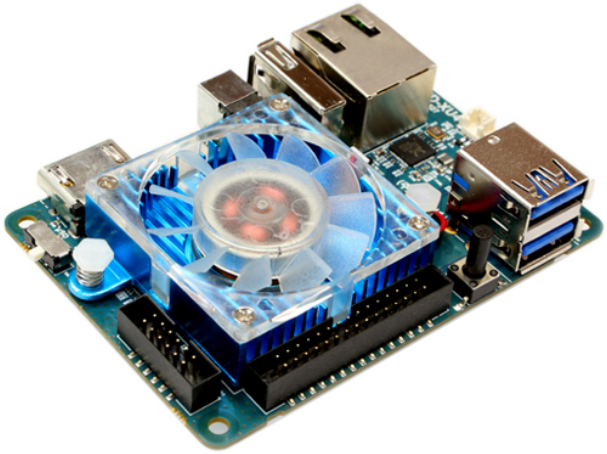

Figura A.1: Odroid-XU4.

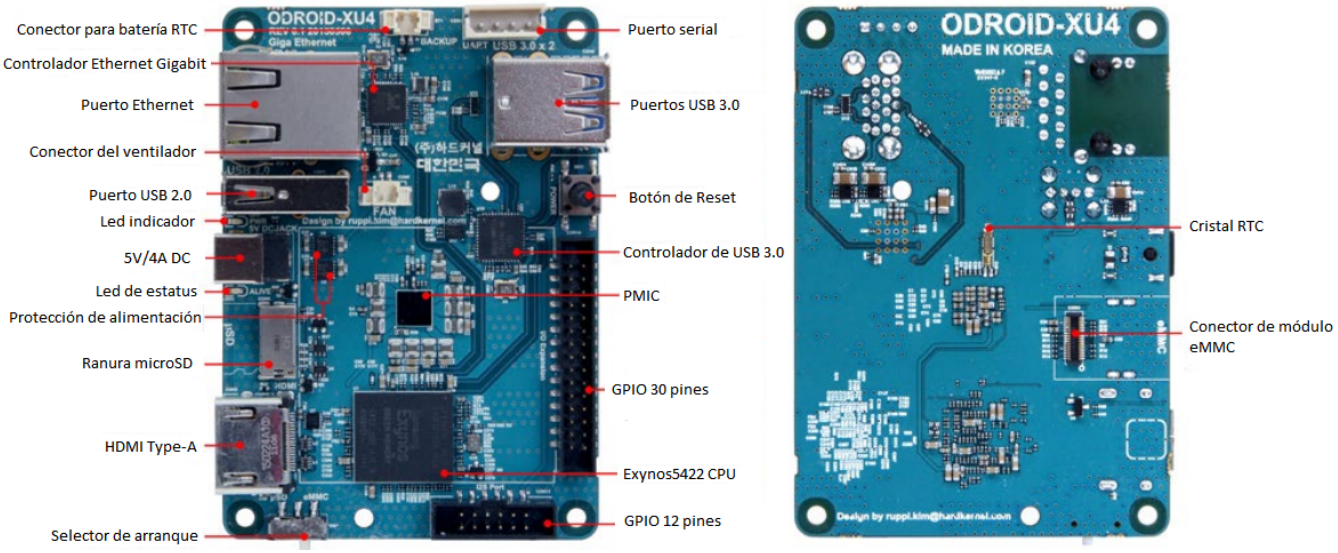

Figura A.2: Imagen descriptiva de la tabla del Odroid-XU.

Cuenta con 1 puerto USB 2.0, 2 puertos USB 3.0, un puerto Ethernet compatible con las velocidades de transferencia de Gigabit, un conector HDMI para pantallas de 720p y 1080p, y un conector de 5V/4A para alimentación. Adicionalmente, cuenta con un puerto de 40 pines GPIO, un conector para batería RTC externa, un puerto USB-UART serial, un conector para módulo eMMC y una ranura para tarjeta micrSD.

El Odroid-XU4 puede utilizar diferentes sistemas operativos de acceso abierto basados en Linux. Dos de los más populares son Ubuntu y Android, disponibles para su descarga en la página web de Hardkernel, aunque también existen otros sistemas como ARCH Linux, FreeBSD, Fedora, CentOS, Open-SUSE, Slackware y Mint. Todos estos cuentan con un kernel personalizado que puede comunicarse con el Odroid. Herdkernel publica kernels específicos para Odroid y los mantiene en un repositorio. Para el caso de este trabajo de tesis, se utilizó el sistema operativo *Ubuntu/Debian*.

Para la instalación del sistema operativo, se utilizó una memora microSD de 16Gb y un adaptador microSD-USB. Utilizando el programa *balenaEtcher* para convertir la memoria en un sistema ejecutable portable. En este caso, se utilizó la imagen del sistema operativo *Ubuntu Mate-Odroid 18.04.1-4.14*. Ubuntu está basado en el sistema operativo *Debian*, que ofrece una gran cantidad de aplicaciónes gratuitas para descargar.

Figura A.3: GUI de Ubuntu Mate 18.04.

Una vez instalada la imagen del sistema operativo, el Odroid-XU4 mostrará la interfaz de usuario, ver Figura A.3. Para acceder a la consola de mandos y poder realizar la instalación de aplicaciones y librerías se deberá utilizar *Ctrl+t*, o bien a través del menú principal.

Los comandos básicos para la consola de Ubuntu son:

- *mkdir*: Crea un nuevo directorio.
- *cd*: Navega a través de los directorios.
- *ls*: Lista los archivos de un determinado directorio o del directorio actual.
- *mv*: Mueve archivos o subdirectorios de un directorio a otro.
- *su* o *sudo*: Con este comando accedemos al sistema como root.
- *apt-get install*: Instala nuevos paquetes.
- *apt-get upgrade*: Actualiza la lista de paquetes disponibles para instalar.
- *apt-get update*: Instala las nuevas versiones de los diferentes paquetes disponibles.

Apéndice B

Cámara IDS UI-1241LE-C-HQ

La UI-1241LE, mostrada en B.1, es una cámara de calidad industrial muy potente gracias a su sensor CMOS e2v de 1.31 megapíxeles. Este sensor es uno de los más sensibles disponible en IDS y se ofrece en las tres versiones: monocromática, color y en versión infrarroja o NIR (por el inglés, *Near Infra-Red*). Además de su elevada sensibilidad lumínica en calidad Dispositivo de Carga Acoplada o CCD (*Charged Coupled Device*, por sus siglas en inglés), la cámara ofrece una gran variedad de funciones adicionales, por ejemplo, las dos variantes de Global y Rolling Shutter del sensor, que se pueden seleccionar con el equipo en marcha, y que aseguran la máxima flexibilidad ante posibles cambios de los requisitos y condiciones ambientales. Además se dispone de un máximo de cuatro AOI (áreas de interés), que permiten verificar simultáneamente varias características, o bien captar las AOI en una serie de exposiciones con distintos parámetros.

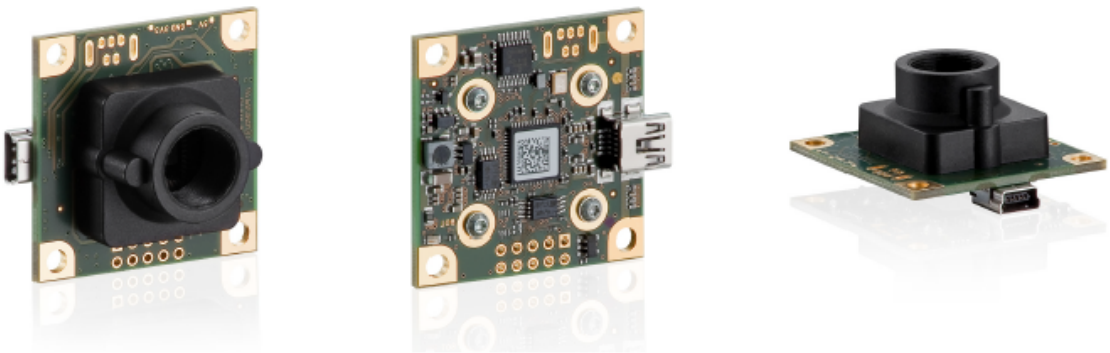

Figura B.1: Cámara IDS UI-1241LE-C-HQ.

El sensor lineal es del tipo CMOS y es de $1/1,8$", con una superficie óptica de 6.784 [mm] x 5.427 [mm], cuenta con Global Shutter para captar objetos en movimiento, y Rolling Shutter para obtener imágenes de mayor contraste y poco ruido. Además, cuenta con una resolución de 1.31 [Mpx] o 1280 x 1024, y el tamaño del pixel es de 5.3 μ_m. La relación de aspecto es de 5:4. El rango de frecuencia de los shutters es de entre 7 [MHz] y 35 [MHz]. El tiempo de exposición tiene un mínimo de 0.009 [ms] y un máximo de 2000 [ms]. Su consumo de potencia es de 0.3 a 0.7 [W]. Sus dimensiones son 36 [mm] x 36 [mm] x 20.2 [mm] y su peso es de 16 [g].

Es importante mencionar que las cámaras de IDS manejan sus propios comandos de programación. A continuación, se mencionan algunos de mayor relevancia.

- $is\_InitCamera$: Se utiliza para inicializar la cámara.
- $is\_SetCameraID$: Asigna una nueva identificación a la cámara.
- $is\_AllocImageMem$: Asigna la memoria de la imagen.
- is_SetAOI: Define el tamaño y posición del área de interés de la cámara.
- $is\_GetSensorInfo$: Adquiere la información del sensor de la cámara.
- $is\_SetColorMode$: Selecciona el modo de color (Para el caso de la cámara utilizada se puede seleccionar modo monocromático y modo color).
- $is\_SetFrameRate$: Define la velocidad de los fotogramas (FPS).
- $is\_SetDisplayMode$: Selecciona el modo de visualización de la imagen.

Para conocer más de estos comandos, refiérase a [39].

Apéndice C

Controlador de vuelo mRo Pixhawk

Los controladores de vuelo Pixhawk, son un proyecto independiente de hardware abierto que tiene como objetivo proporcionar un diseño estándar de hardware para pilotos automáticos. Un piloto automático típico consiste de un controlador de vuelo conectado a un sistema de alimentación, un GPS, una brújula externa (opcional), un sistema de control de radio frecuencia y/o un sistema de radio telemetría. Algunas de las características clave de los Pixhawk son:

- Control de diversos marcos de vehículos, incluyendo: aeronaves (multicópteros, aeronaves de ala fija y VTOL´s), vehículos terrestres y vehículos acuáticos.
- Modos de vuelo flexibles con medidas de seguridad.

El Pixhawk de *mRo*, como el mostrado en la Figura C.1, es un hardware compatible con la versión original de *Pixhawk 1*, la principal diferencia es que el segundo se basa en el *Pixhawk-project FMUv2* mientras que el primero se basa en el *Pixhawk-project FMUv3*, lo cual corrige un error que limitaba la memoria flash del *Pixhawk 1* a 1 MB.

Cuenta con un microprocesador de 32-bits, con un núcleo STM32F427 Cortex M4 con unidad de coma flotante (*Floating Point Unit*, por sus siglas en inglés), memoria flash de 2 MB y memoria RAM de 256 KB. Además, incluye un giroscopio ST Micro L3GD20 de 3 ejes y 16 bits, un acelerómetro/magnetómetro ST Micro LSM303D de 3 ejes y 14 bits, un acelerómetro/giroscopio Invensense MPU 6000 de 3 ejes y un barómetro MEAS MS5611. De igual manera, cuenta con 5 puertos seriales UART, uno de ellos con capacidad de alta potencia, dos puertos con protocolo de comunicaciones CAN (*Controller Area Network*,

por sus siglas en inglés), puerto I2C, ente otras características. Tiene un peso de 38 [g] y sus dimensiones son 50 [mm] x 15.5 [mm].

Para poder utilizar este controlador de vuelo, es importante que se defina bien la orientación del mismo sobre el vehículo, esto debido a que los controles de vuelo deberán utilizar el vector de movimiento. En la figura C.2 se muestra un ejemplo de esta orientación.

De igual manera, para montar el controlador de vuelo, debe considerarse que las vibraciones pueden llegar a afectar la medición, debido a que los acelerómetros son sensibles a ellas. Es por ello que debe existir un medio para amortiguar estas vibraciones.

En la Figura C.3, se muestra la conexión básica de un controlador de vuelo Pixhawk. Estas conexiones son las mínimas necesarias para que el funcionamiento del controlador sea el correcto. Sin embargo es posible realizar conexiones con otros sensores, o bien con dispositivos como Raspberry u Odroid a través de los puerto de comunicación serial.

Figura C.1: Piloto automático Pixhawk.

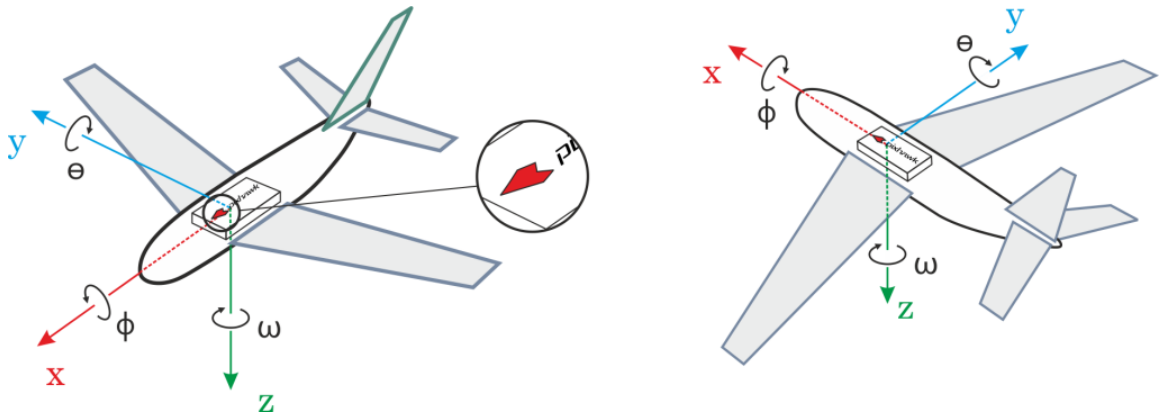

Figura C.2: Correcta colocación del controlador de vuelo.

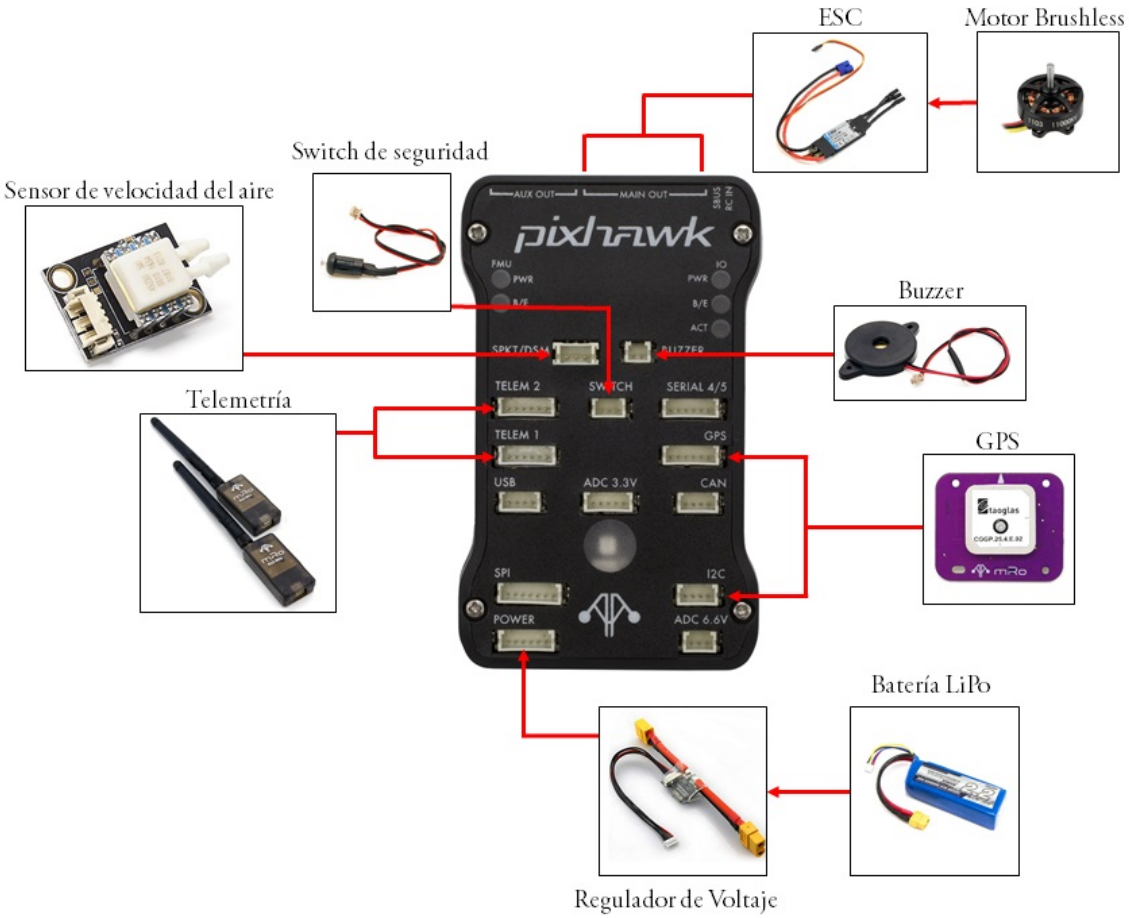

Figura C.3: Conexión básica de un controlador de vuelo Pixhawk.

Bibliografía

[1] S. Ge and F. Lewis, *"Autonomous Mobile Robots: Sensing, Control, Decision Making and Applications"*. Taylos and Francis Group LLC, 2006.

[2] L. G. Carrillo, A. D. López, R. Lozano, and C. Pégard, *"Quad Rotorcraft Vision-Based Hovering and Navigation"*. Springer-Verlag London, 2013.

[3] J. de Jesus Rubio, J. Humberto Perez Cruz, A. J. Salinas, and Z. Zamudio, "Comparison of two quadrotor dynamic models," *IEEE Latin America Transactions*, vol. 12, pp. 531–537, June 2014.

[4] K. Hu, X. Sun, and Y. Wu, "Modeling and control design for quadrotors: A controlled hamiltonian systems approach," *IEEE Transactions on Vehicular Technology*, vol. 67, pp. 11365–11376, Dec 2018.

[5] H. Liu, X. Wang, and Y. Zhong, "Quaternion-based robust attitude control for uncertain robotic quadrotors," *IEEE Transactions on Industrial Informatics*, vol. 11, pp. 406–415, April 2015.

[6] M. Fanni and A. Khalifa, "A new 6-dof quadrotor manipulation system: Design, kinematics, dynamics, and control," *IEEE/ASME Transactions on Mechatronics*, vol. 22, pp. 1315–1326, June 2017.

[7] H. Ahn and Y. Choi, "Nonlinear control of quadrotor for point tracking: Actual implementation and experimental tests," *IEEE/ASME Transactions on Mechatronics*, vol. 20, pp. 1179–1192, June 2015.

[8] N. Fethalla, J. Ghommam, H. Michalska, and M. Saad, "Robust observer-based dynamic sliding mode controller for a quadrotor uav," *IEEE Access*, vol. 6, pp. 45846–45859, 2018.

[9] D. F. Astudillo, L. I. Minchala, P. Ortiz, M. J. Reinoso, and D. Verdugo, "Trajectory tracking of a quadrotor using sliding mode control," *IEEE Latin America Transactions*, vol. 14, pp. 2157–2166, May 2016.

[10] P. Castillo, C. Izaguirre-Espinosa, A. J. Muñoz-Vázquez, V. Parra-Vega, and A. Sánchez-Orta, "Attitude control of quadrotors based on fractional sliding modes: theory and experiments," *IET Control Theory Applications*, vol. 10, no. 7, pp. 825–832, 2016.

[11] D. Cabecinhas, R. Cunha, L. Marconi, R. Naldi, and C. Silvestre, "Robust landing and sliding maneuver hybrid controller for a quadrotor vehicle," *IEEE Transactions on Control Systems Technology*, vol. 24, pp. 400–412, March 2016.

[12] S. K. Saksena, S. Mannar, S. Omkar, and M. Thummalapeta, "Vision-based control for aerial obstacle avoidance in forest environments," *IFAC-PapersOnLine*, vol. 51, no. 1, pp. 480 – 485, 2018. 5th IFAC Conference on Advances in Control and Optimization of Dynamical Systems ACODS 2018.

[13] H. Hu, Y. Liu, Q. Wang, and Y. Zhuang, "A novel trail detection and scene understanding framework for a quadrotor uav with monocular vision," *IEEE Sensors Journal*, vol. 17, pp. 6778–6787, Oct 2017.

[14] B. Xian, X. Zhang, B. Zhao, and Y. Zhang, "Autonomous flight control of a nano quadrotor helicopter in a gps-denied environment using on-board vision," *IEEE Transactions on Industrial Electronics*, vol. 62, pp. 6392–6403, Oct 2015.

[15] W. Du, Y. Tang, C. Wu, and B. Zhao, "Vision-based tracking control of quadrotor with backstepping sliding mode control," *IEEE Access*, vol. 6, pp. 72439–72448, 2018.

[16] F. Baret, A. Comar, M. Hemerlé, X. Jin, and S. Liu, "Estimates of plant density of wheat crops at emergence from very low altitude uav imagery," *Remote Sensing of Environment*, vol. 198, pp. 105 – 114, 2017.

[17] J. R. Alves, B. J. Mederos, W. E. Santiago, and I. H. Yano, "Identification of weeds in sugarcane fields through images taken by uav and random forest classifier," *IFAC-PapersOnLine*, vol. 49, no. 16, pp. 415 – 420, 2016. 5th IFAC Conference on Sensing, Control and Automation Technologies for Agriculture AGRICONTROL 2016.

[18] O. Hassaan, A. Kamal, M. F. Khan, Nasir, and H. Roth, "Precision forestry: Trees counting in urban areas using visible imagery based on an unmanned aerial vehicle," *IFAC-PapersOnLine*, vol. 49, no. 16, pp. 16 – 21, 2016. 5th IFAC Conference on Sensing, Control and Automation Technologies for Agriculture AGRICONTROL 2016.

[19] I. Nourbakhsh and R. Siegwart, *"Autonomous Mobile Robots"*. The MIT Press Cambridge, Massachusetts, 2004.

[20] F.Kendoul, D. Nakazawa, K. Nonami, S.Suzuki, and W.Wang, *"Autonomous Flying*

Robots: Unmanned Aerial Vehicles and Micro Aerial Vehicles". Springer Tokyo Dordrecht Heidelberg London New York, 2010.

[21] A. J. Keane, J. Scanlan, and A. Sóbester, *"Small Unmmanned Fixed-wing Aircraft Design: A Practical Approach"*. WILEY, 2017.

[22] K. Ogata, *"Ingeniería de Control Moderna"*. PEARSON EDUCACIÓN, S.A., 2010.

[23] M. Johnson and M. Moradi, *"PID Control: New Identification and Design Methods"*. Springer-Verlag London Limited, 2005.

[24] M. Krstic and A. Smyshlyaev, *"Boundary Control of PDE´s: A course on Backstepping Design"*. Society for Industrial and Applied Mathematics Philadelphia, 2008.

[25] H. Khalil, *"Nonlinear Systems"*. Prentice Hall, Inc., 2002.

[26] H. Lu and Y. Li, *"Artificial Intelligence and Computer Vision"*. Springer International Publishing Switzerland, 2013.

[27] T. Breckon and C. Solomon, *"Fundamentals of Digital Image Processing"*. WILEY-BLACKWELL A John Wiley and Sons, Ltd, 2011.

[28] D. Baggio, S. Enami, D. Escrivá, K.Ievgen, J. Saragih, and R. Shilkrot, *"Mastering OpenCV 3"*. Packt Publishing, 2017.

[29] K. Dawson-Howe, *"A Practical Introduction to Computer Vision With OpenCV"*. John Wiley and Sons Ltd, 2014.

[30] M. Fairchild, *"Color Appearance Models"*. John Wiley and Sons, Ltd, 2017.

[31] A. Rosebrock, *"Deep Learning for Computer Vision with Python"*. PyImageSearch, 2017.

[32] S. Gowrishankar and A. Veena, *"Introduction to Python Programming"*. CRC Press Taylor and Francis Group, 2019.

[33] D. J. Pine, *"Introduction to Python for Science and Engineering"*. CRC Press Taylor and Francis Group, 2019.

[34] A. A. Najm and I. K. Ibraheem, "Nonlinear pid controller design for a 6-dof uav quadrotor system," *Engineering Science and Technology, an International Journal*, 2019.

[35] Z. Zuo, "Trajectory tracking control design with command-filtered compensation for a quadrotor," *IET Control Theory Applications*, vol. 4, pp. 2343–2355, November 2010.

[36] P. Varaiya and R. Liu, "Bounded-input bounded-output stability of nonlinear time-varying differential systems," *SIAM Journal on Control*, vol. 4, no. 4, pp. 698–704, 1966.

[37] M. Krsti, I. Kanellakopoulos, and V. Petar, *Nonlinear and adaptive control design*. Wiley New York, 1995.

[38] Pixhawk, "mro pixhawk flight controller (pixhawk 1)." https://docs.px4.io/en/flight_controller/mro_pixhawk.html#mro-pixhawk--flight-controller-pixhawk-1. [Online; accessed 16-Junio-2019].

[39] IDS Imaging Development Systems GmbH. Alle Rechte vorbehalten, *uEye Software Development Kit (SDK)*.