Neiva, 15 de febrero de 2021

Señores

CENTRO DE INFORMACIÓN Y DOCUMENTACIÓN

UNIVERSIDAD SURCOLOMBIANA

Ciudad

El (Los) suscrito(s):

David Ernesto Reina Munar_____, con C.C. No. 1026591813_____,

Karen Yulieth Cerón Manrique_____, con C.C. No. 1075314515_____,

_____, con C.C. No. _____,

_____, con C.C. No. _____,

Autor(es) de la tesis y/o trabajo de grado o _____

titulado Sistema de detección y evasión de obstáculos en un ambiente controlado, realizado con AR.Drone 2.0

mediante ROS y Python_____

_____

presentado y aprobado en el año __2021___ como requisito para optar al título de

Ingeniero Electrónico_____;

Autorizo (amos) al CENTRO DE INFORMACIÓN Y DOCUMENTACIÓN de la Universidad Surcolombiana para que, con fines académicos, muestre al país y el exterior la producción intelectual de la Universidad Surcolombiana, a través de la visibilidad de su contenido de la siguiente manera:

- Los usuarios puedan consultar el contenido de este trabajo de grado en los sitios web que administra la Universidad, en bases de datos, repositorio digital, catálogos y en otros sitios web, redes y sistemas de información nacionales e internacionales “open access” y en las redes de información con las cuales tenga convenio la Institución.

- Permita la consulta, la reproducción y préstamo a los usuarios interesados en el contenido de este trabajo, para todos los usos que tengan finalidad académica, ya sea en formato Cd-Rom o digital desde internet, intranet, etc., y en general para cualquier formato conocido o por conocer, dentro de los términos establecidos en la Ley 23 de 1982, Ley 44 de 1993, Decisión Andina 351 de 1993, Decreto 460 de 1995 y demás normas generales sobre la materia.

- Continúo conservando los correspondientes derechos sin modificación o restricción alguna; puesto que, de acuerdo con la legislación colombiana aplicable, el presente es un acuerdo jurídico que en ningún caso conlleva la enajenación del derecho de autor y sus conexos.

De conformidad con lo establecido en el artículo 30 de la Ley 23 de 1982 y el artículo 11 de la Decisión Andina 351 de 1993, “Los derechos morales sobre el trabajo son propiedad de los autores” , los cuales son irrenunciables, imprescriptibles, inembargables e inalienables.

EL AUTOR/ESTUDIANTE:

Firma: Karen Cedón M.

EL AUTOR/ESTUDIANTE:

Firma: David E Reina Munar

EL AUTOR/ESTUDIANTE:

Firma: _____

EL AUTOR/ESTUDIANTE:

Firma: _____

TÍTULO COMPLETO DEL TRABAJO: Sistema de detección y evasión de obstáculos en un ambiente controlado, realizado con AR.Drone 2.0 mediante ROS y Python

AUTOR O AUTORES:

Primero y Segundo Apellido	Primero y Segundo Nombre
Cerón Manrique	Karen Yulieth
Reina Munar	David Ernesto

DIRECTOR Y CODIRECTOR TESIS:

Primero y Segundo Apellido	Primero y Segundo Nombre
Robayo Betancourt	Faiber Ignacio

ASESOR (ES):

Primero y Segundo Apellido	Primero y Segundo Nombre

PARA OPTAR AL TÍTULO DE: Ingeniero Electrónico

FACULTAD: Ingeniería

PROGRAMA O POSGRADO: Electrónica

CIUDAD: Neiva **AÑO DE PRESENTACIÓN:** 2021 **NÚMERO DE PÁGINAS:** 60

TIPO DE ILUSTRACIONES (Marcar con una **X**):

Diagramas___ Fotografías___ Grabaciones en discos___ Ilustraciones en general_**X**_ Grabados___ Láminas___ Litografías___ Mapas___ Música impresa___ Planos___ Retratos___ Sin ilustraciones___ Tablas o Cuadros_**X**_

Vigilada Mineducación
La versión vigente y controlada de este documento, solo podrá ser consultada a través del sitio web Institucional www.usco.edu.co, link Sistema Gestión de Calidad. La copia o impresión diferente a la publicada, será considerada como documento no controlado y su uso indebido no es de responsabilidad de la Universidad Surcolombiana.

SOFTWARE requerido y/o especializado para la lectura del documento: Word

MATERIAL ANEXO:

PREMIO O DISTINCIÓN *(En caso de ser LAUREADAS o Meritoria)*:

PALABRAS CLAVES EN ESPAÑOL E INGLÉS:

	Español	Inglés		Español	Inglés
1.	Visión por computador	Computer vision	**6.**	UAV	UAV
2.	Algoritmo	Algorithm	**7.**	ROS	ROS
3.	Python	Python	**8.**	OpenCV	OpenCV
4.	Dron	Drone	**9.**	AR.Drone 2.0	AR.Drone 2.0
5.	VANT	VANT			

RESUMEN DEL CONTENIDO: (Máximo 250 palabras)

En este trabajo se desarrolla e implementa un algoritmo escrito en lenguaje de programación Python, con el fin de realizar la detección y evasión de obstáculos de color rojo. Este algoritmo se basado en el uso de OpenCV para realizar la detección de obstáculos teniendo en cuenta la característica del color del objeto, así como también el uso de Robot System Operating (ROS) como entorno de desarrollo de programación para realizar el intercambio de datos entre el dron y la estación en tierra, y el uso de Python como lenguaje de programación en estas dos herramientas.

Una finalidad importante de este trabajo es contribuir con el desarrollo tecnológico de la región mediante el uso de drones, así como también sentar las bases para futuras investigaciones y proyectos no sólo con drones sino con cualquier robot que pueda ser programado con ROS, ya que la lógica del algoritmo desarrollado funciona de manera similar para todos, brindando así una amplia utilidad y flexibilidad a las personas que quieran adentrarse en esta área.

Vigilada Mineducación
La versión vigente y controlada de este documento, solo podrá ser consultada a través del sitio web Institucional www.usco.edu.co, link Sistema Gestión de Calidad. La copia o impresión diferente a la publicada, será considerada como documento no controlado y su uso indebido no es de responsabilidad de la Universidad Surcolombiana.

ABSTRACT: (Máximo 250 palabras)

In this work a python algorithm was developed, in order to carry out the detection and avoidance of red obstacles. This algorithm is based on the use of OpenCV to perform obstacle detection taking into account the color characteristic of the object, as well as the use of Robot System Operating (ROS) as programming development environment in order to data exchange between the drone and the ground station, and the use of Python as programming language in these two tools.

An important purpose of this work is to contribute to the technological development of the region by using drones, as well as to blaze the way for future research and projects not only with drones but with any robot that can be programmed with ROS because logic of the developed algorithm works in a similar way for every robot, thus providing a wide utility and flexibility to people who want go deeper into this area.

APROBACION DE LA TESIS

Nombre Presidente Jurado:

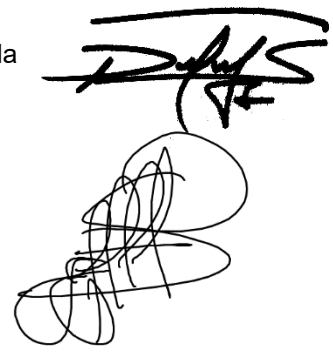
Firma:

Nombre Jurado: Diego Fernando Sendoya Losada

Firma:

Nombre Jurado: José de Jesús Salgado Patrón

Firma:



SISTEMA DE DETECCIÓN Y EVASIÓN DE OBSTÁCULOS EN UN AMBIENTE CONTROLADO, REALIZADO CON AR.DRONE 2.0 MEDIANTE ROS Y PYTHON.

DAVID ERNESTO REINA MUNAR
KAREN YULIETH CERÓN MANRIQUE

UNIVERSIDAD SURCOLOMBIANA
FACULTAD DE INGENIERÍA
PROGRAMA DE INGENÍERIA DE ELECTRÓNICA
NEIVA HUILA
2021

SISTEMA DE DETECCIÓN Y EVASIÓN DE OBSTÁCULOS EN UN AMBIENTE CONTROLADO, REALIZADO CON AR.DRONE 2.0 MEDIANTE ROS Y PYTHON.

DAVID ERNESTO REINA MUNAR
KAREN YULIETH CERÓN MANRIQUE

Trabajo de grado presentado como requisito para optar al Título de Ingeniero Electrónico.

Director
ING. FAIBER ROBAYO BETANCOURT, MSc.

UNIVERSIDAD SURCOLOMBIANA
FACULTAD DE INGENIERÍA
PROGRAMA DE INGENÍERIA DE ELECTRÓNICA
NEIVA HUILA
2021

Primero que nada, quiero dar gracias a Dios por permitirme lograr esta meta, guiar mi camino siempre y mostrarme que, aunque a veces parezca difícil el camino, con Él de la mano se aligeran las cargas.

En segundo lugar, quiero agradecer a mis padres quienes han sido fundamentales en este proceso por brindarme su amor y apoyo incondicional, además de inculcarme los valores que me hacen la persona que soy hoy en día así como también enseñarme que sin Dios nada es posible.

Quiero agradecer también a mis hermanos porque a pesar de la distancia siempre me han apoyado y ayudado en mi formación personal, así como también porque siempre han sido un ejemplo a seguir.

David

En primer lugar, doy gracias a Dios y a la Virgen por estar a mi lado en todo momento ofreciéndome lo mejor y presentándome el camino correcto durante mi carrera sin importar las circunstancias.

A mis padres por todos sus sacrificios y esfuerzos, por darme una carrera para mi futuro, por todo el amor y comprensión a lo largo de este largo camino, siempre tuvieron una palabra de aliento para seguir adelante.

A mi hermano por su cariño y apoyo durante todo este proceso, quien siempre con sus palabras me hace sentir orgullosa de lo que soy.

Karen

AGRADECIMIENTOS

Agradecemos a los docentes del programa de Ingeniería Electrónica de la Universidad Surcolombiana, por compartir sus conocimientos durante la preparación de nuestra profesión, en especial al ingeniero Faiber Robayo director de nuestro proyecto de grado, quien, con su conocimiento, enseñanza y tiempo dedicado a nosotros, contribuyó al desarrollo de este proyecto.

A nuestros compañeros Luisa, Marylin, Yimmi, Camilo, Sebastián, Maicol y David que caminaron con nosotros durante todo este proceso, compartiendo no sólo alegrías sino tristezas, dándonos su apoyo incondicional, aportando su granito de arena para culminar nuestra meta propuesta.

RESUMEN

TÍTULO:

SISTEMA DE DETECCIÓN Y EVASIÓN DE OBSTÁCULOS EN UN AMBIENTE CONTROLADO, REALIZADO CON AR.DRONE 2.0 MEDIANTE ROS Y PYTHON.

AUTORES:

DAVID ERNESTO REINA MUNAR
KAREN YULIETH CERÓN MANIQUE

PALABRAS CLAVES:

Visión por computador, Algoritmo, Python, Dron, VANT, UAV, ROS, OpenCV, AR.Drone 2.0.

DESCRIPCIÓN:

En este trabajo se desarrolla e implementa un algoritmo escrito en lenguaje de programación Python, con el fin de realizar la detección y evasión de obstáculos de color rojo. Este algoritmo se basado en el uso de OpenCV para realizar la detección de obstáculos teniendo en cuenta la característica del color del objeto, así como también el uso de Robot System Operating (ROS) como entorno de desarrollo de programación para realizar el intercambio de datos entre el dron y la estación en tierra, y el uso de Python como lenguaje de programación en estas dos herramientas.

Una finalidad importante de este trabajo es contribuir con el desarrollo tecnológico de la región mediante el uso de drones, así como también sentar las bases para futuras investigaciones y proyectos no sólo con drones sino con cualquier robot que pueda ser programado con ROS, ya que la lógica del algoritmo desarrollado funciona de manera similar para todos, brindando así una amplia utilidad y flexibilidad a las personas que quieran adentrarse en esta área.

ABSTRACT

TITLE:

OBSTACLE DETECTION AND AVOIDANCE SYSTEM IN A CONTROLLED ENVIRONMENT WITH AR.DRONE 2.0 USING ROS AND PYTHON.

AUTHORS:

DAVID ERNESTO REINA MUNAR
KAREN YULIETH CERÓN MANRIQUE

KEYWORDS:

Computer vision, Algorithm, Python, Drone, VANT, UAV, ROS, OpenCV, AR.Drone 2.0.

DESCRIPTION:

In this work a python algorithm was developed, in order to carry out the detection and avoidance of red obstacles. This algorithm is based on the use of OpenCV to perform obstacle detection taking into account the color characteristic of the object, as well as the use of Robot System Operating (ROS) as programming development environment in order to data exchange between the drone and the ground station, and the use of Python as programming language in these two tools.

An important purpose of this work is to contribute to the technological development of the region by using drones, as well as to blaze the way for future research and projects not only with drones but with any robot that can be programmed with ROS because logic of the developed algorithm works in a similar way for every robot, thus providing a wide utility and flexibility to people who want go deeper into this area.

CONTENIDO

Pág.

LISTA DE TABLAS

Pág.

LISTA DE FIGURAS

Pág.

1. CAPÍTULO UNO: FUNDAMENTOS BÁSICOS

1.1. DRONES

Según Hernández *et al*: "Un dron se puede definir como un vehículo aéreo no tripulado por sus siglas VANT o por sus siglas en inglés UAV (Unmanned Aerial Vehicle). Existen varios tipos y formas de VANT's que pueden desarrollar diversos tipos de tareas…"[1]

> Los drones son vehículos aéreos de talla reducida, menos caros y más simples de construir que un avión. También son más discretos y su pérdida no es tan sensible o costosa como la de un vehículo convencional. El tamaño de los drones puede variar (desde algunos centímetros hasta varios metros), al igual que su forma y su tipo de propulsión, por ejemplo, algunos están equipados de reactores, otros de hélices o rotores, etc. Las aplicaciones de los drones son varias, las cuales abarcan desde las civiles hasta las militares, siendo estas últimas las más empleadas o conocidas. Los drones han sido en su mayor parte desarrollados en los conflictos militares. Dentro de sus aplicaciones civiles, las más deseables, tenemos la vigilancia de tráfico de carreteras, las operaciones de búsqueda aérea y salvamento, la recolección de información para la predicción meteorológica, la vigilancia de bosques o detección de fuegos, etc[2].

1.2. ROBOT OPERATING SYSTEM (ROS)

Ortego define el entorno de desarrollo ROS de la siguiente manera:

> Robot Operating System también conocido como ROS es una colección de frameworks para el desarrollo de software de robots. ROS se desarrolló inicialmente en 2007 bajo el nombre de switchyard por el Laboratorio de Inteligencia Artificial de Stanford para dar soporte al proyecto del Robot con Inteligencia Artificial de Stanford (STAIR2). Desde 2008, el desarrollo continuó principalmente en Willow Garage, un instituto de investigación robótico con más de veinte instituciones colaborando en un modelo de desarrollo federado[3].

Núñez, León y Cárdenas afirman que: "Aunque su nombre es la sigla para sistema operativo, en realidad ROS es un meta-sistema operativo, ya que,

---

[1] HERNÁNDEZ, Christian, *et al*. Dron polinizador de cultivos. Tecnologías aplicadas, para alternativas sustentables. En: *Revista Mexicana de Ciencias Agrícolas* [en línea]. Acapulco: Instituto Nacional de Investigaciones Forestales, Agrícolas y Pecuarias, febrero de 2015. vol.1, p. 67-71. [Consultado: 24 de marzo de 2020]. Disponible en https://www.redalyc.org/pdf/2631/263139243009.pdf. ISSN: 2007-0934.

[2] CASTILLO, Pedro, et al. Modelado y estabilización de un helicóptero con cuatro rotores. En: *Revista Iberoamericana de Automática e Informática Industrial* [en línea]. Valencia: Universidad Politécnica de Valencia, enero de 2007. vol.4, p. 41-57 [Consultado: 24 de marzo de 2020]. Disponible en https://polipapers.upv.es/index.php/RIAI/article/view/8176/8319. ISSN: 1697-7912.

[3] ORTEGO, Daniel. Qué es ROS (Robot Operating System) [blog]. OpenWebinars. 21 de septiembre de 2017. [Consultado: 15 de diciembre 2019]

aunque ofrece las funciones de un sistema operativo debe ser instalado sobre la base de otro sistema operativo –basado en UNIX-“[4]

1.3. OPEN SOURCE COMPUTER VISION (OPENCV)

Según lo plantea el equipo de redacción de Cheblender[5] OpenCV es una biblioteca de visión artificial libre desarrollada por Intel, que emplea todo tipo de aplicaciones que requieren incorporar el reconocimiento de objetos desde el año 1999.

OpenCV quiere decir “Open Source Computer Vision Library”, que se puede entender como una librería para el procesamiento de imágenes que tienen como propósito aplicaciones de visión en tiempo real. Es importante resaltar que, aunque esta biblioteca está escrita en códigos C y C++, también existen librerías como opencv-python que permite utilizar todas sus características en el lenguaje de programación Python. OpenCV es multiplataforma, es decir, está disponible para sistemas operativos como Linux, Mac OS X, Windows, Android y iOS.

1.4. PYTHON

Robledano plantea que Python: “Es un lenguaje de programación versátil multiplataforma y multiparadigma que se destaca por su código legible y limpio. Una de las razones de su éxito es que cuenta con una licencia de código abierto que permite su utilización en cualquier escenario. Esto hace que sea uno de los lenguajes de inclinación de muchos programadores siendo impartido en escuelas y universidades de todo el mundo”[6]

Por otro lado, Álvarez afirma lo siguiente: “Python es un lenguaje de scripting independiente de plataforma y orientado a objetos, preparado para realizar cualquier tipo de programa, desde aplicaciones Windows a servidores de red o incluso, páginas web. Es un lenguaje interpretado, lo que significa que no se necesita compilar el código fuente para poder ejecutarlo, lo que ofrece ventajas como la rapidez de desarrollo e inconvenientes como una menor velocidad”[7]

---

[4] NUÑEZ, Manuel; LEÓN, Carlos y CÁRDENAS, Pedro. Congreso Internacional de Electrónica y Tecnología de Avanzada [en línea]. En: (10: 26-28, marzo, 2014: Pamplona, Colombia). ROS Sistema operativo para robótica, nociones y aplicaciones. En: Revista Colombiana de Tecnologías de Avanzada. Pamplona: Universidad de Pamplona, marzo de 2014. p.1-7. [Consultado: 24 de marzo de 2020]. Disponible en: http://www.unipamplona.edu.co/unipamplona/portalIG/home_79/recursos/01general/06052014/memorias.jsp.

[5] ¿Qué es OpenCV? [blog]. Cheblender. 12 de marzo de 2018. [Consultado: 15 de diciembre 2019] Disponible: https://www.cheblender.org/que-es-opencv/.

[6] ROBLEDANO, Ángel. Que es Python: Características, evolución y futuro [blog]. OpenWebinars. 23 de septiembre de 2019, 2. [Consultado: 15 de diciembre de 2019]. Disponible en: https://openwebinars.net/blog/que-es-python/.

[7] Álvarez, Miguel. Qué es Python [blog]. Desarrollo Web. 19 de noviembre de 2003, 2. [Consultado: 15 de diciembre de 2019]. Disponible: https://desarrolloweb.com/articulos/1325.php

1.5. AR.DRONE 2.0

Es un vehículo aéreo no tripulado fabricado por la empresa francesa Parrot. Posee una autonomía de 12 minutos, un alcance de 50 metros, cámara de alta definición de 720p a 30fps que además de tomar fotos permite grabar videos también, el pilotaje se realiza a través de su aplicación diseñada para Smartphone o tabletas, posee un procesador ARM Cortex A8 de 32 bits a 1 GHz con video DSP a 800 MHz TMS320DMC64x, también cuenta con un sistema operativo Linux 2.6.32, así mismo cuenta con sensores abordo como acelerómetro, magnetómetro, barómetro, sensor de altura ultrasónico, y giroscopio.
Albornoz y Calahorrano afirman: “Debido al abaratamiento de los componentes electrónicos en los últimos años ha sido posible que estos aparatos lleguen al mercado como equipos para el ocio y recreación”[8]

Figura 1. AR.Drone 2.0 con protección para interiores

Fuente: Parrot, AR.Drone 2.0 elite edition [imagen]. Francia: 2017. [Consultado: 15 de diciembre de 2019]. Disponible en: https://www.parrot.com/es/drones/parrot-ardrone-20-elite-edition

[8] ALBORNOZ, Michael y CALAHORRANO, Darwin. Seguimiento de objetos basado en visión artificial para cuadrirrotor parrot AR.Drone 2.0 [en línea]. Proyecto previo a la obtención del titulo de ingeniero en electrónica y control. Quito: Escuela politécnica nacional. Facultad de ingeniería eléctrica y electrónica. 124 p. [Consultado: 24 de marzo 2020]. Disponible en: https://bibdigital.epn.edu.ec/bitstream/15000/16978/1/CD-7555.pdf

2. CAPÍTULO DOS: TRANSFERENCIA DE DATOS MEDIANTE ROS

2.1. ROS TOPIC

El término topic en ROS hace referencia a los buses sobre los cuales se intercambian mensajes. Un robot tiene muchos topics como por ejemplo de velocidad, de odometría, de sensores como la cámara, entre otros. Los topics son muy importantes ya que al permitir el intercambio de mensajes no sólo hacen posible la visualización en tiempo real de los datos del robot, sino que también permiten realizar una acción de control mediante el uso de los llamados nodos.

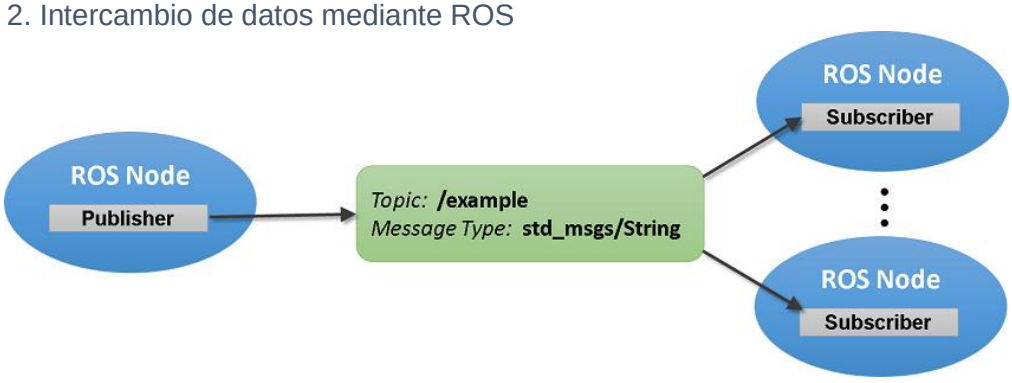
2.2. ROS PUBLISHER NODE

El nodo Publisher es un término usado en ROS para referirse a un archivo ejecutable (en este caso escrito en lenguaje Python) que contiene información como procesamiento de imágenes, redes neuronales o simplemente una cadena de caracteres y que envía esta información mediante un topic específico, esto con el fin de realizar una acción de control mediante el nodo Subscriber.

2.3. ROS SUBSCRIBER NODE

Al igual que el nodo Publisher el nodo Subscriber es un archivo ejecutable que realiza la suscripción (de ahí deriva su nombre) a un topic deseado para obtener los datos en tiempo real que están siendo publicados en dicho topic. Este nodo Subscriber es muy importante ya que en este proyecto es el encargado de obtener los datos y realizar una acción de control basado en el valor de estos datos obtenidos.

Figura 2. Intercambio de datos mediante ROS



Fuente: MathWorks, Exchange Data with ROS Publishers and Subscribers [imagen].Estados Unidos: 2018. [Consultado: 24 de marzo 2020]. Disponible en: https://es.mathworks.com/help/ros/ug/exchange-data-with-ros-publishers-and-subscribers.html

En la figura 2 se muestra el diagrama de funcionamiento de los nodos Publisher-Subscriber, los cuales permiten la configuración en tiempo real de las velocidades angulares (para el giro del robot) y lineales (para el avance del robot).

3. CAPÍTULO TRES: ALGORITMO DETECCIÓN Y EVASIÓN DE OBSTÁCULOS DE COLOR ROJO.

3.1. LIBRERÍAS.

Para la implementación del algoritmo propuesto en este trabajo de grado fue necesario realizar algunas operaciones como establecer el color (rango en formato Hue Saturation Value - HSV) del objeto que será determinado como obstáculo, la obtención de la imagen en tiempo real del AR.Drone 2.0, y la obtención de coordenadas del obstáculo.

Figura 3. Librerías de Python usadas para el desarrollo del algoritmo en el nodo Publisher.

```
import sys
import rospy
import cv2
from std_msgs.msg import String, Empty, Float32
from sensor_msgs.msg import Image
from collections import deque
from cv_bridge import CvBridge, CvBridgeError
import numpy as np
```

Figura 4. Librerías de Python usadas para el desarrollo del algoritmo en el nodo Subscriber.

```
import rospy
from std_msgs.msg import Float32, Empty
from geometry_msgs.msg import Twist
```

3.2. ALGORITMO DE DETECCIÓN DE OBSTÁCULOS DE COLOR ROJO.

El algoritmo realizado consta de dos códigos (Publisher y Subscriber). Por una parte, el código Publisher es el encargado de obtener la imagen del AR.Drone 2.0 en tiempo real y realizar el procesamiento de dicha imagen para determinar si hay un obstáculo presente y así conocer sus coordenadas. Por otro lado, el código Subscriber es el encargado de realizar las acciones de control pertinentes sobre el dron basado en el valor de las coordenadas obtenidas por el código Publisher.

3.2.1 Creación de la clase y definición de funciones

En el código Publisher (llamado CvBridge.py) se crea una clase llamada "*image_converter*" en la cual se definen dos funciones. La primera la función es el constructor de la clase *"__init__"*, esta función permite la creación de los publicadores, tanto de coordenadas en *x* e *y*, así como también el publicador

para el aterrizaje del dron; además de esto se realiza en esta función la suscripción al *topic* de la cámara del dron para obtenerla en tiempo real, y se crea el objeto *"CvBridge"* que permite la conversión del tipo de imagen obtenida en ROS al tipo de imagen usada en OpenCV.

Figura 5. Constructor de la clase "image_converter".

```
class image_converter:
def _init(self):
self.image_pub = rospy.Publisher("image_topic_2",Image,queue_size=10)
self.pub=rospy.Publisher('Coordenadas_Y',Float32,queue_size=1)
self.pub1=rospy.Publisher('Coordenadas_X',Float32,queue_size=1)
self.pub3=rospy.Publisher('Aterrizaje',Float32,queue_size=1)
self.bridge = CvBridge()
self.image_sub = rospy.Subscriber("/ardrone/front/image_raw",Image,self.callback)
```

La segunda función utilizada en esta clase es *"callback"* la cual se detalla a fondo en el siguiente ítem.

3.2.2 Establecimiento de los rangos de color en formato HSV para la detección de obstáculos

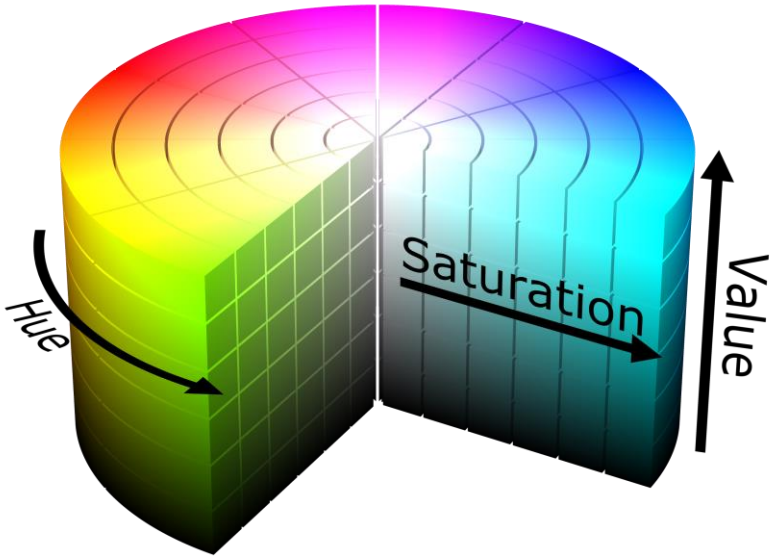
Para establecer el color que se usa como referente para determinar un obstáculo es necesario utilizar la librería "*numpy"*, ya que permite crear los arreglos que contienen los valores de los rangos en formato HSV (Hue Saturation Value), este tipo de formato de color está compuesto de las siguientes tres características:

Hue (Tono): Se debe concebir como un círculo de colores que varía desde el color rojo (0º) a rojo nuevamente (360º), pasando por colores como el verde (120º), el azul (240º) y los demás colores "puros" como el amarillo, cian, morado, entre otros.

Saturation (Saturación): Esta característica indica qué tanta cantidad de color escogido en el valor Hue aparecerá en la mezcla. Tiene un rango que varía entre 0 y 100%.

Value (Valor): La componente Value varía desde 0 a 100% y representa la cantidad de negro presente en el color escogido en la componente Hue (donde 0% es completamente negro y 100% es el color en su brillo máximo).

Figura 6. Representación del formato de color HSV (py2py).



Fuente: Py2py, HSV Model [imagen]. We already have RGB so why we nees HSV?. 2019. Disponible en: https://py2py.com/we-already-have-rgb-so-why-we-need-hsv/

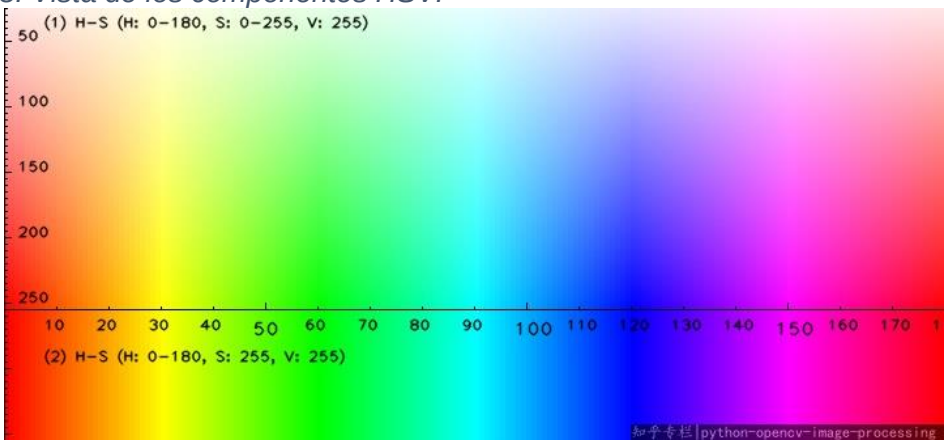
En la figura 7 se muestra la primera sección de la función "*callback*", esta función recibe como argumento la variable "*data*", que es la imagen obtenida en tiempo real por la función "*__init__"* con el fin de realizar su conversión a tipo de imagen usada en OpenCV para también realizar el procesamiento de esta, y así mismo, llevar a cabo la detección de los obstáculos presentes en la imagen de la cámara. Para esto, en primer lugar, usando la librería "*numpy*" se definen los rangos de color en los cuales debe estar el objeto de la imagen para ser detectado como un obstáculo.

Figura 7. Establecimiento de los rangos para detección de obstáculos.

```
def callback(self, data):
Lower = np.array([0, 100, 20],np.uint8)
Upper = np.array([5, 255, 255],np.uint8)
Lower1 = np.array([175, 100, 20],np.uint8)
Upper1 = np.array([179, 255, 255],np.uint8)
```

Cabe aclarar que para el software OpenCV las características *Saturation* y *Value* mencionadas anteriormente tienen un rango de posibles valores entre 0 y 255, mientras que la característica *Hue* abarca un rango desde 0 hasta 179. El color rojo en el formato HSV está presente dos veces como se muestra en la figura 8, por lo que es necesario realizar dos rangos diferentes.

Figura 8. Vista de los componentes HSV.



Fuente: Solano, Gabriela. Vista de los componentes HSV [imagen]. Detección de colores OpenCV - Pyhton (En 4 pasos). 2019. [Consultado: 24 de marzo de 2020]. Disponible en: https://omes-va.com/deteccion-de-colores/

3.2.3 Conversión de tipo de imagen y creación de las máscaras

En la segunda sección de la función *"callback"* se realiza la conversión de tipo de imagen ROS a tipo de imagen OpenCV, para esto se utiliza el módulo *"CvBrige"* importado de la clase *"cv_bridge"* junto con la funcionalidad *"imgmsg_to_cv2"*. Esta conversión proporciona una imagen OpenCV en formato bgr8, la cual posteriormente es convertida al formato HSV explicado anteriormente, con ayuda de la librería de OpenCV para Python conocida como *"cv2"*. Seguidamente se realiza la creación de las máscaras para cada rango de la figura 9 con la ayuda de la función *"inRange"*, la cual tiene como parámetros la imagen sobre la cual se realizará la detección de color, así como el rango inferior y superior del color que se desea detectar. Como era de esperarse, se realizan dos máscaras debido a que son dos rangos de color rojo diferentes para el formato HSV.

Figura 9. Conversión tipo de imagen y creación de máscaras.

```
try:
cv_image = self.bridge.imgmsg_to_cv2(data, "bgr8")
hsv = cv2.cvtColor(cv_image, cv2.COLOR_BGR2HSV)
mask1 = cv2.inRange(hsv, Lower, Upper)
mask2 = cv2.inRange(hsv, Lower1, Upper1)
```

3.2.4 Unificación de máscaras y operaciones morfológicas

Luego de la creación de las máscaras se realiza la unificación de estas para trabajar con una sola máscara que contenga los dos rangos establecidos anteriormente, para esto es necesario el uso de la función *"add"*, cuyos parámetros en este caso son las máscaras mostradas en la figura 9. Posteriormente, se hace uso de los operadores morfológicos *"erode"* (erosión) y *"dilate"* (dilatación) para completar la totalidad de la máscara.

Figura 10. Unificación de máscaras y operaciones morfológicas.

```
mask = cv2.add(mask1, mask2)
mask = cv2.erode(mask, None, iterations=2)
mask = cv2.dilate(mask, None, iterations=2)
```

3.2.5 Creación del contorno

Para poder visualizar el contorno del obstáculo detectado se hace uso de la función *"findContours"* de la librería *"cv2"*. Esta característica recibe como parámetros la máscara binaria obtenida anteriormente, el modo de recuperación del contorno y el método de aproximación del contorno.

Figura 11. Creación del contorno.

```
cnts = cv2.findContours(mask.copy(),
cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)[-2]
```

3.2.6 Obtención de las coordenadas del obstáculo

Inicialmente se crea la variable *"center"* pero no se le asigna ningún valor momentáneamente. Luego se crea un ciclo *"if"* que permite realizar las siguientes acciones si la variable *"cnts"* (el contorno) es mayor que cero: obtener el contorno más grande presente en la imagen mediante la función *"max"* de Python, obtener el centro (coordenada x, coordenada y) y radio del círculo más pequeño posible que encierre el obstáculo detectado. Seguido de esto se calcula los momentos de la imagen mediante la función *"moments"* de OpenCV, esta función permite calcular ciertas propiedades de una imagen como por ejemplo el radio, el área, el centroide, entre otras. Teniendo esta función se calcula el centroide del círculo que encierra el obstáculo. Seguido de esto, se crean dos variables *"b"* y *"a"*, las cuales contienen las coordenadas x e y respectivamente del círculo que encierra al obstáculo detectado. Posteriormente, mediante la librería *"rospy"* se imprime en el terminal de Linux las debidas coordenadas x e y del obstáculo. Por último, se utiliza la función *"sleep"* de la librería *"rospy"* la cual permite ejecutar el ciclo a una velocidad o tasa determinada (en este caso especificada en la variable *"rate"*).

Figura 12. Obtención de las coordenadas del obstáculo.

```
center = None
if len(cnts) > 0:
c = max(cnts, key=cv2.contourArea)
((x, y), radius) = cv2.minEnclosingCircle(c)
M = cv2.moments(c)
center = (int(M["m10"] / M["m00"]), int(M["m01"] / M["m00"]))
a=(int(M["m01"] / M["m00"]))
b=(int(M["m10"] / M["m00"]))
rospy.loginfo('%i %s %i', int(M["m10"] / M["m00"]),",",int(M["m01"] / M["m00"]))
rate.sleep()
```

3.2.7 Visualización del borde del circulo y el centroide

Para poder mostrar el borde del círculo y su centroide se tiene en cuenta el tamaño del radio obtenido por la función *"minEnclosingCircle"* almacenado en la variable *"radius"*; si este valor es mayor a 10 pixeles (10px) se crea un círculo con la ayuda de la función *"circle"* de la librería *"cv2"* a la cual se le pasa como parámetros la imagen de entrada (cv_image), las coordenadas del círculo, el radio del círculo, el color deseado (en formato BGR) y el espesor para el borde del círculo. Para visualizar el centroide se procede del mismo modo, utilizando como parámetros la misma imagen de entrada, el centroide obtenido por la función *"moments"*, el radio del círculo, el color deseado y el espesor.

Figura 13. Visualización del borde del circulo y el centroide.

```
if radius > 10:
cv2.circle(cv_image, (int(x), int(y)), int(radius),(0, 255, 255), 2)
cv2.circle(cv_image, center, 5, (0, 0, 255), -1)
```

3.2.8 Obtención de coordenadas negativas

Las siguientes líneas de código tienen como finalidad conocer cuándo no hay presencia de obstáculos. Si la longitud de la variable *"cnts"* es menor que cero, indica que no hay obstáculo y por tanto no hay contorno ni coordenadas detectadas, por lo que es necesario que las variables *"b"* y *"a"* (coordenadas del centroide) tengan valores negativos, para poder realizar las acciones de control cuando no haya obstáculo.

Figura 14. Obtención de coordenadas negativas.

```
else:
a=(int(-1))
b=(int(-2))
```

3.2.9 Detección de error en la conversión de imagen

Con el fin de detectar un error en el proceso de conversión de imagen tipo ROS a imagen tipo OpenCV (BGR) se utiliza la declaración *"except"* y se publica en el terminal.

Figura 15. Detección de error en la conversión de imagen.

```
except CvBridgeError as e:
print(e)
```

3.2.10 Publicación de la imagen final

Mediante la función *"imshow"* de la librería *"cv2"* se realiza la publicación de la imagen final, la cual recibe como parámetros el nombre de la ventana que mostrará la imagen y la imagen final.

Figura 16. Publicación de la imagen final.

```
cv2.imshow("Image window", cv_image)
```

3.2.11 Publicación de la imagen final convertida a formato tipo ROS y de la coordenada X

Utilizando la función *"publish"* así como la función *"CvBrige"* se realiza la conversión de formato OpenCV (BGR) a formato tipo ROS y se realiza tanto la publicación de esta imagen como la coordenada X del obstáculo.

Figura 17. Publicación de la imagen final convertida a formato tipo ROS y de la coordenada X.

```
try:
self.image_pub.publish(self.bridge.cv2_to_imgmsg(cv_image, "bgr8"))
self.pub1.publish(b)
except CvBridgeError as e:
print(e)
```

3.2.12 Definición de la función principal (main)

En esta función se realiza la ejecución de la clase *"image_converter"*. Además, utilizando la función *"spin"* de la librería *"rospy"* permite ejecutar el nodo hasta que sea interrumpido, en este caso por interrupción de teclado.

Figura 18. Definición de la función principal (main).

```
def main():
ic = image_converter()
try:
rospy.spin()
except KeyboardInterrupt:
print("Shutting down")
cv2.destroyAllWindows()
```

3.2.13 Iniciador de la función principal (main)

En este condicional se crea el nodo llamado *"image_converter"* así como también se declara como global la variable *"rate"* y se inicializa esta variable mediante la función *"Rate"* la cual proporciona la velocidad o tasa de publicación de datos, por último, se ejecuta la función *"main"*.

Figura 19. Iniciador de la función principal (main).

```
if _name_ == '_main_':
rospy.init_node('image_converter', anonymous=True)
global rate
rate = rospy.Rate(20)
main()
```

3.3. ALGORITMO DE EVASIÓN DE OBSTÁCULOS DE COLOR ROJO.

3.3.1. Creación de la clase y su constructor

En primer lugar, se crea una clase llamada *"Detector"* y seguidamente su constructor, esto con el fin que cuando se cree un objeto de esta clase se inicialicen sus atributos definidos en esta sección, los cuales son *"self.coordenadax"* haciendo referencia a la variable que almacena la coordenada x, así como *"self.coordenaday"* que hace referencia a la variable almacenada la coordenada y, en donde ambas son inicializadas con valor *"None"*, con el fin de crear las variables pero sin asignarle valor alguno.

Figura 20. Creación de la clase y su constructor.

```
class Detector():
def __init__(self):
self.coordenadax = None
self.coordenaday = None
```

3.3.2. Definición de la función de acción de control

En este punto se crea la función llamada *"coordenadax_callback(self,data)"* que es la encargada de realizar las acciones de control pertinentes dependiendo si

recibe coordenadas positivas o negativas. Para esto en primer lugar se crean dos publicadores *"pub2"* y *"pub3"* a los topics *"/cmd_vel"* y *"ardrone/land"* respectivamente, esto con el fin de publicar mensajes o enviar datos a estos topics mencionados anteriormente. Los datos que recibe esta función (data) son almacenados en la variable *"self.coordenadax"*.

Figura 21. Definición de la función de acción de control.

```
def coordenadax_callback(self, data):
global pub2
pub3=rospy.Publisher('ardrone/land',Empty,queue_size=10)
pub2=rospy.Publisher('/cmd_vel',Twist,queue_size=10)
self.coordenadax = data
```

3.3.3. Condicional para realizar la acción de control

En esta sección se propone un condicional *"if"* que es el encargado de evaluar los valores obtenidos de la coordenada x proveniente del algoritmo de detección de obstáculos, si el valor recibido es igual a -2 quiere decir que no hay obstáculo presente, por tanto, la velocidad lineal tendrá un valor diferente de cero, para que de esta manera el dron avance y la velocidad angular será igual a cero para que no realice ningún giro. Pero si es diferente de -2 hace referencia a que se detecta un obstáculo, por tanto, aquí la velocidad lineal se hace igual a cero para que el dron no avance, y la velocidad angular toma un valor diferente de cero, haciendo que el dron gire, y de esta manera evadir el obstáculo. Igualmente, en cada parte del ciclo (if y else) se inicia un conteo para cada variable (*i* y *j*), cuyas funciones se definen en la sección 3.3.5.

Figura 22. Condicional para realizar la acción de control.

```
if data.data == -2:
rospy.loginfo('NO HAY OBSTACULO')
msg.angular.z = 0
msg.linear.x = 0.03
i=0
j=j+1
else:
rospy.loginfo('SI HAY OBSTACULO')
msg.linear.x = 0
msg.angular.z = 0.1
i=i+1
j=0
```

3.3.4. Publicación de las velocidades y visualización de los temporizadores

Una vez establecidos los valores de las velocidades lineales y angulares, se procede a publicar en el respectivo nodo (/cmd_vel) dichos valores, con el fin de que el dron realice las instrucciones dadas en la sección 3.3.3. Además de esto, se visualiza en el terminal los temporizadores correspondientes al avance y giro del dron.

Figura 23. Publicación de las velocidades y visualización de los temporizadores.

```
pub2.publish(msg)
rospy.loginfo('%s %i','Temporizador de giro: ',i)
rospy.loginfo('%s %i','Temporizador de avance: ',j)
```

3.3.5. Aterrizaje del dron de acuerdo con los temporizadores

En la sección 3.3.3 se inicia el conteo de dos variables (*i* y *j*), las cuales hacen las veces de un temporizador que permite al dron aterrizar cuando el conteo de alguna de las variables llegue al valor establecido, para esto se realiza la publicación de un mensaje vacío al topic *land* que es el encargado de llevar a cabo el aterrizaje del dron.

Figura 24. Aterrizaje del dron de acuerdo a los temporizadores.

```
if i > 350:
rospy.loginfo('ATERRIZANDO')
pub3.publish(Empty())
if j > 350:
rospy.loginfo('ATERRIZANDO')
pub3.publish(Empty())
```

3.3.6. Iniciador de la función principal

Se inicializa un nodo llamado *"listener"* necesario para ejecutar las publicaciones. Así como también se crea un objeto de la clase *"Detector"*, se declaran como globales las variables *i* y *j*, se realiza la subscripción al topic de la coordenada X y de la coordenada Y creados en el algoritmo de detección de obstáculos de color rojo en la sección 3.2.1.

Figura 25. Iniciador de la función principal.

```
if _name_ == '_main_':
rospy.init_node('listener')
detector = Detector()
global i,j
i=0
j=0
rospy.Subscriber('Coordenadas_X', Float32, detector.coordenadax_callback)
rospy.Subscriber('Coordenadas_Y', Float32, detector.coordenaday_callback)
rospy.spin()
```

4. RESULTADOS Y DISCUSIONES

Antes que nada, es preciso aclarar que el aporte de este proyecto de grado es de tipo académico, debido a que se usan tecnologías que además de interesantes son útiles y que en esta región hasta el momento han sido poco exploradas. En esta sección se presentará los resultados del funcionamiento del algoritmo propuesto y a su vez se comentará el resultado de su desempeño.

Es necesario resaltar que la manera como está compuesto ROS, es decir, su arquitectura, proporciona utilidades como la obtención de datos (imágenes y sensórica) para su posterior procesamiento, así como también la modificación de valores de variables (velocidades de motores) para realizar acciones de control deseadas. Es por esto, que los resultados obtenidos al utilizar este entorno de desarrollo fueron los esperados, permitiendo al dron realizar las acciones pertinentes en los momentos determinados.

Además de lo dicho anteriormente, se puede destacar la gran ventaja y eficacia del paquete *cv_bridge* en cuanto a la obtención de imágenes en tiempo real sobre otras herramientas como el software *ffmpeg* y la función *VideoCapture* de OpenCV. Es posible realizar esta afirmación debido a que durante todo el proceso de construcción del algoritmo se realizaron pruebas con estas tres herramientas encontrando que la obtención de las imágenes por medio de *ffmpeg* y *VideoCapture* tenían un retraso significativo (aproximadamente 15 segundos) que a su vez retrasaban todo el mecanismo de control del sistema haciéndolo ineficiente, mientras que al usar el paquete *cv_bridge* la obtención de la imagen del robot se realiza de inmediato.

Es importante señalar que el uso del software OpenCV para procesamiento de imágenes proporciona muchas facilidades y funcionalidades que hacen de esta una librería muy completa, ideal para los diversos casos en donde se requiera realizar el procesamiento de imágenes. Sin embargo, la eficacia de la detección de los obstáculos en el algoritmo presentado depende en gran medida del correcto establecimiento de los rangos de colores a detectar, ya que el ajuste de un rango muy amplio se traduce en la detección de colores no deseados y, por el contrario, el ajuste de un rango muy reducido equivaldría a no detectar todas las posibles gamas del color escogido.

La eficacia del algoritmo se puede ver afectada por el estado del dron, ya que debido al desgaste de sus componentes puede hacer que haya variaciones en el desempeño de este y por consiguiente obtener resultados no deseados. Así como también las condiciones climáticas pueden incurrir en el funcionamiento del robot y en el rendimiento del algoritmo. En la tabla 1 se realiza una comparación entre las características del AR.Drone 2.0 nuevo con las del AR.Drone 2.0 usado en este proyecto de grado, que confirman que el desgaste en sus componentes afecta el desempeño del algoritmo creado.

Tabla 1. Características AR.Drone 2.0 nuevo y usado.

Características	AR.Drone 2.0 nuevo	AR.Drone 2.0 usado
Alcance de la red wifi	50 metros aproximadamente	10 metros aproximadamente
Altura por defecto usando ROS	1 metro	Menor a 1 metro
Tiempo de carga de baterías	1 hora y 30 minutos aproximadamente	2 horas aproximadamente
Tiempo de vuelo	12-15 minutos	5-7 minutos

Se realizaron un total de nueve (9) pruebas del algoritmo presentado variando el parámetro de la velocidad lineal respecto al eje x (velocidad de avance del dron), así como también la forma de los obstáculos y su tamaño.

En la tabla 2 se presenta la convención usada para distinguir los obstáculos usados en las pruebas.

Tabla 2. Convención para los obstáculos utilizados.

Nombre	Área (cm^2)	Forma
Obstáculo A	875	Rectángulo
Obstáculo B	187	Rectángulo
Obstáculo C	49.2	Rectángulo
Obstáculo D	320.47	Círculo

En la tabla 3 se presenta la convención usada para representar cada una de las pruebas realizadas.

Tabla 3. Convención para las pruebas realizadas.

Prueba	Obstáculo	Velocidad lineal en x (m/s)	Velocidad angular en z (rad/s)	Temporizadores de aterrizaje (s)
1	A	0.05	0.1	17.5
2	B	0.01	0.1	17.5
3	B	0.05	0.1	17.5
4	C	0.03	0.1	17.5
5	C	0.03	0.1	17.5
6	C	0.03	0.1	17.5
7	D	0.01	0.1	17.5
8	D	0.03	0.1	17.5
9	D	0.03	0.1	17.5

Con la finalidad de realizar la captura de los datos en cada prueba para su posterior análisis se hace uso de la funcionalidad *"rosbag"* de ROS, que a su vez también permite almacenarlos en un archivo de texto.

Una vez obtenido el archivo de texto para cada prueba, se procede a exportarlos a Excel con el fin de organizarlos y elegir los datos que se van a analizar, tales

como tiempo y altura. Seguidamente los datos son importados directamente desde MatLab para poder graficarlos y así tener una mejor perspectiva de estos.

Cabe aclarar que el uso de la herramienta MatLab es sólo para realizar el análisis de estos datos, es decir, no hace parte del algoritmo realizado.

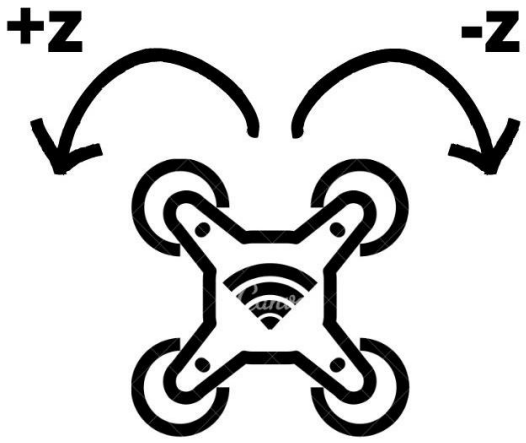
En la figura 26 se muestran los ejes del dron con sus respectivos nombres.

Figura 26. AR.Drone 2.0 con sus respectivos ejes.

Fuente: Maravall, D., de Lope, J., Fuentes, P.

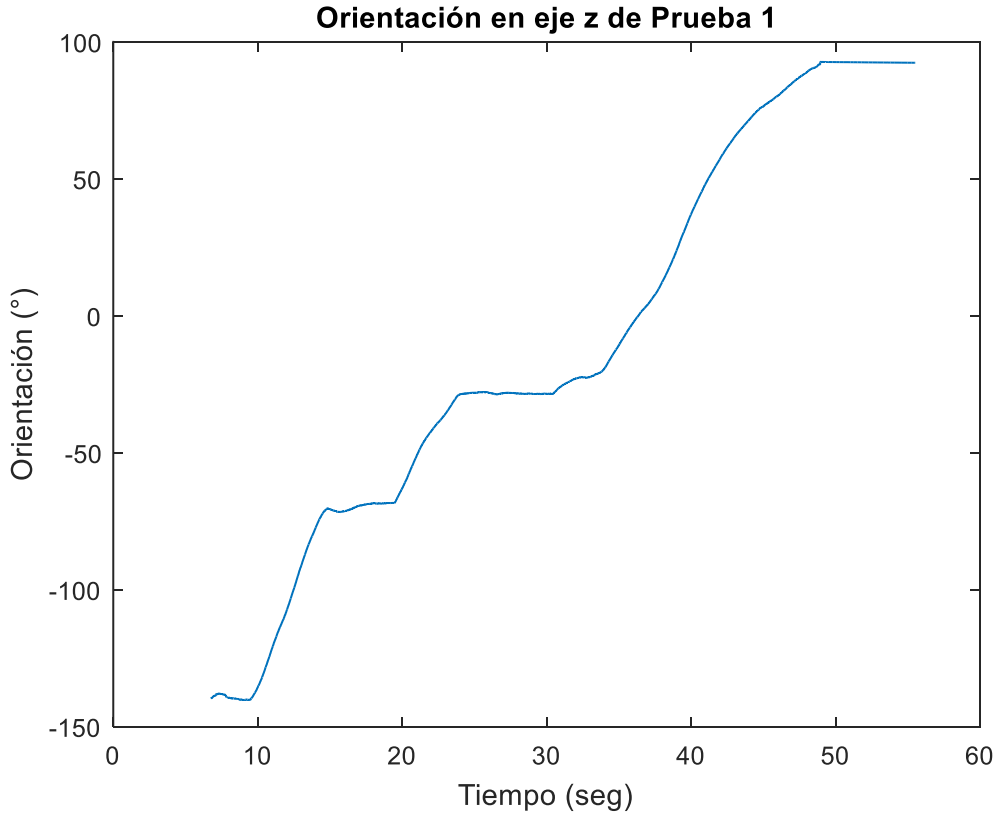
El algoritmo presentado permite al dron realizar la detección y evasión de obstáculos de color rojo girando siempre en sentido positivo respecto del eje z (sentido antihorario). En la figura 27 se pueden apreciar los sentidos de giro respecto dicho eje.

Figura 27. Dirección de giro del dron respecto del eje z (yaw).



La figura 28 muestra los datos de orientación vs tiempo obtenidos para la Prueba 1. Esta prueba tiene una duración de 55.51 segundos, en donde el dron evade 3 obstáculos y posteriormente aterriza. Para esta prueba el dron registra su despegue en el segundo 6.73, por lo cual el análisis se realiza a partir de este punto. Para todas las pruebas realizadas el valor inicial de orientación para el dron oscila entre los -150° y -130°, y teniendo en cuenta que el giro respecto del eje z se realiza en sentido positivo es correcto afirmar que los tramos ascendentes de la curva mostrada en la figura 28 se traduce en la detección de un obstáculo ya que el dron se encuentra girando, mientras que los tramos "planos" reflejan los momentos en que el dron se encuentra avanzando en línea recta. Desde el segundo 31 aproximadamente se coloca de manera intencionada un obstáculo al frente del dron para que realice el aterrizaje pasados los 17.5 segundos correspondientes al temporizador, por lo que en el segundo 58 aterriza. Por el análisis realizado anteriormente se puede corroborar el correcto funcionamiento del algoritmo realizado en este proyecto de grado.

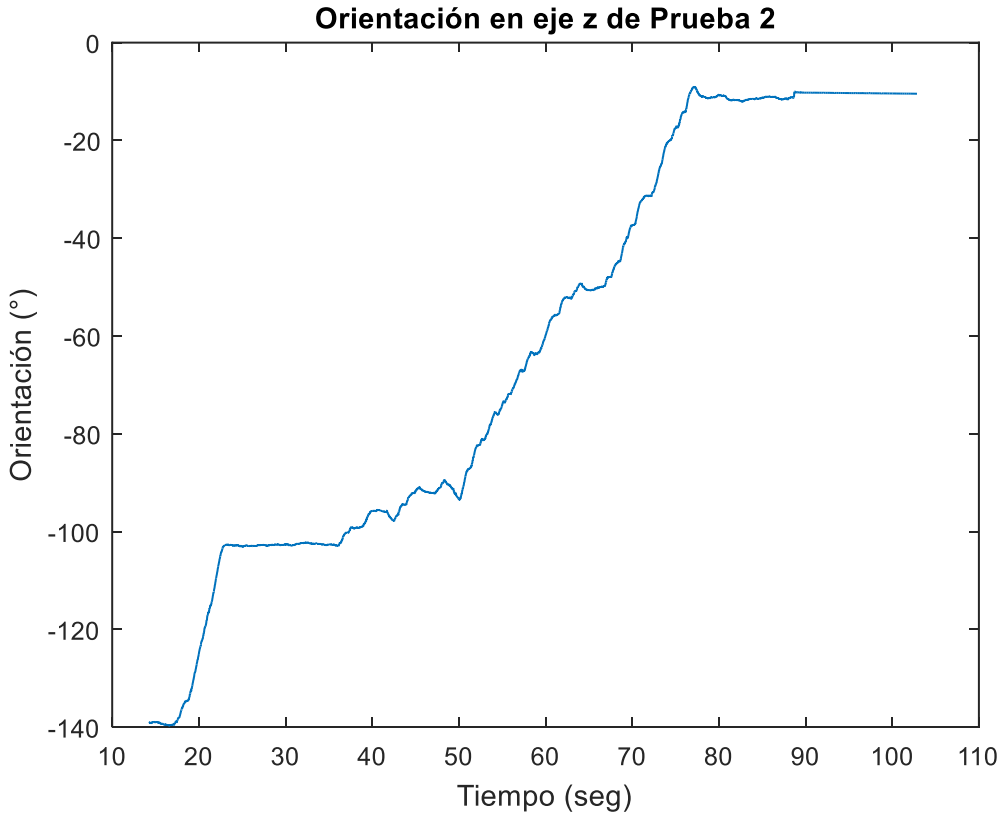
Figura 28. Gráfica de orientación vs tiempo para Prueba 1.



En la figura 29 se puede apreciar los datos de orientación respecto del eje z (yaw) para la Prueba 2. Esta prueba tiene una duración de 102.8 segundos (1 minuto y 43 segundos aproximadamente), en donde el dron evade 3 obstáculos y seguidamente aterriza. Para esta prueba el dron registra su despegue en el segundo 14.25, por lo que los datos obtenidos antes de este punto no son analizados. La orientación inicial para esta prueba es de -140° aproximadamente. Se puede observar claramente las tres evasiones que realiza

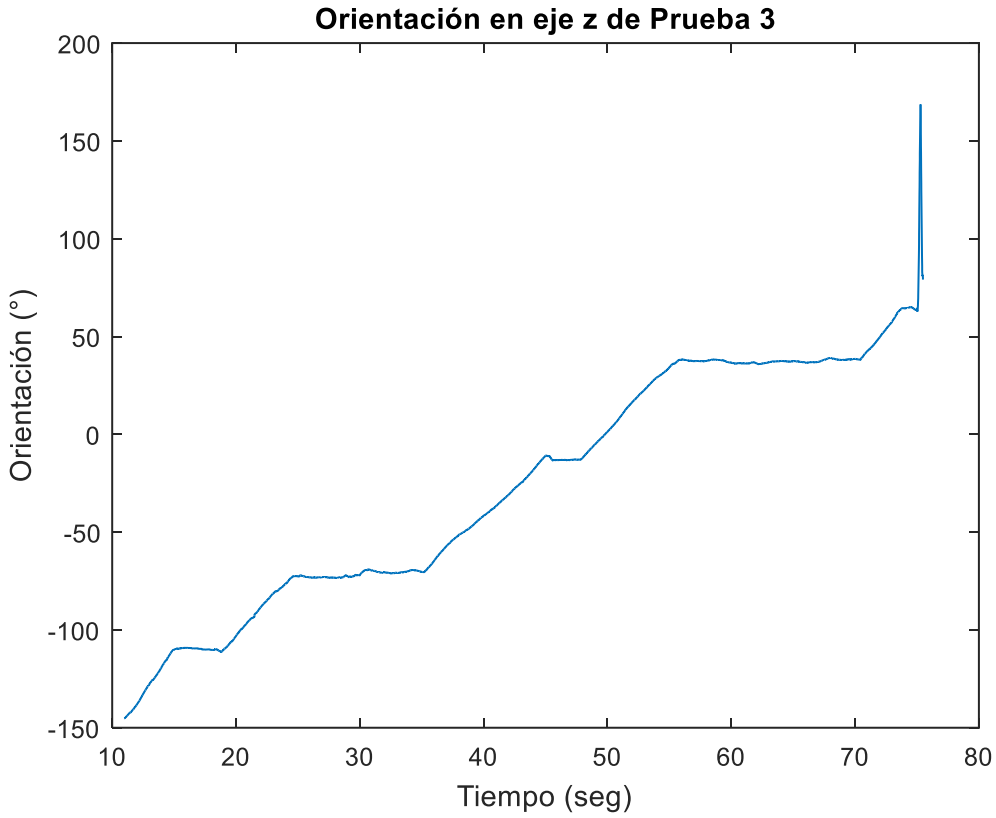
el dron para esta prueba reflejadas en los tramos ascendentes de la curva mostrada. Desde el segundo 71 hasta el segundo 88 se puede apreciar que la curva se aplana, por lo que es correcto afirmar que durante este lapso de tiempo el dron avanza y pasados los 17.5 segundos del temporizador finalmente aterriza.

Figura 29. Gráfica de orientación vs tiempo para Prueba 2.



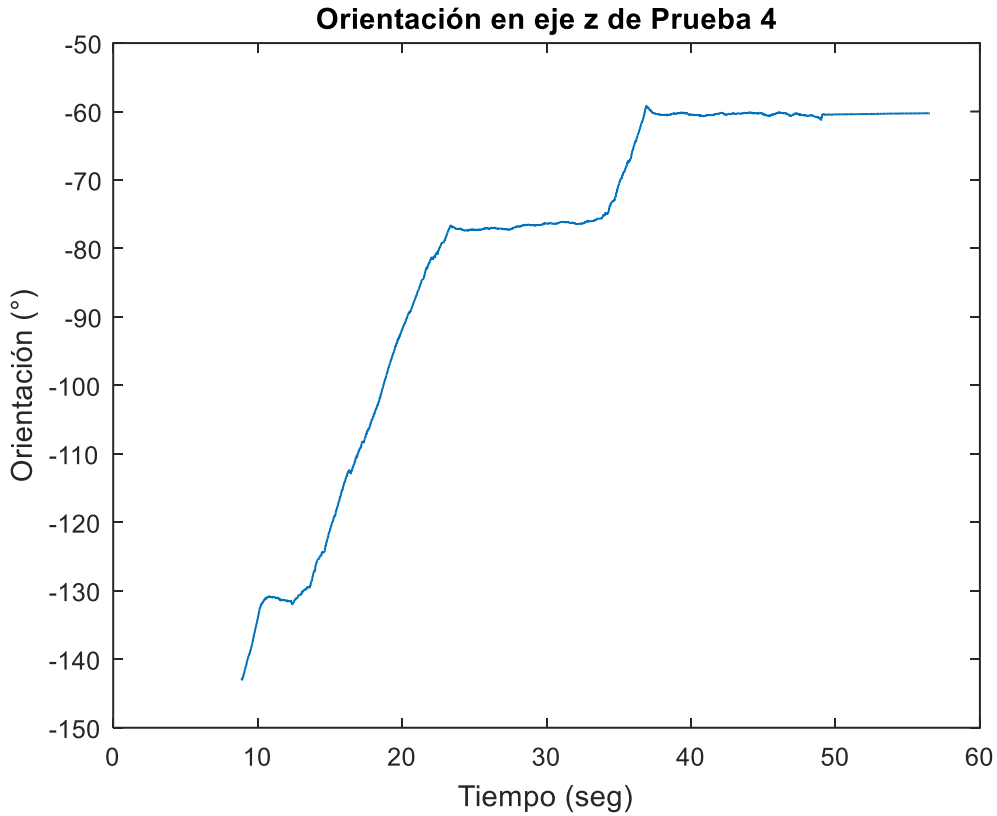
De igual forma la figura 30 muestra los datos de orientación respecto del eje z vs tiempo para la Prueba 3. Este caso tiene una duración de 117.5 segundos, el dron registra su despegue en el segundo 10.96 y el dron evade 3 obstáculos, sin embargo, en la figura 4.5 se observan 5 tramos ascendentes, es decir, 5 evasiones realizadas, esto se debe a que al finalizar la segunda evasión el dron no queda en línea de vista con un obstáculo sino que sigue derecho hacia la pared del recinto por lo cual es necesario colocar un obstáculo externo para evitar una colisión, seguido de esto el dron queda en línea de vista con el tercer obstáculo del recinto y realiza su correspondiente evasión (cuarto tramo ascendente). Llegando a la parte final de la gráfica 30 es posible visualizar un quinto tramo ascendente correspondiente a la evasión un obstáculo externo colocado de manera intencionada porque el dron va a colisionar con una pared del recinto, sin embargo, el dron ya está muy cerca de la pared y no alcanza a cambiar de dirección resultando en una colisión reflejada en el pico mostrado al final de la gráfica 30.

Figura 30. Gráfica de orientación vs tiempo para Prueba 3.



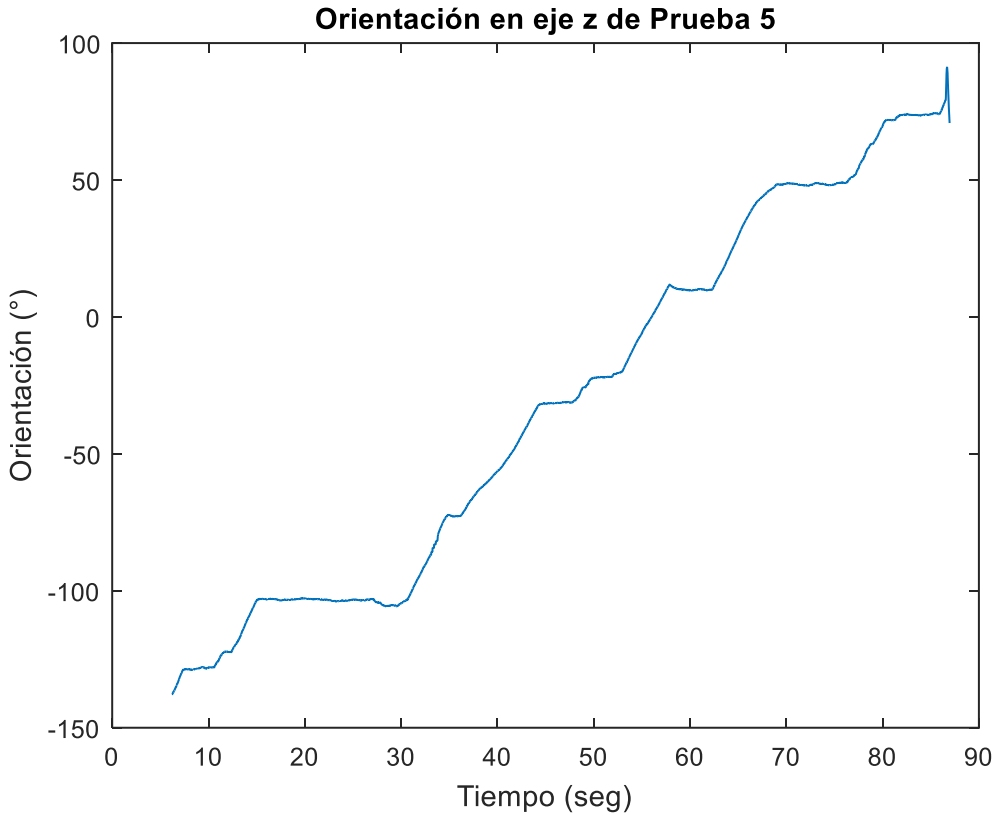
El siguiente análisis corresponde a los datos obtenidos para la Prueba 4, la cual tiene una duración de 56.5 segundos, el dron tiene una orientación inicial de -145° aproximadamente, y realiza el despegue en el segundo 8.88, por lo que el análisis se realizará a partir de este valor. En esta prueba el dron evade dos obstáculos y aterriza 12 segundos después de la última evasión debido a que el dron va a sufrir una colisión por lo que es aterrizado de manera remota por un comando externo al algoritmo ingresado desde la estación en tierra. En la figura 31 se muestran los datos de orientación respecto del eje z vs tiempo para esta prueba.

Figura 31. Gráfica de orientación vs tiempo para Prueba 4.



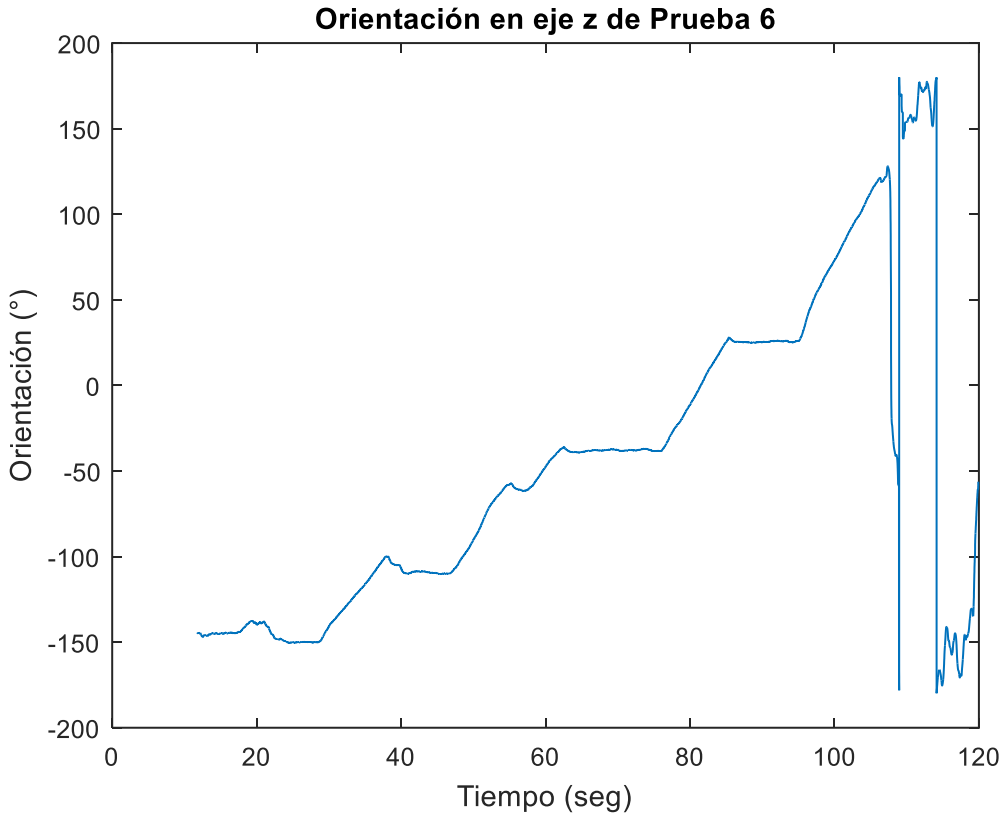
La Prueba 5 tiene una duración de 87 segundos, la orientación inicial del dron es -135° aproximadamente y el despegue se registra en el segundo 6.19. En la figura 4.7 se plasman los datos obtenidos de orientación respecto del eje z contra el tiempo para esta prueba. En esta figura se aprecian 6 tramos ascendentes que corresponden con 6 evasiones realizadas; de estos 6 tramos, el tercero y el quinto están relacionados con obstáculos colocados de manera intencional para guiar al dron hacia los obstáculos del recinto (primer, segundo, cuarto y sexto tramo). Al final de la figura 32 se observa un pico que está ligado a una perturbación que sufre el dron en una de sus hélices en el momento en que está realizando la última evasión, por lo cual, después de este suceso se termina la transmisión de datos de manera automática.

Figura 32. Gráfica de orientación vs tiempo para Prueba 5.



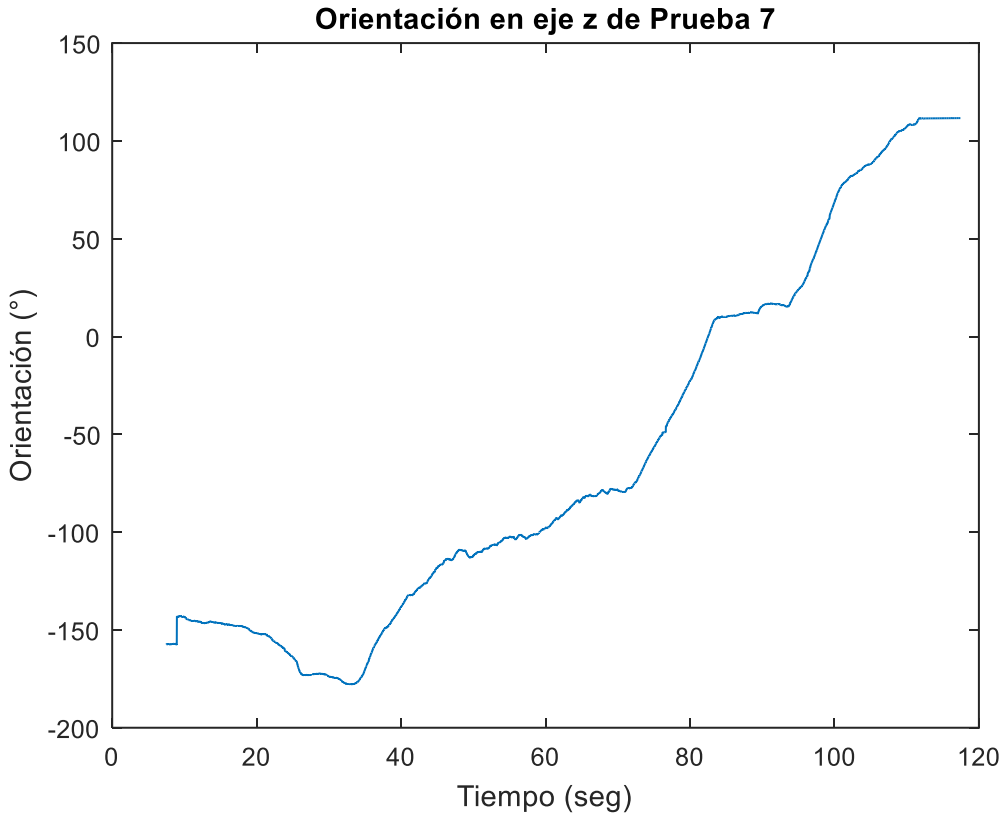
En la figura 33 se muestran los datos obtenidos de orientación respecto del eje z contra el tiempo para la Prueba 6. Esta prueba tiene una duración de 129 segundos, el dron tiene una orientación inicial de -145° aproximadamente, y realiza la evasión de 2 obstáculos del recinto; sin embargo, en la figura 33 se aprecian 5 tramos ascendentes, de los cuales el primero y segundo están relacionados con obstáculos colocados intencionadamente para guiar el dron hacia los obstáculos del recinto. El último tramo ascendente hace referencia a la evasión del obstáculo final para realizar el aterrizaje, no obstante, es necesario realizar un aterrizaje forzoso, debido a la falta de espacio en el recinto para poder llevar a cabo el aterrizaje programado 17.5 segundos después de giro o avance constante; lo anterior se ve reflejado en la parte final de la gráfica mostrada en la figura 33.

Figura 33. Gráfica de orientación vs tiempo para Prueba 6.



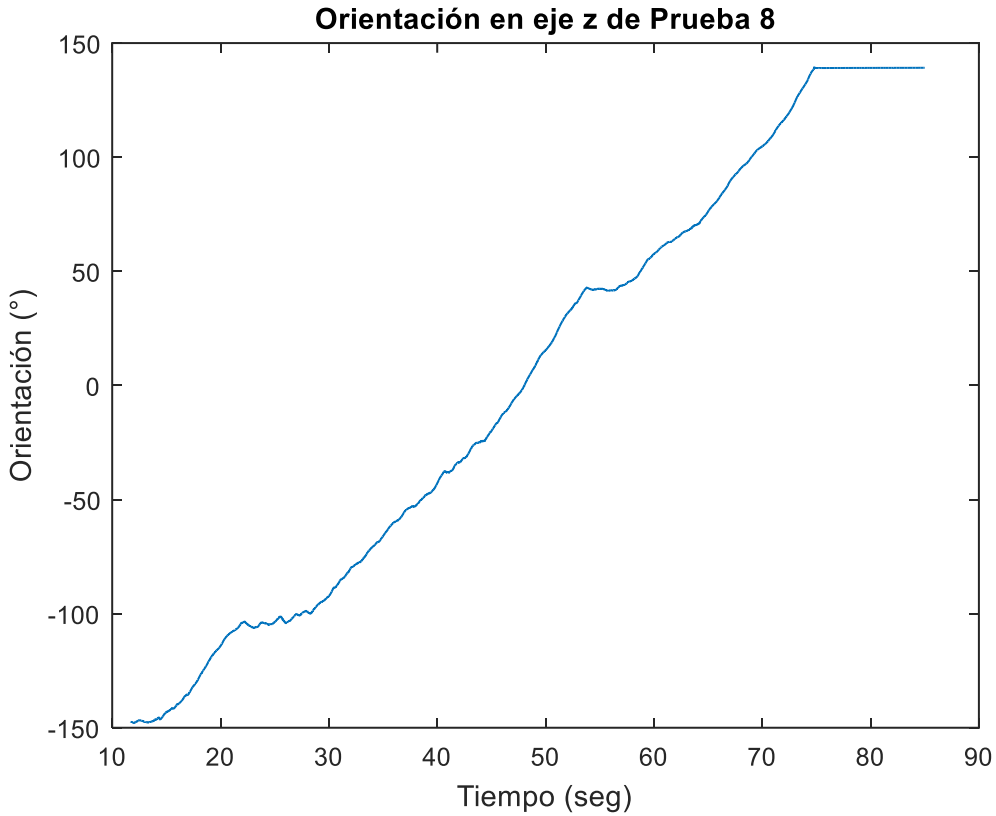
La siguiente prueba tiene una duración de 117.43 segundos, el dron tiene una orientación inicial de -145° aproximadamente, y registra su despegue en el segundo 9; en la gráfica 34 se presentan los datos obtenidos para la Prueba 7 de orientación respecto del eje z contra el tiempo. En esta gráfica se puede observar que justo después del momento del despegue el dron gira en sentido horario (sentido negativo del eje z), esto se debe a inestabilidad del dron al momento de despegar, sin embargo, es necesario hacer uso de un obstáculo externo para redirigirlo hacia los obstáculos del recinto. En la figura 34 se pueden observar 4 tramos ascendentes, correspondiente a 4 evasiones, sin embargo, parte del primer tramo corresponde a un obstáculo externo para guiar al dron hacia el primer obstáculo del recinto, el segundo tramo se ubica entre 60 y 75 segundos aproximadamente, correspondiente a la evasión del segundo obstáculo. Seguido de esto, el tercer tramo ascendente está relacionado con la detección y evasión del tercer y cuarto obstáculo del recinto, es decir, el dron al momento de evadir el tercer obstáculo inmediatamente queda en línea de vista con el cuarto obstáculo por lo que sigue girando hasta alrededor del segundo 90. Finalmente, se coloca un obstáculo final al frente del dron por 17.5 segundos para que realice su respectivo aterrizaje.

Figura 34. Gráfica de orientación vs tiempo para Prueba 7.



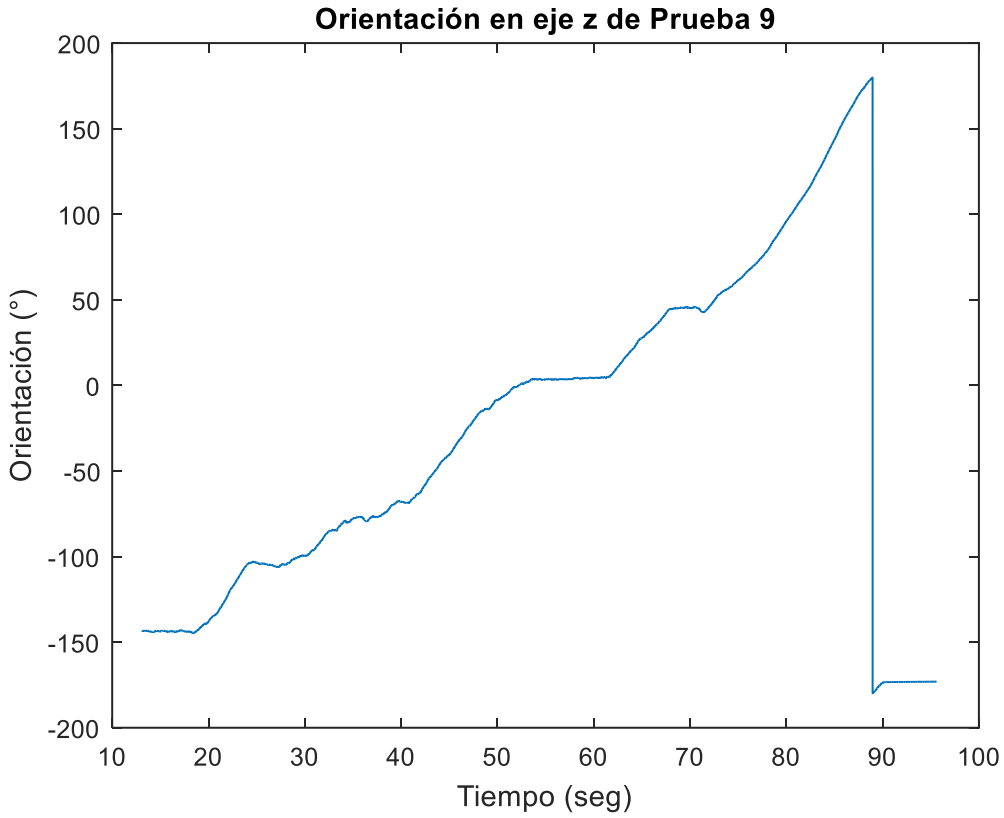
En la figura 35 se representan los datos de orientación respecto del eje z versus el tiempo para la Prueba 8. Esta prueba tiene una duración de 85 segundos, a su vez el dron tiene una orientación inicial de -145° aproximadamente, presenta el despegue en el segundo 10, realiza la evasión de los 4 obstáculos del recinto y finalmente aterriza con ayuda de un obstáculo externo. En la figura 35 se pueden apreciar 3 tramos ascendentes, el primero de ellos corresponde a la evasión del primer obstáculo del recinto, el segundo está relacionado con la evasión del segundo, tercer y cuarto obstáculo del recinto, ya que al terminar de evadir el segundo queda inmediatamente en línea de vista con el tercer obstáculo y de igual manera cuando termina de evadir el tercer obstáculo continua en línea de vista con el último. Concluyendo el análisis de esta prueba, el tercer tramo ascendente coincide con los 17.5 segundos de evasión de un obstáculo externo colocado con la intención de realizar el aterrizaje del dron.

Figura 35. Gráfica de orientación vs tiempo para Prueba 8.



Finalmente se presenta la figura 36 que corresponde a la última prueba realizada donde se plasman los datos de orientación en el eje z respecto del tiempo. Para este caso la duración es de 95 segundos, el dron tiene una orientación inicial de -145° aproximadamente, presenta el despegue en el segundo 13 y realiza la evasión de los 4 obstáculos del recinto. En la figura 36 es posible observar 4 tramos ascendentes, el primero de ellos corresponde a la evasión del primer obstáculo del recinto, el segundo corresponde a la evasión del segundo y tercer obstáculo respectivamente, el tercer tramo ascendente corresponde a la evasión del último obstáculo del recinto y finalmente, el último tramo ascendente corresponde a los 17.5 segundos de evasión del obstáculo externo para realizar el aterrizaje.

Figura 36. Gráfica de orientación vs tiempo para Prueba 9.

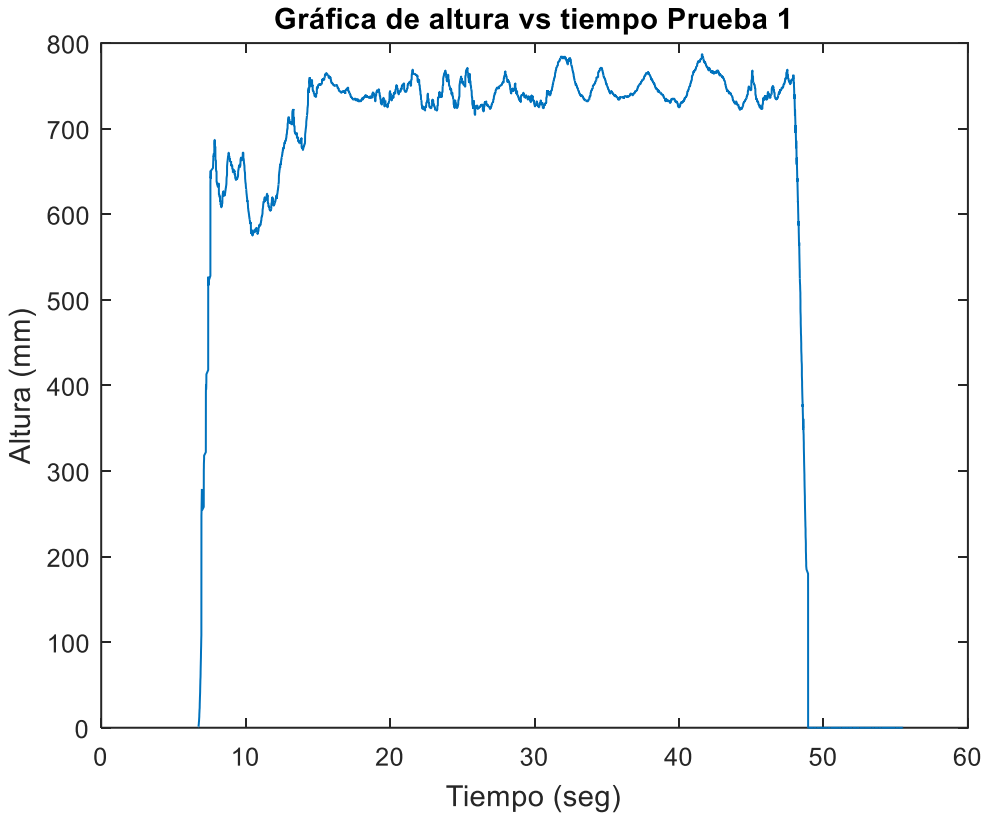


A continuación, se muestran las gráficas de altura con respecto al tiempo para cada una de las nueve pruebas realizadas, esto con el fin de realizar un análisis más completo de estas pruebas. Es importante aclarar que el eje de la altura en todas las figuras está dado en milímetros.

En la figura 37 se muestran los datos de altura versus tiempo para la Prueba 1. En esta figura se observan dos tramos en los cuales el valor de altura (eje y) es cero; el primer tramo corresponde al lapso de tiempo desde el cual se ejecuta el comando para la grabación de datos, hasta que el momento antes que el dron despegue, así como el segundo tramo corresponde al intervalo de tiempo desde que el dron aterriza hasta que es finalizado el comando de grabación de datos. Estos intervalos no se tienen en cuenta para el análisis debido a que no hacen parte del rango tiempo durante el cual se ejecuta el algoritmo, por lo que, se analiza sólo los momentos en los que la altura es diferente de cero. Para esta prueba la carga inicial de la batería es de 40% y la carga final es de 32%.

El promedio obtenido para la altura de esta prueba teniendo en cuenta sólo el intervalo de tiempo en el cual la altura es diferente de cero es de 718.91 mm.

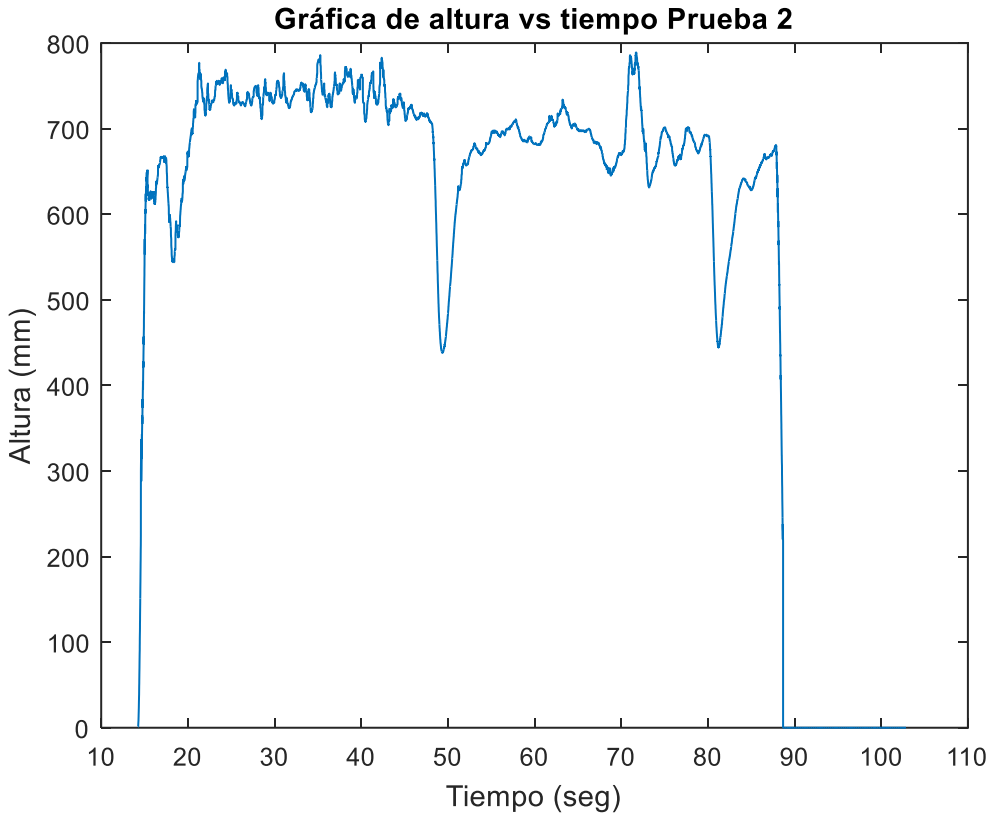
Figura 37. Gráfica de altura vs tiempo para Prueba 1.



De igual manera la figura 38 muestra los datos de altura respecto del tiempo para la Prueba 2. Al igual que en el análisis anterior, no se tienen en cuenta los intervalos de tiempo para los cuales la altura es cero. En la Prueba 2 la carga inicial de la batería es de 47% y la carga final es de 18%.

El promedio obtenido para la altura de esta prueba teniendo en cuenta sólo el intervalo de tiempo en el cual la altura es diferente de cero es de 680.41mm, esto se debe posiblemente a que el dron en esta prueba tuvo mayor tiempo de vuelo y, por tanto, mayor desgaste en sus baterías, ocasionando un promedio de altura menor para esta prueba respecto de la Prueba 1.

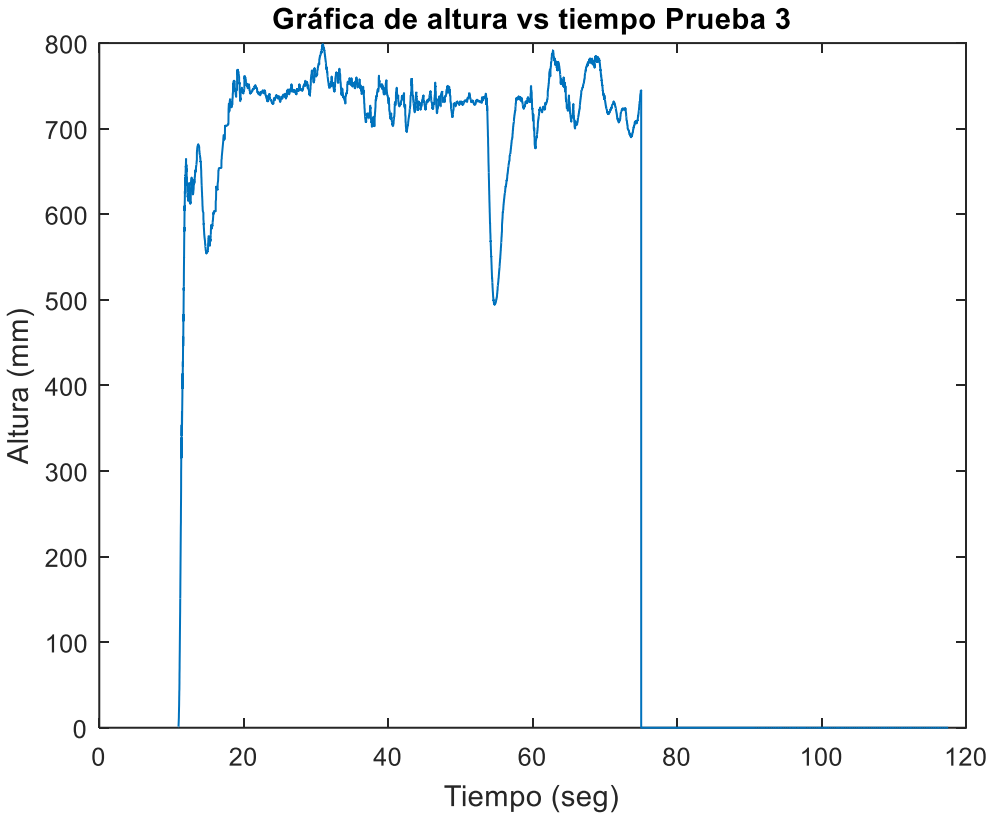
Figura 38. Gráfica de altura vs tiempo para Prueba 2.



Los datos de altura vs tiempo obtenidos para la Prueba 3 son mostrados en la figura 39. Se analiza el intervalo de tiempo en el cual la altura es diferente de cero, obteniendo que el promedio de altura para esta prueba es de 715.64 mm.

El porcentaje inicial de la carga de la batería para esta prueba es de 66%, y el porcentaje final es de 58%. Al comparar el promedio de altura de la Prueba 2 con el de la Prueba 3 se puede concluir que el porcentaje de carga de la batería tiene efecto sobre la altura directamente, ya que el porcentaje de carga final en la Prueba 2 es de 18% y el promedio de altura es menor al de la Prueba 3 el cual tiene un porcentaje final de 58%.

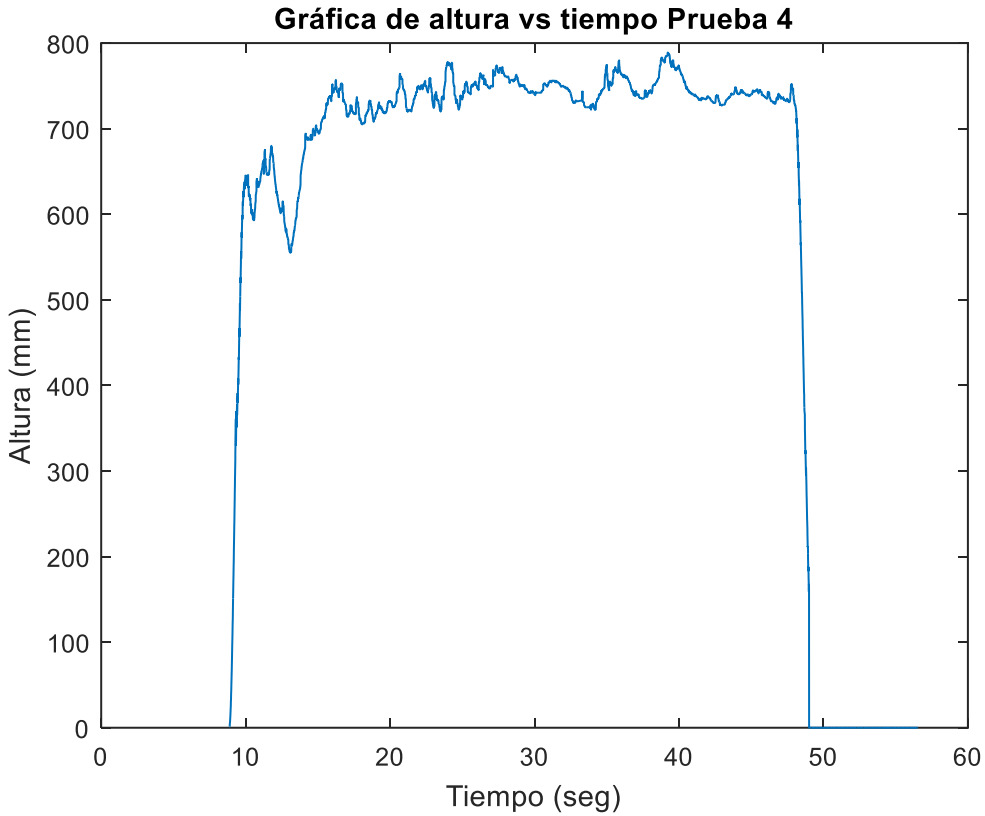
Figura 39. Gráfica de altura vs tiempo para Prueba 3.



La figura 40 contiene la curva que representa los datos de altura referente del tiempo para la Prueba 4. El análisis que se realiza abarca el rango de tiempo para el cual la altura es diferente de cero, obteniendo que el promedio de altura para esta prueba es de 710.22 mm.

El porcentaje inicial de la carga de la batería para esta prueba es de 64%, y el porcentaje final es de 41%. El promedio de altura de la Prueba 2 sigue siendo el menor respecto a los promedios de altura mostrados hasta este punto, teniendo una diferencia que oscila entre los 298 mm y 385 mm.

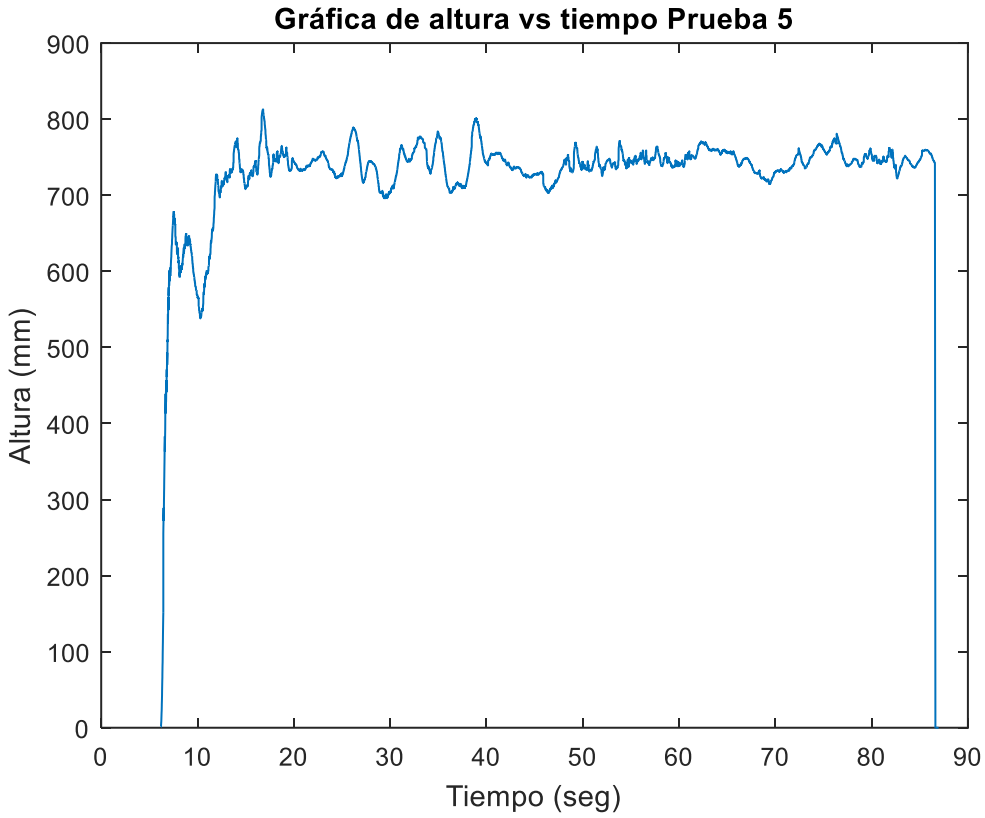
Figura 40. Gráfica de altura vs tiempo para Prueba 4.



En la figura 41 se presentan los datos de altura en relación con el tiempo para la Prueba 5. La carga inicial de la batería del dron para esta prueba es de 41% y la carga final es de 33%.

En este caso se realiza el análisis para el intervalo de tiempo desde el segundo 6.19 hasta el segundo 86.57, debido a que en este rango los valores de altura son diferentes de cero, obteniendo que el promedio de altura para esta prueba es de 730.60 mm. Comparando este promedio con respecto al menor obtenido hasta ahora se concluye que existe una diferencia de 50.19 mm.

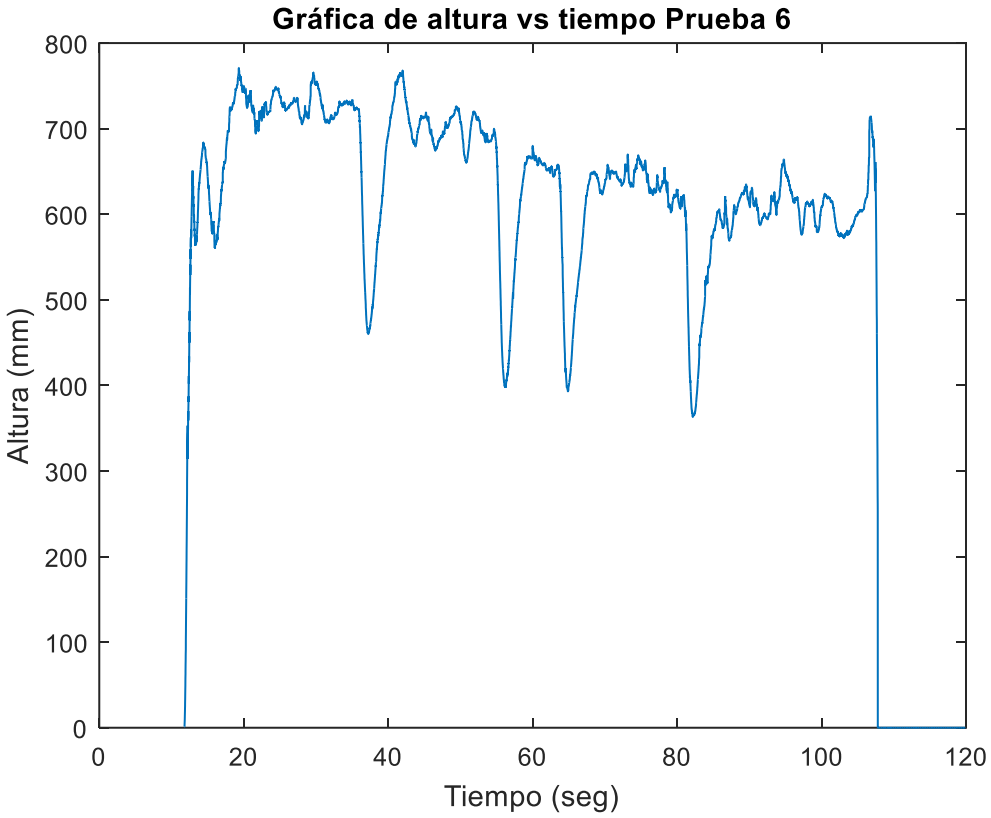
Figura 41. Gráfica de altura vs tiempo para Prueba 5.



Seguidamente se presenta la figura 42 la cual contiene la curva que representa los datos de altura versus tiempo para la Prueba 6. La carga inicial de la batería del dron para esta prueba es de 49% y la carga final es de 11%.

El intervalo de tiempo a ser analizado es comprendido desde el segundo 11.78 hasta el segundo 107.78, obteniendo que el promedio de altura para esta prueba es de 635.60 mm. Al revisar la figura 42 es de esperarse que el promedio de altura para esta prueba sea muy bajo debido a los valles que presenta la curva, esto se debe principalmente a la reducción del porcentaje de carga de la batería del dron durante la prueba.

Figura 42. Gráfica de altura vs tiempo para Prueba 6.



La gráfica que se muestra en la figura 43 la cual contiene la curva que representa los datos de altura versus tiempo para la Prueba 7. La carga inicial de la batería del dron para esta prueba es de 28% y la carga final es de 11%.

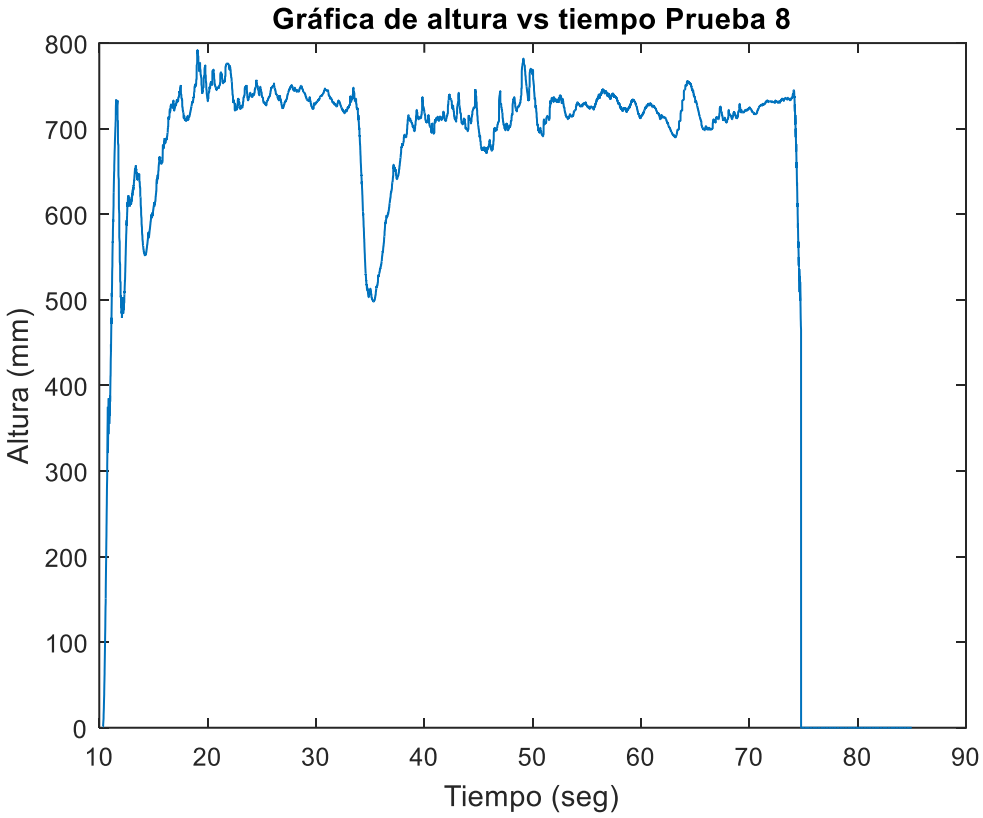
El análisis para esta prueba se realiza desde el segundo 8.98 hasta el segundo 111.77, obteniendo que el promedio de altura para esta prueba es de 740.43 mm. Debido al porcentaje de carga de las baterías tanto inicial como final en esta prueba es de esperarse que el promedio esté situado entre uno de los más bajos, sin embargo, la repercusión que tienen estos porcentajes de carga se ven reflejados directamente en la figura 4.9, ya que al momento de iniciar el despegue el dron gira en sentido contrario a lo indicado en el algoritmo, es decir, realiza un giro de -35°.

Figura 43. Gráfica de altura vs tiempo para Prueba 7.

La figura 44 es una representación gráfica de los datos obtenidos para la Prueba 8 de altura con respecto del tiempo. La carga inicial de la batería del dron para esta prueba es de 93% y la carga final es de 68%.

El análisis para esta prueba se realiza desde el segundo 11.7 hasta el segundo 74.77, obteniendo que el promedio de altura para esta prueba es de 706.94 mm. Es de esperarse que debido al alto porcentaje de carga de la batería para esta prueba sea la que mejor desempeño presenta, no obstante, para esta prueba y la siguiente (Prueba 9) la batería usada se había puesto a cargar muy poco tiempo, por lo que no está cargada en su totalidad y, por ende, el dato obtenido de porcentaje de batería corresponde a una carga falsa, es decir, que para estas dos últimas pruebas el porcentaje de batería real es menor al obtenido. Por esta razón, se presenta el valle aproximadamente en el segundo 35, porque la batería no tiene una carga del 93% sino que en realidad su porcentaje de carga esta entre 50% y 55%.

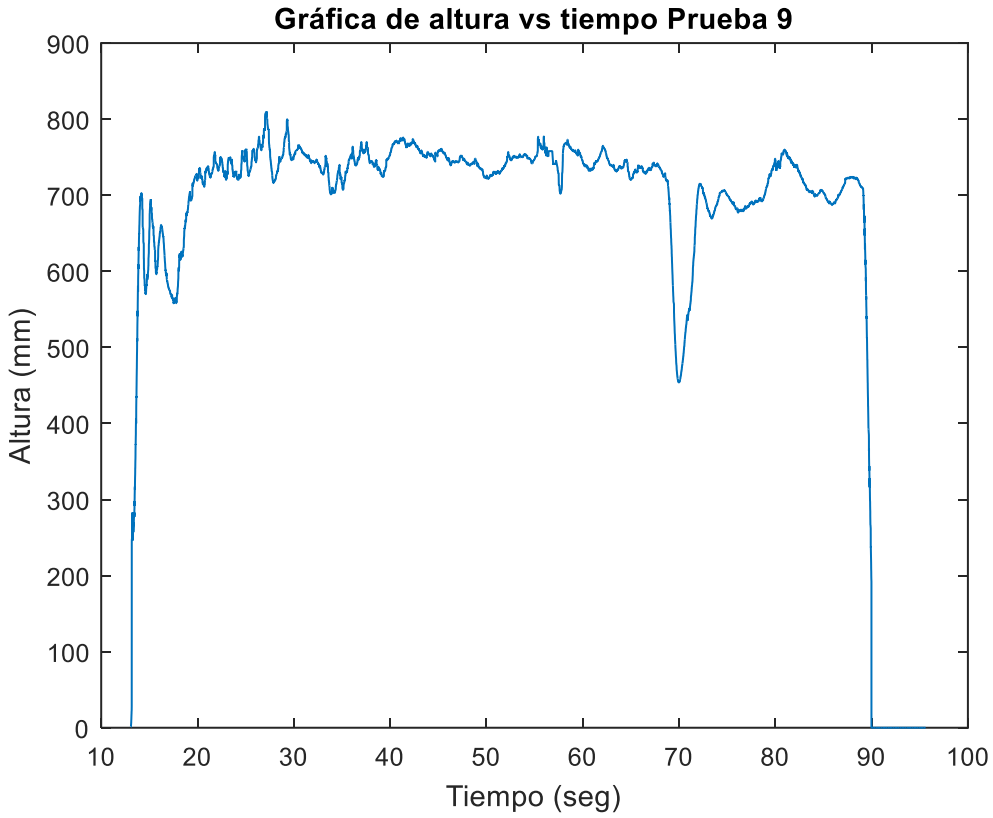
Figura 44. Gráfica de altura vs tiempo para Prueba 8.



En la figura 45 se observa la curva representativa de los datos de altura vs tiempo para la Prueba 9. El porcentaje inicial de carga de la batería es de 68% y el porcentaje final es de 54%.

El rango de tiempo sobre el cual se realiza el análisis inicia en el segundo 13.07 y finaliza en el segundo 89.98, obteniendo que el promedio de altura para esta prueba es de 712.52 mm. Al igual que en la Prueba 8 para esta prueba la batería usada se había puesto a cargar muy poco tiempo, por lo que no está cargada en su totalidad y, por ende, el dato obtenido de porcentaje de batería corresponde a una carga falsa. Por esta razón, se presenta el valle aproximadamente en el segundo 68, porque la batería no tiene una carga del 68% sino que en realidad su porcentaje de carga oscila alrededor del 30%.

Figura 45. Gráfica de altura vs tiempo para Prueba 9.



Este algoritmo es una representación simple y sencilla de todo un conjunto de herramientas útiles, que sirven de base para llevar a cabo futuros proyectos, el cual es uno de los resultados de mayor importancia, porque de este modo se incentiva a las personas interesadas en este campo a hacer uso de las nuevas tecnologías y así mismo incurrir en el desarrollo de estas, así como a su vez lograr tener un aporte académico a la región.

Al realizar las diferentes pruebas del sistema se tiene como resultado que la distancia entre el dron y la estación en tierra es de gran importancia, debido a que esto logra influir en el correcto funcionamiento y, por tanto, en el desempeño del sistema, ya que al alejarse demasiado el dron, la imagen que se obtiene desde este a la base en tierra se congela, y por tanto no es posible el procesamiento de la imagen en tiempo real, y a su vez no se logra realizar la detección del obstáculo y por ende no es posible que se tomen las acciones de control para realizar una evasión correcta. Lo anterior se puede apreciar en las figuras 46 y 47; en esta última figura se observa al dron encerrado en un óvalo de color negro detectando al obstáculo encerrado en un círculo de color rojo, sin embargo, la imagen observada en la estación en tierra en la figura 46 ya se encuentra congelada debido a la distancia de alcance de la red wifi creada por el AR.Drone 2.0.

Figura 46. Imagen congelada del AR.Drone 2.0 por exceder alcance de su red wifi.

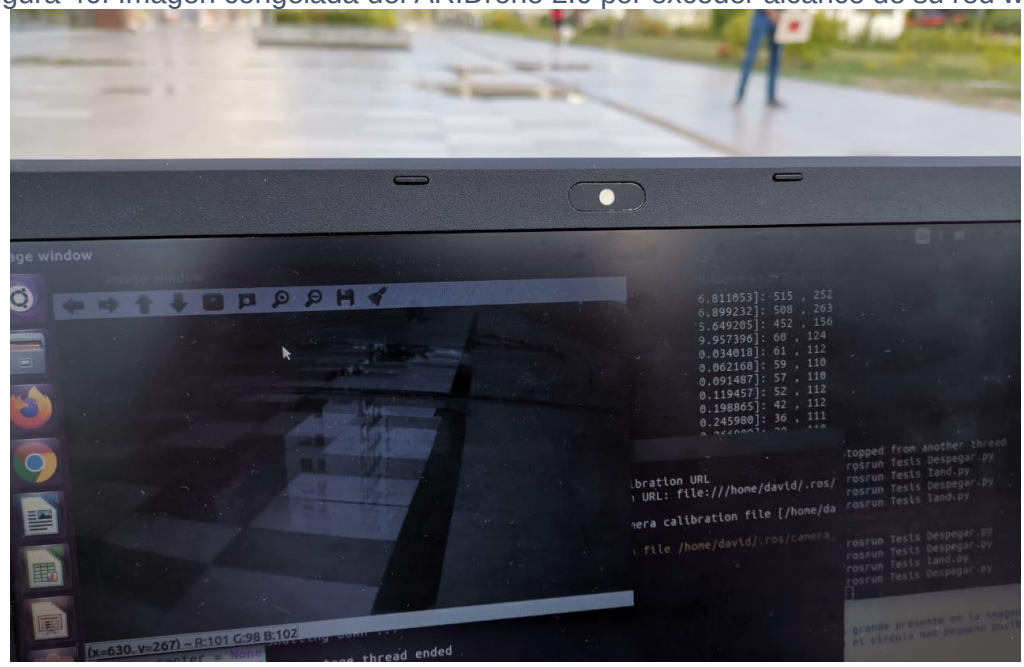
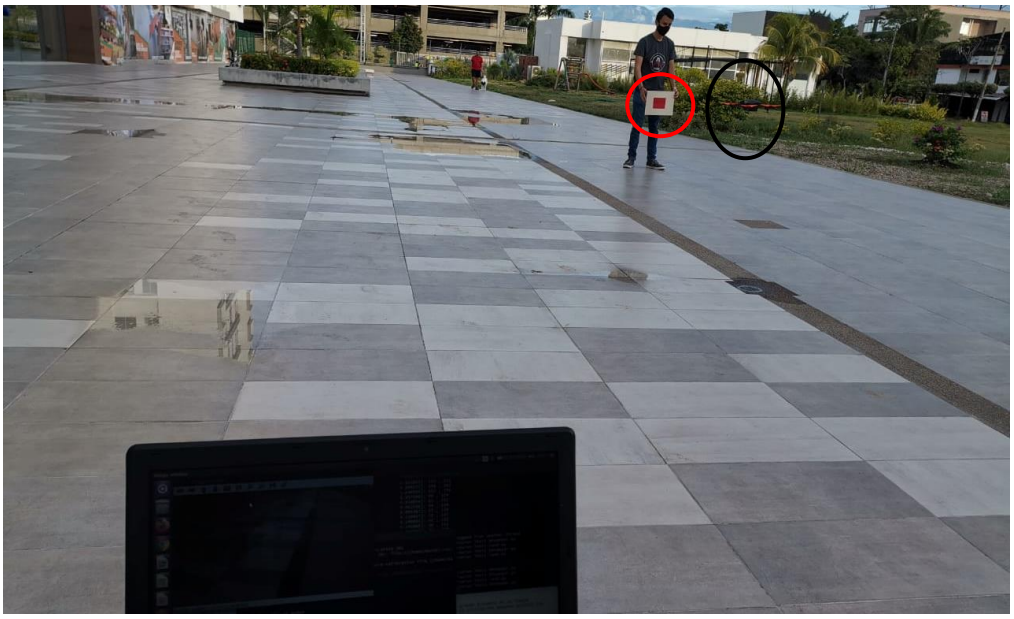


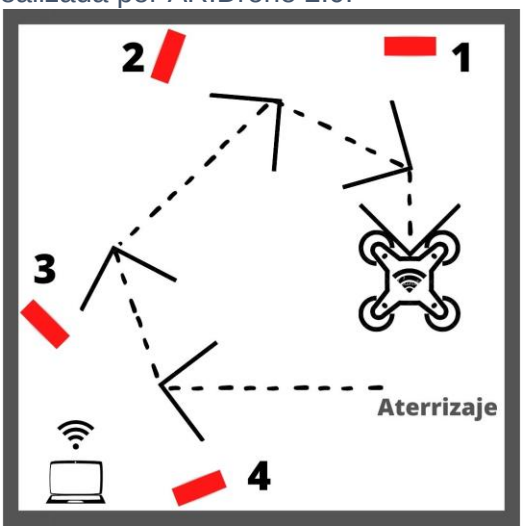
Figura 47. Escenario completo de imagen congelada del AR.Drone 2.0 por exceder alcance de su red wifi.



Finalmente, al llevar el algoritmo al campo y realizar algunas pruebas se puede apreciar el correcto funcionamiento del sistema, es decir, que logra percibir un objeto de color rojo, y según las pruebas realizadas con el obstáculo de mayor tamaño es capaz de reconocerlo hasta a una distancia más o menos de dos metros y medio, para posteriormente evadirlo mediante las acciones control implementadas en el algoritmo tales como la variación de la velocidad angular hasta que el obstáculo se encuentre fuera del rango visual del dron para modificar así la velocidad lineal y seguir un trayectoria recta, hasta que detecta otro obstáculo de color rojo en su camino. Es preciso aclarar que como medida

de seguridad implementada para el sistema se establecen unos tiempos límites que al ser sobrepasados hacen posible que se genere un aterrizaje seguro del dron. Al estar el dron girando durante 17.5 segundos seguidos, este aterriza de manera satisfactoria en las pruebas realizadas. De igual manera al continuar en línea recta durante este mismo periodo de tiempo el dron realiza su aterrizaje de forma correcta.

Figura 48. Trayectoria realizada por AR.Drone 2.0.



En la figura 49 se evidencia la simulación del escenario de detección de obstáculo con AR.Drone 2.0 mediante Gazebo, el cual es una herramienta que permite simular entornos en tres dimensiones para estimar el comportamiento de un robot en un entorno virtual. En el costado izquierdo de la figura se observa al dron y al obstáculo frente a él, mientras que en el costado derecho de la figura se evidencia la detección del obstáculo, así como en la parte inferior las coordenadas x e y de su centro.

Figura 49. Simulación de escenario de detección de obstáculo con AR.Drone 2.0 mediante Gazebo.

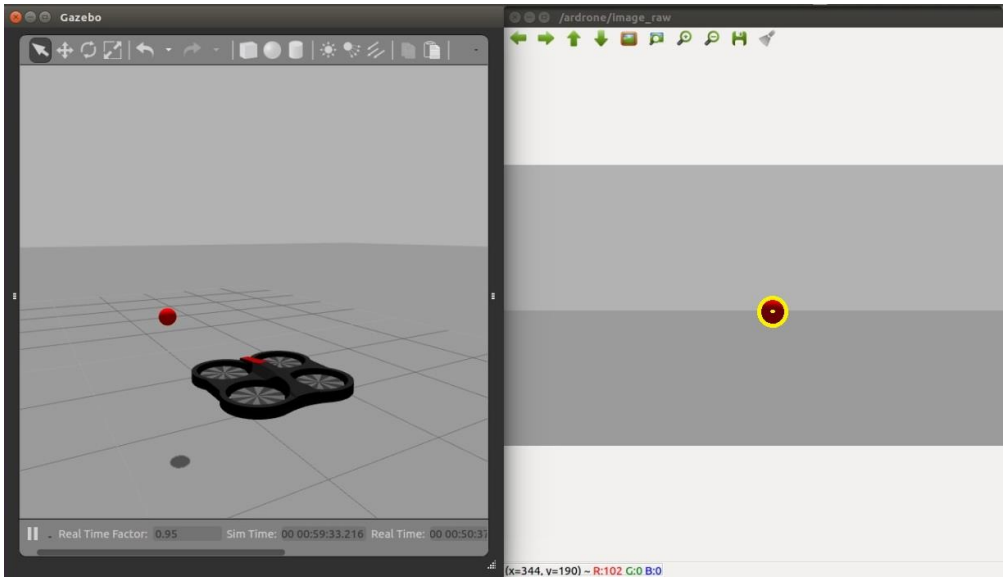
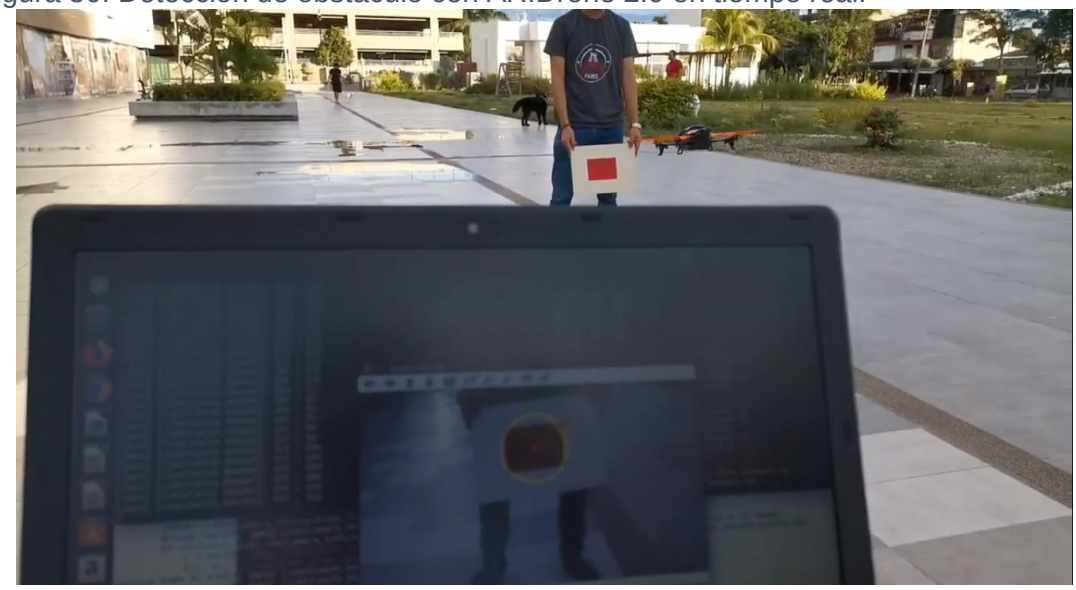


Figura 50. Detección de obstáculo con AR.Drone 2.0 en tiempo real.



La figura 50 ratifica la detección de obstáculos en tiempo real usando el AR.Drone 2.0. Esta figura se puede dividir en dos escenarios, el primero de ellos es el dron con el obstáculo ubicado en frente de él, y el segundo la estación en tierra obteniendo la imagen de la cámara frontal del dron con su respectiva detección.

Con lo dicho anteriormente es posible concluir que los resultados obtenidos fueron los esperados, logrando implementar satisfactoriamente un sistema de detección y evasión de obstáculos haciendo uso de herramientas de fuente abierta bajo condiciones específicas.

5. CONCLUSIONES

- Se logró desarrollar un algoritmo de detección y evasión de obstáculos de color rojo utilizando el lenguaje de programación Python en conjunto con el entorno de desarrollo Robot Operating System (ROS) y la librería para procesamiento de imágenes OpenCV, teniendo la gran ventaja de ser asequible a cualquier persona ya que las herramientas mencionadas anteriormente son de código abierto.

- Se encontraron varios factores que influyeron negativamente en el desempeño del algoritmo, los cuales son externos a este, tales como el estado del dron (aspas, motores y baterías) que dificultaron ver el rendimiento esperado del algoritmo realizado.

- Es de suma importancia tener en cuenta la cobertura (alcance) de la red WiFi creada por el AR.Drone 2.0 al momento de ser energizado, ya que al sobrepasar este alcance se pierde la comunicación entre el dron y la estación en tierra (computador), generando que se pierdan los paquetes de la transmisión del video y esta manera no se pueda seguir realizando las acciones control para el dron.

- La estructura de ROS ofrece una gran cantidad de ventajas entre las cuales cabe resaltar la adquisición de datos en tiempo real del robot en estudio (AR.drone 2.0), como la obtención de imágenes de las cámaras, datos de sensores, así como también su gran comunidad existente que provee información de gran utilidad en la web.

- Se utilizaron dos métodos para la obtención de imágenes en tiempo real (mediante ROS realizando la suscripción al topic *"/ardrone/front/image_raw"* y mediante OpenCV utilizando la función *"VideoCapture")* encontrando que las imágenes obtenidas mediante OpenCV tienen un tiempo de retardo considerable, mientras que las imágenes por medio de ROS si se obtenían en tiempo real, verificando una de las ventajas de este entorno de desarrollo.

- Al tener que definir unos rangos de colores en formato HSV es de suma importancia tener en cuenta la luminosidad del entorno ya que la última característica de este formato (Value) indica qué tanto brillo debe tener el color para ser detectado, por tanto, no todos los rojos serán detectados como obstáculos.

- Este proyecto de grado sienta las bases en un tema muy poco conocido en la región que es de gran utilidad como lo es el uso de ROS para la implementación de algoritmos en robots, debido a que el algoritmo presentado proporciona información útil y funciona como guía para futuros proyectos o investigaciones, además de que la lógica del algoritmo

funciona con mucha similitud en otros robots no solo aéreos sino también terrestres y acuáticos.

6. RECOMENDACIONES

- Con el fin de mejorar la eficacia del sistema se recomienda utilizar una herramienta alternativa en cuanto a la detección de los obstáculos como las redes neuronales, debido a que estas pueden reconocer objetos tanto de diferentes formas y colores, gracias a su proceso de entrenamiento.

- Dependiendo de la aplicación en la que se vaya a incurrir, se recomienda ajustar las acciones de control (velocidades angulares y lineales), así como también los tiempos para el aterrizaje seguro.

- Al momento de realizar las pruebas del sistema, se recomienda estar seguro de que las baterías se encuentren con más del 80% de carga, debido que a la alta cantidad de corriente que necesitan los motores del dron generan un desgaste alto en un corto tiempo en las baterías, limitando así el tiempo de uso, el número de ejecuciones y el rendimiento del sistema.

Bibliografía

Bradski, Gary. (2000). The OpenCV Library. Dr. Dobb's Journal of Software Tools.

Mihelich, P., Bowman, J. 2012. cv_bridge: package to convert between ROS Image messages and OpenCV images.

Monajjemi, M., et al. 2012. ardrone_autonomy: a ROS driver for AR.Drone 1.0 & 2.0.

ROS.org. Converting between ROS images and OpenCV images (Python). [en línea]. http://wiki.ros.org/cv_bridge/Tutorials/ConvertingBetweenROSImagesAndOpenCVImagesPython. 24 de marzo de 2020

ROS.org. Topics. [en línea]. http://wiki.ros.org/Topics. 24 de marzo de 2020.

ROS.org. Writing a Simple Publisher and Subscriber (Python). [en línea]. http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28python%29. 24 de marzo de 2020.

SOLANO, Gabriela. DETECCION DE COLORES en OPENCV [en 4 Pasos] – Parte1 [video]. Ecuador: Youtube. (2 de marzo de 2019). 7:11 minutos. [Consultado: 24 de marzo de 2020]. Disponible en https://www.youtube.com/watch?v=giwtDYcIXKA.

Stanford Artificial Intelligence Laboratory et al. (2018). Robotic Operating System. Disponible en https://www.ros.org.

ANEXOS

ALGORITMO DE DETECCIÓN DE OBSTÁCULOS DE COLOR ROJO

```
#!/usr/bin/env python2

import sys
import rospy
import cv2
from std_msgs.msg import String, Empty, Float32
from sensor_msgs.msg import Image
from collections import deque
from cv_bridge import CvBridge, CvBridgeError
import numpy as np

class image_converter:

  def _init(self):
    self.image_pub = rospy.Publisher("image_topic_2",Image,queue_size=10)
    self.pub=rospy.Publisher('Coordenadas_Y',Float32,queue_size=1)
    self.pub1=rospy.Publisher('Coordenadas_X',Float32,queue_size=1)
    self.pub3=rospy.Publisher('Aterrizaje',Float32,queue_size=1)
    self.bridge = CvBridge()
    self.image_sub = rospy.Subscriber("/ardrone/front/image_raw",Im-
age,self.callback)

  def callback(self,data):
    Lower = np.array([0, 150, 20],np.uint8)
    Upper = np.array([1, 255, 255],np.uint8)
    Lower1 = np.array([178, 150, 20],np.uint8)
    Upper1 = np.array([179, 255, 255],np.uint8)
    pts = deque(maxlen=64)
    try:
        cv_image = self.bridge.imgmsg_to_cv2(data, "bgr8")
        hsv = cv2.cvtColor(cv_image, cv2.COLOR_BGR2HSV)
        mask1 = cv2.inRange(hsv, Lower, Upper)
        mask2 = cv2.inRange(hsv, Lower1, Upper1)
        mask = cv2.add(mask1,mask2)
        mask = cv2.erode(mask, None, iterations=2)
        mask = cv2.dilate(mask, None, iterations=2)
        cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTER-
NAL,cv2.CHAIN_APPROX_SIMPLE)[-2]
        center = None
        if len(cnts) > 0:
            c = max(cnts, key=cv2.contourArea)
            ((x, y), radius) = cv2.minEnclosingCircle(c)
            M = cv2.moments(c)
            center = (int(M["m10"] / M["m00"]), int(M["m01"] / M["m00"]))
```

```
            a=(int(M["m01"] / M["m00"]))
            b=(int(M["m10"] / M["m00"]))
            rospy.loginfo('%i %s %i', int(M["m10"] / M["m00"]),",",int(M["m01"] /
M["m00"]))
            rate.sleep()
            if radius > 10:
                cv2.circle(cv_image, (int(x), int(y)), int(radius),(0, 255, 255), 2)
                cv2.circle(cv_image, center, 5, (0, 0, 255), -1)

        else:

            a=(int(-1))
            b=(int(-2))

    except CvBridgeError as e:
      print(e)

    cv2.imshow("Image window", cv_image)
    cv2.waitKey(3)

    try:
      self.image_pub.publish(self.bridge.cv2_to_imgmsg(cv_image, "bgr8"))
      self.pub1.publish(b)
    except CvBridgeError as e:
      print(e)
def main():
  ic = image_converter()
  try:
    rospy.spin()
  except KeyboardInterrupt:
    print("Shutting down")
  cv2.destroyAllWindows()

if _name_ == '_main_':
    rospy.init_node('image_converter', anonymous=True)
    global rate
    rate = rospy.Rate(20) # Taza o velocidad de 20Hz
    main()
```

ALGORITMO DE EVASIÓN DE OBSTÁCULOS DE COLOR ROJO

```
#!/usr/bin/env python
import rospy
from std_msgs.msg import Float32,Empty
from geometry_msgs.msg import Twist
msg=Twist()

class Detector():


    def _init_(self):

        self.coordenadax = None
        self.coordenaday = None

    def coordenadax_callback(self, data):

        global pub2
        pub3=rospy.Publisher('ardrone/land',Empty,queue_size=10)
        pub2=rospy.Publisher('/cmd_vel',Twist,queue_size=10)
        self.coordenadax = data
        if data.data == -2:

            global i,j
            rospy.loginfo('NO HAY OBSTACULO')
            msg.angular.z = 0
            msg.linear.x = 0.01
            i=0
            j=j+1
        else:

            rospy.loginfo('SI HAY OBSTACULO')
            msg.linear.x = 0
            msg.angular.z = 0.1
            i=i+1
            j=0

        pub2.publish(msg) #Realiza la publicacion de las velocidades
        rospy.loginfo('%s %i','Temporizador de giro: ',i)
        rospy.loginfo('%s %i','Temporizador de avance: ',j)
        if i > 350:
            rospy.loginfo('ATERRIZANDO')
            pub3.publish(Empty())
        if j > 350:
            rospy.loginfo('ATERRIZANDO')
            pub3.publish(Empty())
```

```
    def coordenaday_callback(self, data):

        self.coordenaday = data


if _name_ == '_main_':
    rospy.init_node('listener')

    detector = Detector()
    global i,j
    i=0
    j=0
    rospy.Subscriber('Coordenadas_X', Float32, detector.coordenadax_callback)
    rospy.Subscriber('Coordenadas_Y', Float32, detector.coordenaday_callback)

    rospy.spin()
```