UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS
INSTITUTO DE INVESTIGACIÓN EN CIENCIAS BÁSICAS Y APLICADAS
CENTRO DE INVESTIGACIÓN EN CIENCIAS

ROBOT AÉREO SEGUIDOR DE LINEA USANDO VISIÓN ARTIFICIAL

T E S I S

QUE PRESENTA:

IGNACIO GABRIEL LOAEZA RODRIGUEZ

Para obtener el Grado de
LICENCIADO EN CIENCIAS
ÁREA TERMINAL EN COMPUTACIÓN

Director de la Tesis:
Jorge Alberto Fuentes Pacheco

Sinodales:
Juan Manuel Rendón Mancha (Presidente)
Jorge Alberto Fuentes Pacheco (Secretario)
Paul Hernández Herrera (Vocal)

CUERNAVACA, MORELOS **AGOSTO 2019**

Agradecimientos

A mis padres y a mi hermano por siempre brindarme su apoyo, en esta ocasión por apoyar mi decisión de estudiar la licenciatura en ciencias la cual, si bien no fue fácil, ha sido para mí uno de los logros más importantes.

A mis profesores, por haber reconocido mi potencial a lo largo de la carrera y haberme puesto a prueba constantemente, esto me permitió desarrollar una disciplina que será clave para el próximo reto que vaya a emprender.

Resumen

Seguir una línea es una tarea básica en la Robótica Móvil, donde se hace uso de diferentes sensores para llevarla a cabo. Las tareas de la Robótica Móvil pueden tener una amplia variedad de aplicaciones, como en exploración, búsqueda aérea, combate de incendios, e incluso en la agricultura, permitiendo que un dron pueda identificar objetos de interés, sobrevolar la zona mientras libera alguna sustancia ya sea para combatir fuego, o un agroquímico como fertilizante, plaguicida, fungicida, entre otros.

Este trabajo tiene como objetivo principal desarrollar un sistema de visión artificial que permita a un dron identificar en tiempo real una línea de color contrastante al del suelo, determinar su ubicación y seguirla. El sistema fue desarrollado para un dron modelo Bebop de la empresa francesa Parrot, lo que permite sentar las bases para comenzar a desarrollar más aplicaciones para este tipo de drones dentro del Centro de Investigación en Ciencias (CInC) de la Universidad Autónoma del Estado de Morelos (UAEM).

Índice

Índice de figuras

1. Introducción

Desde hace varias décadas, la Robótica [1] ha sido un campo de gran interés para múltiples disciplinas, desde la Sociología hasta la Ingeniería y las Ciencias Exactas, incluyendo a las Ciencias Computacionales, puesto que hay un gran interés que ha impulsado que se realicen investigaciones para aprovechar las últimas tecnologías a las que se tiene acceso, lo que permitirá dar soluciones a todo tipo de problemáticas del mundo actual.

En tiempos recientes, gracias al desarrollo de algoritmos que están revolucionando las Ciencias Computacionales, al surgimiento de herramientas de código abierto, al acceso a hardware de menor costo con relación a sus contrapartes de años atrás, así como a materiales cada vez más baratos que permiten construir hardware propio; se han vuelto más accesibles las investigaciones en Robótica, pues todo lo anterior ha facilitado el desarrollo de programas que permiten hacer uso de robots en distintos escenarios. A esto se le puede agregar, el desarrollo de computadoras más potentes y de menor tamaño, al igual que el desarrollo de robots cada vez más pequeños, cuya capacidad de procesamiento es muy superior a la de sus predecesores.

Algunas de las aplicaciones que se pueden dar a los robots incluyen, exploración, inspección y mantenimiento, realización de tareas que podrían beneficiarse de la automatización de procesos, detección de objetos, agricultura, entre otros.

Los robots son especialmente útiles para llevar a cabo tareas en zonas de difícil acceso para los humanos, por lo que se ha recurrido al uso de drones (Aeronaves no tripuladas) [2] para labores de exploración, búsqueda aérea, combate de incendios, etc. De esto viene la importancia de desarrollar programas para vuelo autónomo de drones.

1.1 Planteamiento del problema

En el presente trabajo, se presenta el desarrollo de un programa que permita a un dron, utilizando la herramienta ROS (Robot Operating System) [3], realizar el seguimiento de una línea.

Seguir una línea es una tarea elemental en la Robótica Móvil, al poseer sensores que les permiten recopilar información de su entorno, los robots son capaces de interactuar con este [4]. Algunos de los sensores que pueden tener los drones incluyen a los sensores ultrasónicos que permiten medir distancia mediante el uso de ondas ultrasónicas y la evaluación del eco recibido, sensores infrarrojos que se basan en la medición de distancia al utilizar un sistema de emisión/recepción de radiación lumínica, cámaras que permiten recolectar información visual y puede utilizarse para crear sistemas de visión complejos [5]. Adicional a las cámaras de video, los robots pueden incorporar cámaras térmicas que permiten detectar información basada en temperatura [6].

Gracias a los sensores que poseen los robots, es posible llevar a cabo la detección de una línea y seguirla.

Con este trabajo se permitirá sentar las bases para desarrollar más programas para drones en el Centro de Investigación en Ciencias (CInC), ya que es la primera vez que se usa el hardware utilizado en este trabajo en conjunto con las librerías utilizadas. Esto presenta un reto ya que la documentación disponible es escasa y se requiere realizar pruebas previas al desarrollo de sistemas más robustos.

Una de las aplicaciones de interés es la Agricultura de Precisión, la cual consiste en la optimización de la gestión de cultivos haciendo uso de múltiples tecnologías para lograr una mayor eficiencia de aplicación de insumos [7]. Normalmente las plantas son cultivadas en filas, si el dron puede localizarlas por medio de sus sensores podría desplazarse encima de ellas y aplicar de forma precisa algún agroquímico (fertilizante, plaguicida, fungicida, entre otros).

1.2 Objetivo general

El objetivo general consiste en el desarrollo de un sistema de visión artificial que permita al dron identificar una línea situada en el suelo de un color contrastante, en un entorno interior con condiciones de iluminación controlada, esta información debe ser procesada en tiempo real en una estación en tierra, determinando la orientación de la línea para posteriormente actualizar la posición del dron.

1.3 Objetivos específicos

Entre los objetivos específicos se destaca:

- Creación de un módulo de conexión entre el dron y una estación en tierra (computadora de escritorio) para el envío y recepción de información por medio de una transmisión de datos inalámbrica.
- Procesamiento digital de la imagen capturada por la cámara del dron con la finalidad de detectar de forma rápida y precisa una línea.
- Obtención de información relevante para el dron, la cual consiste en las coordenadas de la línea las cuales son utilizadas para que el dron pueda desplazarse tomando en cuenta su estado actual y la ubicación de la línea.

2. Estado del Arte

En cuanto a seguidores de línea, existe una gran variedad de vehículos terrestres que realizan esta tarea a través de sensores infrarrojos principalmente, tales son los casos de algunos vehículos construidos con placas electrónicas como Arduino [8, 9].

La ventaja principal que tienen los robots terrestres sobre los aéreos es la estabilidad, tanto estando en reposo como en movimiento. A los robots terrestres difícilmente les afectan las condiciones ambientales, como los flujos de aire que se encuentran en el entorno donde están operando, lo cual es un reto principal para los vehículos aéreos, los cuáles deben tener un sistema de control más complejo para tener una buena estabilidad al momento de volar [10].

En cuanto al desarrollo de programas para drones se puede encontrar una variedad de software que realizan diversas tareas tales como el seguimiento de trayectorias previamente programadas [11] o detección y seguimiento de objetos [12].

Entre los drones más utilizados, se destaca el dron Parrot Bebop [13], el cual cuenta con un kit de desarrollo que facilita enormemente programar para este tipo de dron [14].

Existen repositorios en línea, como es el caso de Github [15] en los cuales hay ejemplos de aplicaciones para el Bebop, tal como el repositorio del usuario Alexis Guijarro [16] que contiene varios ejemplos de programas de los cuales se destaca un programa que realiza un seguimiento de colores.

También hay trabajos de tesis en los cuales se han desarrollado programas para el dron Bebop, tal como es el caso de la Universidad de las Fuerzas Armadas de Ecuador, donde se desarrolló un programa que implementa un sistema de visión para realizar la detección y seguimiento de personas en un ambiente exterior aplicando descriptores de HOG (Histogramas de gradientes orientados) los cuales se usan para conseguir la máxima invariancia posible ante cambios de posición o iluminación y posteriormente clasificando los datos con una máquina de soporte vectorial para separar las clases “persona” y “no persona” [17].

Por lo anterior, se puede apreciar que son muchas las posibilidades al desarrollar programas para drones, esto puede beneficiar al CInC, ya que permitirá abrir paso a nuevas líneas de investigación.

3. Marco teórico

Como ya se mencionó anteriormente, el objetivo principal del presente trabajo consiste en desarrollar un sistema de visión que permita a un dron seguir una línea situada en el suelo en un entorno interior con condiciones de iluminación controlada. Antes de presentar el sistema, es necesario mencionar qué herramientas se utilizaron, así como explicar en términos generales el funcionamiento de algunas de estas.

Las herramientas de hardware utilizadas son:

Dron Parrot Bebop, el cual cuenta con cámara 4K, adaptador Wifi integrado

Computadora (Utilizada como estación en tierra)

Antena para Wifi.

Las herramientas de software utilizadas son:

Robot Operating System (ROS)

Driver bebop_autonomy [18]

OpenCV [19]

CV_Bridge [20]

Para empezar, hay que definir qué es un cuadricóptero para comprender su funcionamiento para poder emprender vuelo y las partes que lo conforman, esto permite tener una visión general del hardware (en este caso el dron Bebop) para el cual se está programando.

Posteriormente, se explicará brevemente la herramienta ROS ya que esta funciona como una interfaz para realizar la conexión entre el dron y la computadora. Además, permitirá el desarrollo del sistema propuesto directamente en la computadora para que esta simplemente reciba información del dron y le envíe instrucciones después de procesar la información visual. Por lo anterior, ROS simplifica todo el proceso de conexión y comunicación entre el dron y la computadora.

Una vez analizado ROS, se analizarán otras herramientas que son de utilidad para desarrollar el sistema de visión de forma óptima y eficiente, al permitir implementar diferentes técnicas de Visión por Computadora. Tal es el caso de la librería OpenCV, la cual es de código abierto y provee una gran cantidad de funciones de procesamiento de imágenes previamente programadas; la herramienta CV_Bridge la cual permite realizar la conexión entre OpenCV y ROS para transmitir la información visual capturada por el dron a OpenCV.

3.1 Vehículos Aéreos No Tripulados

Los VANTs son máquinas aéreas que como su nombre lo indica, no tienen operadores humanos a bordo [21]. Hacen uso de hélices, a las cuáles se les transmite movimiento por medio de un motor para que el VANT pueda elevarse y desplazarse en el aire.

Un robot móvil cuenta con un sistema de locomoción que define cómo se mueve, un sistema de transmisión que actúa con el sistema de locomoción para permitirle moverse, un sistema de sensores que le permite interactuar con el mundo, además de un sistema que permite procesar señales y un sistema de control [22].

Un dron puede ser visto como un robot móvil aéreo, donde el sistema de locomoción consta del conjunto de elementos que le permiten moverse en el aire, tales como motores y hélices, el sistema de sensores, por lo general es una cámara, además de contar con un sistema que permite recibir señales enviadas ya sea por un control manual, o por un programa de computadora, estas señales son recibidas por el sistema de control automático que procesa dichas señales y las envía a los motores (como velocidades), lo cual le permite moverse. Estos elementos son explicados en las siguientes secciones.

3.2 Cuadricóptero

Un cuadricóptero es un helicóptero con cuatro motores para su elevación y movimiento (ver figura 1).

Figura 1.- Ejemplo de un cuadricóptero, Dron T1 de la marca TYH.

Los drones se han vuelto muy populares en los últimos años por su uso para fotografía y video, permitiendo capturar imágenes que serían difíciles de capturar sin la ayuda de estos.

Las partes que conforman un cuadricóptero se describen a continuación [23].

Chasis. - Es la estructura central del cuadricóptero, da cuerpo y soporte a cada una de las demás partes. La apariencia básica es en forma de X (ver figura 2), con 4 brazos sobre los cuáles van montados los motores y las hélices.

Figura 2.- Chasis de un cuadricóptero con forma de X.

Motores. – Son los encargados de dar movimiento a las hélices para que el cuadricóptero pueda elevarse, mantenerse suspendido en el aire, moverse o girar en alguna dirección. Estos se clasifican en unidades de kilovoltios, los cuáles determinan la velocidad a la que puede girar el motor.

Hélices. – Las hélices son las partes encargadas de mover el cuadricóptero, estas afectan en gran medida la velocidad a la que el cuadricóptero puede volar, la carga que puede soportar y la velocidad a la que puede maniobrar. Hélices grandes permiten al cuadricóptero cargar objetos más pesados a bajas Revoluciones Por Minuto (RPM), pero toman más tiempo en acelerar y en desacelerar, por otro lado, hélices pequeñas permiten cambiar rápidamente la velocidad del cuadricóptero y suelen ser mejores para realizar maniobras. Sin embargo, requieren más energía para girar, lo que representa un mayor esfuerzo para los motores y podría causar que su vida útil sea menor.

Controladores Electrónicos de Velocidad. – Los Controladores Electrónicos de Velocidad, los cuales son conocidos como ESC por sus siglas en inglés, se encargan de hacer girar los motores, un cuadricóptero tiene 4 ESCs (ver figura 3), uno para cada uno de sus motores.

Estos proveen la corriente propiamente modulada a los motores, los cuales producen el ritmo apropiado de giro tanto para la elevación como para las maniobras.

Figura 3.- Controladores Electrónicos de Velocidad (ESC).

Controlador de vuelo. – El controlador de vuelo como su nombre lo indica, controla el vehículo e interpreta las señales que el dispositivo transmisor y receptor (transceptor) [24] envía para guiar el cuadricóptero.

Radio receptor. – El radio receptor (ver figura 4) es el encargado de recibir las señales del transmisor (control remoto, aplicación móvil). Este consiste en un componente que se conecta al controlador de vuelo para recibir señales y a un controlador para transmitirlas.

Figura 4.- Radio receptor.

Baterías. - Un cuadricóptero debe tener una fuente de energía, el estándar es una batería recargable de Polímero de Litio la cual es de bajo peso y ofrece alta densidad energética, así como tasas de carga más rápidas lo cual la hace ideal para vehículos de radio control y drones [25].

Hay otros elementos que puede contener un cuadricóptero, sin embargo, los mencionados anteriormente son los básicos con los que todo cuadricóptero debe contar para poder volar y realizar acciones básicas.

El movimiento de un cuadricóptero consiste en traslación y rotación, estos se controlan mediante variaciones en la velocidad de giro de sus motores lo cual produce una fuerza de empuje vertical en cada uno de los motores.

Los motores y las hélices se dividen en 2 pares, de los cuales uno gira en sentido de las manecillas del reloj y el otro con dirección opuesta a las manecillas del reloj (ver figura 5).

Figura 5.- Dirección de giro de cada una de las hélices del dron cuadricóptero Phantom 3.

Para ascender, se aumenta la velocidad de rotación de los cuatro motores simultáneamente de tal forma que genera una fuerza neta que provoca que el dron se eleve, pero a su vez, por la forma de las hélices y el flujo del aire alrededor de estas, genera un cambio de presión encima y debajo de las hélices; aumentando la presión debajo de estas y disminuyendo la presión encima de estas [26] (ver figura 6).

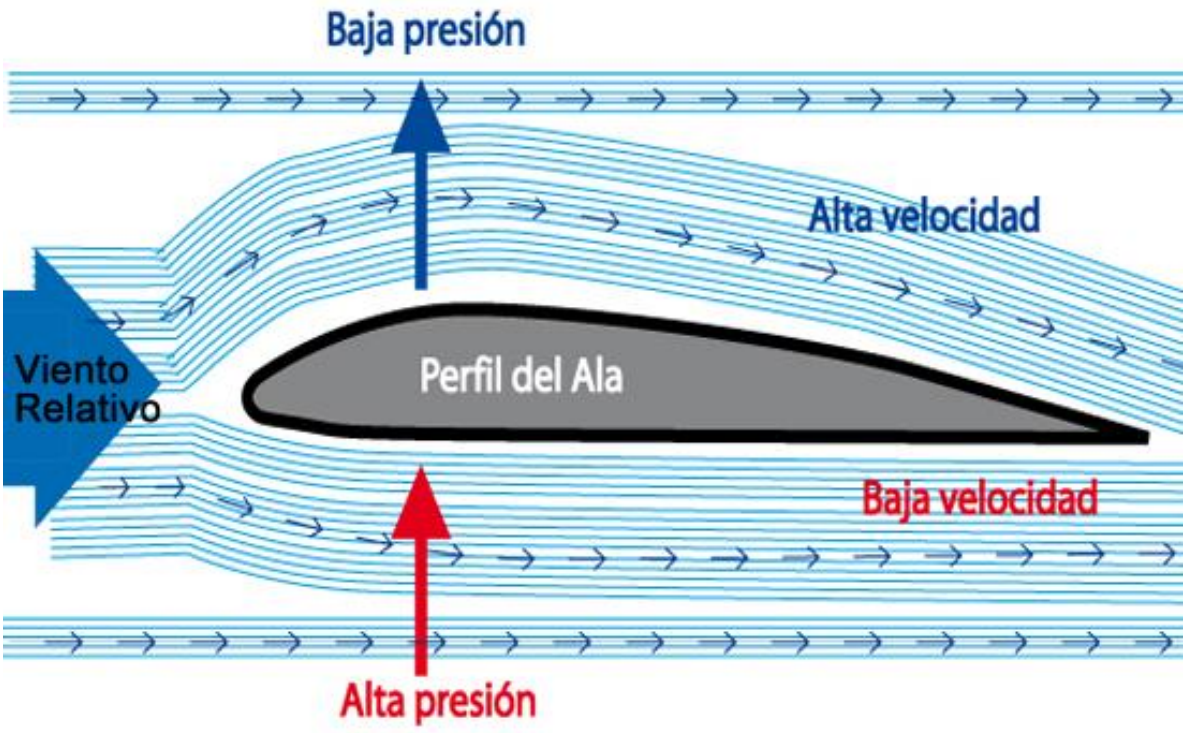

Figura 6.- Cambio de presión con respecto a velocidad del viento alrededor de las hélices [27].

Para mantenerse flotando, la velocidad de rotación de los motores debe generar una fuerza hacia arriba que iguale la de su propio peso, permitiéndole mantener un equilibrio.

Antes de explicar las rotaciones, es importante definir los tipos de giro con respecto a los ejes X, Y, Z; los cuales por convención reciben los nombres en inglés de *Roll*, *Pitch* y *Yaw* respectivamente (ver figura 7). Estos son importantes puesto que además de describir movimientos en un espacio tridimensional, sirven como referencia para explicar cómo el dron avanza con respecto a un eje determinado, o cómo gira sobre su propio eje.

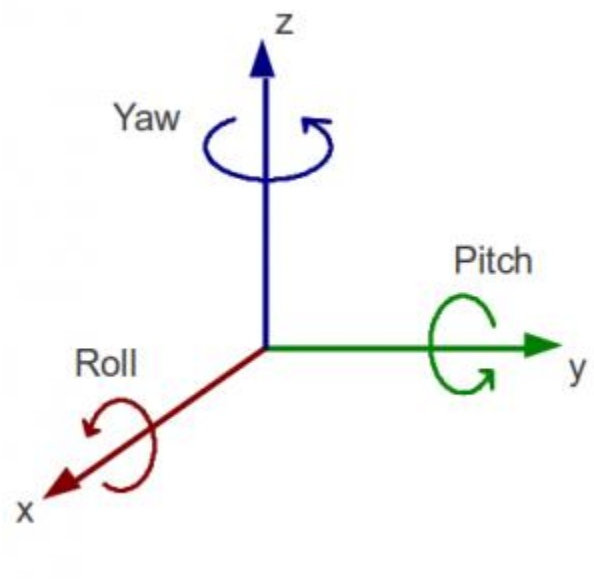

Figura 7.- Rotación sobre los 3 ejes.

Para generar un giro *Roll* hacia la izquierda, se incrementa la velocidad de rotación de ambos motores de la derecha, y para generarlo hacia la derecha se incrementa esta velocidad en ambos motores de la izquierda. Para generar un giro *Pitch* hacia el frente, se incrementa la velocidad de ambos motores de la parte atrás del cuadricóptero, y para generarlo hacia atrás, se aumenta la velocidad de ambos motores de la parte de enfrente. Para generar un giro *Yaw* en sentido de las manecillas del reloj únicamente se incrementa la velocidad de los motores que giran en dicho sentido, análogamente para girar en sentido contrario a las manecillas del reloj. Al generar un giro de tipo *Roll*, debido a la forma de un cuadricóptero, sin importar en qué dirección esté el frente del cuadricóptero, este se desplazará en la dirección del giro *Roll*, lo mismo sucede con el giro *Pitch*, que hará que se desplace hacia adelante o hacia atrás. La única diferencia es con el giro *Yaw*, en ese caso, el cuadricóptero sólo generará una rotación sobre el eje Z, sin desplazarse en alguna dirección.

Finalmente, para descender se disminuye la velocidad de los 4 motores de tal forma que la fuerza generada sea menor a la fuerza de gravedad.

3.3 Parrot Bebop

El Bebop es un dron cuadricóptero fabricado por la empresa francesa Parrot SA, introducido inicialmente en 2014, es un dron de peso ligero con una cámara HD integrada.

En 2015, Parrot introdujo al mercado el dron Bebop 2 (ver figura 8). Entre sus diferencias principales está que el Bebop 2 es más grande que el Bebop original (ver figura 9), con un marco de 290mm contra el de 250mm del Bebop original, cuenta con una batería de 2700 mAh, lo que le permite tener un tiempo de vuelo mayor al del Bebop original el cual contaba con una batería de 1200 mAh.

Figura 8.- Parrot Bebop 2.

Figura 9.- Diferencias de tamaño entre Bebop 2 y Bebop.

Otras diferencias notorias que se pueden mencionar son la cámara y la velocidad máxima que alcanza el Bebop 2 con respecto al Bebop original. El dron Bebop 2 pesa 500g y ofrece 25 minutos de tiempo de vuelo autónomo, además la cámara tiene una buena captación de luz con lo cual es capaz de tomar fotos y videos de buena calidad tanto en interiores como en exteriores.

La cámara del Bebop 2 cuenta con una lente de ojo de pez con una resolución de 14 megapíxeles que permite capturar fotos con resolución 4096 x 3072 pixeles, y permite capturar video en full HD (resolución 1920 x 1080) a 30 fps, con codificación de video H.264.

Para conectividad cuenta con un adaptador Wi-Fi de banda dual que emite señal de 2.4 y 5 GHz, lo que permite a otros dispositivos conectarse a este, además tiene un rango de señal de 300 m.

La velocidad máxima horizontal que alcanza es de 16 m/s y velocidad hacia arriba de 6 m/s.

El Bebop 2 cuenta con una luz LED en la parte trasera (ver figura 10), que además es el botón de encendido y apagado, esta luz es visible desde largas distancias, permitiendo ubicar al Bebop 2 durante el vuelo, así como revisar su orientación.

Figura 10.- Luz LED del Bebop 2.

Los motores del Bebop 2 tienen una velocidad de rotación de 7,500 RPM en el arranque y pueden alcanzar una velocidad de rotación de hasta 12,000 RPM. Los motores del Bebop 2 giran de la siguiente manera: Los motores de la parte delantera izquierda y de la parte trasera derecha giran en sentido contrario a las manecillas del reloj. Los motores de la parte delantera derecha y la trasera izquierda giran en sentido de las manecillas del reloj (ver figura 11).

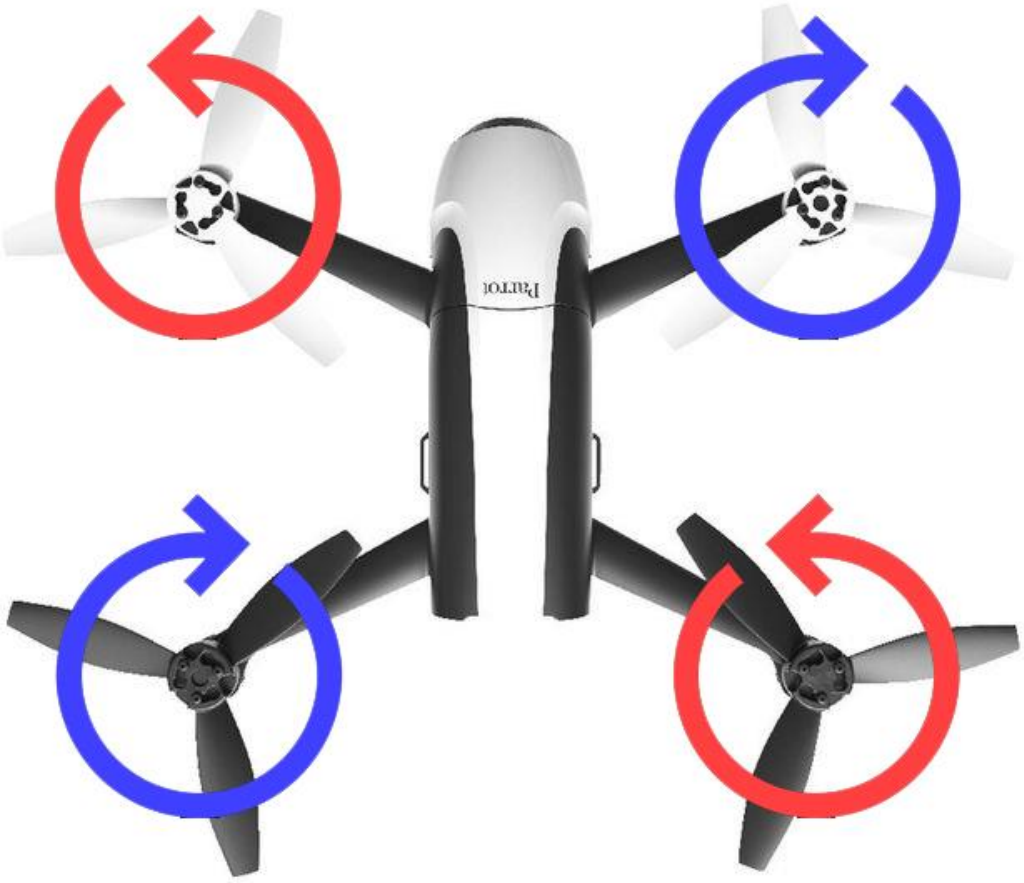

Figura 11.- Dirección de giro de los motores del Bebop 2.

3.4 Robot Operating System (ROS)

ROS es un metasistema operativo de código abierto que ofrece los mismos servicios que un Sistema Operativo, incluyendo abstracción, control de dispositivos a bajo nivel, implementación de funcionalidad comúnmente usada, paso de mensajes entre procesos y administración de paquetes que ofrece herramientas y librerías para obtener, construir, escribir y ejecutar código.

ROS provee un marco de trabajo (*framework*) flexible para escribir software para robots, gracias a sus herramientas y librerías que buscan simplificar la tarea de crear comportamientos complejos para robots a través de una amplia variedad de plataformas robóticas.

Por lo general, desarrollar programas para robots es una tarea muy complicada puesto que suele hacerse para el tipo de hardware y arquitectura específicos del robot en cuestión, por esta razón reutilizar código puede ser muy complicado en algunos casos o no es posible en otros.

La estructura con la que funciona ROS se explica a continuación:

a) El sistema de archivos de ROS: El Software en ROS está organizado en paquetes, los paquetes son la unidad de construcción atómica en este sentido, esto es, un paquete es la unidad de construcción mínima de ROS y es la forma en que el software es empaquetado.

Los paquetes pueden contener procesos de tiempo de ejecución de ROS (nodos), librerías dependientes de ROS, conjuntos de datos, archivos de configuración y algún otro elemento que puede ser útilmente organizado en conjunto.

Además, los paquetes contienen un archivo llamado *package.xml*, que provee metadatos sobre el paquete, incluyendo su nombre, versión, descripción, información de licencia, dependencias y otra metainformación como paquetes exportados.

Otros elementos importantes que contiene un paquete son descripciones de mensajes y de servicios, los cuales definen estructuras de datos para envío de mensajes de ROS y para servicios de ROS, respectivamente.

b) Grafo de cómputo de ROS: El grafo de cómputo es la red *peer-to-peer* de procesos ROS que procesan datos en conjunto, sus elementos, que proveen datos al grafo son:

- Nodos: Procesos que realizan cálculos. Un sistema de control robótico está compuesto de varios nodos que sirven para distintos propósitos en específico, como controlar partes de un robot o realizar planeación de caminos.
- Master: El ROS Master provee registro de nombres y búsquedas para el resto del grafo de cómputo. Sin el Master, los nodos no podrían encontrarse entre ellos, intercambiar mensajes o invocar servicios.
- Mensajes: Los nodos se comunican entre sí por paso de mensajes. Un mensaje es una estructura de datos que compone campos con tipos primitivos (*int*,

float, *bool*, etc.), al igual que arreglos de tipos primitivos. Un mensaje puede incluir estructuras y arreglos arbitrariamente anidados (tal como estructuras en C).

- Tópicos: Los mensajes son enrutados por medio de sistemas de transporte con semánticas de publicar y subscribir. Un nodo envía un mensaje "publicándolo" a un tópico dado. El Tópico es un nombre usado para identificar el contenido del mensaje. Igualmente, un nodo con interés en cierto tipo de datos se subscribirá al tópico apropiado (ver figura 12). Puede haber múltiples publicadores y subscriptores concurrentes en un solo tópico.
- Servicios: Un nodo proveedor puede ofrecer un servicio bajo un nombre y un cliente utiliza el servicio enviando mensajes de solicitud y esperando la respuesta (comunicación de solicitud / respuesta).

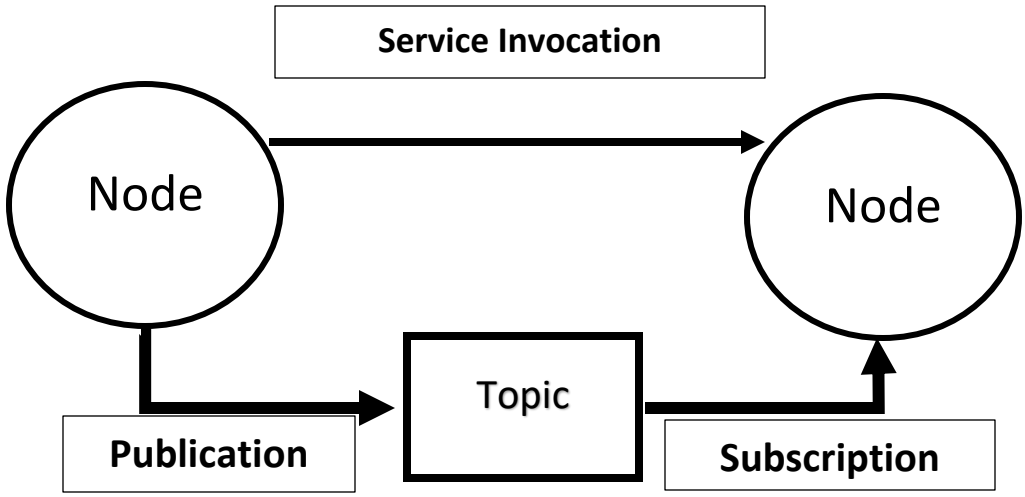

Figura 12.- Modelo de publicar / subscribir tópicos en ROS [28].

Los elementos anteriormente mencionados son los más relevantes para el presente trabajo, por lo que se omiten otros elementos que no son utilizados.

Los tipos de mensaje que son de especial interés para el presente trabajo, son los mensajes de tipo *sensor_ msgs/Image, std_ msgs/Empty* y *geometry_ msgs/Twist*.

La definición de los mensajes del tipo *sensor_ msgs/Image* es la siguiente [29]:

std_msgs/Header header
uint32 height
uint32 width
string encoding
uint8 is_bigendian
uint32 step
uint8[] data

Este mensaje contiene información de una imagen sin comprimir y mantiene información de sus propiedades que son alto y ancho de la imagen, codificación, el contenido, etc.

El tipo de mensaje *std_ msgs/Empty* es como el nombre lo indica, un mensaje vacío que sirve para enviar una señal a algún tópico [30].

El tipo de mensaje *geometry_ msgs/Twist* se define de la siguiente manera [31]:

geometry_msgs/Vector3 linear
geometry_msgs/Vector3 angular

Este mensaje expresa velocidad en un espacio 3D y se divide en sus partes linear y angular.

3.4.1 Rosdep

Rosdep es una herramienta de línea de comando usada para instalar dependencias de ROS, en este trabajo se utiliza para instalar el driver bebop_autonomy (el cuál se explica más adelante) [32].

3.4.2 Caktin

Catkin es el sistema de construcción oficial de ROS [33]. Catkin combina Macros de CMake [34] (herramientas de construcción de código abierto multiplataforma usadas para construir software) y scripts de Python [35] para proveer funcionalidades sobre el flujo de trabajo de CMake.

Al ser un sistema de construcción, Catkin se encarga de generar 'objetivos' para código fuente que pueden utilizarse por un usuario final. Estos objetivos pueden ser librerías, programas ejecutables, scripts generados, interfaces o alguna otra cosa que no es código estático.

Catkin funciona con una estructura ordenada de carpetas la cuál será nuestro espacio de trabajo (ver figura 13).

Catkin viene incluido por defecto al instalar ROS.

```
workspace_folder/         -- WORKSPACE
  src/                    -- SOURCE SPACE
    CMakeLists.txt        -- The 'toplevel' CMake file
    package_1/
      CMakeLists.txt
      package.xml
      ...
    package_n/
      CATKIN_IGNORE       -- Optional empty file to exclude package_n from being processed
      CMakeLists.txt
      package.xml
      ...
  build/                  -- BUILD SPACE
    CATKIN_IGNORE         -- Keeps catkin from walking this directory
  devel/                  -- DEVELOPMENT SPACE (set by CATKIN_DEVEL_PREFIX)
    bin/
    etc/
    include/
    lib/
    share/
    .catkin
    env.bash
    setup.bash
    setup.sh
    ...
  install/                -- INSTALL SPACE (set by CMAKE_INSTALL_PREFIX)
    bin/
    etc/
    include/
    lib/
    share/
    .catkin
    env.bash
    setup.bash
    setup.sh
    ...
```

Figura 13.- Estructura de un espacio de trabajo de Catkin.

3.5 Bebop_autonomy

Bebop_autonomy es un driver de ROS para los drones Parrot Bebop 1.0 y 2.0, basado en el kit oficial de desarrollo de parrot ARDroneSDK3. Este driver, permite conectar el Bebop con ROS, brindando todas las ventajas que ofrece ROS, creando nodos para las partes del Bebop que puedan comunicarse de alguna manera con el ordenador (sistema de locomoción, sistema de sensores), publicando tópicos con sus mensajes correspondientes (como por ejemplo, la imagen capturada por la cámara) y además permitiéndole subscribirse a tópicos que pudieran ser de utilidad (por ejemplo, un tópico que mande mensajes de velocidad para la rotación de los motores).

Esto es especialmente útil para desarrollar programas para vuelo autónomo del Bebop, pudiendo aprovechar la abstracción, las librerías y el framework en general que ofrece ROS para el desarrollo de software para robots.

3.6 OpenCV

Open Source Computer Vision Library, es una librería de código abierto que provee una infraestructura común para aplicaciones de visión por computadora, la cual contiene más de 2500 algoritmos optimizados [36].

Se hace amplio uso del objeto cv::Mat de openCV [37], el cual representa un arreglo numérico n-dimensional y sirve para almacenar imágenes.

Además, se aprovechan las funciones para manipular imágenes de OpenCV, en particular, funciones que permiten cambiar de modelos de color, ya que OpenCV funciona con un modelo de color BGR, así como el algoritmo de los momentos de Hu que provee.

3.6.1 Modelos de color

Un modelo de color es un sistema que se utiliza para describir colores en combinación de colores básicos llamados colores primarios [38]. Haciendo uso de un modelo de color, una computadora puede interpretar colores de acuerdo a los valores que tome un pixel en un arreglo numérico.

Algunos ejemplos de modelos de colores son:

a) RGB, cuyos componentes son Rojo, Verde y Azul.
b) HSV que, a diferencia del modelo RGB, en lugar de combinar colores primarios utiliza escalas de Matiz, Saturación y Valor.

Como se mencionó con anterioridad, OpenCV por defecto funciona con un modelo de color BGR (Azul, Verde y Rojo) que cambia de posición los valores con respecto a la posición de los valores del modelo RGB. Como la mayoría de las imágenes funcionan con el modelo RGB, al importar una imagen a OpenCV, se pueden apreciar los colores rojo y azul invertidos (ver figura 14) [39].

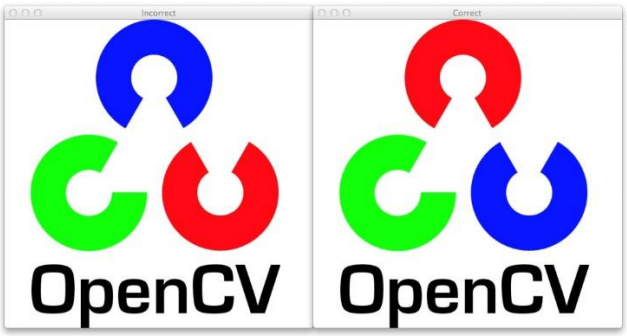

Figura 14. Diferencia de colores en el logo de OpenCV. Izquierda – Imagen en modelo BRG, Derecha – Imagen en modelo RGB.

OpenCV provee funciones que permiten cambiar de escala de color [40], en particular se utiliza la función para convertir la imagen al modelo HSV, ya que este es de particular utilidad para discriminar colores percibidos por una cámara debido a las posibles variaciones de los valores en otro modelo de color causados por los cambios de la iluminación y el material del objeto a detectar.

También se utiliza la función para convertir la imagen a escala de grises, teniendo así una imagen en la que los valores de los pixeles varían únicamente en intensidades de pixel en lugar de valores de colores dados por un arreglo.

3.6.2 Modelo de color HSV

El modelo de color HSV es un modelo en el cual se interpretan los colores de acuerdo a 3 valores numéricos correspondientes a Matiz (*Hue*), Saturación (*Saturation*) y Valor (*Value*) [41].

Este modelo de color se representa comúnmente como un cono invertido donde sus componentes corresponden a una posición en dicho cono (ver figura 15).

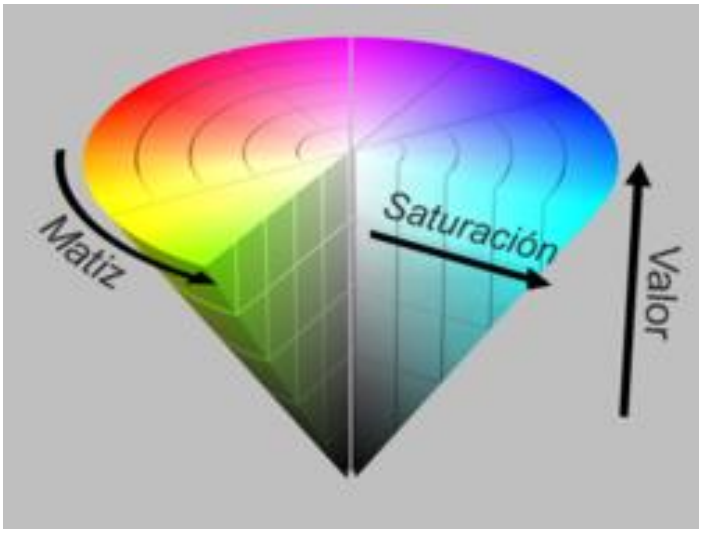

Figura 15. Cono del modelo de color HSV.

El Matiz se representa como un ángulo, donde sus valores se encuentran en el rango (0, 360), y cada valor corresponde a un color.

La saturación se representa como la distancia al eje, donde sus valores se encuentran en el rango (0, 100) y corresponden al brillo del color.

El valor se representa como la altura, donde sus valores se encuentran en el rango (0, 100) y corresponden a la amplitud de luz que define el color, en este caso 0 siempre es negro y conforme más alto es el valor más claro se va volviendo el color.

Haciendo uso de los componentes del modelo HSV se puede elegir cualquier color tomando en consideración variaciones en iluminación a través de la saturación y el valor.

3.6.3 Momentos de Hu

Los momentos de Hu consisten en descriptores invariantes a traslación, rotación y escala utilizados para describir objetos en una imagen [42]. Estos descriptores son particularmente útiles el para reconocimiento de patrones visuales y caracteres.

OpenCV provee la función *moments* [43], la cual devuelve momentos que consisten en distribuciones probabilísticas de densidad de una imagen de entrada [44]. Estos son de particular interés para el presente trabajo debido a que se utilizan para calcular el centroide de la línea que se busca detectar ya que con esta información se puede interpretar su ubicación a partir de su centroide.

3.7 Cv_bridge

Cv_bridge es una paquetería de ROS que permite la conexión entre un mensaje de tipo *sensor_msgs/Image* y OpenCV, convirtiendo a tipo cv::Mat, una imagen recibida por un sensor y publicada en un tópico (ver figura 16).

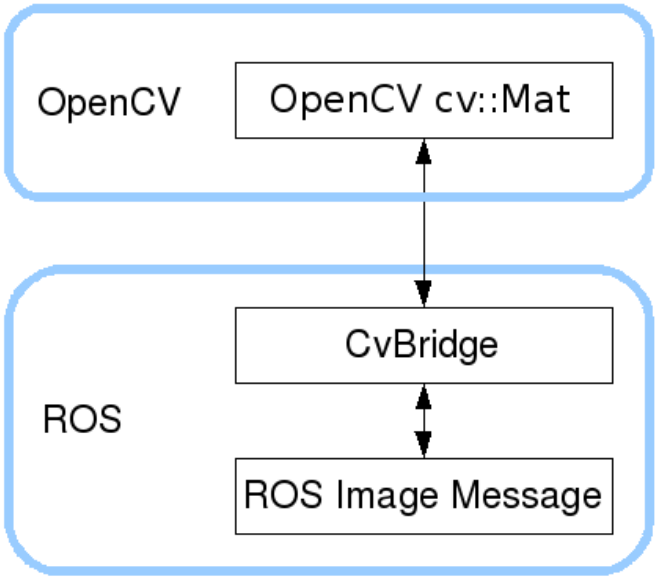

Figura 16. Flujo de trabajo de cv_bridge para conectar con OpenCV.

Cv_bridge define un tipo de imagen CvImage que contiene una imagen de OpenCV, su codificación y un encabezado de ROS. CvImage contiene exactamente la información de *sensor_msgs/Image*, por lo que se puede convertir de una representación a la otra.

La clase de formato CvImage está definida de la siguiente manera:

```
namespace cv_bridge {

class CvImage
{
public:
  std_msgs::Header header;
  std::string encoding;
  cv::Mat image;
};

typedef boost::shared_ptr<CvImage> CvImagePtr;
typedef boost::shared_ptr<CvImage const> CvImageConstPtr;

}
```

CvBridge provee las funciones toCvCopy para hacer una copia de la imagen, esto es por si se tiene que modificar la información de la imagen, se realiza una copia, y toCvShare para compartir la imagen, esta función se utiliza cuando no se va a modificar la imagen original.

4. Metodología.

A continuación, se presenta el método de resolución utilizado para cumplir los objetivos propuestos en este trabajo.

Haciendo uso de la estructura de grafo de ROS, se crean dos nodos que son ejecutados en paralelo en distintas terminales (ver figura 17), uno que se encargará de realizar el procesamiento de las imágenes capturadas por la cámara del Bebop, y otro que, de acuerdo a la información obtenida en el procesamiento de las imágenes, mandará comandos al Bebop. Se escriben por separado para aprovechar el procesamiento en paralelo, minimizando el tiempo en que se actualiza la información de detección de la línea y el tiempo en el que se mandan las velocidades a los motores del Bebop.

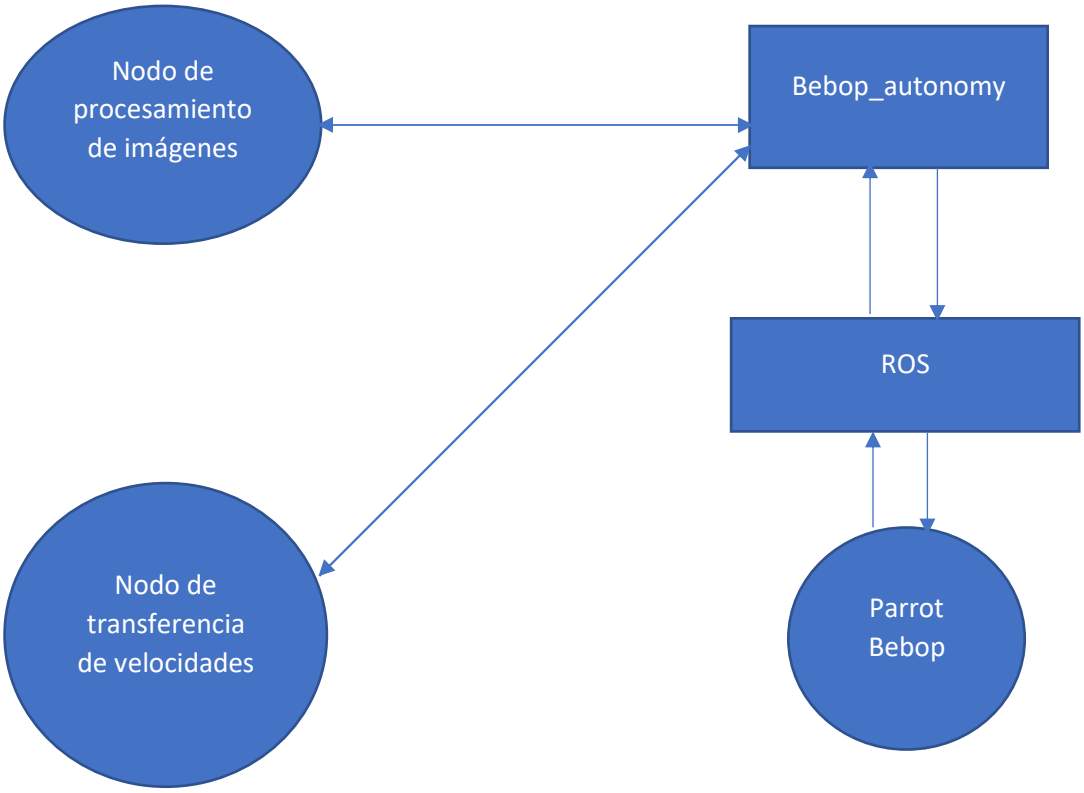

Figura 17.- Grafo que representa el método de resolución.

Estos nodos o programas publican tópicos de ROS a los que los demás nodos pueden subscribirse y hacer uso de la información relevante. El primero se subscribe al tópico

publicado por la cámara del Bebop (a través del driver bebop_autonomy), recibiendo la imagen en un tipo de mensaje de imagen, posteriormente convirtiéndola a un objeto de tipo cv::Mat, el cuál es la representación matricial de una imagen en el modelo de color BGR, esta se convierte a un modelo de color HSV el cual sirve para separar el color relevante de la imagen original de los colores que no son de interés.

Una vez convertida la imagen al modelo de color HSV, se obtiene una máscara usando la función *InRange* de OpenCV la cual se aplica a la imagen HSV para separar los colores de interés de la imagen de los colores que no son de interés [45], logrando aislar la línea para comenzar a extraer información relevante para el Bebop como la posición de la línea. Posteriormente se realiza una umbralización a través de la función *threshold* de OpenCV aplicando una operación de umbralizado inverso [46] la cual por cada pixel de la imagen de entrada y dado un valor de intensidad de pixel que se utiliza como umbral revisa si el pixel de la imagen es mayor al umbral entonces al pixel se le asigna valor 1 (negro), de lo contrario se le asigna valor 0 (Blanco), obteniendo así una imagen binarizada la cual es utilizada para calcular el centroide de la línea a través de los momentos de Hu con la función *moments* de OpenCV.

Ya calculado el centroide, se publica su posición como mensaje de tipo punto (geometry_msgs::Point), al igual que la imagen procesada para tener acceso a sus propiedades desde otros nodos.

El segundo nodo se subscribe a los tópicos publicados por el primer nodo, y dependiendo de la posición del punto, determina si el Bebop tiene que mantenerse suspendido en el aire, avanzar, o girar.

Lo anterior se realizó siguiendo el plan descrito a continuación, el cual fue previamente establecido:

a) Instalación de ROS, OpenCV, y Catkin en la computadora con sistema operativo Ubuntu versión 16.04.

b) Instalación del driver bebop_autonomy.

c) Actualización del firmware del Bebop 2.

d) Conexión del Bebop con el ordenador haciendo uso del driver bebop_autonomy.

e) Desarrollo del programa de procesamiento de imágenes

f) Desarrollo del programa de transferencia de velocidades.

4.1 Comunicación con el Bebop

Como se mencionó en el marco teórico, las herramientas utilizadas incluyen al dron Parrot Bebop 2, una computadora, una antena Wifi por el lado del hardware, y por el lado del software, el sistema operativo Ubuntu, ROS, el controlador bebop_autonomy y OpenCV.

Se eligió el dron Parrot Bebop por la cámara que tiene integrada, la cual, al ser el sensor principal de percepción, cuenta con una alta resolución de imagen, además de la posibilidad de conectarlo a ROS, lo que permite que el programa sea más eficiente tanto en procesamiento como en tiempo de respuesta.

El uso del sistema operativo Ubuntu se debe a que el driver bebop_autonomy está probado en dicho sistema operativo [47] junto con la versión Kinetic de ROS [48]. El driver bebop_autonomy ya viene previamente compilado, por lo que basta con seguir las instrucciones de instalación para comenzar a utilizarlo. Una vez iniciado el driver, provee los nodos de ROS relevantes para el Bebop, estos nodos a su vez publican los tópicos con los que se logrará establecer comunicación entre el sistema desarrollado y el Bebop.

A continuación, se describen los tópicos de mayor interés:

a) El tópico */bebop/image_raw* con el que se puede acceder a la imagen capturada por la cámara del Bebop.
b) El tópico */bebop/takeoff* con el que se despega al Bebop.
c) El tópico */bebop/land* con el que se aterriza al Bebop.
d) El tópico */bebop/cmd_vel* el cual es utilizado para mandar velocidades al bebop.
e) El tópico */bebop/reset* que reinicia todas las velocidades de los motores Bebop. Este tópico se utiliza para realizar aterrizajes de emergencias.

Con el primer tópico se recibirá la imagen para procesar que se utilizará por el primer nodo, los demás nodos servirán para enviar comandos para controlar el bebop desde el segundo nodo publicando mensajes del tipo requerido a los tópicos correspondientes.

Los comandos para despegar, aterrizar y reinicio se pueden enviar publicando un mensaje de tipo *std_msgs/Empty* a los tópicos *takeoff*, *land* y *reset* respectivamente.

Para mover el Bebop, se publican mensajes del tipo *geometry_ msgs/Twist* al tópico *cmd_ vel* cuando el Bebop está suspendido en el aire, este mensaje contiene 3 tipos de velocidad lineal para cada eje coordenado (*x, y, z*) y un tipo de velocidad angular para el eje coordenado *z*. La velocidad lineal permite al bebop desplazarse sobre el eje sobre el cual se manda la velocidad, mientras que la velocidad angular le permite realizar un giro sobre su propio eje.

Los valores que toman las velocidades están definidos en un rango de [-1, 1], el Bebop ejecuta el último comando recibido mientras el driver esté en ejecución, la dirección de desplazamiento y giro según el valor recibido se definen de la siguiente manera:

a) linear.x positivo: Desplazamiento al frente.
b) linear.x negativo: Desplazamiento hacia atrás.
c) linear.y positivo: Desplazamiento a la izquierda.
d) linear.y negativo: Desplazamiento a la derecha.
e) linear.z positivo: Desplazamiento hacia arriba (ascenso).
f) linear.z negativo: Desplazamiento hacia abajo (descenso).
g) angular.z positivo: Giro con sentido contrario a las manecillas del reloj.
h) angular.z negativo: Giro con sentido de las manecillas del reloj.

4.2 Procesamiento Digital de la Imagen

A continuación, se explica el procesamiento de imagen que se realiza en el primer nodo.

En modo de resumen, los pasos son los siguientes:

1. Recorte de la imagen para obtener una región de interés.
2. Conversión de la imagen a modelo de color HSV.
3. Obtención de una máscara para discriminar colores.
4. Aplicación de la máscara a la imagen original.
5. Binarización de la imagen por medio de umbralización.
6. Obtención del centroide de la línea utilizando los momentos de Hu.

Al tener como entrada una imagen digital, esta se puede visualizar como una matriz de números, los cuales representan pixeles y sus intensidades de color. Para realizar

el Procesamiento Digital de Imágenes (PDI) en algunas tareas suele ser mejor trabajar en escalas de grises.

OpenCV trabaja en un formato de canales de color BGR (*Blue*, *Green*, *Red*), pero provee funciones que permiten transformar la imagen a otro modelo de color como HSV o a escala de grises.

Nuestro objetivo al realizar el procesamiento de la imagen es obtener un punto con coordenadas (x, y) el cual corresponde al centroide de la línea. Una vez que se haya encontrado se le enviarán los comandos apropiados al Bebop para desplazarse a dichas coordenadas.

Este centroide se obtiene a través de los momentos de Hu, usando los momentos de orden 0 y de primer orden.

4.2.1 Recorte de la imagen.

Primero es necesario delimitar una región de interés de la imagen, esto es necesario para disminuir elementos de la imagen que pudieran causar ruido al procesamiento, puesto que sólo se requiere analizar la región de la imagen donde aparece el suelo y la línea sobre la cual se desplazará el dron.

Además, una imagen de menor tamaño es más fácil de procesar que una imagen más grande, al realizarse las operaciones sobre esta en un tiempo menor que una imagen más grande.

Utilizando la función preprogramada de OpenCV cv::Rect se puede obtener una imagen recortada a partir de la imagen de entrada. Esta función está definida de la siguiente manera: Rect(*int* x, *int* y, *int width*, *int height*). Donde los argumentos corresponden a las coordenadas (x, y) de la esquina superior izquierda de la imagen, y el ancho y largo del rectángulo que define el área de interés. En particular se decidió recortar la imagen a un tamaño de 200 x 80 pixeles partiendo del pixel (308, 400) de la imagen original ya que tras varias pruebas se notó que esta imagen recortada mantenía suficiente información relevante de la línea además que no era perdida de vista por el Bebop al avanzar.

En la figura 18 se muestra una imagen captada por la cámara del Bebop y en la figura 19 la misma imagen recortada.

Figura 18.- Imagen captada por la cámara del Bebop.

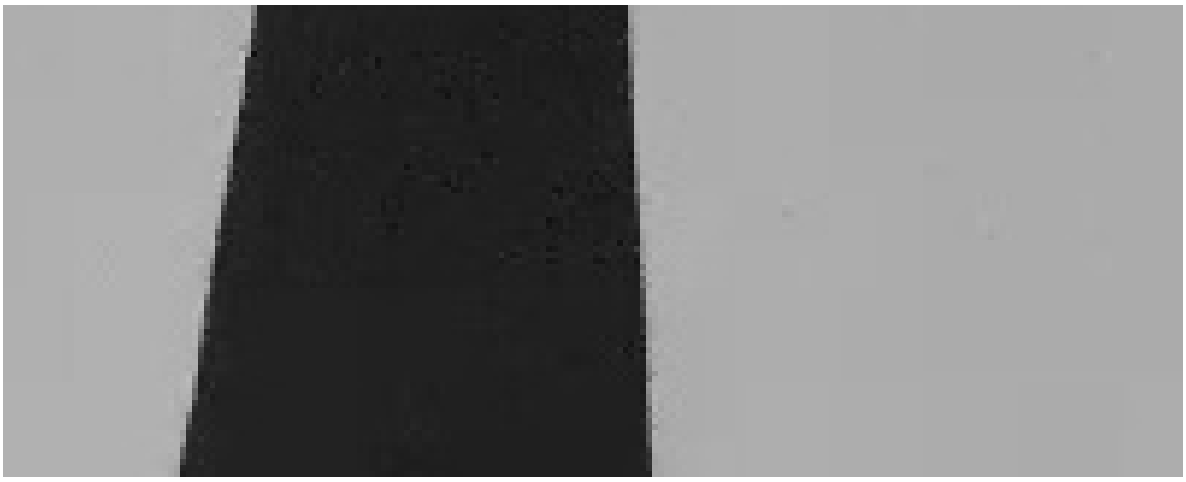

Figura 19.- Imagen recortada para definir una región de interés.

4.2.2 Conversión de la imagen al modelo de color HSV.

Una vez definida la región de interés, es necesario separar los objetos de interés de la imagen de los objetos que no lo son, en este caso se trata de una línea de un color contrastante al color del suelo.

La escala de color HSV es particularmente útil para procesamiento de imágenes porque se pueden separar los colores por tonalidad, intensidad e iluminación, factores que en el mundo real pueden afectar la forma en cómo se percibe un color determinado según el material y la iluminación del entorno donde se realiza el experimento.

Como se ha mencionado con anterioridad, OpenCV trabaja por defecto en un formato BGR, por lo que es necesario convertir las imágenes a la escala de color deseada antes de empezar con el procesamiento, de lo contrario, podría haber discrepancia entre los valores ingresados y los colores procesados.

OpenCV cuenta con la función cv::cvtColor para convertir una imagen a otro modelo de color. La función se define de la siguiente manera:

cvtColor(InputArray src, OutputArray dest, int code, int dstCn)

donde los argumentos corresponden a una imagen de entrada que puede ser un arreglo o un objeto tipo cv::Mat, un arreglo de salida, el espacio de conversión y un número de canales opcional para la imagen destino. Si este último parámetro no es ingresado, el número de canales se deriva automáticamente de la imagen de entrada y del espacio de conversión.

La transformación CV_BGR2HSV como tercer argumento permite convertir la imagen a un espacio de color HSV, la cual será utilizada para realizar el procesamiento.

4.2.3 Obtención de una máscara para discriminar colores

Una vez convertida la imagen a modelo de color HSV, el siguiente paso consiste en obtener una máscara, la cuál será aplicada a la imagen original recortada para eliminar los elementos no relevantes y separarlos de los que sí son de interés.

Con la función cv::inRange, se revisa si los valores de los elementos de un arreglo se encuentran entre los valores de elementos de otros dos arreglos.

Es necesario definir un rango entre el valor mínimo y el valor máximo del color que se busca discriminar. Es decir, el valor del color de interés en su menor intensidad y en su mayor intensidad, de esta manera se crea una máscara que contenga

únicamente los pixeles con valores en este rango a los cuales se les asigna un valor de 255, los demás pixeles son ignorados asignando a la máscara un valor de 0.

La función se define de la siguiente manera:

inRange(InputArray src, InputArray lowerb, InputArray upperb, OutputArray dst)

donde los argumentos corresponden a la imagen de entrada, el valor inferior del color elegido, el valor superior del color elegido y como salida la máscara calculada con un tamaño igual al de la imagen de entrada la cual puede almacenase en un arreglo o en un objeto cv::Mat.

Donde dado un arreglo de entrada de un solo canal, cada elemento de la salida se define de la siguiente manera:

$$dst(I) = \begin{cases} 255 \; si \; lowerb(I)_0 \leq src(I)_0 \\ 0 \; de \; otra \; forma \end{cases} \tag{4.1}$$

Para dos canales:

$$dst(I) = \begin{cases} 255 \; si \;\; lowerb(I)_0 \leq src(I)_0 \wedge lowerb(I)_1 \leq src(I)_1 \\ 0 \; de \; otra \; forma \end{cases} \tag{4.2}$$

Y así sucesivamente para n canales.

Para las pruebas finales se creó la máscara usando arreglos con los valores (0, 0, 128) para el valor inferior del color y (255, 255, 255) para el valor superior para obtener una máscara que dejará pasar los distintos valores de intensidad que podía tener la línea a pesar de las variaciones de iluminación.

4.2.4 Aplicación de la máscara a la imagen original

Una vez obtenida la máscara, esta se aplica a la imagen original, esto con el fin de aislar los segmentos de la línea, lo cual permitirá analizarla con mayor facilidad además de ahorrar tiempo de procesamiento de imagen. Para el sistema aquí desarrollado se optó por utilizar una función bitwise_and, la cual permite revisar si el pixel existe en la imagen y en la máscara, de ser así lo conserva, de lo contario, lo descarta. OpenCV provee la función cv::bitwise_and con la cual se calcula la conjunción de bits de dos arreglos.

Esta función se define de la siguiente manera:

bitwise_and(InputArray src1, InputArray src2, OutputArray dst, InputArray mask)

Donde los argumentos corresponden a dos imágenes de entrada, un arreglo de salida, y un arreglo que corresponde a la máscara a aplicar.

Y para cada elemento del arreglo de salida se define de la siguiente manera:

$$dst(I) = src1(I) \wedge src2(I) \;\; si\; mask(I) \neq 0 \tag{4.3}$$

Tomando la imagen recortada original como ambos elementos de entrada y aplicando la máscara, se descartan de la imagen original los pixeles no relevantes.

En las figuras 20 y 21 se presenta un ejemplo del resultado de este proceso sobre una superficie de color distinto a blanco, con lo que se puede ver que funciona para cualquier color de suelo, siempre y cuando se definan de manera correcta los valores de intensidad e iluminación.

Figura 20.- Imagen original.

Figura 21.- Imagen resultante tras aplicar la máscara.

4.2.5 Binarización de la imagen

Como se ha mencionado con anterioridad, el objetivo principal de este procesamiento es calcular un punto con coordenadas (x, y) correspondientes a la ubicación de la línea en un espacio 2D. El centroide de la línea bastará para localizarla y así indicar al Bebop en qué dirección tiene que desplazarse.

Para calcular el centroide se requiere hacer uso de los momentos de Hu. La función que provee OpenCV para calcular los momentos requiere que la imagen de entrada sea una imagen binaria, es decir, una imagen que sólo tiene 2 posibles valores para cada pixel.

Para lograrlo, se realiza un proceso de umbralización (o *threshold*, en inglés), el cual sirve para particionar imágenes en regiones basadas en valores de intensidad y/o propiedades de dichos valores. Para esto se utiliza la función *threshold* de OpenCV que se define de la siguiente manera:

threshold(InputArray src, OutputArray dest, double thresh, double maxval, int type)

donde los argumentos corresponden a una imagen de entrada, un arreglo de salida, un valor en escala de grises que se utilizará como el umbral, un valor para dar a los pixeles que cumplan las condiciones definidas y un modo de umbralización.

Dada la intensidad de pixel en un punto de la imagen denotada como *f(x, y)* se elige un umbral T tal que si $f(x, y) \geq T$ a este punto se le denomina punto de objeto, de lo contrario el punto es denominado punto de fondo. Por lo anterior, la umbralización está dada por:

$$g(x,y) = \begin{cases} 1 \; si \; f(x,y) \geq T \\ 0 \; si \; f(x,y) < T \end{cases} \tag{4.4}$$

Para el sistema aquí desarrollado, se utilizó el modo de umbralización THRESH_BINARY_INV el cual invierte los valores de los pixeles obtenidos y para los valores del umbral se utilizó el valor 60, buscando conservar todos los pixeles cuyo color es cercano a negro.

En la figura 22 se puede ver el resultado de la imagen recortada de la figura 17 una vez realizada la umbralización.

Figura 22.- Imagen recortada umbralizada e invertida.

4.2.6 Obtención del centroide utilizando los momentos de Hu

Una vez realizada la umbralización, el único paso restante es calcular el centroide. Como ya se mencionó en la sección anterior, para lograr esto se hace uso de los momentos de Hu.

Los momentos de una imagen consisten en promedios ponderados de las intensidades de los pixeles, tomando como entrada una imagen binaria. Estos capturan información de la forma de una mancha (*blob* en inglés) presente en una imagen binaria.

Los momentos espaciales de una imagen centrados respecto al origen se calculan de la siguiente manera:

$$m_{ij} = \sum_{x,y}\left(array(x,y) \cdot x^i \cdot y^i\right) \tag{4.5}$$

El centroide de una mancha es su centro de masa, sus coordenadas se calculan haciendo uso de los momentos espaciales m_{00}, m_{01} y m_{10}:

$$\bar{x} = \frac{m_{10}}{m_{00}} \tag{4.6}$$

$$\bar{y} = \frac{m_{01}}{m_{00}} \tag{4.7}$$

Para obtener los momentos, se utiliza la función de OpenCV moments(InputArray array, bool binaryImage=false)

Donde los argumentos son, un arreglo de entrada, el cuál debe ser binario, y un indicador de tipo booleano para definir si se trata de una imagen binaria.

Este regresa un arreglo con los momentos de Hu, con los cuales se calcula el centroide como se describió anteriormente. En la figura 23 se puede ver el centroide calculado de la imagen umbralizada, una vez obtenido este punto, es publicado al tópico correspondiente para que el nodo de transferencia de velocidades realice su función.

Figura 23.- Centroide calculado.

4.3 Nodo de control/Transmisión de velocidades

Una vez calculada la coordenada del centroide y publicada como un mensaje de tipo *geometry_ msgs*, se hace uso del nodo de transmisión de velocidades para enviar comandos al Bebop con el fin de que este se desplace hacia la posición calculada del centroide, manteniendo constante la altura a la que vuela el Bebop y manteniendo una velocidad de desplazamiento constante.

A continuación, se explica el funcionamiento del nodo de transmisión de velocidades:

Este nodo toma como entrada un punto con coordenadas (x, y) que representa la posición del centroide de la línea detectada en la imagen procesada y como salida envía velocidades a los motores del Bebop según la información de entrada.

La metodología con la que funciona este nodo consiste en definir las coordenadas correspondientes al centro del Bebop, estas se utilizan para determinar qué tipo de movimiento deberá realizar el Bebop de acuerdo con la posición del centroide de la línea. Posteriormente se definen regiones de coordenadas para las cuales la acción

requerida por el Bebop deberá ser avanzar o girar según sea el caso para finalmente enviar las velocidades correspondientes a través del tópico */bebop/cmd_ vel*.

En la figura 24 se muestra de manera gráfica una generalización del funcionamiento de este nodo, donde partiendo del centro de la figura con coordenadas (0, 0), se define una región en Y positivo para la cual el Bebop deberá desplazarse al frente (verde), regiones a las orillas para las cuales el Bebop deberá girar para posicionarse de tal manera que deba simplemente avanzar al frente (azul) y regiones a los lados para las cuales el Bebop deberá desplazarse a los lados (verde). La región en color rojo no corresponde a una coordenada válida, por lo que no es considerada.

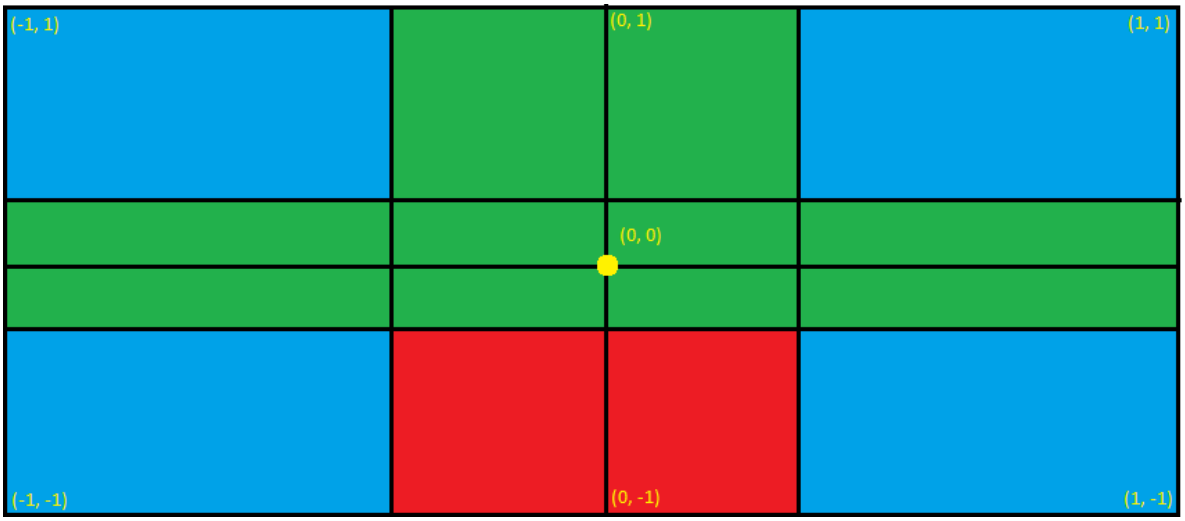

Figura 24.- Regiones definidas para el rango de movimiento.

Para definir el centro del Bebop, se hace uso de las propiedades de la imagen procesada, al ser un objeto del tipo cv::Mat, se puede acceder a sus propiedades como las dimensiones de alto y ancho en pixeles (*cols* y *rows* respectivamente). Se considera el punto medio de la imagen procesada como el centro del Bebop, a partir del cual se pueden definir fácilmente las regiones de coordenadas para determinar el movimiento requerido.

Para definir las regiones de coordenadas anteriormente mencionadas se considera el centro del bebop como punto de inicio, y se procede a establecer las regiones de la siguiente manera:

a) Regiones para las cuales el Bebop deberá desplazarse al frente (verde).
b) Regiones para las cuales el Bebop deberá desplazarse a los lados (verde).

c) Regiones para las cuales el Bebop deberá girar sobre su eje (azul).

El movimiento más sencillo es cuando la coordenada (x, y) del centroide se encuentra desplazada con dirección en la coordenada Y positiva sin desplazamiento en X, en este caso se considera que el punto está inmediatamente al frente y el Bebop solamente tendrá que desplazarse hacia él.

Otro caso a considerar es en el que la coordenada del centroide (x, y) se encuentra desplazada con dirección en la coordenada Y positiva y un ligero desplazamiento en X (positivo o negativo). El movimiento óptimo para el Bebop sería igualmente desplazarse al frente, ya que si se intenta girar al Bebop, este podría perder el punto o girar indefinidamente intentando centrar su coordenada X con la coordenada X del punto, esto último derivado de un retraso que hay entre el cálculo de la coordenada del centroide de la línea y el envío de las velocidades a los motores del Bebop.

El siguiente caso es cuando el movimiento óptimo sería desplazarse hacia los lados, esto puede suceder cuando el punto está desplazado una distancia considerable en la coordenada X (positivo o negativo) pero su desplazamiento en Y es mínimo, en este caso, girar podría causar que el Bebop pierda el punto o tarde más tiempo en centrar el punto en la coordenada X.

El último caso es cuando el movimiento óptimo para el Bebop es girar para posicionarse en una dirección en la que tenga que desplazarse al frente. Este caso se considera cuando la coordenada del punto (x, y) está desplazada en gran medida tanto en X como en Y.

Definidos estos casos de movimientos, se procede a definir las regiones de coordenadas para las cuales se realiza el movimiento que permita al Bebop posicionarse y desplazarse hacia el punto del centroide detectado.

Las velocidades se publican cada cuadro de video que captura la cámara del Bebop, enviando las velocidades correspondientes, según lo definido con anterioridad (*linear.x* positivo para avanzar, *angular.z* para girar o *linear.y* para desplazar).

5. Pruebas y resultados

Se realizaron varias pruebas con el sistema desarrollado, las primeras se realizaron para probar el nodo de procesamiento de imagen sobre superficies de cualquier color (ver figuras 25 y 26), y otras sobre una superficie color blanco, ambas con la línea de color negro.

Figura 25.- Prueba sobre un piso de azulejos.

Figura 26.- Línea eficientemente detectada.

Se pudo observar que, en algunos casos, el nodo podía discriminar eficientemente la línea del color de la superficie, siempre y cuando esta fuera lisa o no tuviera muchos detalles con colores dentro del rango definido para la línea, casos en los cuales el nodo corría el riesgo de interpretar otros elementos del entorno como segmentos de línea.

Un problema frecuente que se encontró fue en el que no se detectaban correctamente segmentos de la línea debido a la iluminación no uniforme en el entorno en conjunto con el material de la línea, en el caso de la figura 28 se aprecia que hay regiones dentro de la línea que no son detectados por el nodo por el problema de la iluminación.

En la figura 27 se puede observar que el nodo era capaz de calcular el centroide de la línea con exactitud sobre la imagen.

Figura 27.- Centroide de la línea detectada.

Las pruebas relevantes para el presente trabajo fueron aquellas con la línea ubicada sobre una superficie de color blanco. Se utilizaron dos lonas de color blanco sobre las cuales se construyó un circuito utilizando cinta adhesiva de color plateado tal como se ve en la figura 26.

Se pudo notar que en la mayoría de los casos la línea es detectada de forma correcta, con lo que se procedió a probar el nodo de transferencia de velocidades.

Por seguridad se creó un pequeño menú para despegar, iniciar y aterrizar el bebop. Además de esto, tal como se mencionó con anterioridad, siempre se mantuvo una terminal con el comando para realizar un aterrizaje de emergencia, puesto que en algunas ocasiones por errores durante el desarrollo un nodo podía terminar su ejecución y el Bebop tenía que ser aterrizado de emergencia.

La primera prueba que se realizó una vez construido el nodo de transmisión de velocidades, consistió en seguir una línea recta, y verificar que cuando el punto salía de vista del Bebop, este se detenía por completo, lo cual se cumplió sin muchas complicaciones.

Posteriormente se procedió a hacer un semi-circuito, esto con la intención de probar que el Bebop giraba en la posición indicada. Durante el tramo recto el Bebop seguía avanzando de forma correcta sin mayores problemas, sin embargo, el primer problema se presentó al momento de girar puesto que en algunas ocasiones el cambio repentino de velocidades en el Bebop junto con los problemas relacionados a la estabilidad en el aire, provocaba que el Bebop comenzara a retroceder aun cuando el nodo de transmisión de velocidades mandaba velocidades al Bebop para desplazarse al frente.

La última prueba consistió en hacer el circuito completo, donde se buscaba que el Bebop pudiera seguirlo indefinidamente. No en todas las pruebas se logró que el Bebop siguiera el circuito debido a los mismos problemas presentados en la prueba anterior durante el giro, además que por elementos de ruido externo los cuales fueron la iluminación no uniforme y las visibles uniones de las dos lonas, que en ocasiones por la iluminación el nodo de detección de línea consideraba como nuevos segmentos de línea. Sin embargo, en las pruebas que habían fallado en un punto, sí se posicionaba el Bebop donde había fallado y se volvía a realizar la prueba, este podía continuar el recorrido nuevamente hasta presentar los problemas mencionados anteriormente.

En las figuras 28 y 29 se puede ver al Bebop desplazándose correctamente al frente, en la figura 30 se puede ver al Bebop después de comenzar a girar sobre su eje para posicionarse con el centroide en la región definida para que avance al frente.

Figura 28.- Bebop en movimiento.

Figura 29. Bebop tras desplazarse al frente.

Figura 30. Bebop antes de iniciar un giro hacia la izquierda.

6. Conclusiones

Tras desarrollar los nodos que conforman el sistema de visión propuesto en este trabajo y realizar las pruebas se concluye lo siguiente:

- El dron pudo seguir una línea recta sin muchos problemas.
- El dron fue capaz de girar sobre su propio eje (Yaw) y desplazarse a los costados (en el eje Y).
- Se presentaron algunas complicaciones debido a la iluminación y ruido ocasionado por superficies distintas a la superficie blanca.
- Se presentaron complicaciones tras los giros sobre Yaw, pese que se le enviaban velocidades para avanzar sobre X positivo al Bebop, este se movía hacia X negativo.
- En algunas ocasiones, se presentaron problemas con las velocidades, puesto que no se mostraban constantes, pese a que se le enviaba siempre la misma velocidad, en ocasiones el bebop avanzaba muy rápido, y en otras tardaba el doble en recorrer el mismo segmento de recta.

El objetivo propuesto se pudo cumplir:

- Se pudo conectar el dron con una estación de trabajo en tierra.
- Se logró realizar el procesamiento de la imagen capturada por la cámara del dron para extraer información relevante.
- Se consiguió que el sistema desarrollado pudiera realizar el seguimiento de la línea, con algunas ligeras complicaciones.

En conjunto el procesamiento de la imagen junto con el nodo de transmisión de velocidades conforman el sistema de visión propuesto como el objetivo general.

Algunas áreas a mejorar incluyen:

- Refinamiento del nodo de procesamiento de imagen, para discriminar colores de forma más eficiente, permitiendo identificar objetos a seguir en todo tipo de superficies y de cualquier color.
- Mejorar la lógica del nodo de control, para lograr indicar de mejor manera en que momento el dron deberá girar o en qué momento deberá avanzar, sin importar el desplazamiento del centroide en X o Y.

- Profundizar en el mecanismo de vuelo del Bebop para corregir errores de estabilidad, y algunos problemas presentados en el vuelo (como el movimiento opuesto tras un giro).

Al ser la Agricultura uno de los puntos de especial interés que motivó el desarrollo del sistema, dado que las plantas se siembran en hileras que se pueden visualizar como líneas rectas, se requiere que el sistema sea capaz de detectar estas líneas rectas, del color más sobresaliente de las plantas, donde el piso, si bien es de color contrastante a las líneas de interés, no es un piso completamente liso, y puede haber elementos que causen más ruido en la imagen.

El aporte de este trabajo consiste en sentar las bases para desarrollar un sistema de visión más complejo, que permita al dron ser utilizado en los escenarios requeridos, así como para seguir desarrollando más programas de vuelo autónomo para drones en el Centro de Investigación en Ciencias.

ANEXO A

Pseudocódigo del sistema desarrollado.

En formato de pseudocódigo, se presenta cómo se construyeron los nodos que conforman el sistema de visión aquí propuesto.

Nodo de procesamiento de imagen:

```
Procedimiento ProcesarImagen(Img)
    RegionDeInteres <- Recortar(Img)
    HSVImg <- CambiarBGRaHSV(RegionDeInteres)
    Mascara <- CalcularRango(Img, ColorMin, ColorMax)
    DiscriminarColor <- AplicarMascara(HSVImg, Mascara)
    ImagenBinaria <- Umbralizar(DiscriminarColor, Valor, Color)
    MomentosHu <- CalcularMomentos(ImagenBinaria)
    Punto_x <- MomentosHu[1,0]/MomentosHu[0,0]
    Punto_y <- MomentosHu[0,1]/MomentosHu[0, 0]
    Publicar (Punto_x, Punto_y)
    Publicar (ImagenBinaria)
FinProcedimiento
```

Nodo de transmisión de velocidades:

```
Procedimiento TransmisionVel(Punto, Imagen)
    //Comentario: Punto = (Punto_x, Punto_y); Imagen = Imagen procesada
    // K = valor arbitrario, en el sistema se usó K=60.
    Centro_bebop_x <- Imagen.ancho/2
    Centro_bebop_y <- Imagen.alto/2
    Repetir
        Si Punto_y > 0 y Punto_x > 0:
            Si Punto_y > 0 y (Punto_x está entre Centro_bebop_x-K,
            Centro_bebop_x+K):
                Avanzar al frente
```

Si Punto_ y > 0 y (Punto_ x no está entre Centro_ bebop_ x-K, Centro_ bebop_ x+K):
Si Punto_ x < Centro_ bebop_ x-K:
Si Punto_ y está entre (Punto_ y-J, Punto_ y+j):
Desplazar a la izquierda
De lo contrario:
Girar a la izquierda
Si Punto_ x > Centro_ bebop_ x+K:
Si Punto_ y está entre (Punto_ y-J, Punto_ y+j):
Desplazar a la derecha
De lo contrario:
Girar a la derecha
De lo contrario:
Mantenerse suspendido
De lo contrario:
Mantenerse suspendido
FinProcedimiento

BIBLIOGRAFÍA:

[1] Craig, J. 2005. Introduction to Robotics. Pearson Prentice Hall.

[2] REAL ACADEMIA ESPAÑOLA: *Diccionario de la lengua española*, 23.ª ed., [versión 23.2 en línea]. <https://dle.rae.es> Consultado el 5 de octubre de 2018.

[3] Página web de ROS. Consultado el 6 de octubre de 2018. http://www.ros.org/.

[4] Blog 3D Insider https://3dinsider.com/drone-sensors/ Consultado el 14 de Marzo de 2020.

[5] Morcillo, Lucía. 2018. Sistemema de detección de obstáculos para drones basado en sensor láser. Escuela Técnica Superior de Ingeniería de Diseño, Universitat Politècnica de València.

[6] Blog DroneZon https://www.dronezon.com/learn-about-drones-quadcopters/9-heat-vision-cameras-for-drones-and-how-thermal-imaging-works/ Consultado el 14 de Marzo de 2020.

[7] Rodolfo, B. et al. 2006. Agricultura de precisión: integrando conocimientos para una agricultura moderna y sustentable. PROCISUR, Montevideo.

[8] Sitio official Arduino https://www.arduino.cc/en/guide/introduction Consultado el 14 de Marzo de 2020.

[9] Blog Rootsaid https://rootsaid.com/line-follower-robot-using-arduino/ Consultado el 14 de Marzo de 2020.

[10] Jonathan, C. et al. 2009. Visual Navigation of a quadrotor Aerial Vehicle.

https://ieeexplore.ieee.org/abstract/document/5354494.

[11] Tien Do, et al. 2015. Autonomous Flights through Image-defined Paths.

https://www-users.cs.umn.edu/~stergios/papers/ISRR-2015-Quadrotor_Visual_Servoing-Tien-Luis.pdf.

[12] Jesús, P. et al. 2013. Vision based GPS-denied Object Tracking and following for unmanned aerial vehicles.
https://ieeexplore.ieee.org/abstract/document/6719359 consultado el 5 de octubre de 2018.

[13] Sitio oficial Parrot, dron Bebop https://www.parrot.com/us/drones/parrot-bebop-2 Consultado el 5 de octubre de 2018.

[14] Repositorio Github de Parrot, Kit de desarrollo para drones Parrot https://github.com/Parrot-Developers consultado el 5 de octubre de 2018.

[15] Sitio oficial Github https://github.com/ consultado el 5 de octubre de 2018

[16] Repositorio Github, usuario Alexis Guijarro, ejemplos de programas para drones Parrot Bebop https://github.com/TOTON95/Bebop_ROS_Examples/ Consultado el 5 de octubre de 2018.

[17] Morales, L. y Paucar C. 2017. Investigación e implementación de un sistema de seguridad fijo y móvil mediante un dron, usando visión artificial para detección y seguimiento de personas en un ambiente externo específico de la Universidad de las Fuerzas Armadas ESPE-L, Universidad de las fuerzas armadas Ecuador.

[18] Página web de bebop_autonomy, https://bebop-autonomy.readthedocs.io/. Consultado el 8 de octubre de 2018.

[19] Página web de OpenCV. Consultado el 8 de octubre de 2018. https://opencv.org/

[20] Página web Wiki ROS. Consultado el 8 de octubre de 2018. http://wiki.ros.org/cv_bridge/Tutorials/UsingCvBridgeToConvertBetweenROSImagesAndOpenCVImages

[21] José G. 2017. Vehículos Aéreos No Tripulados del LANAMMEUCR: una herramienta multidisciplinaria adaptada para todo tipo de condiciones al servicio del país. Laboratorio Nacional de Materiales y Modelos Estructurales. Universidad de Costa Rica.

[22] Morales, E. y Sucar E. Introducción a la Robótica Móvil, Instituto Nacional de Astrofísica, Óptica y Electrónica INAOE: https://ccc.inaoep.mx/~esucar/Clases-ia/Laminas2014/intro-robotica.pdf. consultado el 8 de octubre de 2018.

[23] Mandado, E. et. al. 2018. Tabla de equivalencias entre el inglés y el español de términos de automatización, utilizados en control lógico, control de procesos, gestión de la tecnología, sensores, actuadores, comunicaciones digitales, comunicaciones industriales y confiabilidad. Real Academia Española.

[24] Quadcopter academy, http://www.quadcopteracademy.com/quadcopter-parts-what-are-they-and-what-do-they-do/. Consultado el 8 de octubre de 2018.

[25] RC Battery Guide: The Basics of Lithium-Polymer Batteries, https://www.tested.com/tech/502351-rc-battery-guide-basics-lithium-polymer-batteries/. Consultado el 8 de octubre de 2018.

[26] Como funcionan y vuelan los drones, http://inforepuesto.com/como-funcionan-y-vuelan-los-drones/. Consultado el 8 de octubre de 2018.

[27] Blog Midronedecarreras.

https://www.midronedecarreras.com/dron/porque-vuelan-los-drones/ Consultado el 8 de octubre de 2018.

[28] Página web Wiki ROS, ROS Concepts. http://wiki.ros.org/ROS/Concepts. Consultado el 10 de octubre de 2018.

[29] Documentación de ROS, sensor_msgs/Image Message. http://docs.ros.org/melodic/api/sensor_msgs/html/msg/Image.html

[30] Página web Wiki ROS, std_msgs, http://wiki.ros.org/std_msgs.

[31] Documentación de ROS, geometry_msgs/Twist Message. https://docs.ros.org/api/geometry_msgs/html/msg/Twist.html.

[32] Página web Wiki ROS, rosdep. http://wiki.ros.org/rosdep. Consultado el 10 de octubre de 2018.

[33] Página web Wiki ROS, catkin. http://wiki.ros.org/catkin. Consultado el 10 de octubre de 2018.

[34] Página web CMake, https://cmake.org/. Consultado el 10 de octubre de 2018.

[35] Página web Python, https://www.python.org/. Consultado el 10 de octubre de 2018.

[36] Página web de OpenCV, acerca de, https://opencv.org/about/. Consultado el 12 de octubre de 2018

[37] Documentación de OpenCV, cv::Mat Class Reference. https://docs.opencv.org/3.4/d3/d63/classcv_1_1Mat.html Consultado el 12 de octubre de 2018.

[38] Latoschik, M. 2005. *Color Models. Realtime 3D Computer Grapics / Virtual Reality.*

[39] Blog Pyimagesearch. *Convert URL to image with Python and OpenCV.* https://www.pyimagesearch.com/2015/03/02/convert-url-to-image-with-python-and-opencv/

[40] Documentación de OpenCV, *Miscellaneous Image Transformations.* https://docs.opencv.org/2.4/modules/imgproc/doc/miscellaneous_transformations.html Consultado el 12 de octubre de 2018.

[41] Gonzalez, R. y Woods R. 2008. Digital Image Processing. Pearson Prentice

Hall, New Jersey.

[42] Hu, M. 1962. *Visual Pattern Recognition by Moment Invariants.* IRE Transactions on Information Theory.

[43] Documentación de OpenCV, cv::Moments Class Reference. https://docs.opencv.org/3.4/d8/d23/classcv_1_1Moments.html Consultado el 12 de octubre.

[44] *Wolfram MathWorld, Central Moments.* http://mathworld.wolfram.com/CentralMoment.html Consultado el 12 de octubre de 2018.

[45] Documentación de OpenCV, cv::InRange. https://docs.opencv.org/master/d2/de8/group__core__array.html#ga48af0ab51e36436c5d04340e036ce981 Consultado el 14 de marzo de 2020.

[46] Documentación de OpenCV, cv::threshold. https://docs.opencv.org/master/d7/d1b/group__imgproc__misc.html#gae8a4a146d1ca78c626a53577199e9c57 Consultado el 14 de marzo de 2020.

[47] Página web de bebop_autonomy, Instalación.

https://bebop-autonomy.readthedocs.io/en/stable/installation.html Consultado el 8 de octubre de 2018

[48] Página web Wiki ROS, ROS Kinetic Kame. http://wiki.ros.org/kinetic. Consultado el 12 de octubre de 2018.

VOTOS DE APROBATORIOS

Presidente del Consejo Directivo del Instituto de Investigación en Ciencias Básicas Aplicadas de la Universidad Autónoma del Estado de Morelos.
P r e s e n t e .

Por medio de la presente le informamos que después de revisar la versión escrita de la tesis que realizó la C. **LOAEZA RODRIGUEZ IGNACIO GABRIEL** con número de matrícula **20154006387** cuyo título es:

“Robot Aéreo Seguidor de Línea usando Visión Artificial”.

Consideramos que **SI** reúne los méritos que son necesarios para continuar los trámites para obtener el título de **Licenciado Ciencias Área Terminal en Ciencias Computacionales y Computación Científica.**

Cuernavaca, Mor a 03 de diciembre del 2020

Atentamente
Por una universidad culta

Se adiciona página con la e-firma UAEM de los siguientes:

DR. JUAN MANUEL RENDON MANCHA	**PRESIDENTE**
DR. JORGE HERMOSILLO VALADEZ	**SECRETARIO**
DR. JORGE ALBERTO FUENTES PACHECO	**VOCAL**
DR. PAUL HERNANDEZ HERRERA	**PRIMER SUPLENTE**
DRA. LORENA DÍAZ GONZALEZ	**SEGUNDO SUPLENTE**

Se expide el presente documento firmado electrónicamente de conformidad con el ACUERDO GENERAL PARA LA CONTINUIDAD DEL FUNCIONAMIENTO DE LA UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS DURANTE LA EMERGENCIA SANITARIA PROVOCADA POR EL VIRUS SARS-COV2 (COVID-19) emitido el 27 de abril del 2020.

El presente documento cuenta con la firma electrónica UAEM del funcionario universitario competente, amparada por un certificado vigente a la fecha de su elaboración y es válido de conformidad con los LINEAMIENTOS EN MATERIA DE FIRMA ELECTRÓNICA PARA LA UNIVERSIDAD AUTÓNOMA DE ESTADO DE MORELOS emitidos el 13 de noviembre del 2019 mediante circular No. 32.

Sello electrónico

JORGE HERMOSILLO VALADEZ | Fecha:2020-12-03 14:34:43 | Firmante

PfPn7S5pXjwe4Mppng17/V0rKfoLZ7BWd1+X3GJV5Z5ZwEi5UbRcfOBCBX521MnuEBHp4AjX8MpECSoi0rs0mJ3kbU9QXfr1cjUC0JUqYdu6ucokvhAOHc1yIl0P5+b1E4AVDqvXB6eoZMvJTDVzbqkUVUrGCPr2DVky7QYYCMkEvgOf1hbt18pRNour903g5VT3kmkUh6sxeNx+7Q5slvJD4MXOC4CthrLT18m2BDjXTFlBQ/ZyYl2+7VFxCh+5aYT0sOsbByUJdRJp1GDLI7Azo51fnEkRMaMJTxkRJu8kk2g1uHni98rP52vQsS2VHF8JarCqIyC5klgnTzzI4Q==

JUAN MANUEL RENDON MANCHA | Fecha:2020-12-03 15:15:40 | Firmante

U2zRyepTMjsR1LesrC8y6exC+bOXIWMViiWboTXKdNTWf81ahI5bWRVB/6EaQknLjt3Mv4qurhIELNY9clcbkOiavs8klZwwKNaU6Qeet3Hi/iRKofBKtzpNWYm20tkTTCkWdfqgrTZAISCsGEtOUUTUbFLkFyavPZfeh6E5XneZnR3RuYN4RKDkxjGNhZqv1TBRgCKVG07QYq37ZUibSg8BxVLuixeKIgn6S17tWdzyBbOhHxmPyFFoLqCZq6mGAUDP2rfADfA7Omhiu9IRps1b0t+ueWihytKYsGVMlc0LW5jWkK68f9XmY1xEBYbqu4my/EKrrJGLNTDth5g18A==

JORGE ALBERTO FUENTES PACHECO | Fecha:2020-12-03 16:39:24 | Firmante

lQFNcS9tdjOCumMd7PwHqi8Jm6kZThgzdKh03HrZebwg1Cd4JYdbxpGr7cPbTecgcde6c/8VuuLGYCTaqFZcihYO9adqVAsKWHE2wkHrXG13OnY9/fvGDDfxS6fo4IFKyWMqHJFTb61QSwXGpdjRC/OJxxyIn+Qlt+bbO5v/8q0NdO2V2vPneSSXD0KsLwLCIldSm5p+Z86JJ0UErBYQYekNsUe1tZMhRSCt4pWj+ct/7NVvSvMhv3rcZcfq/ag/Uw5hkY2EZXVyD2BxmYRGsUxD7awXRLeVykcurj98UE0jffH2zwtsNIhe0QrPmAwXqSvhDxXowQny+R2lGOrFew==

LORENA DIAZ GONZALEZ | Fecha:2020-12-04 18:16:14 | Firmante

eJ0uk8OIiowetUqHaIMav6zhkcoMJVKVeJHvtTwGOcosx+1ZCvsFT4yVUo/J5qp/IC5s0JRAm/ophhUxSRakhVHVkGvf+Qv0b22oATwo4Bg7E++Qtit4irkOboaOh6t3aVHoyWHEpoiz2O8g9omzuv9p4u3zPHwTqQ8/btnjw5DofR0iR5hk+fW5/vFFOBTHDqWNKSh5EA7+lJoQOXfmikg8Ijlh4VRVT45APXoHjuy2enFRi03k2mXvCyWX4X2Et75yz0eGA3JCvHRHXx7o4AHAfLuKpEzKECbdh4b1qjIsqDbuFMyWlNbdxLJWBtw+HSlJRfW8weHqRoMwxlYkMQ==

Puede verificar la autenticidad del documento en la siguiente dirección electrónica o escaneando el código QR ingresando la siguiente clave:

GjLUma

https://efirma.uaem.mx/noRepudio/cV1ioOlF4QfYlJx5347IFABYZoAFt9Ai