DESDE 2013
https://repository.uaeh.edu.mx/revistas/index.php/icbi/issue/archive
Pädi Boletín Científico de Ciencias Básicas e Ingenierías del ICBI
ISSN: 2007-6363

Publicación Semestral Pädi Vol. 10 No. Especial 6 (2022) 33–41

Sistema de visión artificial y vuelo autónomo para un cuadricóptero en ROS 2
Autonomous flight and computer vision system for a quadcopter in ROS 2

A. Ramirez-Linarez [a,*], M. Torres-Rivera [a]

[a] *Universidad Aeronáutica en Querétaro, 76278, Colón, Querétaro, México.*

Resumen

Se propone e implementa un framework de software in the loop enfocado a la simulación de un algoritmo de detección de compuertas por medio de visión artificial, basado en operaciones morfológicas para segmentación de color, y un algoritmo de misión de vuelo con seguimiento de trayectoria a partir de waypoints, para un cuadricóptero autónomo virtual. Además, se integra un conjunto de software libre de última generación para validar el funcionamiento de los algoritmos propuestos dentro de un circuito de vuelo desarrollado en un ambiente de simulación 3D. Se observa que el desempeño del algoritmo de visión artificial es aceptable bajo condiciones ideales y a distancias cortas, y que el cuadricóptero es capaz de completar el circuito de vuelo utilizando la metodología propuesta para la gestión de trayectoria.

Palabras Clave: visión artificial, misión de vuelo, waypoints, software in the loop, cuadricóptero

Abstract

A software in the loop framework is proposed and implemented, focused on the simulation of a gate detection algorithm via computer vision, based on morphological operations for color segmentation, and a flight mission algorithm with trajectory tracking via waypoints, for a virtual autonomous quadcopter. In addition, a state-of-the-art open-source software set is integrated to validate the operation of the proposed algorithms within a flight circuit developed in a 3D simulation environment. It is observed that the performance of the artificial vision algorithm is acceptable under ideal conditions and at short distances, and that the quadcopter is capable of completing the flight circuit using the proposed methodology for trajectory management.

Keywords: computer vision, flight mission, waypoints, software in the loop, quadcopter

1. Introducción

Las carreras de drones se han convertido en un deporte bastante popular en los últimos años. Resulta increíble pensar que, haciendo uso de única y exclusivamente una cámara de vuelo, los pilotos son capaces de abstraer la información necesaria del ambiente para ejecutar maniobras de vuelo con alta precisión y agilidad.

A partir de lo anterior, la comunidad científica, en especial aquella dedicada al campo de la robótica, se ha visto bastante interesada en sustituir al piloto humano por únicamente unidades de cómputo y componentes electrónicos; es decir, hoy en día existe la tendencia a dotar de autonomía el vuelo de estos vehículos aéreos no tripulados, de tal manera que, a partir de computadoras de placa única, sensores y algoritmos sofisticados de visión artificial, odometría y gestión y control de trayectorias de vuelo, se pueda obtener el mismo desempeño de vuelo que el otorgado por un piloto humano, e incluso, en algún punto, superarlo de manera significativa.

Sumando a lo ya expresado, se han creado una serie de instituciones y eventos con el fin de financiar, potenciar y motivar el desarrollo tecnológico en este campo emergente, dando lugar a lo que se conoce como carreras y competencias de drones autónomos. Dentro de los eventos o competencias más significativas se encuentra el Autonomous Drone Racing (ADR) (Moon et al., 2017), llevado a cabo cada año en la Conferencia Internacional de Sistemas y Robots Inteligentes (IROS, por sus siglas en inglés), el AlphaPilot Challenge (APC) (Foehn et al., 2020), organizado por Lockheed Martin en colaboración con Nvidia y la Liga de Carrera de Drones (DRL); y Game of Drones (GOD) (Madaan et al., 2020), gestionada por Microsoft para la Conferencia Anual de Sistemas de Procesamiento de Información Neuronal (NeurIPS) de 2019.

*Autor para correspondencia: 5296@soyunaq.mx
Correo electrónico: 5296@soyunaq.mx (Áxel Ramírez-Linarez), moises.torres@unaq.mx (Moisés Torres-Rivera)

Historial del manuscrito: recibido el 15/04/2022, última versión-revisada recibida el 01/07/2022, aceptado el 14/07/2022, publicado el 30/11/2022. **DOI:**https://orcid.org/10.29057/icbi.v10iEspecial6.9021

Los eventos mencionados representan un punto de encuentro a nivel internacional que ha permitido dirigir los esfuerzo e intelectos alrededor del mundo, a la propuesta de soluciones, ya sea de forma parcial o general, a los retos enfrentados en el campo de estudio, de tal forma que buscan poner a prueba las implementaciones propuestas por los participantes, dentro de circuitos y retos con distintas características.

Dicho lo anterior, dentro de los ambientes de simulación utilizados por los equipos participantes para entrenar y validar sus algoritmos, existen algunas soluciones bastante específicas, que se enfocan exclusivamente en la simulación de drones autónomos, tales como: FlightGoggle desarrollado por (Guerra et al., 2019); Flightmare creado por (Song et al., 2020) o AirSim (Shah et al., 2017). Sin embargo, los ambientes anteriores requieren un hardware con poder computacional bastante alto, de tal manera que es necesario contar con una tarjeta gráfica dedicada para su ejecución; además, pese a que FlightGoggle y Flightmare ofrecen compatibilidad con ROS (Robot Operating System), solo lo hacen con ROS 1. En este trabajo se utiliza Gazebo en conjunto con ROS 2 para la implementación de los algoritmos desarrollados, por lo que se propone un ambiente de simulación abierto que utiliza tecnologías de última generación, con requerimientos de hardware básicos. A continuación se mencionan los módulos de software que integran el framework propuesto.

1.1. ROS

De acuerdo con su sitio oficial (Open-Robotics, 2021b), ROS (del inglés, Robot Operating System) es un conjunto de herramientas y librerías de software para robótica desarrolladas por Open Robotics (Open-Robotics, 2021a) bajo el paradigma de software libre u open-source. Este entorno de trabajo destaca por contener algoritmos de última generación y herramientas de desarrollo avanzadas, que permiten la creación, implementación y reutilización de código para todo tipo de proyectos de robótica.

Si bien es cierto que en los últimos años ROS ha adquirido una importante relevancia, su uso se ha visto limitado a aplicaciones con fines académicos y no ha podido despegar en el sector industrial debido a las limitaciones con las que cuenta, dentro de ellas cabe mencionar el hecho de no estar diseñado para trabajar con sistemas en tiempo real, además de carecer de estándares de seguridad para llevar a cabo certificaciones de este tipo. A partir de lo anterior, ROS 2 es una nueva versión del framework con el que se busca ampliar la gama de aplicaciones de ROS, dotándolo de herramientas para solventar los problemas mencionados.

1.2. OpenCV

OpenCV (del inglés, Open Source Computer Vision Library) es una librería multiplataforma de código abierto que fue diseñada para realizar procesamiento de imágenes en tiempo real, contiene una cantidad considerable de algoritmos de visión artificial, por lo que, es considerada un estándar para implementar aplicaciones en donde es necesario utilizar visión por computadora; está escrita en C/C++ y puede ser instalada en Linux, Windows y Mac OS X. Además, cuenta con interfaces para Python, Ruby, MATLAB y otros lenguajes de programación. (OpenCV-Team, 2021).

1.3. ArduPilot

ArduPilot es definido por sus autores (ArduPilot-Dev-Team., 2022) como un sistema de piloto automático confiable, versátil y de código abierto, capaz de funcionar en diversos vehículos, tales como: multicópteros, helicópteros, aeronaves de ala fija, botes, submarinos, rovers y más. El firmware de ArduPilot permite desarrollar sistemas de navegación para vehículos autónomos no tripulados prácticamente de cualquier tipo, ha adquirido una relevancia tal que, es usado ampliamente en vehículos desarrollados por instituciones y corporaciones de gran importancia, tales como la NASA, Intel, Boeing y una gran cantidad de universidades y colegios de alrededor del mundo.

1.4. PymavLink

De acuerdo con su documentación oficial (Lorenz Meier, 2021), MAVLink es un protocolo para comunicación con drones y sus componentes; sigue un patrón de diseño híbrido de publicación-subscripción y comunicación punto-a-punto, en donde la trama de datos es enviada mediante el primer paradigma y los protocolos de configuración, como misiones y parámetros de vuelo, son enviados mediante el segundo.

En otras palabras, el uso del protocolo MAVLink es necesario para entablar comunicación con la simulación del controlador de vuelo provisto por Ardupilot, y de esta forma obtener los parámetros de vuelo de dron simulado así como el envío de comandos de vuelo.

Por otro lado, Pymavlink es una implementación en Python del protocolo MAVLink, lo cual posibilita el envío de comandos de vuelo y recepción de datos de sensores a través de un script en Python.

1.5. Gazebo

Gazebo es un software de simulación que ofrece la posibilidad de simular de forma precisa y eficiente, conjuntos de robots en ambientes complejos de interiores y exteriores. Además, ofrece un motor de físicas robusto, gráficos de alta calidad e interfaces gráficas y programáticas convenientes. (Open-Robotics, 2014).

2. Arquitectura propuesta

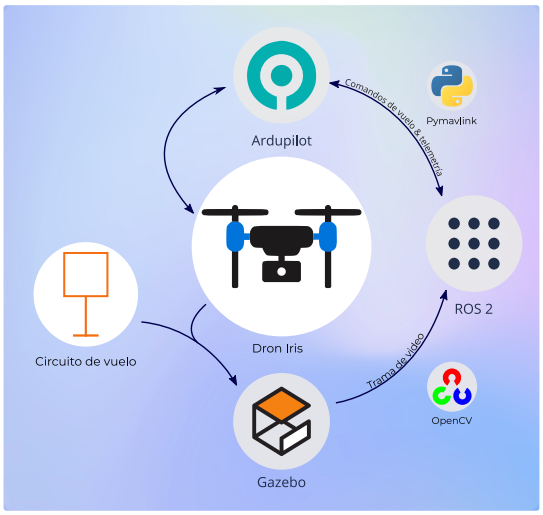

Figura 1: Diagrama de alto nivel de la arquitectura propuesta.

La figura 1 muestra un esquema en donde se observan el software y librerías que componen el sistema propuesto, así como la interacción que se tiene entre estos.

De forma más específica, el sistema está conformado por 3 grandes módulos de software que pueden ser utilizados de forma independiente, pero existe la posibilidad de integrarlos a partir de plugins.

Por un lado el circuito de vuelo así como el modelo y las físicas del cuadricóptero fueron desarrollados en Gazebo. En cuanto al control del cuadricóptero, se lleva a cabo por medio del framework de software in the loop provisto por ArduPilot, este firmware está pensado para implementarse en hardware físico, sin embargo, permite simular el piloto automático, de tal manera que existe la posibilidad de acceder a la telemetría del vehículo y enviarle comandos e instrucciones por medio de protocolo de comunicación de MAVLink, como si se tratase de un sistema real.

Por último, para la gestión de los procesos (nodos) correspondientes a los algoritmos de visión artificial y misión de vuelo se utilizó ROS 2.

Ahora bien, retomando el esquema de la figura 1, la interacción entre los 3 módulos de software se realiza a partir del modelo del cuadricóptero; este se encuentra desarrollado en un archivo de tipo SDF (Simulation Description Format), un formato especial que sirve para describir las características físicas y visuales de un robot, y dentro del código fuente del cuadricóptero se encuentra especificado el uso de 2 plugins que permiten la comunicación entre los módulos de software.

ArduPilot cuenta con un plugin que permite que la simulación del piloto automático interactúe de forma directa con el modelo en el ambiente de Gazebo; es decir, los comandos e instrucciones enviadas al firmware se ven reflejadas dentro de la simulación de Gazebo, y a la vez, el estado del modelo del cuadricóptero corresponde con la telemetría leída por el piloto automático. Por otro lado, los comandos de vuelo son generados por medio de un nodo de ROS, el cual hace uso de la librería Pymavlink para comunicarse con el firmware de ArduPilot a través de un script en Python 3.

Finalmente, el modelo del cuadricóptero cuenta con una cámara simulada debajo de su chasis, y es posible acceder a los fotogramas captados por esta a partir de un plugin que permite la comunicación entre Gazebo y ROS, de la forma que la trama de video puede ser recibida en un nodo de ROS, en donde es procesada con el algoritmo de visión artificial implementado con OpenCV en C++.

2.1. Características del hardware utilizado

Previo al desarrollo realizado es importante indicar las características del hardware donde se realizaron las simulaciones y que se enlistan en la tabla 1.

Parámetro	Descripción
Procesador	Intel Core i3-7100U Dual-core 2.40 GHz
Memoria RAM	12 GB DDR4
Disco duro	1 TB Toshiba HDD
Coprocesador de gráficos	Intel HD Graphics 620

Tabla 1: Características de hardware usadas en la implementación.

3. Ambiente de simulación

La figura 2 presenta un diagrama en donde se especifica la estructura del circuito elaborado, en él se pueden apreciar las cotas con las distancias presentes entre cada una de las compuertas, así como el orden del recorrido realizado por el dron. Cabe destacar que el modelo de compuerta que se seleccionó para el circuito de vuelo, fue el utilizado en las distintas competencias del IROS.

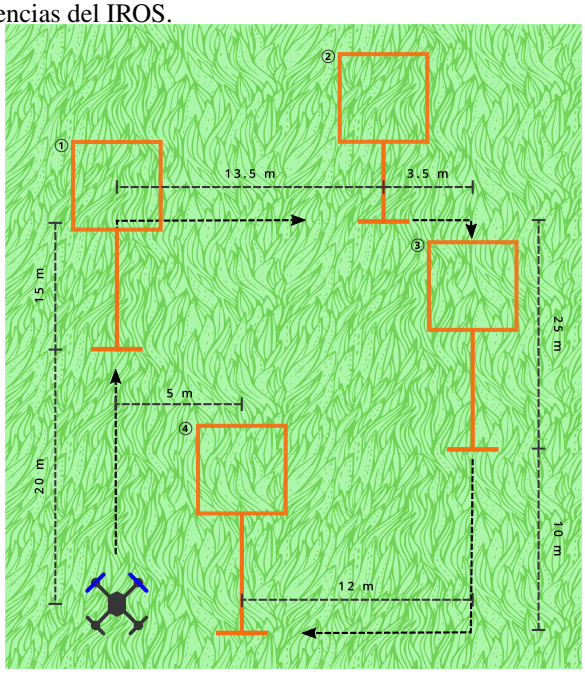

Figura 2: Esquema detallado del circuito de vuelo elaborado en simulación.

A partir del esquema antes mencionado, la elaboración del circuito dentro de la simulación se llevó a cabo utilizando algunos modelos ya elaborados por la comunidad de Gazebo. Para el terreno y el modelo de la compuerta se utilizaron los modelos elaborados por (Rojas-Perez and Martinez-Carranza, 2020).

Adicionalmente, la figura 3 presenta el modelo 3D del dron Iris utilizado, el cual fue provisto por (Johnson, 2018). Cabe destacar que la principal diferencia entre este modelo y el modelo incluido en la simulación de demostración del plugin de Ardupilot para Gazebo, es la orientación de la cámara. En la simulación de demostración, la cámara se encuentra en dirección hacia el suelo, mientras que en el modelo mostrado en la figura, la cámara enfoca hacia el frente de la dirección de vuelo del dron.

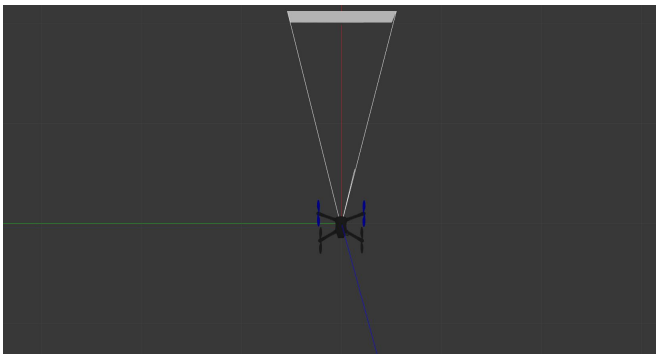

Figura 3: Vista superior del modelo del dron utilizado.

Por último, la figura 4 muestra una captura del proyecto creado para el circuito de vuelo implementado, en ella se presenta un panorama general del circuito de vuelo, con las cuatro compuertas del circuito, el dron Iris y el modelo de un edificio; este último se incluyó para fines del algoritmo de visión artificial.

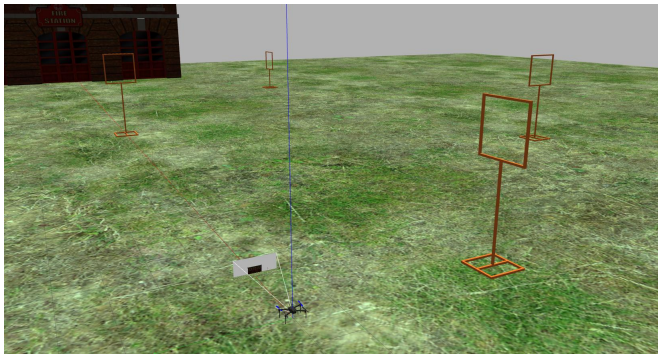

Figura 4: Captura del ambiente de simulación implementado.

4. Sistema de visión artificial

El tipo de compuertas seleccionadas para la formación del circuito de vuelo fue un modelo similar al utilizado en las competencias del IROS, con un característico color naranja. Ahora bien, debido a que el algoritmo de visión artificial está basado en la detección de color y no utiliza redes neuronales, el aspecto más importante para adecuar el algoritmo es encontrar el rango de color en la escala HSV para nuestro objeto.

Para lograr sintonizar lo mejor posible la escala de color, se utilizó como base un rango de color adecuado para detectar el color naranja, pues este es el color principal del objeto de interés; sin embargo, el color de las compuertas es uno de la infinidad de tonalidades derivadas del naranja, por lo que fue necesario ajustar más el rango para que, de ser posible, el algoritmo fuera capaz de detectar exclusivamente las compuertas y no diera falsos positivos con objetos con una tonalidad naranja similar. Por otro lado, dentro del circuito de vuelo el único objeto con una tonalidad similar al color naranja es el edificio que se encuentra detrás de la primera compuerta, por lo que, se están asumiendo condiciones prácticamente ideales para la detección de las compuertas.

A partir de la escala base se realizaron las pruebas del algoritmo de detección sobre una colección de imágenes, las cuales se obtuvieron directamente del ambiente de simulación. Entonces, para detectar el color naranja se definió el rango *[H:5-25; S:75-255; V:25-255]*, el cual fungió como escala base, de tal forma que los parámetros del modelo de color se fueron variando para reajustar el rango, hasta que se observó que el algoritmo era capaz de segmentar correctamente las compuertas en las imágenes prueba.

La figura 5 muestra los resultados obtenidos en una de las imágenes utilizadas para la sintonización, obteniéndose así el rango de color siguiente:

$$H : 5 - 25;\ S : 99 - 255;\ V : 77 - 171$$

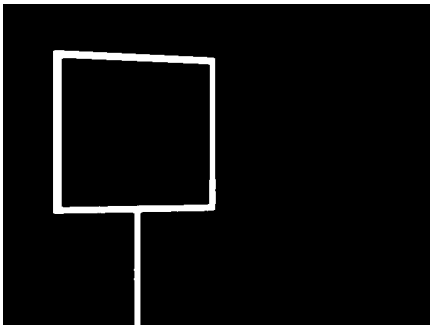

Figura 5: Resultados de detección de compuertas.

Como se puede observar, la detección logra aislar con gran precisión la compuerta del fondo, el suelo y el fragmento visible de pala de uno de los rotores del dron.

4.1. Algoritmo de visón artificial

Para complementar los resultados anteriores la estructura de la lógica implementada puede observarse en el algoritmo

1. *Adquirir* imagen
2. *Convertir* espacio de color: RGB → BGR
3. *Convertir* espacio de color: BGR → HSV
4. *Aplicar* el método del valor umbral sobre la imagen
5. *Aplicar* una operación de apertura sobre la imagen umbralizada
6. *Aplicar* una operación de cerradura sobre la imagen umbralizada
7. *Mostrar* la imagen original y la imagen procesada

Dado que OpenCV ofrece una gran variedad de funciones y herramientas para la ejecución de algoritmos de visión artificial, el programa resultante es bastante compacto en cuanto a líneas de código, pues se utilizaron funciones nativas de la librería. El código desarrollado para esta etapa puede ser consultado en el repositorio oficial del presente trabajo (Ramírez-Linarez, 2022), dentro del directorio de recursos.

5. Misión de vuelo

El algoritmo de seguimiento de trayectoria se trabajó en pequeñas etapas, en donde se probó una funcionalidad distinta del piloto automático, de tal forma que al final, todas las etapas se deben de integrar en una sola. Los códigos fuente pertenecientes a cada etapa se encuentran en el respectivo repositorio de GitHub, perteneciente al presente trabajo. La tabla 2 muestra la relación que existe entre cada etapa y su script en el repositorio de GitHub.

Etapa	Código fuente
Establecimiento de la conexión entre Pymavlink y ArduPilot	listen.py
Configuración del modo de vuelo del dron	mode.py
Armado de motores y despegue	takeoff.py
Seguimiento de trayectoria con waypoints	movement.py
Implementación de la misión de vuelo	mission.py

Tabla 2: Relación entre etapa y su código fuente.

A continuación se muestran los resultados obtenidos en cada una de las etapas mencionadas.

5.1. *Conexión entre Pymavlink y ArduPilot*

La primera consistió en entablar la comunicación entre el script de Pymavlink y el SIL (Software in the Loop) de ArduPilot, lo cual se logró con la librería Pymavlink y la creación de un objeto que sirve como intermediario entre el script y el piloto automático simulado.

Una vez que se entabló la conexión entre ambas partes es posible tener acceso a toda clase de datos de vuelo, entre ellos la actitud del dron. Cabe mencionar que, los mensajes que recibe Pymavlink por parte de ArduPilot contienen una trama de datos de gran longitud, y debido a que se creó un objeto de Python para entablar la conexión, es posible acceder a un dato en específico como se accede a un atributo en la instancia de una clase. En este caso, se decidió imprimir en consola el dato del ángulo de roll de dron con el fin de comprobar que la comunicación se dio de forma exitosa, lo cual se observa en la figura 6.

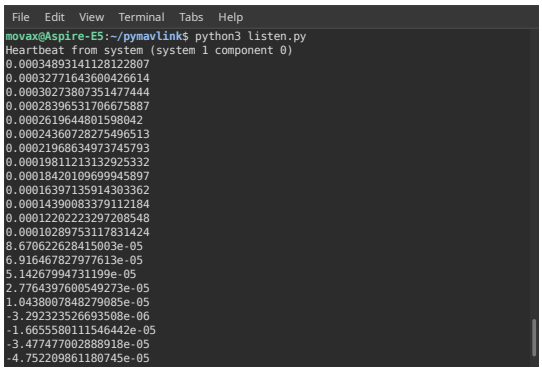

Figura 6: Datos obtenidos a partir de la conexión entre ArduPilot y Pymavlink.

5.2. *Configuración de modo de vuelo*

De acuerdo con la documentación oficial de ArduPilot (Johnson, 2022), el Arducopter cuenta con una gran variedad de modos de vuelo, no obstante, en este trabajo se utilizan únicamente dos modos, *Guided* y *Land*; el primero permite enviar comando de vuelo de forma directa al piloto automático a través de mensajes, mientras que el segundo ejecuta una rutina de aterrizaje para el dron.

Previo a la secuencia de despegue del dron, es necesario configurar el modo de vuelo en Guided, pues al iniciar el SIL de ArduPilot, este inicializa con el modo de vuelo *Stabilize*, de tal forma que el piloto automático no es capaz de recibir comandos de vuelo ingresados directamente por el usuario. Esto resulta ser un inconveniente, pues es necesario que el dron reciba ejecute los comandos de vuelo para realizar el seguimiento de trayectoria.

El método utilizado para establecer el modo de vuelo fue *set_mode_send*, el cual recibe solamente 3 parámetros, el atributo del sistema del objeto de la conexión, el comando para establecer el modo de vuelo, y el código de identificación del modo de vuelo especificado. La figura 7 muestra el resultado que se obtuvo al ejecutar el código desarrollado para esta subsección, el mensaje de ejecución del script como tal, y por otro lado, la aceptación del comando y el cambio de modo de vuelo reflejado en la terminal de ArduPilot.

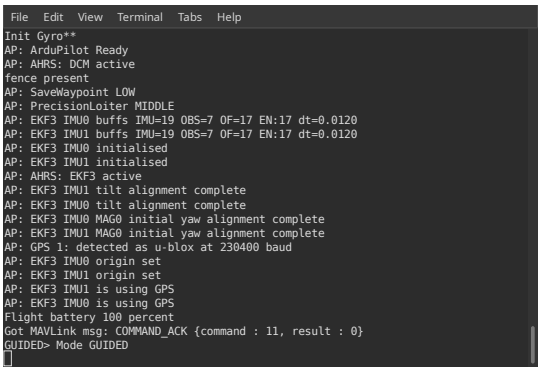

Figura 7: Ejecución del script para la configuración del modo de vuelo.

5.3. *Secuencia de despegue*

El script perteneciente a esta subsección no cuenta con la etapa de la selección del modo de vuelo, pues es necesario ejecutar el programa en el momento en el que los filtros del sistema han sido inicializados, y si se corre el script al mismo tiempo que se inicia el SITL, el comando de despegue es rechazado. Para ejecutar el programa, el usuario tiene que cambiar el modo de vuelo de forma manual dentro de la terminal de ArduPilot, de esta forma se asegura que el sistema está listo para recibir la instrucción de despegue.

Los comandos de vuelo utilizados en esta etapa fueron *MAV_ CMD_COMPONENT_ARM_DISARM* para el armado de los motores y *MAV_CMD_NAV_ TAKEOFF* para la orden de despegue. La figura 8 muestra el comportamiento observado en la simulación tras la ejecución del script con la secuencia de despegue, al haber alcanzado la altura indicada en el comando de vuelo.

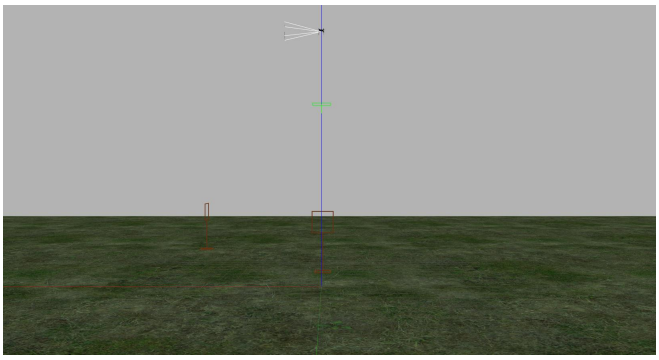

Figura 8: Comportamiento de la simulación ante el comando de despegue.

Si bien la figura 8 muestra de manera acertada la correcta ejecución de la secuencia de despegue, a simple vista resulta difícil decir con exactitud si el dron alcanzó, o no, la altura deseada. Debido a lo anterior, se recopilaron los datos correspondientes a la posición y velocidad del dron durante su ascenso, a partir de uno de los mensajes de MAVLink que permite obtener acceso a esta información manejada por el piloto automático. En consecuencia, la figura 9 es una gráfica que muestra la altura del dron durante su despegue, de tal forma que es posible apreciar que, en efecto, el dron logra alcanzar la altitud

deseada durante su despegue con un ligero sobre paso, aproximadamente a los 13 segundos después de la ejecución del comando de vuelo.

Figura 9: Respuesta en la altura para el comando de despegue.

Por último, la gráfica de la figura 10 se obtuvo a partir de la diferencia entre la referencia de altura y la respuesta del sistema; es decir, el comportamiento del error de altura. Se puede apreciar una pendiente negativa bastante pronunciada en los segundos 4 y 12, que corresponden, precisamente, al lapso de tiempo en el que el dron asciende a la altura definida. Entonces, también es posible corroborar que el dron llegó a la referencia en el segundo 13 aproximadamente, pues es cuando el error llega a 0.

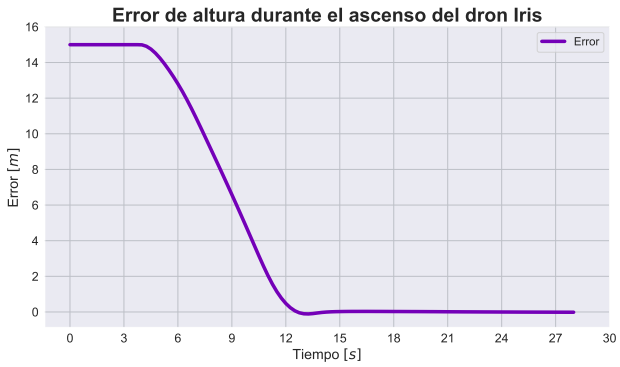

Figura 10: Gráfica de error de altura durante ascenso.

5.4. *Seguimiento de trayectoria*

La última etapa que se tuvo que realizar fue la del seguimiento de trayectoria por medio de comandos de vuelo directos, en forma de waypoints. La prueba realizada fue enviando dos waypoint los cuales fueron definidos de tal manera que se buscó que el dron cruzara por las dos primeras compuertas del circuito de vuelo, dichos waypoint fueron [35,0,3] y [35, 17, 2]. La secuencia pensada para la ejecución de esta prueba fue, primero hacer ascender el dron a una altura de 10 m y después indicarle su trayectoria de vuelo a partir de los waypoints especificados.

Para evitar que el piloto automático ignore el primer waypoint, se utilizó un mensaje de MAVLink definido como *NAV_CONTROLLER_OUTPUT*, el cual publica información relacionada sobre el comando de vuelo enviado al piloto automático, dentro de la cual se encuentra la distancia faltante para que el dron llegue al waypoint especificado al momento de realizar la lectura del mensaje. Entonces, la instrucción para el seguimiento del segundo waypoint es enviada al piloto automático hasta que el mensaje especifica que la distancia restante para llegar al primer waypoint es igual a cero.

De manera similar a la subsección anterior, la figura 11 permite observar de manera rápida y práctica si el comportamiento del dron fue el adecuado; sin embargo, el análisis del desempeño del dron se logra a partir de los datos de vuelo recopilados.

La figura 12 muestra una gráfica en donde se describe la trayectoria de vuelo seguida por el dron, en ella se especifica la ubicación de las compuertas con base en el recorrido descrito.

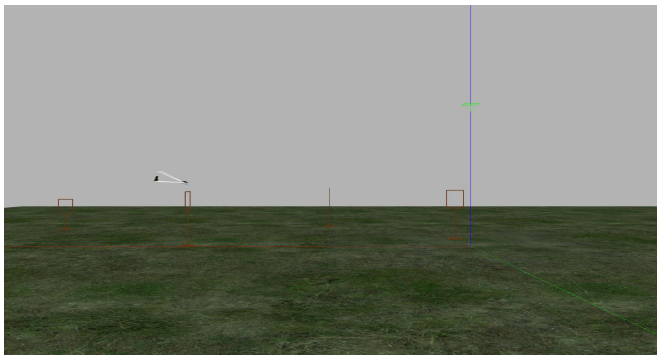

Figura 11: Prueba de seguimiento de trayectoria.

Figura 12: Recorrido realizado por el dron.

5.5. *Misión de vuelo*

Con la etapa anterior se culminaron las pruebas por módulos; en esta subsección se describe el proceso realizado para integrar todas las etapas descritas en un solo módulo o secuencia de instrucciones.

Debido a que se trabajó con un paradigma de módulos, en donde cada etapa descrita corresponde a un módulo y/o funcionalidad específica, la unificación de todo lo anteriormente descrito resultó ser un proceso bastante lineal y práctico, en la mayor parte de las etapas se re-utilizó el mismo código desarrollado para su respectivo script. Por otro lado, en esta etapa se buscó implementar de forma completa la misión de vuelo para el dron, de tal manera que fuera capaz de cruzar a través de las 4 compuertas que componen el circuito de vuelo.

Dicho lo anterior, a manera de recapitulación, se describen los módulos en el orden en el que fueron implementados:

1. Establecimiento de comunicación entre Pymavlink y ArduPilot
2. Cambio de modo de vuelo de stabilize a guided
3. Armado de motores
4. Envío de orden de despegue
5. Envío de comandos de vuelo para seguimiento de waypoints

Adicionalmente, el script de misión de vuelo busca eliminar la necesidad de que el usuario tenga que introducir comando de vuelo de forma manual en la terminal de ArduPilot; es decir, solo es necesario ejecutar el script y el dron será capaz de ejecutar todas las etapas descritas.

La figura 13 cuenta con la ubicación de las cuatro compuertas del circuito de vuelo, así como el sentido de vuelo que siguió el dron durante su recorrido. A partir de esto, se logra apreciar que los waypoints fueron definidos de tal manera que el dron describiera una trayectoria rectangular uniforme, por lo tanto, la trayectoria recorrida por cada waypoint corresponde a cada una de las aristas del rectángulo y la transición entre cada uno está representado por los vértices de este.

Figura 13: Seguimiento de misión de vuelo completa.

6. Implementación del sistema de detección y vuelo con ROS 2

En esta última sección de resultados se desglosa el proceso llevado a cabo para la implementación de los nodos del algoritmo de visión artificial y el seguimiento de trayectoria.

El paradigma para la conversión entre un programa de programación estructura a un nodo de ROS es sencillo en mucho de los casos, pues la consideración más importante que se tiene que realizar es que, al ser un nodo de ROS, el programa se estará ejecutando como un proceso dentro del sistema operativo, de tal manera que su ejecución se hará de forma indefinida a manera de ciclo.

6.1. *Visión artificial*

La figura 14 presenta el desempeño general del algoritmo de visión artificial. La imagen muestra los fotogramas de la trama de video transmitida con la cámara simulada durante el vuelo del cuadricóptero; se logra apreciar que el algoritmo es capaz de aislar el contorno de la compuerta del fondo del fotograma.

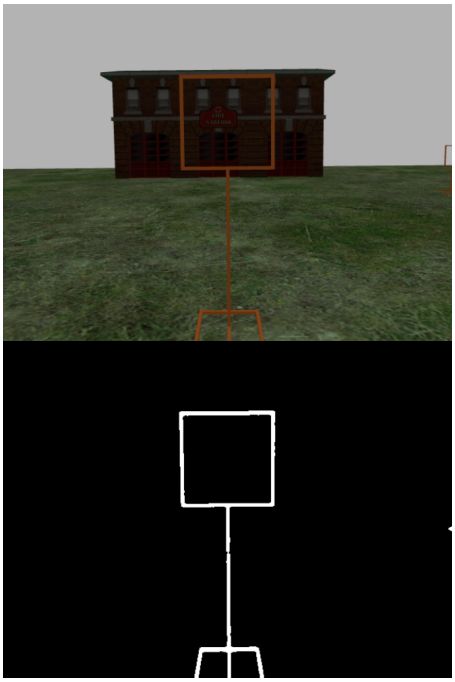

Figura 14: Detección de la primera compuerta en el circuito de vuelo.

Por otro lado, la figura 15 muestra más fotogramas captados por la cámara del cuadricóptero. El objetivo de esta serie de fotogramas es presentar de manera más clara el proceso de detección durante el vuelo y dar a conocer las áreas de oportunidad del algoritmo implementado.

A partir de lo anterior, es posible apreciar que, pese a que la compuerta no. 3 se encuentra dentro del campo de visión de la cámara, el algoritmo no es capaz de detectar la compuerta, sino hasta que el cuadricóptero se acerca más a esta, de tal forma que cuando la compuerta es detectada, esta ya no se encuentra en su totalidad dentro del campo de visión de la cámara. Entonces, es evidente que la detección no es del todo eficiente y requiere distancias cortas con respecto a las compuertas para ejecutarse de manera correcta.

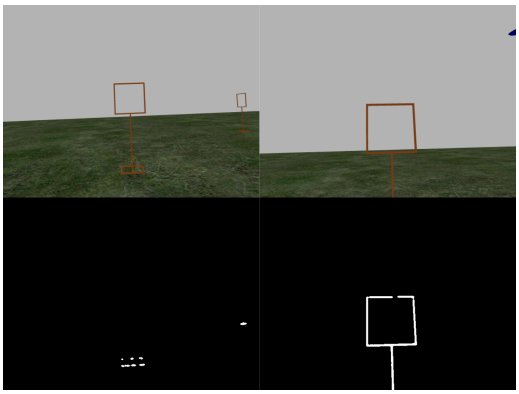

Figura 15: Serie de fotogramas con proceso de detección de compuertas.

6.2. *Misión de vuelo*

Con respecto al algoritmo de misión de vuelo, se tuvo que solventar la problemática derivada de la asincronía de los procesos, pues es necesario esperar a que el firmware de ArduPilot inicialice el conjunto de sensores y telemetría antes de ejecutar

comandos de vuelo, de no ser así, el comando de vuelo simplemente es ignorado; entonces, si el nodo dedicado al algoritmo de seguimiento de trayectoria es ejecutado a la par que el SITL de ArduPilot, es muy posible que los comandos de vuelo no sean ejecutados. Para dar solución a lo anterior, y asegurar que el sistema se haya inicializado por completo se implementaron dos acciones específicas:

1. Leer el estado del sistema y espera a que todos los sensores hayan sido inicializados
2. Esperar el tiempo necesario para que el sistema arme los motores y se pueda proceder a la fase del despegue

El primer punto se logró leyendo uno de los mensajes publicados por el piloto automático; *SYS_STATUS* el cual entrega una serie de datos acerca del estado del sistema, entre ellos los estados de los sensores dentro de la simulación, de tal forma que la lectura de este parámetro devuelve un entero. A partir de las pruebas realizadas, se observó que cuando todos los sensores del sistema han sido inicializados el parámetro devuelve un valor de 1382128815, por lo que se implementó un ciclo que hiciera la lectura del parámetro hasta que este tuviera el valor deseado. Una vez que los sensores han sido inicializados, se puede proceder con el resto de la secuencia de vuelo.

Ahora, previo al despegue es necesario indicarle al piloto automático que prepare los motores del dron para la secuencia del dron; sin embargo, esta acción conlleva un poco de tiempo, y si se envían comandos de vuelo de forma apresurada, el piloto automático rechaza las instrucciones, es por ello que se utilizó el método *motors_armed_wait()* para pausar la secuencia de vuelo hasta que el sistema indique que los motores han sido armados de forma satisfactoria.

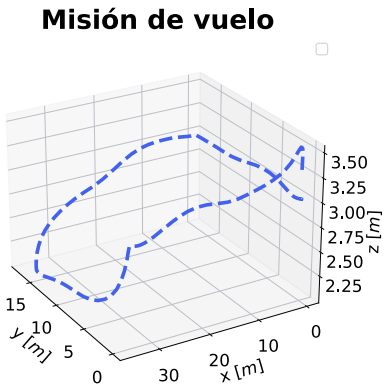

Figura 16: Trayectoria seguida por el cuadricóptero con la implementación en ROS.

Hecho lo anterior, la lógica propuesta para la misión de vuelo se pudo portear a un nodo de ROS sin ningún otro percance. La figura 16 muestra la gráfica generada a partir de los datos recopilados por medio de ArduPilot, en ella es posible observar la trayectoria seguida por el cuadricóptero tras la ejecución del sistema de vuelo. Además, cabe mencionar que el dron fue capaz de completar el circuito de vuelo en aproximadamente 35 segundos, a una velocidad promedio de vuelo de $3{,}15\frac{m}{s}$ u $11{,}34\frac{k}{h}$. Por otro lado, se anexa un enlace a un video en donde se muestra el desempeño del dron durante el seguimiento de trayectoria vuelo; https://youtu.be/YRieuULaRi0.

6.3. *Integración del sistema en ROS 2*

Por último, la integración del ambiente no solo conllevó el conjuntar los nodos creados en las secciones anteriores, sino que, también se buscó la forma de implementar el SIL de ArduPilot y la simulación de Gazebo a partir de un archivo de lanzamiento en ROS.

La figura 17 muestra el diagrama de nodos que se genera al momento de realizar el lanzamiento del sistema con ROS. Se puede observar que, en su conjunto, se desarrolló un sistema sencillo con dos nodos; el nodo designado como *waypoints_node* es el nodo encargado de ejecutar la secuencia de vuelo, mientras que el *computer_vision_node* corresponde al nodo en donde se implementa el algoritmo de detección de compuerta. Adicionalmente, es posible observar otro nodo designado con el nombre de *camera_controller*, el cual es creado por el plugin que se encarga de entablar la comunicación entre ROS y Gazebo, de hecho, se puede observar el topic por donde se envían los fotogramas generados por la simulación hacia el nodo de visión artificial, el cual está designado como *cam_image_raw*. Por último, en el diagrama no se observa que el nodo de seguimiento de trayectoria se comunique con ningún otro nodo; sin embargo, es importante recalcar que, con base en la arquitectura de la figura 1, este nodo se comunica con ArduPilot y debido a que ArduPilot es un software independiente (no se ejecuta por medio de un nodo de ROS). Entonces, el hecho de que la comunicación entre ambos procesos no se vea reflejada en el diagrama no quiere decir que esta interacción no se esté llevando a cabo, pues los resultados presentados confirman que el dron fue capaz de completar la trayectoria definida debido a la interacción entre el *waypoints_node* y ArduPilot. Finalmente, se anexa un video en donde se muestra una vista general del comportamiento del sistema implementado; https://youtu.be/TTaQOsrOBwA.

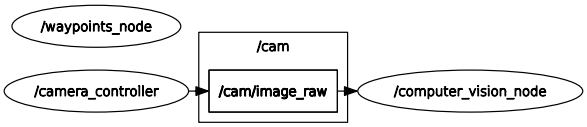

Figura 17: Red de nodos resultante.

A manera de recapitulación, se puede mencionar que se logró realizar el porteo de los algoritmos bases del sistema de vuelo a ROS, de tal forma que el algoritmo de visión artificial y el de seguimiento de trayectoria se ejecutaron como nodos de ROS, además, también fue posible implementar el framework de SIL de ArduPilot desde una archivo de lanzamiento de ROS. Por otro lado, no se pudo realizar la integración del ambiente de simulación dentro del archivo de lanzamiento, pues la simulación no se inicializaba de forma correcta al implementarla de esta manera; sin embargo, lo anterior no quiere decir que la integración final de los módulos de software haya fallado, sino que, de momento el ambiente de simulación tiene que ser ejecutado de forma individual para que interactúe de forma correcta con el resto de programas del sistema de vuelo.

7. Conclusiones

Se desarrolló un sistema visión artificial y misión de vuelo de un cuadricóptero autónomo basado en una ambiente virtual, que hace uso de tecnologías de última generación, con requerimientos de hardware básicos, completamente gratuito y de código abierto.

Con respecto al algoritmo de visión artificial, se implementó una metodología basada en un proceso de segmentación de color con operaciones morfológicas básicas para la detección de las compuertas. El algoritmo fue capaz de realizar el reconocimiento de las 4 compuertas que conforman el circuito de vuelo bajo condiciones ideales, en donde se descartó los cambio de iluminación, clima no favorable, etc.; Asimismo, los resultados mostraron que el algoritmo solo es capaz de detectar las compuertas a partir de 1 m de distancia de estas.

Por otro lado, en cuanto al desempeño del algoritmo de seguimiento de trayectoria, la metodología utilizada permitió que el cuadricóptero fuera capaz de completar el circuito de vuelo pasando a través de todas y cada una de las compuertas.

Está claro que los resultados obtenidos en este proyecto abren las puertas a una serie de mejoras que permitan explotar de mejor manera lo aquí presentado, dentro de los cuales se pueden mencionar la implementación de un algoritmo de visión artificial basado en redes neuronales convolucionales, el uso de la librería de MAVROS en lugar de PymavLink para el envío de comandos de vuelo y recepción de información sobre el vehículo o la implementación de la arquitectura en hardware físico.

Finalmente, se puede destacar que la integración entre los módulos de software desarrollados fue adecuada, y el sistema propuesto pudo ser implementado con base en la arquitectura mencionada.

Referencias

ArduPilot-Dev-Team. (2022). Ardupilot. https://ardupilot.org. Accessed: 2022-02-17.

Foehn, P., Brescianini, D., Kaufmann, E., Cieslewski, T., Gehrig, M., Muglikar, M., and Scaramuzza, D. (2020). Alphapilot: Autonomous drone racing. *arXiv preprint arXiv:2005.12813*.

Guerra, W., Tal, E., Murali, V., Ryou, G., and Karaman, S. (2019). FlightGoggles: Photorealistic sensor simulation for perception-driven robotics using photogrammetry and virtual reality. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE.

Johnson, E. (2018). Iq simulations. https://github.com/Intelligent-Quads/iq_sim.

Johnson, E. (2022). Flight modes. https://ardupilot.org/copter/docs/flight-modes.html.

Lorenz Meier, Andreas Antener, t. (2021). Mavlink developer guide. https://mavlink.io/en/. Accessed: 2022-01-04.

Madaan, R., Gyde, N., Vemprala, S., Brown, M., Nagami, K., Taubner, T., Cristofalo, E., Scaramuzza, D., Schwager, M., and Kapoor, A. (2020). Airsim drone racing lab. In *NeurIPS 2019 Competition and Demonstration Track*, pages 177–191. PMLR.

Moon, H., Sun, Y., Baltes, J., and Kim, S. J. (2017). The iros 2016 competitions. *IEEE Robotics and Automation Magazine*, 24(1):20–29.

Open-Robotics (2014). Gazebo: Robot simulation made easy. http://gazebosim.org/.

Open-Robotics (2021a). Open robotics. https://www.openrobotics.org/.

Open-Robotics (2021b). Ros 2 docuumentation: foxy documentation. https://docs.ros.org/en/foxy/index.html.

OpenCV-Team (2021). Opencv: Introduction. https://opencv.org. Accessed: 2022-02-17.

Ramírez-Linarez, A. (2022). Axolotsil. https://github.com/MOVAX19/AxolotSIL.

Rojas-Perez, L. O. and Martinez-Carranza, J. (2020). Deeppilot: A cnn for autonomous drone racing. *Sensors*, 20(16):4524.

Shah, S., Dey, D., Lovett, C., and Kapoor, A. (2017). Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and Service Robotics*.

Song, Y., Naji, S., Kaufmann, E., Loquercio, A., and Scaramuzza, D. (2020). Flightmare: A flexible quadrotor simulator. In *Conference on Robot Learning*.