UNIVERSIDAD AUTÓNOMA DEL ESTADO DE HIDALGO

INSTITUTO DE CIENCIAS BÁSICAS E INGENIERÍA

CENTRO DE INVESTIGACIÓN EN TECNOLOGÍAS DE INFORMACIÓN Y SISTEMAS

MAESTRÍA EN CIENCIAS EN AUTOMATIZACIÓN Y CONTROL

ATERRIZAJE DE UN MINI HELICÓPTERO DE CUATRO ROTORES EN UNA PLATAFORMA MÓVIL UTILIZANDO VISIÓN ARTIFICIAL.

T E S I S

QUE PARA OBTENER EL GRADO DE MAESTRO EN CIENCIAS EN AUTOMATIZACIÓN Y CONTROL

QUE PRESENTA:

BENJAMÍN NICOLÁS TRINIDAD

DIRECTORES DE TESIS:

DR. CARLOS CUVAS CASTILLO

DR. JORGE SAID CERVANTES ROJAS

PACHUCA HGO., MÉXICO 3 DE ENERO DE 2020

Benjamín Nicolás Trinidad

Aterrizaje de un mini helicóptero de cuatro rotores en una plataforma móvil utilizando visión artificial.

El presente trabajo, es fruto del esfuerzo no solo mio,
es también de mis padres y hermanos,
y sobre todo a mi amada esposa Esbeydi,
a quienes esta dedicado este trabajo,
a ellos quienes me han amado, educado y enseñado,
me han dado su comprensión, cariño y amor,
me han brindado los medios para concluir
de manera exitosa todos mis proyectos.
Gracias por cada uno de sus detalles.

Gracias a

CONACYT por la beca de Maestría otorgada durante el periodo Agosto 2017 - Julio 2019.

CITIS-UAEH, por la formación académica brindada durante el posgrado y el apoyo para la realización de la presente memoria.

Agradecimientos

Les agradezco a mis padres Soledad y Rogelio, a mis hermanos Rogelio, Esthela y Osvaldo, que con amor y tiempo me han dado lo mejor de sus vidas jamás terminaré de dar gracias por bendecirme por contar con una familia tan maravillosa como ustedes.

Pero principalmente debo agradecer a mi amada esposa Esbeydi que siempre me esperó con una abrazo y un beso, con comida caliente cuando llegaba y un café a las 3 de la mañana, por entenderme y tener la paciencia de la mujer más dulce del mundo, te amo y te amare por siempre, siempre te estaré agradecido, mi amor, siempre estaremos juntos, en las buenas en las malas y en las peores, siempre...

Agradezco a mis amigos de maestría, Eduardo, Jose Carlos, Omar y Rodrigo, por las experiencias que vivimos y por explicarme las cosas cuando no las entendía, por apoyarme cuando más lo necesite, como alguien muy apreciado me dijo en algún momento hoy por ti mañana por mi.

A otro amigo que lejos pero siempre presente, mi amigo Juan Francisco, que es como un hermano de trinchera, que gran parte de lo que sé, es gracias a él, por demostrarme que siempre tengo que dar lo mejor, y ser mi mentor para llevar y sacar un proyecto adelante, gracias infitas.

A mis asesores el Dr. Carlos y el Dr. Said que más que eso son mis amigos, que con su paciencia tuvieron el compartirme su conocimiento, sus consejos y enseñanzas.

A mis sinodales, por sus comentarios para mejorar mi trabajo, a los que me dieron clase y que amablemente comparten lo que a ellos les llevo años aprender.

UNIVERSIDAD AUTÓNOMA DEL ESTADO
Instituto de Ciencias Básicas e Ingeniería
School of Engineering and Basic Sciences
Área Académica de Computación y Electrónica
Department of Electronics and Computer Science

150 Años de Vida Institucional 1869-2019

Mineral de la Reforma, Hgo., a 27 de noviembre de 2019

Número de control: ICBI-AACyE/2397/2019
Asunto: Autorización de impresión de tesis.

M. EN C. JULIO CÉSAR LEINES MEDÉCIGO
DIRECTOR DE ADMINISTRACIÓN ESCOLAR DE LA UAEH

Por este conducto le comunico que el comité revisor asignado al Ing. Benjamín Nicolás Trinidad alumno de la Maestría en Ciencias en Automatización y Control, autoriza la impresión del proyecto de tesis titulado "Aterrizaje de un mini helicóptero de cuatro rotores en una plataforma móvil utilizando visión artificial", en virtud de que se han efectuado las revisiones y correcciones pertinentes.

A continuación, se integran las firmas de conformidad de los integrantes del jurado.

PRESIDENTE:	DR. OMAR JACOBO SANTOS SÁNCHEZ	
SECRETARIO:	DR. HUGO ROMERO TREJO	
VOCAL:	DR. CARLOS CUVAS CASTILLO	Carlos Cuvas C.
SUPLENTE 1:	DR. RAÚL VILLAFUERTE SEGURA	
SUPLENTE 2:	DR. JORGE SAID CERVANTES ROJAS	

Atentamente
"Amor. Orden y Progreso"

Dra. Liliam Rodríguez Guerrero
Coordinadora de la Maestría en Ciencias en Automatización y Control

Instituto de Ciencias Básicas e Ingeniería
Área Académica de Computación y Electrónica

LRG/APL

Ciudad del Conocimiento
Carretera Pachuca-Tulancingo km 4.5 Colonia Carboneras, Mineral de la Reforma, Hidalgo, México. C.P. 42184
Teléfono: +52 (771) 71 720 00 ext. 2250, 2251
Fax 2109
aacye_icbi@uaeh.edu.mx

www.uaeh.edu.mx

Resumen

En el tiempo de vida de la robótica, los robots aéreos no tripulados han sido utilizados en un amplio número de tareas, tales como reconocimiento en zonas de difi ´cil acceso, al momento de suscitarse desastres naturales o diferentes tipos de accidentes. Este tipo de vehículos también, son requeridos en fábricas para realizar actividades de mantenimiento en redes eléctricas, y en agricultura de precisión, para el monitoreo de plantaciones, siembra, entre otras. Dentro del ámbito ecológico y la agronomía, exploración de actividad volcánica y el descubrimiento de nuevos yacimientos arqueológicos; actualmente se están utilizando para detectar y combatir criaderos de mosquitos de transmisión del dengue. Debido a que esta tecnología se ha extendido en diversas áreas de la vida cotidiana del ser humano, se ha incrementado su demanda y por tanto se han reducido sus costos de fabricación.

Continuamente investigaciones en vehículos aéreos no tripulados (VANTs), para mejorar aspectos en autonomía, y estabilidad de vuelo, mientras el aterrizaje en superficies en movimiento es un tema con gran auge actualmente, al dar soluciones a esta problemática se daría pie a nuevas aplicaciones para los VANTs, anteriormente se han realizado aterrizajes con ayuda de sistemas con cámaras en ambientes controlados, y/o sistemas de anclaje entre robot terrestre y aéreo, de igual forma con ayuda de comunicación entre el vehículo en tierra y el vehículo a aterrizar.

En este trabajo de tesis, se propone que un VANT, específicamente un mini helic óptero de cuatro rotores, realice el seguimiento de una plataforma móvil, y aterrice sobre esta una vez que se ha detenido, mediante el uso de algoritmos de visión artificial y la implementación de técnicas de control clásicas.

Se plantea utilizar la información visual proveniente de una cámara monocular embebida en el vehículo aéreo para determinar su posición respecto a uno o varios puntos de interés a la plataforma móvil mediante algoritmos de visión artificial. Una vez conocida la posición del objeto de interés, se aplica una ley de control clásica para que el mini helicóptero sobre vuele esta, a través de la corrección del error de posición entre ambos vehículos corregir el error de posición entre el vehículo y la plataforma.

Acrónimos

VANT: Vehículo Aéreo no Tripulado (UAV, Unmanned Aerial Vehicle del inglés.).

CPU: Unidad Central de Procesamiento (Central Processing Unit del inglés).

ROS: Sistema Operativo Robótico (Robot Operating System del inglés).

ROM: La memoria de solo lectura (Read Only Memory del inglés).

SDK: Kit de desarrollo de software (Software Development Kits del inglés).

GPS: Sistema de Posicionamiento Global (Global Positioning System del inglés).

WIFI: Fidelidad Inalámbrica (Wireless Fidelity del inglés).

FPS: Fotogramas por segundo.

HSV: Hue, Saturation, Value del ingles (Matiz, Saturación, Valor).

HLS: Hue, Saturation, Lightness del inglés (Matiz, Saturación, Luminosidad).

HSI: Hue, Saturation, Intensity del inglés (Matiz , saturacion, Intensidad).

RGB: Red, Green, Blue del inglés (Verde, Rojo, Azul).

ORB: Oriented FAST and Rotated BRIEF.

SIFT: Scale-Invariant Feature Transform.

SURF: Speeded-Up Robust Features.

GLONASS: Global'naya Navigatsionnaya Sputnikovaya Sistema en Ruso, que se traduce como Sistema de Navegación Global por Satélite en español.

MD5: Es un algoritmo que proporciona un código asociado a un archivo o un texto concretos.

Glosario

- **Visión Artificial.** Es una disciplina científica que incluye métodos para adquirir, procesar y analizar imágenes del mundo real con el fin de producir información que pueda ser tratada por una máquina.
- **Sensor.** Un sensor es un dispositivo que está capacitado para detectar acciones o estímulos externos y responder en consecuencia.
- **Firmware.** Conjunto de instrucciones de un programa informático que se encuentra registrado en una memoria ROM
- **Aruco.** Librería dedicada al reconocimiento de marcadores en tiempo Real.
- **AprilTag.** Es un marcador de referencia que se utiliza para sistemas de visión
- **Hardware.** En informática, se puede definir como los componentes físicos, disco duro, placa madre, microprocesador, circuitos integrados, cables, sensores, etc.
- **Sensor Ultrasónico.** Dispositivo que mide la altura del suelo en un rango de 5 metros por medio de un pulso a altas frecuencias (Frecuencia 40Khz).
- **Presión.** Sensor que convierte la magnitud física presión en una seña eléctrica
- **Giroscopio.** Dispositivo electrónico que mide los movimientos de un dispositivo con un brazo de accionamiento
- **Acelerómetro.** Dispositivo electrónico responsable de detectar los cambios de orientación
- **Brújula Digital.** Dispositivo electrónico señalizar nuestra orientación respecto al campo magnético terrestre.
- **Cabeceo.** El eje lateral o transversal es un eje imaginario que se extiende de punta a punta de las alas del avión.
- **Alabeo.** El eje longitudinal es un eje imaginario que se extiende desde el frente a la cola del avión.
- **Guiñada.** El eje vertical es un eje imaginario que, pasando por el centro de gravedad del avión, es perpendicular a los ejes transversal y longitudinal. Este eje está contenido en un plano que pasa por el centro de gravedad desde arriba hacia abajo.
- **Linux™.** Es un sistema operativo libre tipo Unix, multiplataforma, multiusuario y multitarea.

- **Tipo UNIX.** Es un sistema que se comporta de manera similar a un sistema Unix, aunque no es necesario que sea certificado en ninguna versión.
- **Código Abierto.** El código abierto es un modelo de desarrollo de software basado en la colaboración abierta, se enfoca más en los beneficios prácticos de la colaboración
- **Rosbuild.** Antiguo compilador de ROS.
- **Cross-Compilling**. Compilación cruzada, cuando un mismo código se puede compilar en arquitecturas diferentes.
- **Variables de entorno.** Cadenas que contienen información acerca del entorno para el sistema y el usuario que ha iniciado sesión en ese momento.
- **Interprete de Código.** Es un programa que ejecuta scripts escritos en un lenguaje interpretado como Python y/o Java.
- **Script.** Es una serie de comandos que se almacenan en un archivo de texto y los cuales se caracterizan por presentar un tamaño muy pequeño.
- **Handle.** Los handles fueron una solución popular para el manejo de memoria en los sistemas operativos desarrollados en los años 1980, tales como Mac OS y Windows.
- **Callback.** En programación, una devolución de llamada o retrollamada (en inglés: callback) es una función .ª"que se usa como argumento de otra función "B". Cuando se llama a "B", ésta ejecuta .ª"
- **Matiz.** El matiz es la definición que se le da a la diferencia de un color a otro: permite clasificarlos en términos de rojo y sus matices, verde y sus matices, azul y sus matices.
- **Saturación.** En la teoría del color, la saturación, colorido o pureza es la intensidad de un matiz específico.

Índice general

Índice de figuras

Capítulo 1

Introducción

En este capítulo se presentan algunos antecedentes relacionados con los vehículos aéreos no tripulados, empleados en tareas de aterrizaje autónomo. También, se definen el problema a resolver, la justificación y el planteamiento de los objetivos, así como los alcances para este trabajo de tesis.

1.1. Antecedentes

Actualmente se han desarrollado diferentes técnicas y métodos para que vehículos aéreos no tripulados realicen aterrizajes sobre diferentes plataformas en movimiento, por ejemplo en [2], sobre un automóvil que cuando se desplaza a alta velocidad mayor a la que podrá alcanzar un robot terrestre, o en movimiento lento usando el sistema de posicionamiento global (GPS del inglés Global Positioning System). En [12] se emplean sistemas de visión en ambientes controlados provistos de un gran número de cámaras, proporcionan un primer ejemplo, usando un marcador visual con círculos concéntricos. Por otro lado en [28] utilizando la librería ArUco creada para permitir la estimación relativa de la posición y velocidad utilizando algoritmos de flujo óptico. Sin embargo, según este trabajo un problema se localiza un problema al utilizar flujo óptico cuando se la posición y velocidad del robot aéreo y el terrestre son similares. También se han realizado investigaciones donde es necesario que la plataforma móvil colabore con el vehículo aéreo para realizar el aterrizaje [16]. En[21] se implementó un algoritmo *blob color* el cual funciona para velocidades de menos de 1 m/s. En [13] se muestra que es posible usar sensores de bajo costo combinados con un marcador de referencia *AprilTag* similar a los códigos *QR* sin embargo en este caso las barras de dos dimensiones son de un mayor tamaño para que sean visibles a una mayor distancia por una cámara convencional y ejecutar un aterrizaje.

Se han utilizado diferentes VANTs, para fines de investigación, algunos fabricantes de vehículos aéreos, cuentan con modelos que cuentan con un kit de desarrollo, como

son DJI y Parrot, este último, con modelos que han sido ampliamente utilizados, como es el modelo ARdrone, y mas actualmente el Bebop 2 Drone, estos se han implementado en diferentes aplicaciones, navegación autónoma, usando lectores RFID esto sin uso de energía en la red de RFID [27], navegación por visión con múltiples drones[7][17], además de interacción con otros robots autónomos [9].

1.2. Planteamiento del problema

En trabajos previos se han diseñado diferentes metodologías para el aterrizaje de helicópteros de cuatro rotores, sobre superficies en movimiento, algunas soluciones proponen el uso de GPS [18], sin embargo estos tienen un margen de error de 3 a 15 mts (existen *GPS* de mayor precisión, pero, su costo es alto) en las mediciones. Por otro lado el uso de sistemas *OptiTrack* para determinar las localizaciones de la zona de aterrízaje y del vehículo, eleva los costos de implementación, de igual manera se limita el uso de este en ambientes estructurados.

1.3. Justificación

Realizar tareas de seguimiento y aterrizaje sobre plataformas móviles representa una tarea que se encuentra en constante evolución, actualmente es un tema de suma relevancia en el ámbito de los VANTs [1], [7], [18]. En la mayoría de estos trabajos se considera el uso de GPS para la localización del vehículo aéreo y/o sistemas de localización por sistemas de cámaras para rastreo de objetos. En este trabajo se propone resolver el problema de seguir la plataforma móvil y aterrizar posteriormente sobre ella una vez que se ha detenido, esto haciendo uso de un helicóptero comercial de 4 rotores.

1.4. Objetivo general

Realizar un seguimiento de una plataforma móvil y su posterior aterrizaje, implementando algoritmos de visión artificial, para realizar una tarea autónoma.

1.4.1. Objetivos particulares

1. Estudiar la estructura de programación de Robot Operating System, revisando la documentación actualizada, para poder desarrollar algoritmos propios.

2. Utilizar comandos de ROS para establecer comunicación con el vehículo aéreo "Bebop", a través del paquete bebop_autonomy.

3. Desarrollar un nodo en C++, para adquirir la imagen de la cámara embebida, utilizando la estructura de ROS.

4. Comunicar el control de XBOX ONE con el vehículo aéreo, desarrollando un nodo en C++ en la estructura de ROS, para el paneo horizontal y vertical de la cámara embebida, despegue, aterrizaje y movimiento traslacional del vehículo.

5. Adquirir la medición del vehículo, a través del desarrollo de un nodo de ROS, para implementar un control de altura.

6. Detectar un marcador de interés, mediante la implementación de algoritmos de visión (de segmentación de color y de detección de características), para propósitos de seguimiento.

1.5. Hipótesis

Una rutina autónoma utilizando algoritmos de visión artificial y leyes de control convencionales, realizará una tarea de seguimiento de una plataforma móvil y posteriormente el aterrizaje.

1.6. Metodología

Para la realización de este trabajo de tesis se sigue una metodología, en la que se hace uso de procesos sistemáticos, críticos y empíricos que se emplean para la resolución de nuestra hipótesis propuesta.

Figura 1.1: Metodología

Capítulo 2

Marco Teórico

En el presente capítulo se presenta el marco teórico necesario para realizar la tarea de seguimiento y aterrizaje.

2.1. Conceptos de vehículos aéreos.

En los últimos años, la tecnología se ha desarrollado a un ritmo bastante acelerado. La invención de las primeras aeronaves tripuladas, las redes de comunicación o la radiocomunicación. Han traído consigo múltiples avances, donde los Vehículos Aéreos No Tripulados son un claro ejemplo de ello. Los VANT's, comúnmente conocidos como drones, tienen la característica de no disponer de un piloto en el interior de la aeronave, por lo que ésta es dirigida por una persona o sistema electrónico externo, que decide en cada momento el siguiente paso a seguir [12]. La habilidad, los sentidos y la pericia del piloto son sustituidas por sensores electrónicos de gran precisión, que aportan una maniobrabilidad similar a la contribuida por todo el personal abordo. No se debe caer en el error de pensar que un dron y un vehículo teledirigido es el mismo aparato, puesto que la principal diferencia radica en que los drones son utilizados para tomar datos a través de diferentes sensores o sistemas de ayuda al vuelo y los vehículos teledirigidos no tienen incorporado ningún sensor [13]. Desde sus orígenes militares los drones han tenido la finalidad de aumentar la seguridad del ser humano por encima de la seguridad de las propias aeronaves, por ello los modos de empleo utilizados siempre han trasladado a las personas que las manejan, lejos de los lugares donde puedan sufrir algún daño [17]. Los primeros modelos radiocontrolados, permitían al piloto tener acceso a todos los datos de vuelo que tendría al estar en el interior del vehículo situándolo en un lugar lo suficientemente alejado que le garantizara no sufrir ningún daño en caso de aparecer algún peligro inmediato [10]. Con el paso de los años y fundamentalmente con la mejora e invención de nuevas tecnologías como el GPS, han ido apareciendo varios modos de utilización hasta llegar al actual modo autónomo,

que permite a un dron despegar, realizar cualquier intervención de forma cíclica, y aterrizar sin la intervención ni presencia humana.

2.1.1. Tipo de configuraciones de un VANT de cuatro motores.

Con el uso de cuatro motores se obtiene un vehículo mas simple desde el punto de vista mecánico, causando una mejor resistencia ante accidentes y una estabilidad más favorable. El control de los ángulos de cabeceo, alabeo y guiñada, son más visuales a causa de la geometría simítrica que presenta la configuración regular de cuatro propulsores.

Configuración "+" y configuración "x"

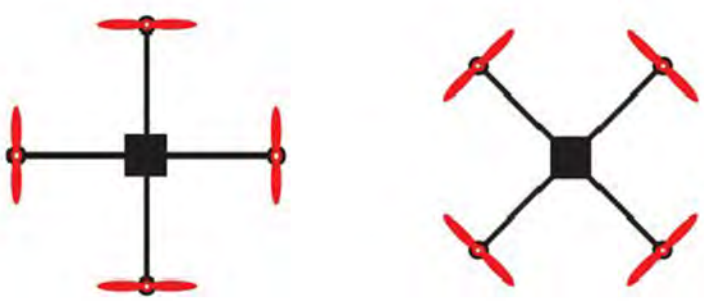

Figura 2.1: Configuración + y X en helicópteros de cuatro hélices.

La diferencia que radica entre ambos es que la configuración "+", Figura 2.1 izquierda, recrea los movimientos de ángulos de forma más directa sobre el control de los motores(debido a que están alineados con los mismos ejes del movimiento), mientras que la configuración en "X", Figura 2.1 derecha, necesita de la participación de los 4 motores para la orientación y movimiento hacia delante, atrás o costados, una de las ventajas que tiene la configuración en X, es que permite el uso de cámaras para filmación o sensores en dirección de avance, sin generar obstrucciones, además al tener que utilizar todos los motores, esto genera un balance de fuerzas más equilibrado.

2.2. Sistema Operativo Robótico (ROS)

Robot Operating System conocido como ROS por sus siglas en Inglés, es un entorno de trabajo que es ampliamente utilizado en robotica,es un software que permite la interacción de forma simple entre diferentes tipos de robots, para aplicar cualquier

tipo de procesamiento, provee las facilidades de un sistema operativo estandar como la abstracción de hardware de bajo nivel.

2.2.1. Uso y manejo de ROS

ROS usa un lenguaje mensajes descriptivos simples para describir los valores de datos que publica por medio de nodos. Con esta descripción, ROS puede generar el código fuente para ese tipo de mensajes a través de diferentes lenguajes de programación. ROS tiene una gran variedad de mensajes predefinidos, pero es posible desarrollar un nuevo tipo de mensaje, definido mediante un paquete personalizado. El funcionamiento del sistema ROS, se ilustra en la siguiente Figura 2.2.

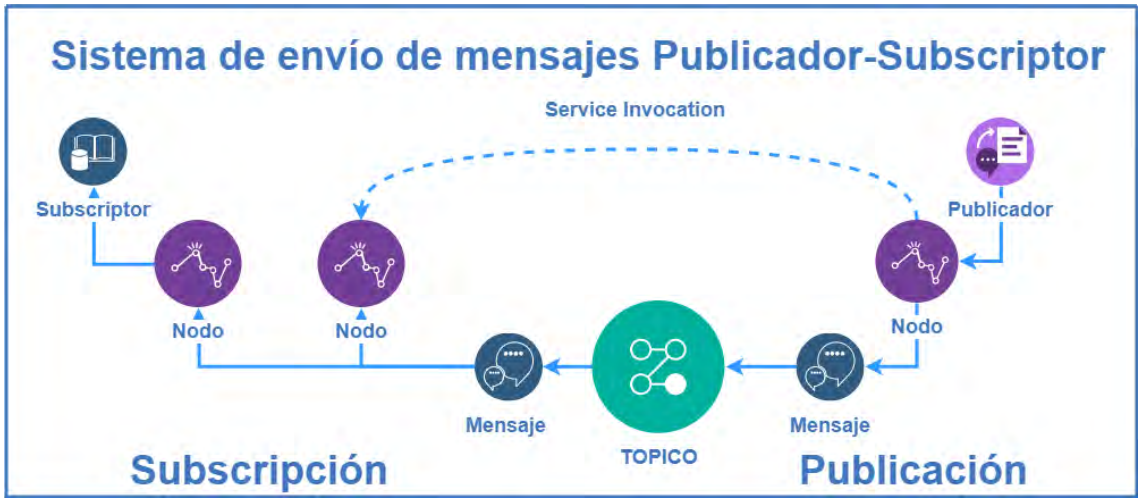

Figura 2.2: Esquema de funcionamiento de ROS

2.2.2. Catkin

Catkin es el sistema de construcción oficial de ROS y el sucesor del sistema original de construcción de ROS *rosbuild*. Catkin combina los macros *CMake* y los *scripts* de *Python*, para proveer de algunas de las mejores funcionalidades de ambos en un ambiente de desarrollo, catkin fue desarrollado para ser más convencional que *rosbuild*, permitiendo una mejor distribución de paquetes, mejor *cross-compilling*, y mejor portabilidad.

2.2.3. Nodos

Los nodos son ejecutables que pueden comunicarse con otros procesos usando tópicos, servicios, o los parámetros, el uso de nodos proporciona el desarrollo de un código

reutilizable para diferentes proyectos, cada nodo debe contener un nombre único dentro del sistema, este nombre lo utiliza el sistema operativo para realizar una comunicación entre otros nodos dentro del sistema evitando así las ambigüedades.

ROS contiene herramientas para que nos proporciona información acerca de los nodos que se encuentran ejecutando.

2.2.4. Tópicos

Los tópicos son buses de datos usados por los nodos para el envío de información, estos pueden transmitir información de manera indirecta entre los nodos. Cada tópico tiene un tipo de mensaje definido que es usado para publicar información a través de él, cabe destacar que si otro nodo se desea suscribir a él es necesario que contenga el mismo tipo de mensaje, cada nodo es capaz de soportar a varios suscriptores conectados simultáneamente.

2.2.5. Mensajes

Un nodo envía información a otro nodos por medio del uso de mensajes que son publicados por tópicos. Los mensajes tienen una estructura simple que pueden ser definidos por el usuario, los nombres de los mensajes son definidos por la siguiente convención: el nombre del paquete, seguido por una barra \, el nombre, finalizado por la extensión *.msg*, ejemplo: *std_msgsmsg\String.msg*.

2.3. Visión Artificial

Visión por computadora es la transformación de datos o aun más de un vídeo, en una nueva representación, donde todas las transformaciones están hechas para lograr un objetivo más específico. La entrada de vídeo incluye algo de información la ubicación de donde se encuentre la cámara, o la cantidad de fotogramas por segundo. Una nueva representación, puede significar cambiar una imagen en color a una representación en escala de grises, o remover el movimiento de una secuencia de imagen. Uno de los tantos problemas que pueden resolverse usando la visión artificial es el seguimiento y la detección de objetos de interés, con la finalidad de resolver tareas autónomas utilizando vehículos aéros. A continuación se revisaran los conceptos relacionados con detección de objetos utilizando diferentes algoritmos de visión artificial, como lo son algoritmos de detección por color y detección por homografia y correspondencia.

2.3.1. Detección de imagen por color.

Unos de los algoritmos mas utilizando para poder detectar objetos son los que esta basados en la detección de color, esto es debido a que el procesamiento necesario para realizar la detección es relativamente sencillo de implementar en comparación con los algoritmos de detección de formas o de detección de características propias de la imagen, a continuación de presentaran tres algoritmos para poder detectar color, HSV, Mean-shift y CAM-shift.

2.3.1.1. Características de modelo de color HSV

El espacio HSV, representa uno de los espacios de coordenadas más clásicos e intuitivos existentes en la literatura. Su interpretación geométrica viene determinada por un cono, Figura 2.3. Con esta interpretación del espacio de color, cada color trabaja con 3 componentes básicas; matiz, saturación y brillo [6] . El Matiz (Hue), hace referencia al valor de la cromaticidad o clase de color, la saturación (Saturation), se refiere a las longitudes de onda que se suman a la frecuencia del color, y determina la cantidad de blanco que contiene un color. Contra menos saturado este es un color contendrá más color blanco, y contra más saturado este un color la cantidad de blanco es menor. La saturación representa la pureza e intensidad de un color. Así la falta de saturación viene dada por la generatriz en la representación del cono HSV [19]. Esa falta de saturación representación la gama de grises desde el blanco hasta el negro. La iluminación Value, se corresponde con la apreciación subjetiva de claridad y oscuridad.

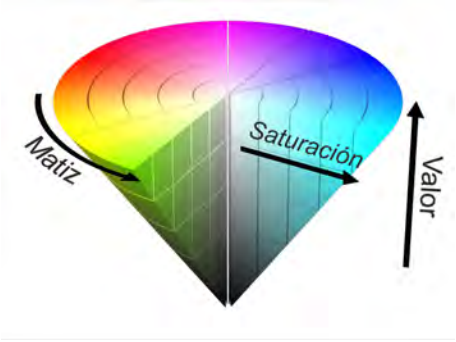

Figura 2.3: Modelo de color HSV

Cuando se requiere representar una imagen en color con un espacio de color HSV, es importante determinar cómo influyen las componentes de color de una cámara RGB sobre el espacio HSV. Así, el sistema HSV viene definido por las ecuaciones, (2.1),

(2.2), (2.3).

$$h_{HSV} = \begin{cases} \frac{g-b}{max(r,g,b)-min(r,g,b)} & si \quad r \quad = \quad max(r,g,b) \\ \frac{b-r}{max(r,g,b)-min(r,g,b)} & si \quad g \quad = \quad max(r,g,b) \\ \frac{r-g}{max(r,g,b)-min(r,g,b)} & si \quad b \quad = \quad max(r,g,b) \end{cases} \tag{2.1}$$

,

$$s_{HSV} \quad = \quad \frac{max(r,g,b) - min(r,g,b)}{max(r,g,b)} \tag{2.2}$$

,

$$v_{HSV} \quad = \quad max(r,g,b) \tag{2.3}$$

2.3.1.2. Detección de imagen por segmentación de color, en modelo HSV

La segmentación empleada en HSV se fundamenta en la detección de umbrales del mapa de componentes HSV. La umbralización de imágenes de objetos en entornos robotizados muchas veces es compleja por la falta de conocimiento a previo del número de objetos a detectar, por la influencia de elementos no deseados como sombras, brillos y la complejidad de colores, texturas y tamaños de los objetos y la posibilidad de solapamientos de estos, así como las variaciones en el fondo, esto hace que la elección de un número bajo de umbrales influya en la segmentación, se detectan menos regiones de las deseadas, o la elección de un número elevado lo haga en una sobre segmentación, se detectan más regiones de las que interesa [6]. Tanto una como otra, se puede deber a dos motivos principalmente, la aparición de sombras y brillos por un lado, y por otro por la similitud de los colores que tienen los objetos presentes en la escena principalmente.

El fundamento empleado para segmentar imagen es similar al usado en el espacio de color RGB. Así se transforma inicialmente, la imagen RGB a HSV, y se separan las tres componentes de color: Matiz, saturación, y valor de la iluminación, para analizarlos y trabajar con ellos.

Haciendo una primera aproximación, para detectar objetos de color en una imagen en entornos robotizados, se ha partido de la suposición que la falta de color en las imágenes adquiridas hace referencia principalmente a dos situaciones básicas zonas de sombras y zonas de brillos en la imagen; información que facilita el histograma bidimensional Valor/Saturación [23].

Es importante, sobre todo para las regiones de sombras que estas no sean añadidas como área de los objetos presentes en la imagen, y que por otro lado, no supongan que una región con distintas zonas de sombra sea segmentada como regiones distintas. De ahí que el espacio de color HSV proporcione una información de falta de cromaticidad más intuitiva y más esencia de segmentar.

2.3.1.3. Algoritmo Mean-Shift.

El cambio medio (del inglés Mean-Shift) fue propuesto por primera ver por fukunaga y hostetler [8], luego fue adaptado por cheng [3] para el análisis de imágenes y más recientemente por Comaniciu, Meer y Ramesh para problemas de visión de bajo nivel, incluyendo, segmentación [5], suavización adaptativa [4], y seguimiento [4].

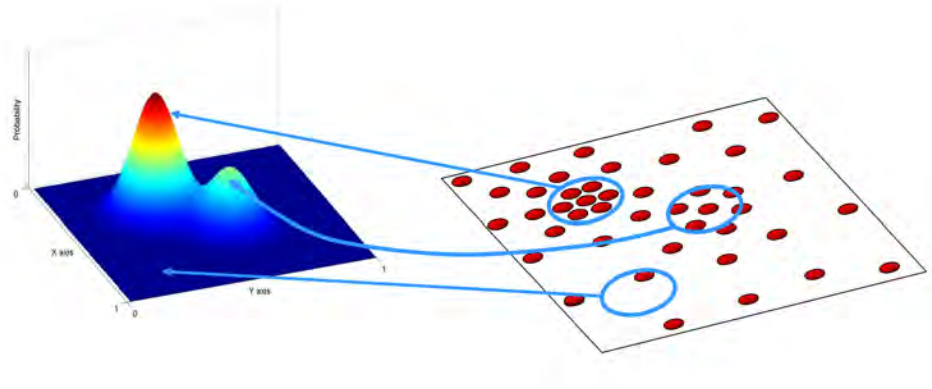

Figura 2.4: El algoritmo busca continuamente, dentro de una ventana de búsqueda local

La idea principal detrás del cambio medio es tratar los puntos en el espacio de la característica d-dimensional como una función de densidad de probabilidad empírica donde las regiones densas en el espacio de la característica corresponden a los máximos locales, como lo podemos ver en la Figura 2.4. Para cada punto de datos en el espacio de características, uno realiza un procedimiento de ascenso por gradiente en la densidad local estimada hasta la convergencia. Los puntos estacionarios de este procedimiento representan los modo de la distribución. Además, los puntos de datos asociados, con el mismo punto estacionario se considera miembros del mismo grupo.

2.3.1.4. Algoritmo CAM-Shift.

Cambio Medio Continuamente Adaptable (del inglés Continuously Adaptive Mean Shift), se diferencia del mean shift en que la ventana de búsqueda se ajusta en tamaño.

Los algoritmos de visión por computadora destinados a formar parte de un sistema de seguimiento, deben ser rápidos y eficientes. Por lo que se ha elegido el algoritmo Cambio Medio Continuamente Adaptable para realizar el seguimiento de un objeto que se haya seleccionado de un video obtenido. Este es una adaptación del algoritmo cambio medio, para poder tratar las distribuciones de probabilidad dinámicas (con cambios en su tamaño y en su posición), que representa los objetos en movimiento.

Todas las imágenes serán transformadas del modelo RGB a HSV ya que se utilizara los componentes de color (canal H del modelo HSV), para segmentar los objetos en el algoritmo Cambio Medio Continuamente Adaptable.

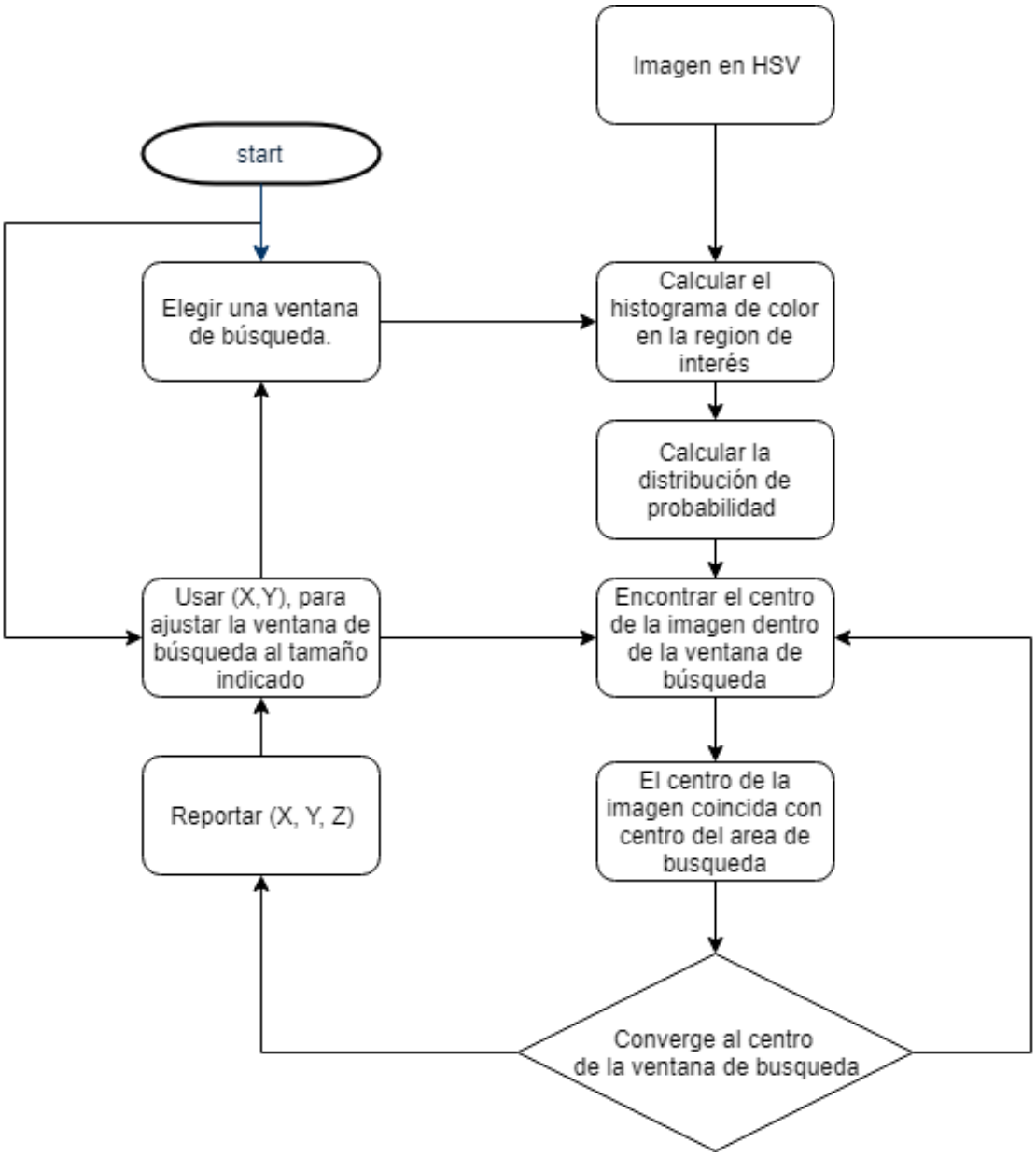

Figura 2.5: Diagrama de flujo, algoritmo “CAM-Shift”

2.3.2. Detección de imagen por homografía y correspondencia

Uno de los principales pasos para realizar el algoritmo de "homografía", es localizar los puntos de interés sobre la imagen, para los cuales se analizaron los algoritmos SIFT, SURF y ORB para la obtención de descriptores, Figura 2.6.

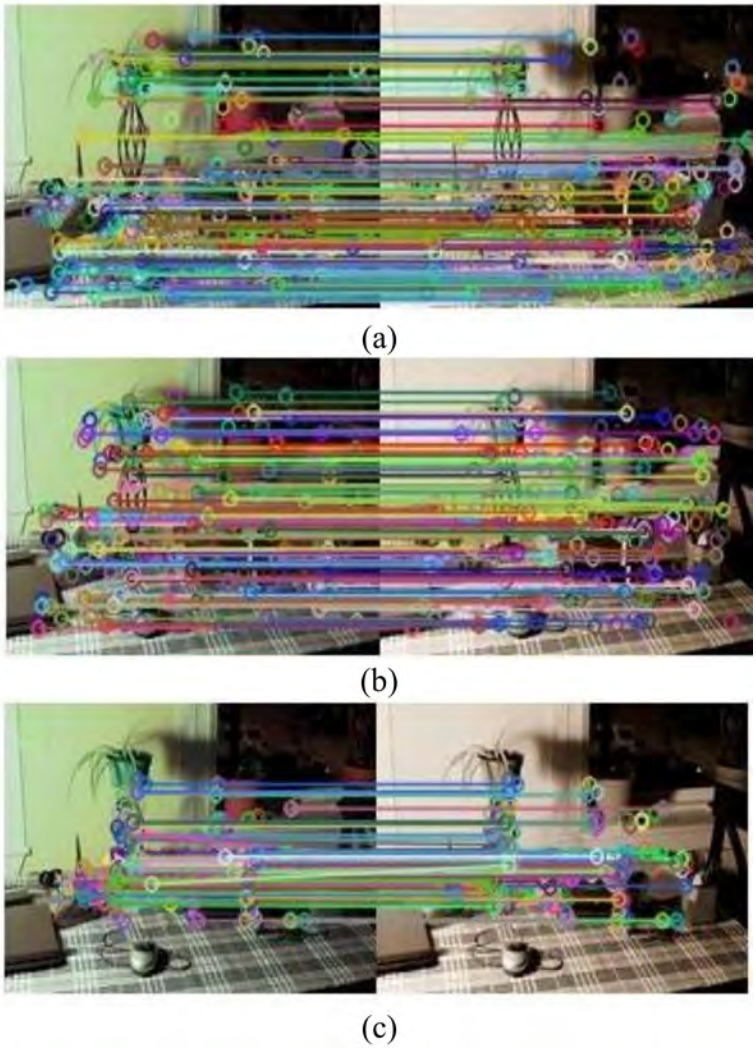

Figura 2.6: Descriptores con algoritmos a) Sift, b) Surf, c) Orb.

2.3.2.1. Algoritmo SIFT (Scale-Invariant Feature Transform)

Transformación de características invariante a la escala SIFT (del ingles Scale-Invariant Feature Transform), es un algoritmo que permite la identificación de similaridad entre imágenes, este es uno de los problemas en los sistemas de visión por

computadora, las características son invariables a la escala y la rotación de imágenes, están bien localizados tanto en el dominio espacial como en el de frecuencia, lo que reduce la probabilidad de interrupción por pérdida de enfoque, desorden o ruido [14]. Se pueden extraer grandes cantidades de funciones de imágenes típicas con algoritmos eficientes. Además, las características son altamente distintivas, lo que permite que una única característica se combina correctamente con una alta probabilidad en contra de una gran base de datos de características, proporcionando una base para el reconocimiento de objetos y escenas. El costo computacional de extraer estas características se minimiza al adoptar un enfoque de filtrado en cascada [11]. Las siguientes son las principales etapas de cómputo utilizadas para generar el conjunto de características de imagen:

- Detección de Extrema en el espacio de escala: La primera etapa busca en todas las escalas y ubicaciones de imágenes. Se implementa de manera eficiente mediante el uso de una función Gaussiana para identificar puntos de interés potenciales que son invariables a la escala y orientación.
- Localización de puntos clave: En cada ubicación candidata, se ajusta un modelo detallado para determinar la ubicación y la escala. Los puntos clave son seleccionados con base a las medidas de su estabilidad.
- Asignación de orientación: Se asignan una o más orientaciones a cada ubicación de punto clave según las direcciones de gradiente de imagen locales. Todas las operaciones futuras se realizan en datos de imagen que se han transformado en relación con la orientación, escala y ubicación asignadas para cada característica, lo que proporciona invariabilidad a estas transformaciones.
- Descriptor de punto clave: Los gradientes de imagen locales se miden en la escala seleccionada en la región alrededor de cada punto clave. Estos se transforman en una representación que permite niveles significativos de distorsión de la forma local y cambio en la iluminación.
- Coincidencia de puntos clave: los puntos clave entre dos imágenes se combinan mediante la identificación de sus vecinos más cercanos. Pero en algunos casos, la segunda coincidencia más cercana puede estar muy cerca de la primera. Puede suceder debido al ruido u otras razones. En ese caso, se toma la relación de la distancia más cercana a la segunda distancia más cercana.

Para la coincidencia y reconocimiento de imágenes, las características de SIFT se extraen primero de un conjunto de imágenes de referencia y se almacenan en una base de datos. Una nueva imagen se compara verificando individualmente cada

característica de la nueva imagen con esta base de datos anterior y encontrando las características coincidentes candidatas basadas en la distancia euclidiana de sus vectores de características.

Los descriptores de puntos clave son altamente distintivos, lo que permite que una sola entidad encuentre su coincidencia correcta con una buena probabilidad en una gran base de datos de características. Sin embargo, en una imagen desordenada, muchas características del fondo no tendrán ninguna coincidencia correcta en la base de datos, lo que dará lugar a muchas falsas coincidencias además de las correctas. Las coincidencias correctas se pueden filtrar del conjunto completo de coincidencias identificando subconjuntos de puntos clave que concuerdan con el objeto y su ubicación, escala y orientación en la nueva imagen [26]. La probabilidad de que varias características coincidan con estos parámetros es mucho menor que la probabilidad de que cualquier coincidencia de características individuales sea errónea. La determinación de estos grupos consistentes se puede realizar rápidamente utilizando una implementación eficiente de la estructura de datos de la transformada de Hough generalizada [25].

2.3.2.2. Algoritmo SURF

Características robustas aceleradas SURF (del ingles Speeded-Up Robust Features), es un algoritmo para la localización de puntos clave así como de su descriptores, las características principales que brinda el algoritmo son: invarianza ante la escala, orientación y distorsión afín así como invarianza parcial a los cambios de iluminación, el algoritmo se encuentra dividido en 4 etapas principales, las cuales son:

1. Imágenes integrales: es uno de los principales aportes que utiliza el algoritmo, debido a que contribuyen a una mejora en el funcionamiento. Dada una imagen de entrada y un pixel de esa imagen, la imagen integral es calculada como la suma de los pixeles comprendidos entre el punto y el origen.

2. Detector basado en la matriz Hessiana: se basa en el determinante de la matriz para la localización y escala de los puntos, se emplea por su rendimiento en la velocidad de cálculo y precisión.

3. Descriptor: es un vector de características que se calcula sobre una pequeña región de interés de la imagen, en el caso del descriptor SURF, describe como la intensidad de los pixeles se distribuye dentro de una escala dependiente de la vecindad de cada punto de interés que detectó la Hessiana.

4. Emparejamiento de puntos de interés: correspondencia de los puntos de interés identificados en dos imágenes.

Estas son algunas ventajas del algoritmo SURF:

- Velocidad de cálculo considerable superior sin ocasionar pérdida del rendimiento.
- Mayor robustez ante posibles transformaciones de la imagen.

2.3.2.3. Algoritmo ORB

Rotación breve y orientación rápida ORB(del ingles Oriented FAST and Rotated BRIEF), funciona tan bien como SIFT en la tarea de detección de características mientras que es casi dos veces más rápido. ORB se basa en el conocido detector de puntos clave FAST y el descriptor BRIEF. Ambas técnicas son interesantes para su estudio debido a su buen rendimiento y bajo costo computacional.

Dado un píxel p en una matriz, compara rápidamente el brillo de p con los 16 píxeles circundantes que están en un pequeño círculo alrededor de p como se puede ver en la Figura 2.7. Los píxeles en el círculo se clasifican en tres clases más claro que p, más oscuro que p o similar a p. Si más de 8 píxeles son más oscuros o más brillantes que p, entonces se selecciona como punto clave. Entonces, los puntos clave encontrados por FAST nos dan información de la ubicación de los bordes determinantes en una imagen

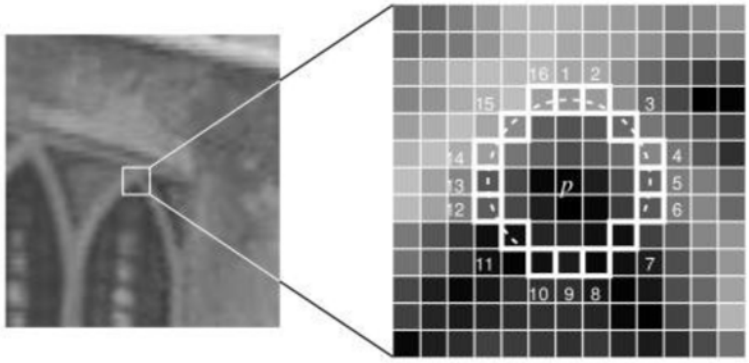

Figura 2.7: Selección de un punto clave.

Sin embargo, las funciones FAST no tienen un componente de orientación y funciones multiescala. Por lo tanto, el algoritmo ORB usa una pirámide de imagen de multiescala, Figura 2.8. Una pirámide de imagen es una representación multiescala de una sola imagen, que consiste en secuencias de imágenes, todas las cuales son versiones de la imagen con diferentes resoluciones. Cada nivel en la pirámide contiene la versión de la imagen con muestreo reducido que el nivel anterior. Una vez que ORB ha creado una pirámide, utiliza el algoritmo FAST para detectar puntos clave en la imagen.

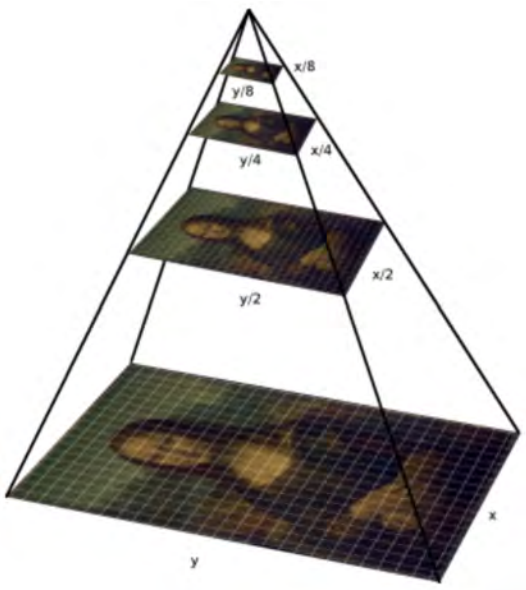

Figura 2.8: Localización de puntos clave y descriptores.

Después de ubicar los puntos clave, ORB ahora asigna una orientación a cada punto clave como hacia la izquierda o hacia la derecha dependiendo de cómo cambien los niveles de intensidad alrededor de ese punto clave. Para detectar el cambio de intensidad, Orb utiliza el centroide de intensidad. El centroide de intensidad supone que la intensidad de una esquina está desplazada de su centro, y este vector puede usarse para imputar una orientación.

En la Figura 2.9, se puede apreciar de una mejor manera como se realiza la rotación en el algoritmo ORB.

Figura 2.9: Localización de puntos clave y descriptores.

Una vez que hemos calculado la orientación del parche, podemos rotarlo y luego calcular el descriptor, obteniendo así invariancia de rotación.

Ahora para obtener los descriptores de los puntos clave se utiliza el algoritmo Brief toma todos los puntos clave encontrados por el algoritmo rápido y los convierte en un vector de características binarias para que juntos puedan representar un objeto. El

vector de características binarias también conocido como descriptor de características binarias es un vector de características que solo contiene 1 y 0. Cada punto clave se describe mediante un vector de características que es una cadena de 128–512 bits.

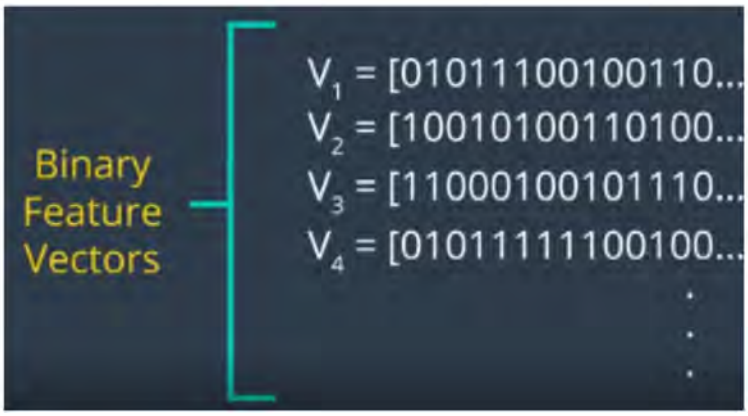

Figura 2.10: Localización de puntos clave y descriptores.

Posteriormente seleccione un par de píxeles aleatorios en un vecindario definido alrededor de ese punto clave. El vecindario definido alrededor del píxel se conoce como parche, que es un cuadrado de cierto ancho y alto de píxel. El primer píxel en el par aleatorio se dibuja a partir de una distribución gaussiana centrada alrededor del punto clave con una desviación. El segundo píxel en el par aleatorio se extrae de una distribución gaussiana centrada alrededor del primer píxel con una desviación estándar o extensión de sigma por dos. Ahora, si el primer píxel es más brillante que el segundo, asigna el valor de 1 al bit correspondiente más 0.

Figura 2.11: Localización de puntos clave y descriptores.

Para un vector de 128 bits, repita el algoritmo BRIEF 128 veces para un punto clave. BRIEF crear un vector como este para cada punto clave en una imagen. Sin

embargo, BRIEF tampoco es invariable para la rotación, por lo que Orb usa rBRIEF (BRIEF para la rotación). ORB intenta agregar esta funcionalidad, sin perder el aspecto de velocidad que caracterisa BRIEF.

ORB discretiza el ángulo a incrementos de 12 grados, y construye una tabla de búsqueda de patrones BRIEF pre-calculados. Mientras la orientación de los puntos clave sea consistente en todas las orientaciones, se utilizará el conjunto correcto de puntos para calcular su descriptor.

En cuanto a rendimiento, podemos comprobar que los descriptores ORB son una opción pertinente para su implementan en aplicación en tiempo real.

Figura 2.12: Localización de puntos clave y descriptores.

2.3.3. Comparación entre métodos para localización de descriptores.

Existen diferentes métodos para la localización de puntos clave y el calculo de sus descriptores, unos de los principales como se menciono en párrafos anteriores son *Transformación de características invariante a la escala*, *Características robustas aceleradas*, y *Rotación breve y orientación rápida*, se han hecho estudios comparativos de estos descriptores para determinar cual es el mas eficiente como en [24], en los cuales se determina que el algoritmo de rotación breve y orientación rápida en los 4 parámetros evaluados, que estos son rotación, manipulación, imagen borrosa, y desplazamiento.

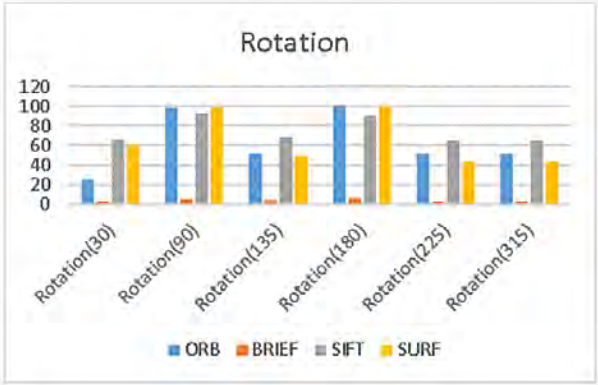

Figura 2.13: Porcentaje de aciertos con diferentes descriptores, en rotación.

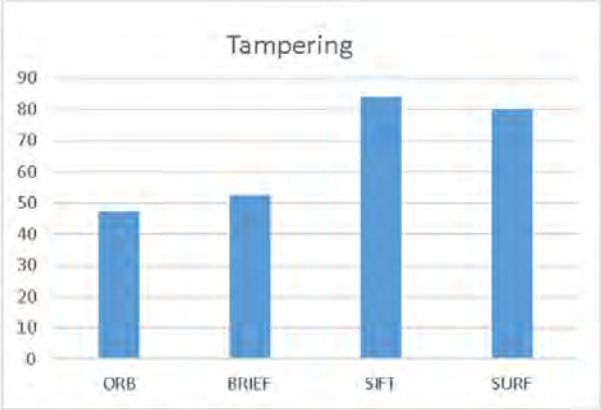

Figura 2.14: Porcentaje de aciertos con diferentes descriptores, en manipulacion.

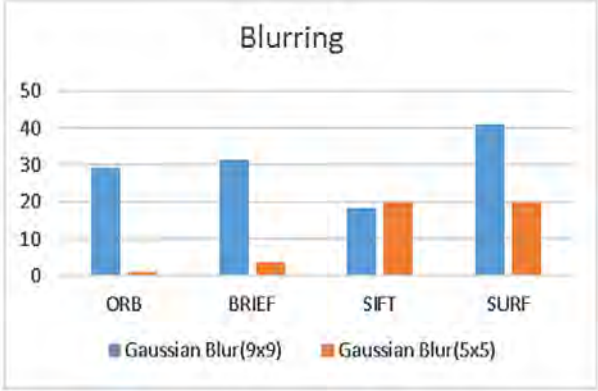

Figura 2.15: Porcentaje de aciertos con diferentes descriptores, en una imagen borrosa.

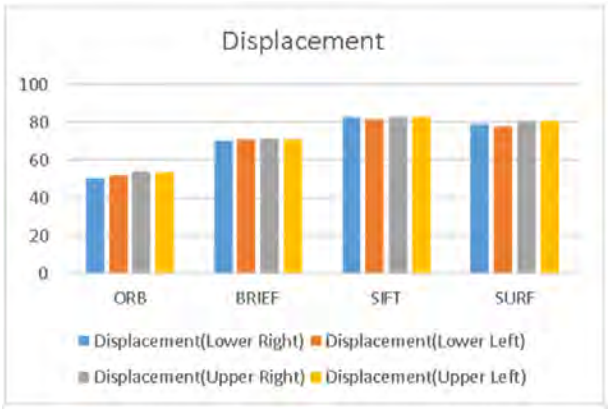

Figura 2.16: Porcentaje de aciertos con diferentes descriptores, en desplazamiento.

2.3.4. Algoritmo FLANN

Librería rápida para el Vecino más Vercano Aproximado (del Inglés FLANN - Fast Library for Approximate Nearest Neighbor), una de las tareas de mas importante para la obtención de un resultado correcto en el algoritmo de homografia, es definir la cercanía entre los puntos clave para ello existen diferentes opciones

Podemos definir el problema de búsqueda de cercanía entre los puntos de interés de la siguiente manera: dado un conjunto de puntos $P = p1$, p2,. . . , pn en un espacio métrico X, estos puntos deben pre-procesarse de tal manera que, dado un nuevo punto de consulta $q \in X$, encontrar el punto en P más cercano a q se pueda hacer rápidamente. El problema de la búsqueda del punto de interés más cercano es de gran importancia en una variedad de aplicaciones, como el reconocimiento de imágenes, la compresión de datos, el reconocimiento y clasificación de patrones, el aprendizaje automático, los sistemas de recuperación de documentos, las estadísticas y el análisis de datos [15]. Sin embargo, resolver este problema en espacios de alta dimensión parece ser una tarea muy difícil y no existe un algoritmo que funcione significativamente mejor que la búsqueda estándar de fuerza bruta. Esto ha llevado a un interés creciente en una clase de algoritmos que realizan búsquedas aproximadas de vecinos más cercanos, que han demostrado ser una aproximación lo suficientemente buena en la mayoría de las aplicaciones prácticas y, en la mayoría de los casos, órdenes de magnitud más rápidas que los algoritmos que realizan las búsquedas exactas. FLANN (Fast Library for Approximate Nearest Neighbors) [20] es una biblioteca para realizar búsquedas rápidas aproximadas a puntos de interés más cercanos. *FLANN* está escrito en el lenguaje de programación C ++. *FLANN* se puede utilizar fácilmente en muchos contextos a través de los enlaces de *C* , *MATLAB* y Python que se proporcionan con la biblioteca.

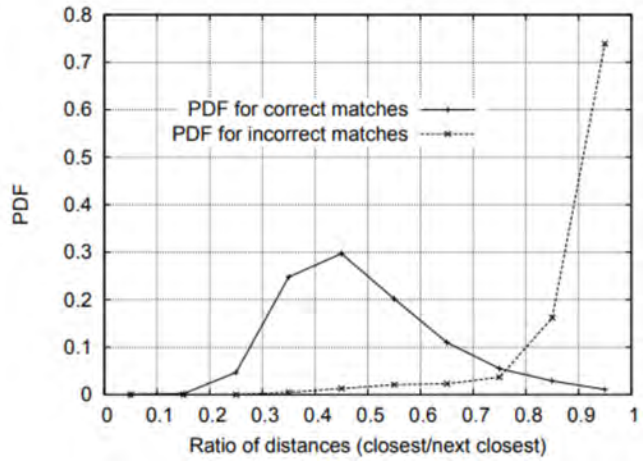

Figura 2.17: Distancia optima entre puntos característicos para identificación de homografia.

Capítulo 3

Desarrollo

En este capitulo se abordaran principalmente la plataforma experimental que se utilizara en las pruebas experimentales así como también los algoritmos de visión artificial a través de resultados experimentales obtenidos de la implementación de los mismos con finalidad de resolver el problema de la plataforma móvil.

3.1. Plataforma robótica

Para realizar las pruebas experimentales de detección y seguimiento de objetos se utilizara una plataforma compuesta por un vehículo aéreo de cuatro rotores, el cual puede ser manipulado en su movimientos de translación y rotación utilizando un paquete dedicado para la plataforma de ROS, a continuación presentamos sus características más importantes.

3.1.1. Parrot Bebop 2.

Parrot™Bebop 2 Drone®, Figura 3.1, es un vehículo aéreo no tripulado de cuatro rotores, se listan las características de dispositivo empleado, fuente *parrot.com.*

Sensor	Descripción
Óptico	Sensor óptico de 14 Mega-pixeles vídeo en formato 1080p a 30 fps.
Ultrasónico	Mide la altura del suelo en un rango de 5 metros(Frecuencia 40Khz).
Presión	Convierte la magnitud física presión en una seña eléctrica
Giroscopio	Mide los movimientos de un dispositivo con un brazo de accionamiento
Acelérometro	Responsable de detectar los cambios de orientación
Brújula Digital	Señalizar nuestra orientación respecto al campo magnético terrestre.
GPS y GLONASS	Sistema de Navegación Global por Satélite

Tabla 3.1: Sensores de Parrot Bebop 2 Drone

Figura 3.1: Bebop vista "explosionada"

cabe señalar que este vehículo aéreo ya cuenta con un control de orientación que le permite permanecer en vuelo estacionario, en este sentido el trabajo propuesto en esta tesis no se ocupara de este problema.

3.1.2. Comunicación con el vehículo a través de ROS

Para comunicarse con Bebop es necesario esta conectado a la misma red de datos, con los protocolos 802.11 b/g/n/ac a las frecuencias de $2.4Ghz$ o $5Ghz$, es posible establecer una contraseña para la conexión de red, en estas pruebas se mantuvo la configuración por defecto, con una conexión de red abierta.
Para lograr la interacción entre el VANT *Parrot Bebop 2 Drone*, ha sido necesario realizar una serie de tareas que comienzan con la familiarizacíon con el vehículo aéreo

no tripulado. Para iniciar, con la interacción es necesario contar con un paquete de ROS denominado *Bebop Autonomy*, con ayuda de este se obtienen los archivos necesarios para la interacción con el vehículo, contiene mensajes que son enviados a través de los diferentes tópicos que vienen cargados dentro del mismo paquete por esta parte los que fueron empleados fueron los siguientes:

- \bebop\takeoff
- \bebop\land
- \bebop\reset
- \bebop\cmd_vel
- \bebop\camera_control
- \bebop\start

Para el envío de mensajes es posible enviar por línea de comandos directamente desde una terminal de linux, se probó el envío de mensajes para lograr su comprensión completa, se hicieron pruebas con los diferentes mensajes para el despegue y el aterrizaje en un ambiente estructurado.

Para el aterrizaje se usa el siguiente comando \bebop\takeoff, cabe señalar que la altitud máxima a la que se probó con mensajes enviados desde una *PC* es de 50 metros.

El aterrizaje se realiza con el mensaje antes mencionado, después de hacer pruebas con estos, se comenzaron a probar los siguientes mensajes, debido a que pueden ser de riesgo el uso de esos mensajes, para la seguridad del usuario o del mismo *hardware*, se llega a la conclusión de que es necesario hacer pruebas desde un código desarrollado en *C++*, para tener un mayor control en el aterrizaje y despegue además de la velocidad a la que se desplaza el vehículo.

3.2. Control manual del vehículo

En esta sección se presentara el desarrollo de los controladores manuales utilizados para modificar la orientación de la cámara así como de los utilizados para controlar la posición translacional *(X,Y,Z)* y de rotación con respecto al eje **Z** de vehículo aéreo.

3.2.1. Movimiento de traslación y orientación mediante un control manual

El control de "Xbox One", Figura 3.2 es un control alámbrico, que puede ser integrado mediante los controladores adecuados al sistema operativo linux, lo que nos permite leer las salidas de este.

Figura 3.2: Control de Xbox One Alámbrico

Para la realizar detección del control de Xbox One por parte del sistema operativo UBUNTU, es necesario que se ejecute el siguiente comando:

Después de esto se requiere que se configure la posición de los controles ya que por defecto los joysticks están intercambiados de posición en los ejes 2 y 3 por los ejes 4 y 6, Figura 3.3.

Cada eje analógico tiene una resolución en cada conversor analógico de 16 bits, por lo que la respuesta es precisa en un rango de -32767 a 32767, los movimientos de cada eje así como la rapidez de respuesta de cada botón es de $5ms$, se realiza la lectura, con una cantidad de 10 botones, y 7 ejes de movimiento.

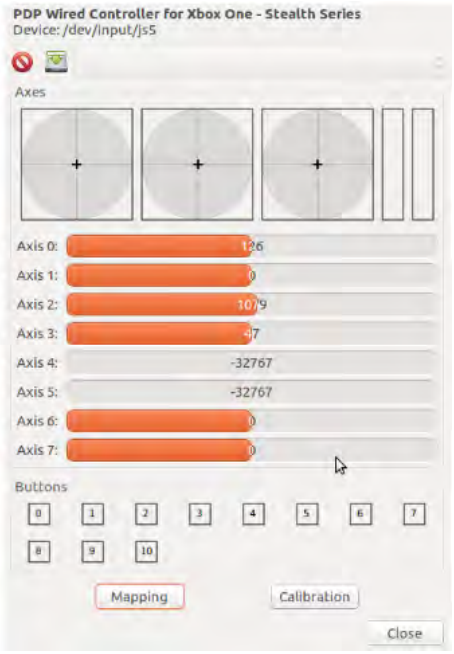

Figura 3.3: Configuración de ejes del control de Xbox One

Para realizar la lectura, de valores de los ejes del control de Xbox One, se realiza un código de*c++* que es ejecutado bajo el núcleo de ROS para la obtención de datos del control, dicho código obtiene y caracteriza los valores de lectura del control y los interpola a un rango de -1 a 1, se desarrolla una interpolación lineal.

- benjamin@benjamin-HP: $ rosrun custom_controller xbox_controller_node
- [INFO] [1541980087.184160922]: Opened joystick /dev/input/js5. deadzone_ 0.050000.

Obteniendo los siguientes resultados.

- ejes [-0.0, -0.012517407536506653, -0.0, -0.0, 0.0, 0.0, 0.0, 0.0]
- botones [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

Para hacer esto se realiza una integración de la lectura de datos con el control de Xbox y el envío de mensajes a través de tópicos creando publicadores, enviando así la información requerida a *Bebop*.

Se logra el control manual interactuando directamente con el control de Xbox One y Bebop, aquí se unen las lecturas de ejes del control de Xbox One la publicación de mensajes a través de tópicos, además de la adquisición de imágenes para generar pruebas de vuelo y de envió de transimición de datos.

3.2.2. Adquisición de imágenes por medio de *image_view*

ROS contiene, un visor de imagen que con el que es posible visualizar una secuencia de vídeo, que consiste en un subscriptor que recibe mensajes de un tópico, en este caso */bebop/image_raw*, en este caso el objetivo principal es obtener la imagen que es obtenida por medio del sensor del dispositivo *bebop*.

```
rosrun image_view image_view image:=/bebop/image_raw
```

Es señalable que para hacer la ejecución del subscriptor es necesario iniciar el núcleo de ROS con el comando *roscore*.

3.2.3. Orientación de la Cámara del vehículo mediante un control manual

Después de obtener la transmisión de video es necesario poder orientar la cámara, por primera instancia lo realizaremos de forma manual esto con un control de Xbox One de tal manera que logremos orientar a cualquier posición nuestra cámara, debido a que el seguimiento de un marcador sea posible hacerlo desde cualquier posición, para ello es factible hacerlo mediante la publicación de un mensaje a través del tópico $/bebop/camera_control mismo que se incluye dentro del driver de bebop en el paquete *Bebop Autonomy*, el tópico acepta mensajes de tipo *Twist*, enviando los movimientos requeridos, en las coordenadas lineales y angulares, se debe señalar que para la orientación se deben enviar en los valores de las coordenadas angulares de **'Y'** y **'Z'** en un rango de -80 a 80 para en **'Y'** y de -35 a 35 en **'Z'**, para esto se realiza un código en *C++*, para hacer un publicador.

```
ros::NodeHandle nh_camera_bebop;
ros::Publisher pub_bebop_camera;
```

Se crea un objeto de tipo *Twist* de la clase geometry_msgs.

```
geometry_msgs::Twist bebop_camera_msgs;
```

```
pub_bebop_camera = nh_camera_bebop.advertise<geometry_msgs::Twist
>("bebop/camera_control",1);
```

Leemos y asignamos los valores a las variables *bebop_camera_msgs.angular.y* provenientes de la lectura de los ejes 6 y 7 del control de Xbox One.

```
bebop_camera_msgs.angular.y = Interpolacion(joy_msg.axes[7],-1,1,-80,80);
bebop_camera_msgs.angular.z = Interpolacion(joy_msg.axes[6],-1,1,35,-35);
```

Y finalmente se hace la publicación a través del método *Publish*.

```
pub_bebop_camera.publish(bebop_camera_msgs);
```

3.3. Adquisición de imágenes y programación de algoritmos de visión

En este capítulo se presentan los métodos para obtener las imágenes provenientes de la cámara embebida de Bebop, y los algoritmos necesarios para efectuar el movimiento requerido en el enfoque del vídeo.

3.3.1. Conversión de mensajes de tipo imagen a imágenes de OpenCV

En primer lugar se obtienen las imágenes de la cámara para poder procesarlas implementando algoritmos de visión con ayuda de OpenCV, para ello se utiliza el suscriptor /bebop/image_raw, que retorna la imagen en formato *RAW*, misma que se utiliza para hacer un procesamiento de imagen posteriormente.

```
frame = self.bridge.imgmsg_to_cv2(data, "bgr8")
```

3.4. Algoritmos para la detección y seguimiento visual de la plataforma de aterrizaje.

En esta sección se presentara lo resultados experimentales al aplicar los algoritmos explicados en el capitulo 2, utilizando diferentes objetos de interés.

3.4.1. Detección de objetos usando el espacio de color *HSV*

El funcionamiento de este bloque se basa en capturar cada imagen dada por la fuente de vídeo y realizar sobre ellas distintas operaciones. Con esto se genera una nueva imagen apta con lo cual es posible realizar el seguimiento con esta nueva información, después de esto se pueden implementar operaciones morfológicas que permitan segmentar la información de color de esta nueva imagen.
La Figura 3.4 representa de una manera ordenada toda la secuencia del procesado:

1. Primero se captura la imagen del vídeo, esta será el único dato para el algoritmo Figura 3.4.

2. Esta imagen está representada en RGB, por lo tanto el siguiente bloque se encarga de convertirla al modelo HSV, y también se separa cada canal para poder operar sobre ellos por separado. Para los siguientes bloques solo se usarán los canales S y V.

3. Para cada canal se genera un fondo que posteriormente pasará a ser utilizado en la segmentación, Figura 3.7.

4. Para esta segmentación, se toma la imagen de cada cal y el correspondiente fondo generado y con ello se sustraen las personas en movimiento de la imagen.

5. Con la nueva imagen binaria obtenida de la segmentación, se aplican las operaciones morfológicas, erosión y dilatación para mejorar el resultado.

6. Una vez que se tiene la imagen dilatada de cada canal, se solapan formando una única imagen, también binaria. Con esto, ya se tiene la imagen objetivo para realizar el seguimiento.

7. Este algoritmo se realizará para todas las imágenes del vídeo.

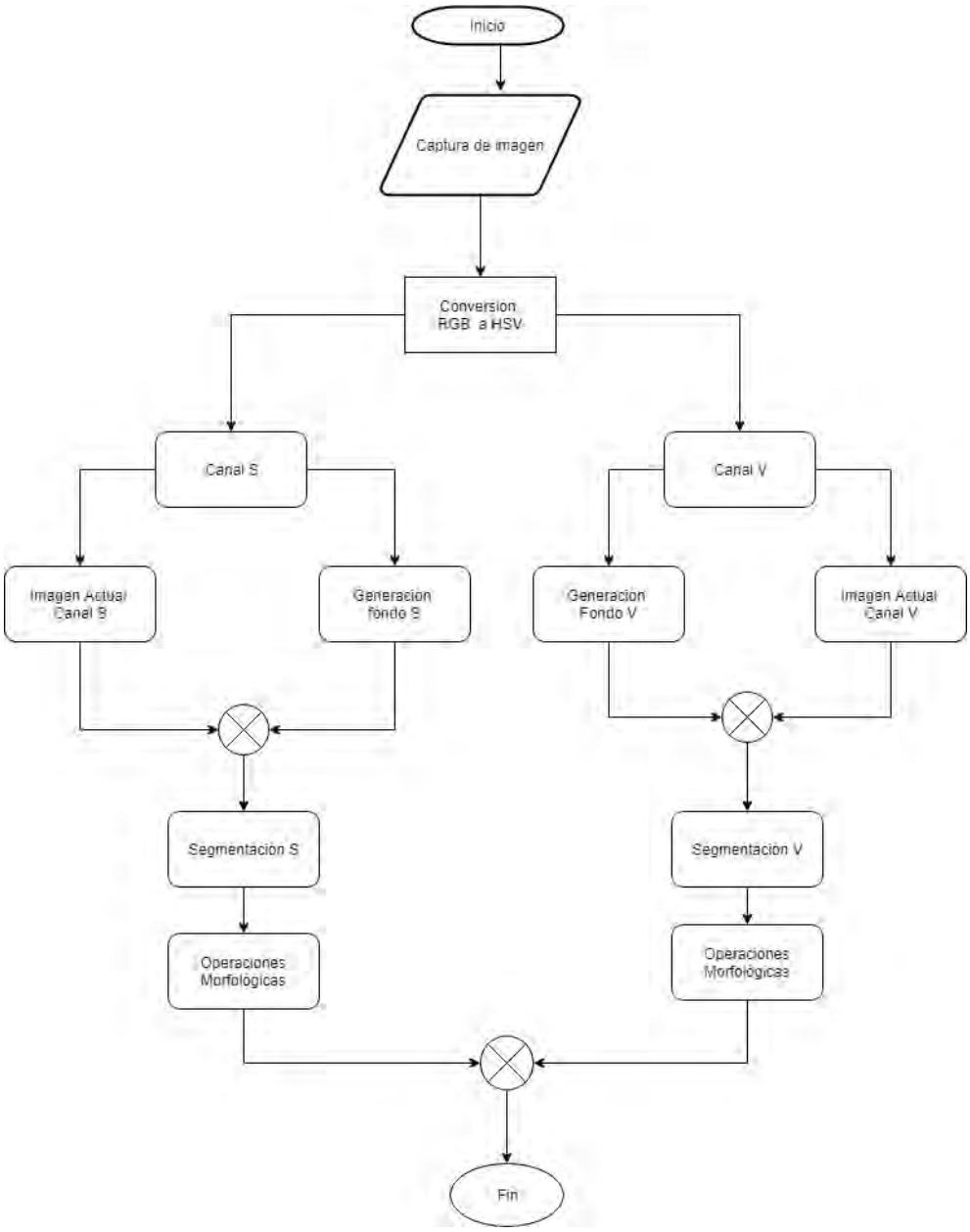

Figura 3.4: Diagrama de flujo discriminación de colores HSV

Las imágenes de entrada que se reciben generalmente vienen en formato de color RGB, es decir, en tres canales solapados de colores rojo, verde, y azul. Es esta aplicación, el objetivo es reconocer un marcador de un color en específico, debido a esto el RGB no resulta del todo útil, por lo que se opta por cambiar ese esquema HSV (Hue, Saturation, Value), Figura 3.6.

El uso de este modelo de color tiene la ventaja de que todos los colores están agrupadas en un único canal por lo que facilitará el reconocimiento de los marcadores en el espacio de visión de la cámara basándose en la cantidad de brillo y saturación de los colores detectados, Figura 3.5.

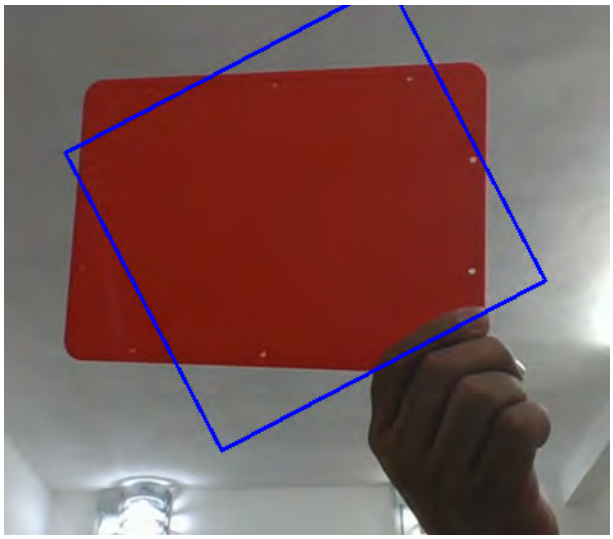

Figura 3.5: Seguimiento de objeto color rojo, por segmentación de color

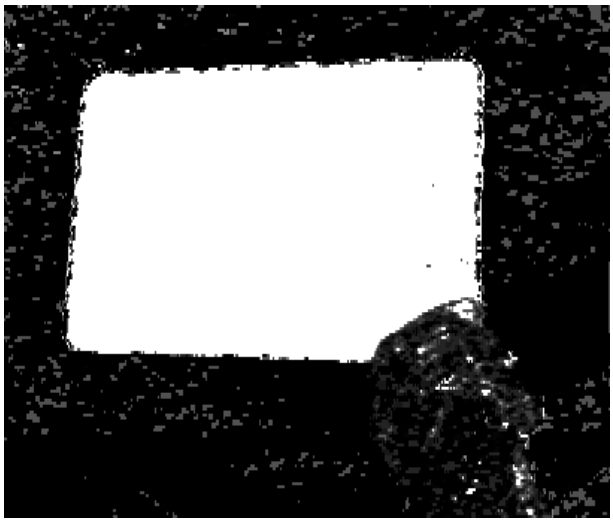

Figura 3.6: Mascara para discriminación de color en modelo *HSV*

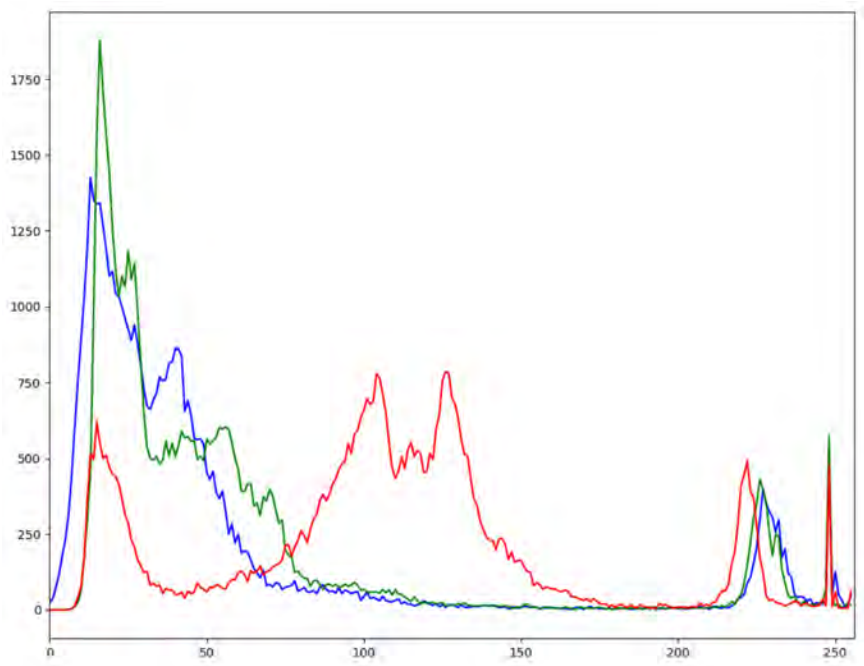

Figura 3.7: Histograma de color RGB

3.4.2. Seguimiento de imagen usando el algoritmo Mean-shift

Es un algoritmo para encontrar una serie de muestras, que se localizar calculando la probabilidad dentro de una función de densidad. Básicamente, es al algoritmo *Hill-climbing* aplicado a un histograma de densidad de datos. Sin embargo, este es un problema menos trivial.

Mean-shift ignora los esbozos en los datos, esto significa que ignora los puntos de datos que están lejos de los picos en los datos, Figura 2.4. Lo hace al procesar solo esos puntos dentro de una ventana de búsqueda local de datos y luego mover esa ventana continuamente.

Esto implica que si la ventana de búsqueda está demasiado distante de los valores pico, no es posible encontrar con base a la probabilidad de la función de densidad.

Pasos para el desarrollo del algoritmo Mean-shift.

1. Escoge una venta de búsqueda.

   a) Su localización inicial.

   b) Su tipo (uniforme, polinomial, exponencial o gaussiano).

   c) Su forma (simétrica o sesgado, posiblemente rotado o rectangular).

2. Calcula la ventana del centro de masa.

3. Centra la ventana del centro de masa.

4. Regresa al paso número 2 hasta que la ventana pare de moverse.

Se calcula el histograma Figura 3.8, posteriormente se normaliza el color dividendo la luminosidad del pixel dejando solo el canal cromático, esto mejora los resultados a cambios de luminosidad, dentro del entorno, Figura 3.9.

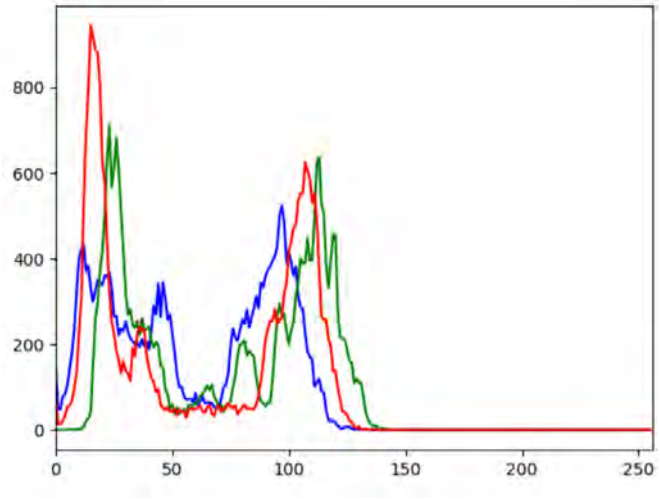

Figura 3.8: Cálculo del histograma del color, para normalizar el color.

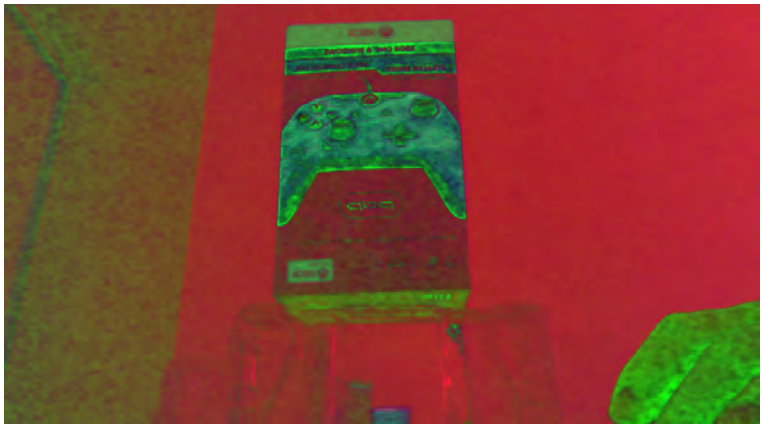

Figura 3.9: Imagen en modelo de color *HSV* para discriminar la luminosidad

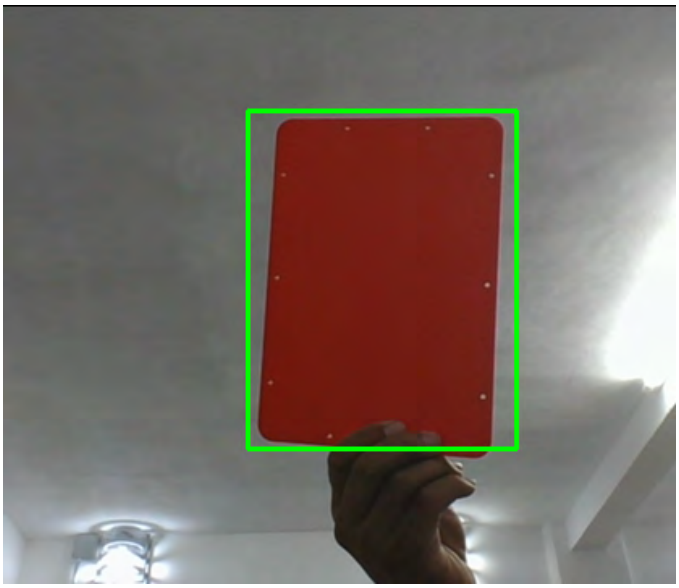

Figura 3.10: Algoritmo Mean-shift en funcionamiento.

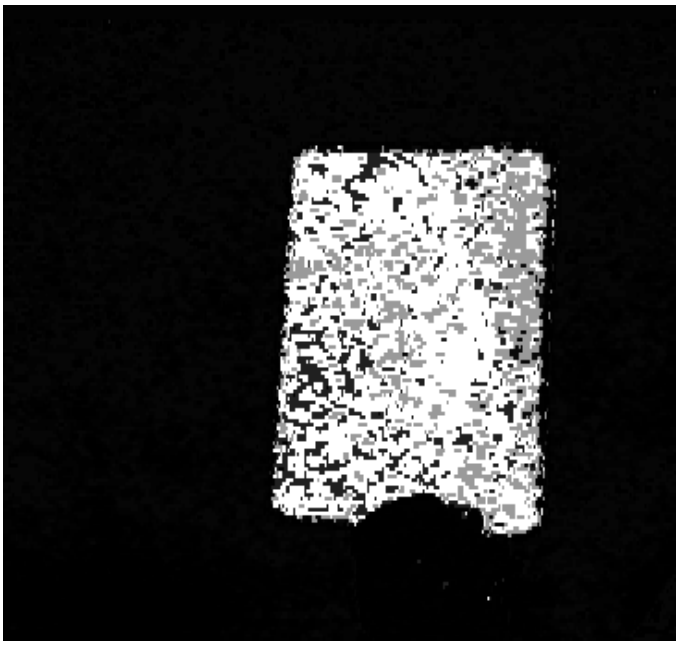

Figura 3.11: Algoritmo Mean-shift en funcionamiento - mascara

3.4.3. Seguimiento de imagen utilizando el algoritmo CAM-shift

Hay tres pasos clave para el análisis de vídeo: la detección de objetos en movimiento, el seguimiento de este tipo de objetos frame a frame, y el respectivo análisis. Para el diseño del algoritmo de seguimiento de objetos propuesto se realiza una implementación basada en la funcionalidad CAM-shift de OpenCV[14], en el cual se agregan características para mejorar la detección del objeto y el seguimiento. Estas mejoras se basan en el filtrado de la imagen inicial dentro de la ROI inicial y en la detección del color del objeto a seguir.

El desarrollo del algoritmo se enfoca a los siguientes paso:

1. Inicialización: Se adquiere la imagen y se realiza la transformación de espacio de colores de RGB a HSV. Esto permite calcular el histograma inicial en base a el matiz del color una vez que es seleccionada la zona donde se encuentra el objeto a seguir.

2. Durante esta etapa se realizan los cálculos de comparación y promedio ponderado del histograma de color.

3. Recibe como parámetro el histograma inicial perteneciente a la etapa de inicialización, y en esta fase realiza los cálculos correspondiente a la posición del objeto dando como resultado una nueva zona o región de interés (region of interest, ROI). Esta información se re-alimenta al bloque cálculo de histograma para obtener el valor del histograma promedio y así, nuevamente, determinar la posición actualizada del objeto en seguimiento.

Figura 3.12: Algoritmo CAM-shift en funcionamiento.

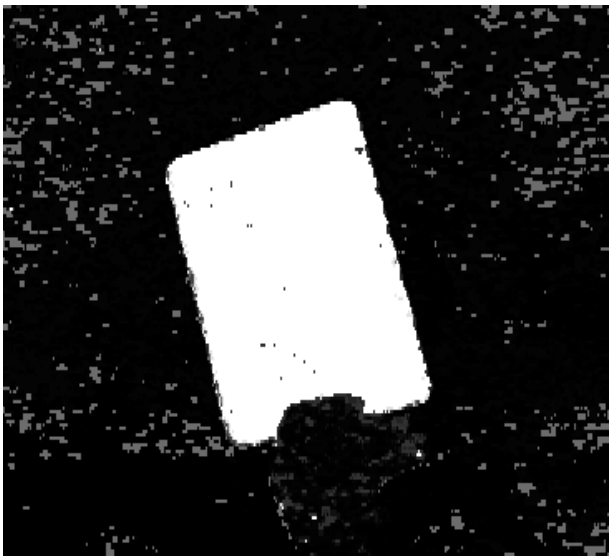

Figura 3.13: Mascara aplicada para el algoritmo CAM-shift

3.4.4. Seguimiento de imagen usando homografia

En el area de visión por computadora una de las técnicas más usadas para el reconocimiento de imágenes está basado en el uso de puntos característicos. El objetivo es reconocer puntos y características invariables en la imagen. En particular, cuando se trata de objetos planos, el planteamiento está basado en encontrar una proyección u *homografia* que lleve los puntos de una imagen a los de la otra.

En este proyecto, se han utilizado los puntos característicos y descriptores ORB, los cuales ofrecen ciertas ventajas:

- Son más rápidos en su procesamiento, ya que sus descriptores son binarios.
- En cuestión de precisión, son equivalentes a los que proporcionan los algoritmos SIFT y SURF.
- El algoritmo ORB es de uso libre, mientras que SIFT y SURF operan bajo derechos de autor.
- El costo computacional es mucho menor en comparación con algoritmos como SIFT y SURF.

En conclusión, como los algoritmos SIFT y SURF no son eficientes para tareas en tiempo real, debido a su gran costo computacional, en este trabajo de tesis se opta por la implementación del algoritmo ORB, puesto cumple con los requerimientos de velocidad de localización de puntos característicos y calculo de descriptores.

3.5. Seguimiento y aterrizaje sobre la plataforma móvil

Hasta ahora hemos abordado el problema de identificar un marcador por medio de homografia y correspondencia utilizando la cámara monocular del Bebop, el movimiento del Bebop y el movimiento de la cámara integrada del mismo, sin embargo queda por resolver el problema de aterrizaje y seguimiento de una plataforma móvil, como se aprecia en la Figura 3.17, esto se obtiene integrando todos pasos que se han realizado hasta ahora. Para realizar un proceso de aterrizaje sobre una plataforma es indispensable el desarrollo de un programa principal utilizando la plataforma de ROS que haga uso del código previamente desarrollado. Las pruebas experimentales realizadas, se llevaron a cabo siguiendo el procedimiento que se describe a continuación:

3.5.1. Calculo de centro de la imagen.

Para la obtención del punto central del marcador detectado, se procede a calcular la distancia entre esquinas de dicho marcador. Posteriormente se divide esta magnitud entre dos para obtener el punto medio de cada eje, ademas de sumar las coordenadas de los puntos inferiores en los ejes X e Y, para obtener el punto medio de la imagen.

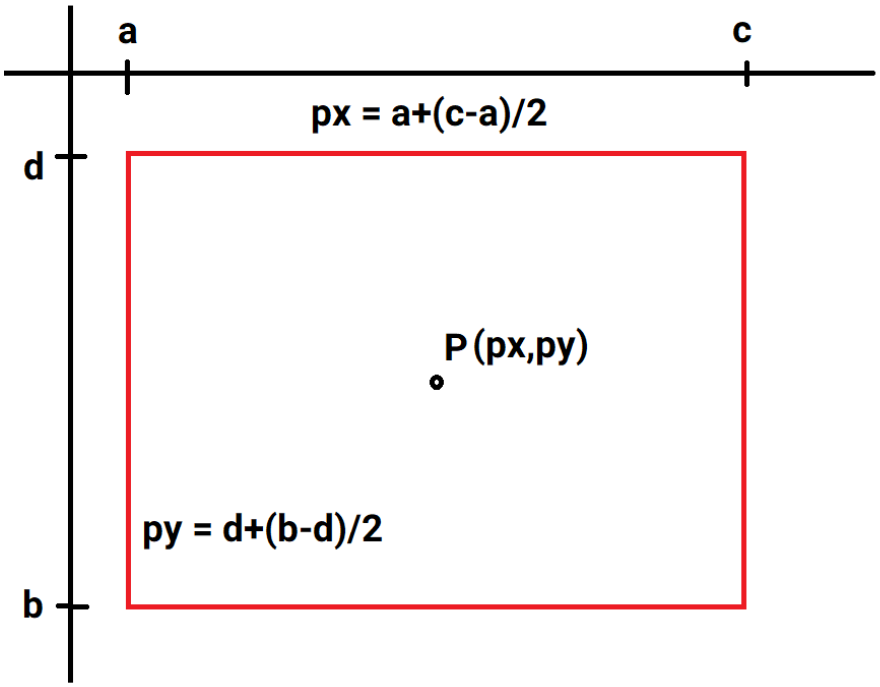

Figura 3.14: Secuencia de Aterrizaje.

3.5.2. Calculo de distancia de marcador

Para poder determinar la distancia entre el Bebop y el marcador se llevaron a cabo mediciones de la altura del marcador en pixeles a diferentes distancias, esta relación entre distancias a la que se encuentra el marcador y la altura en pixeles se utilizo para poder construir un polinomio que se ajustara a todos estos valores. Por lo tanto, conociendo la altura en pixeles del marcador es posible determinar la distancia a la que se encuentra este en un rango de 20 cm a 2 metros.

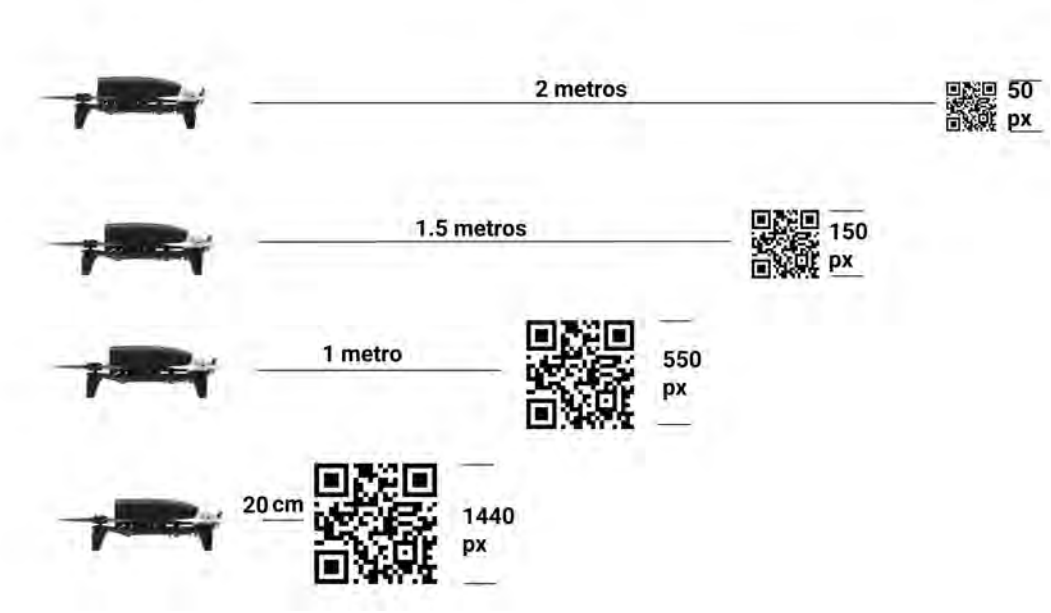

Figura 3.15: Mediciones para calculo de polinomio

Este polinomio nos representa la curva como se muestra en la Figura 3.16.

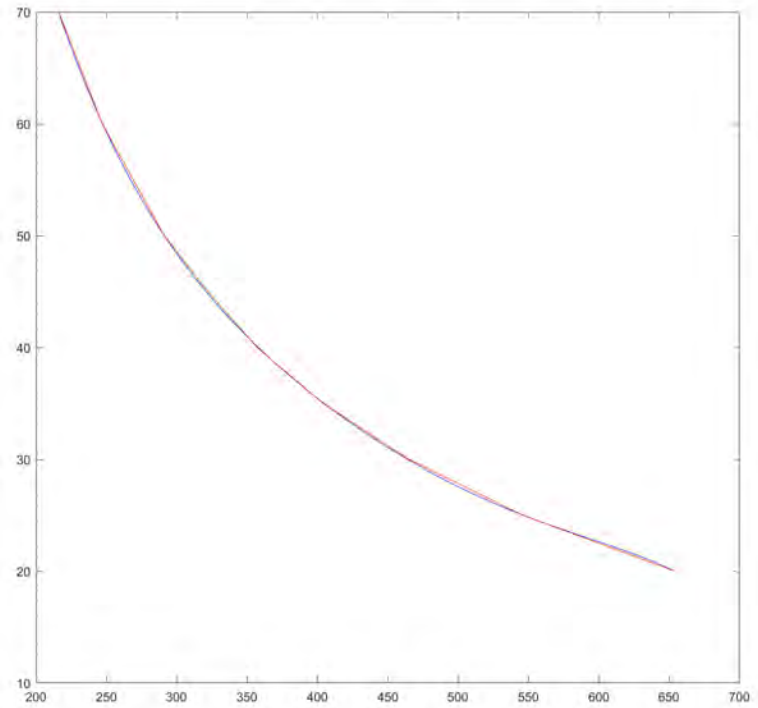

Figura 3.16: Curva de polinomio de grado 5.

3.5.3. Procedimiento para realizar el aterrizaje autónomo

1. Establecer una conexión inalámbrica con una computadora portátil y el Bebop por medio de una red Wifi 2.4 Ghz.

2. Ejecutar el nodo *bebop_driver* que se encuentra dentro del paquete *bebop_autonomy*, para poder enviar comandos de ROS y recibir información a través de la suscripción de los tópicos generados por el Bebop.

3. Despegar el Bebop utilizando el control manual.

4. Orientar el vehículo hacia la plataforma designada para el aterrizaje para que se encuentre dentro de su campo de visión.

5. Una vez que el vehículo ha detectado el punto de interés inicia una tarea de aterrizaje autónomo.

a) Se genera un error entre la coordenada en pixeles del centroide del marcador detectado y las coordenadas del centro del plano imagen capturado con la cámara del Bebop.

b) Una vez centrado y enfocado el marcador, se calcula la distancia con respecto al marcador utilizando un polinomio interpolante grado 5, véase Figura 4.18.

c) Se procede a compensar dicho error mediante un control proporcional, con una saturación de 0.8, de con base a el rango de respuesta del mismo Bebop.

d) Se ajusta continuamente del enfoque de la cámara, para mantener en centro del plano imagen, como se muestra en la Figura 3.18.

e) Se compensa el error en distancia mediante el envío de mensajes por medio del tópico *cmd_vel*.

f) En el momento cuando la orientación de la cámara ha llegado al mínimo valor en el eje Z, se toma como referencia que el marcador se encuentra exactamente bajo el Bebop.

g) Se cambia el algoritmo para medir distancia de manera horizontal, mismos que se lleva a cabo con la cámara integrada del Bebop, por un algoritmo que mide altura, este igualmente con la ayuda de la cámara del Bebop.

h) Se determina una región donde el vehículo puede permanecer para movimientos de corrientes de aire existentes en el entorno.

i) Se espera 5 segundos para verificar que la base continua en posición para realizar un aterrizaje.

j) Si la base realiza un movimiento se volverá a enviar un ajuste en la posición del Bebop.

k) En caso contrario se comenzara a realizar un descenso hasta llegar a la base de aterrizaje, hasta la altura de 10 cm, altura que en la que Bebop aun tiene sustentación con el aire.

l) Se enviar un mensaje de tipo vacío por medio del tópico *land*, este mensaje aterrizara al Bebop de manera suave sobre la plataforma, dando así por concluida nuestra tarea

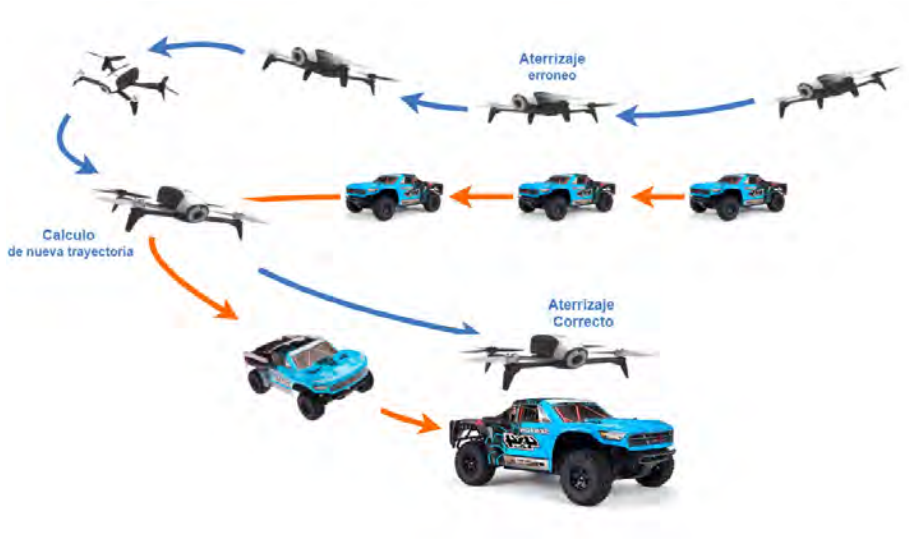

Figura 3.17: Secuencia de Aterrizaje.

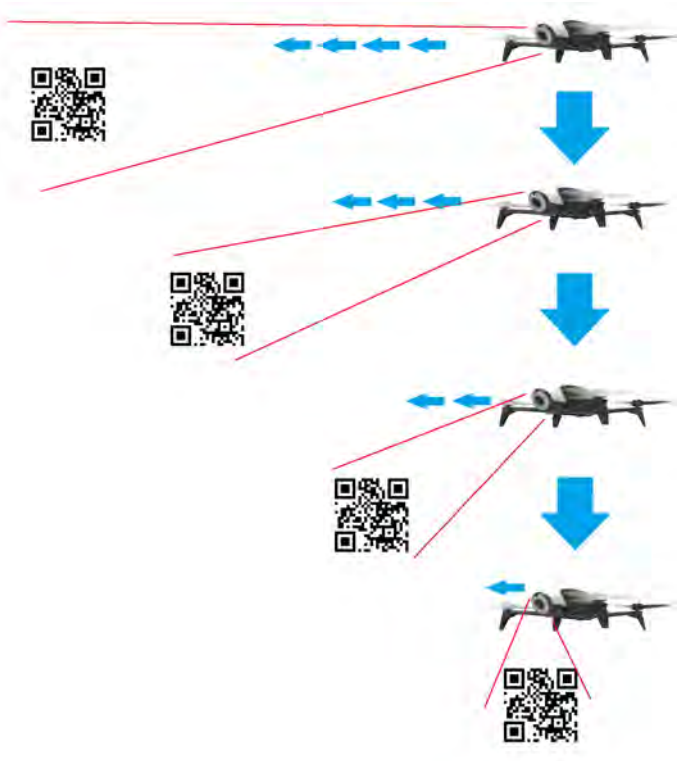

Figura 3.18: Ajuste de angulo de visión de la cámara de Bebop.

Capítulo 4

Resultados Experimentales

En este capitulo se presentaran lo experimentos realizados para el seguimiento de un objeto de interés utilizando el vehículo aéreo esto a través de la integración de los resultados parciales en el capitulo anterior, para resolver la tarea de aterrizaje autónomo.

4.1. Resultados preliminares de adquisición de imágenes y lectura de ejes del control XBOX ONE.

Se obtuvo la lectura de ejes del control de Xbox One con una frecuencia de 100 Hz, también se obtuvieron las imágenes en formato RAW, que son procesadas haciendo un cambio del modelo del color del modelo RGB al modelo HSV y seguimiento por localización de descriptores, para hacer lo siguiente es necesario iniciar el sistema ROS, Figura 4.1, iniciar el espacio de trabajo Figura 4.2, e iniciar el driver de ROS.

```
nicolas@Dell-XPS-15-9560-Signature-Edition:~$ roscore
... logging to /home/nicolas/.ros/log/975a21fc-56f4-11e9-b320-9cb6d0f776ee/roslaunch-Dell-XPS-15-9560-Signature-Edition-
64.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://Dell-XPS-15-9560-Signature-Edition:49295/
ros_comm version 1.13.7


SUMMARY
========

PARAMETERS
 * /rosdistro: lunar
 * /rosversion: 1.13.7

NODES

auto-starting new master
process[master]: started with pid [74]
ROS_MASTER_URI=http://Dell-XPS-15-9560-Signature-Edition:11311/

setting /run_id to 975a21fc-56f4-11e9-b320-9cb6d0f776ee
process[rosout-1]: started with pid [87]
started core service [/rosout]
```

Figura 4.1: Inicio del sistema operativo ROS

```
nicolas@Dell-XPS-15-9560-Signature-Edition: ~/catkin_ws
nicolas@Dell-XPS-15-9560-Signature-Edition:~/catkin_ws$ catkin init
Catkin workspace `/home/nicolas/catkin_ws` is already initialized. No action taken.
------------------------------------------------------------
Profile:                     default
Extending:          [cached] /opt/ros/lunar
Workspace:                   /home/nicolas/catkin_ws
------------------------------------------------------------
Source Space:       [exists] /home/nicolas/catkin_ws/src
Log Space:         [missing] /home/nicolas/catkin_ws/logs
Build Space:        [exists] /home/nicolas/catkin_ws/build
Devel Space:        [exists] /home/nicolas/catkin_ws/devel
Install Space:      [unused] /home/nicolas/catkin_ws/install
DESTDIR:            [unused] None
------------------------------------------------------------
Devel Space Layout:          linked
Install Space Layout:        None
------------------------------------------------------------
Additional CMake Args:       None
Additional Make Args:        None
Additional catkin Make Args: None
Internal Make Job Server:    True
Cache Job Environments:      False
------------------------------------------------------------
Whitelisted Packages:        None
Blacklisted Packages:        None
------------------------------------------------------------
Workspace configuration appears valid.
------------------------------------------------------------
nicolas@Dell-XPS-15-9560-Signature-Edition:~/catkin_ws$
```

Figura 4.2: Inicio del espacio de trabajo.

4.2. Resultados preliminares de seguimiento de marcadores y segmentación de color.

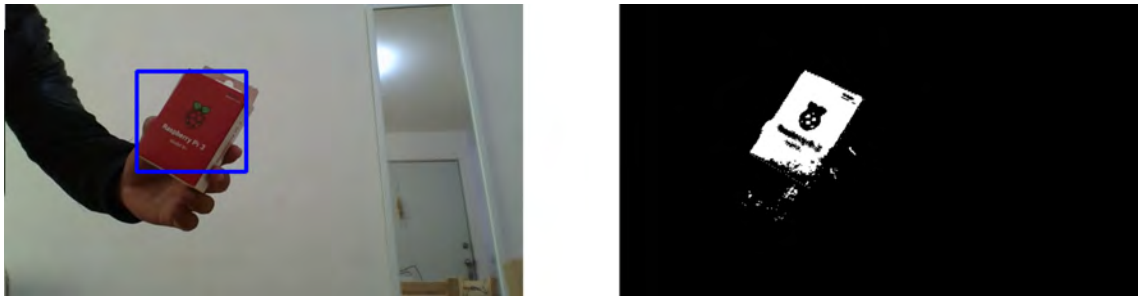

Figura 4.3: Seguimiento por segmentación de color HSV

El seguimiento por segmentación de color en el modelo de color hsv Figura 4.3, se realiza el seguimiento sin embargo este tipo de algoritmo por color es bastante propenso a fallas por cambio de iluminación y matiz en el color, debido a esto se comparó el método meanshift, Figura 4.4, que se ejecuta mediante el histograma discriminando el canal de luminiscencia que permite al algoritmo sea menos propenso a cambios de iluminación, por este motivo presenta un mejor rendimiento en comparación al método por segmentación en el modelo de color hsv.

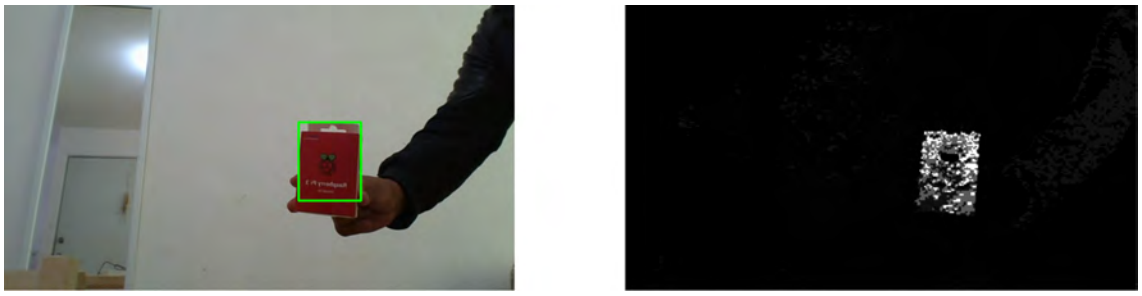

Figura 4.4: Seguimiento por meanshift

El rastreo de la imagen se realiza de modo eficiente, sin embargo para los propósitos de este trabajo carece de todos los elementos que se buscan, esto debido a que es necesario detectar la cercanía Figura 4.6 o el distanciamiento Figura 4.5 de la imagen capturada, que hace insuficiente a nuestro objetivo.

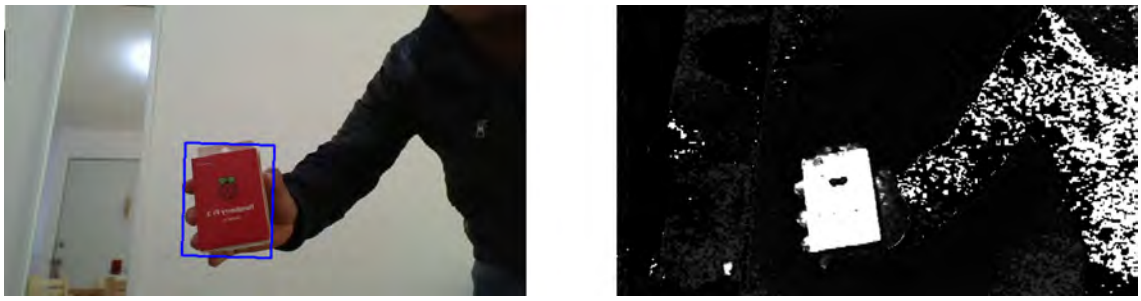

Figura 4.5: Seguimiento por camshift, imagen alejada

Figura 4.6: Seguimiento por camshift, imagen cercana al lente de la cámara

El algoritmo cam-shift, es capaz de ajustar el tamaño de la ventana de búsqueda, gracias a la forma de trabajo del algoritmos pues en cierta manera, es el algoritmo meanshift que ajusta la ventana de búsqueda continuamente hasta su convergencia.

Desafortunadamente aún se tiene el problema de no poder detectar la orientación de la imagen, por lo que se opta por el siguiente método de localización y comparación por descriptores y puntos clave en la imagen Figura 4.7, utilizando descriptores SIFT.

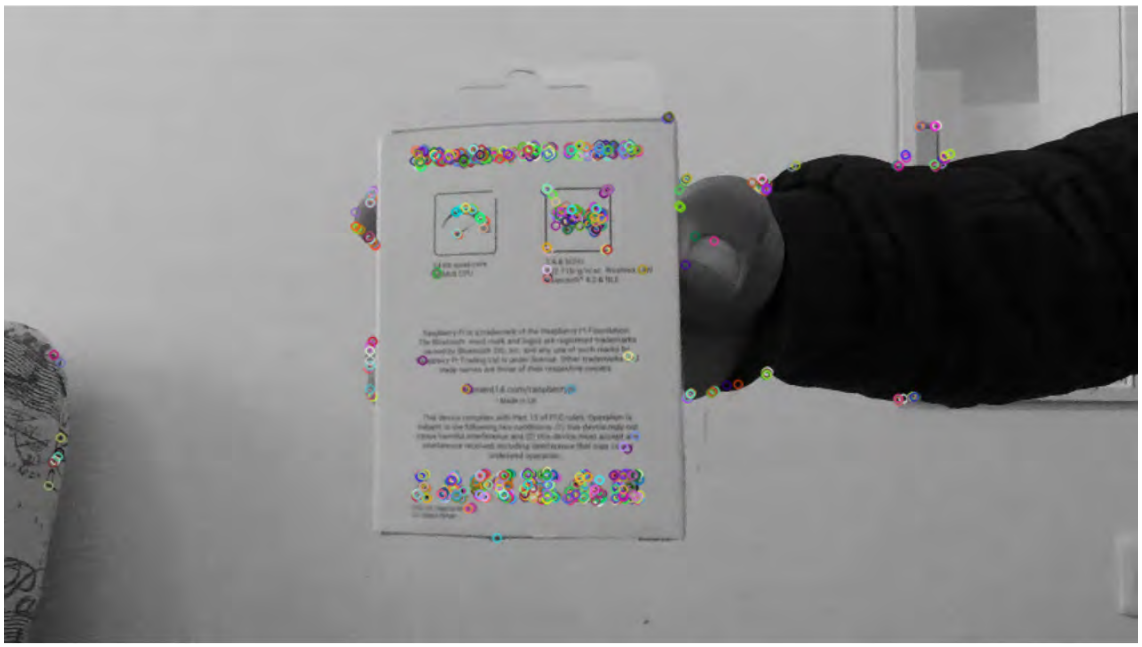

Figura 4.7: Se realiza la localización, los puntos clave.

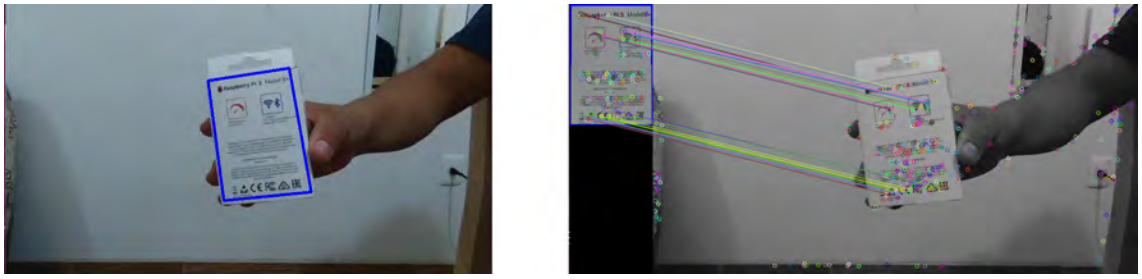

Figura 4.8: comienza el seguimiento de la imagen seleccionada.

Figura 4.9: Seguimiento de la imagen en orientación izquierda.

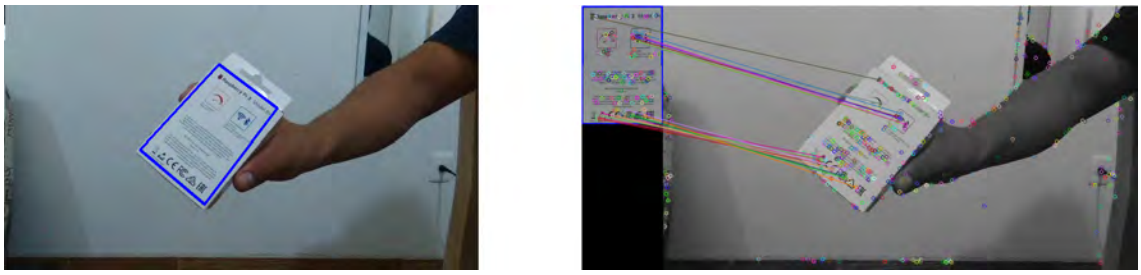

Figura 4.10: Seguimiento de la imagen en orientación derecha.

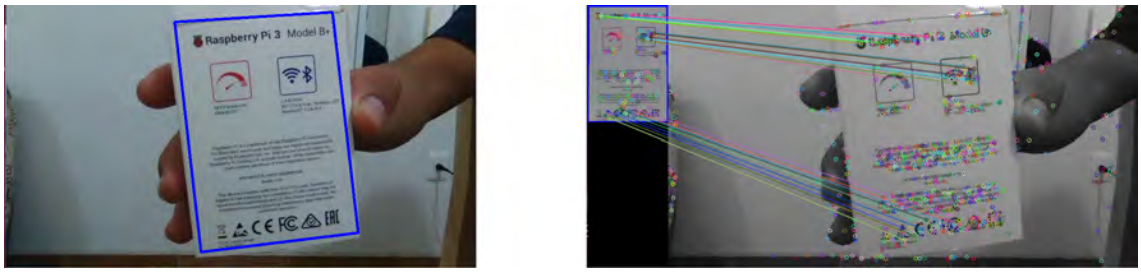

Figura 4.11: Seguimiento de la imagen con un acercamiento a la cámara

Se puede observar que, el seguimiento de la imagen, se realiza de manera exitosa, Figura 4.8, ajustando el objetivo y localización de los descriptores a diferentes ángulos de inclinación y orientación , figuras 4.9 y Figura 4.10, al igual que al hacer un acercamiento y distanciamiento de la imagen, sin embargo, los descriptores SIFT, presentan un gran costo computacional provocando que la imagen presente un retraso de 3 segundos desde el momento de su captura hasta su visualización, es por tal motivo que se decidió investigar otros métodos para localización de descriptores mediante otros algoritmos.

Por este motivo se ha decidido emplear descriptores de denominados **Orient FAST and Rotated BRIEF (ORB)**, los cuales son de uso libre, debido a que no se encuentran patentados, de hecho, se presentaron este tipo de descriptores como opción a los descriptores SIFT y SURF

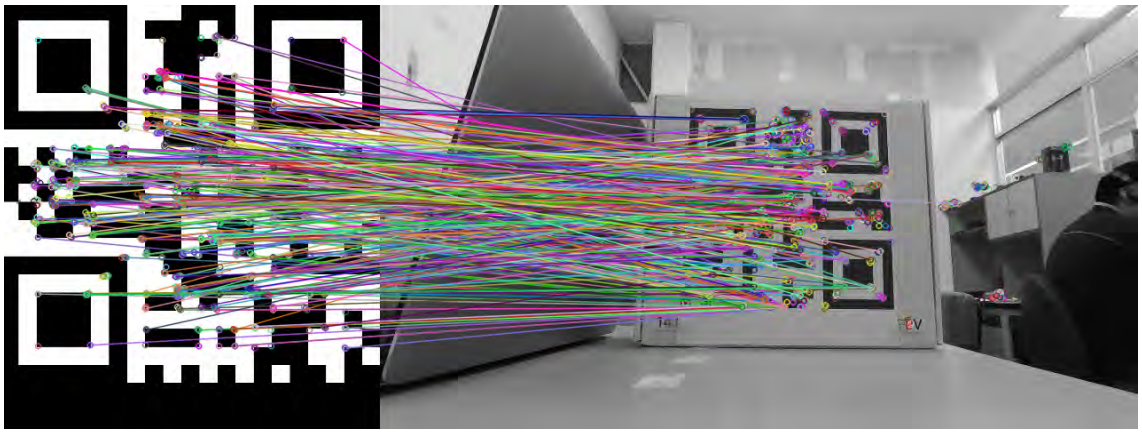

Figura 4.12: Homografia con descriptores ORB

4.3. Seguimiento por control de software de cámara mediante ROS

```
pub = rospy.Publisher("bebop/camera_control", Twist, queue_size = 1)
```

Listing 4.1: Publicador para topico “camera_control” con tipo de mensaje “Twist”.

Por medio de la publicación del mensaje a través del tópico, es posible controlar por software el rango de visualización por parte de la cámara.

- Se calcula el centro de la imagen principal, haciendo la división de la resolución en pixeles entre 2.
- Se Localizar el centroide de la imagen localizada por visión.
- Calcular la distancia entre la imagen localizada y el centro del cuadro de video.
- Calculamos el error en ambos ejes en este caso denominados “z” y “y”.
- Proponemos un control proporcional para minimizar el error en el seguimiento de la imagen.

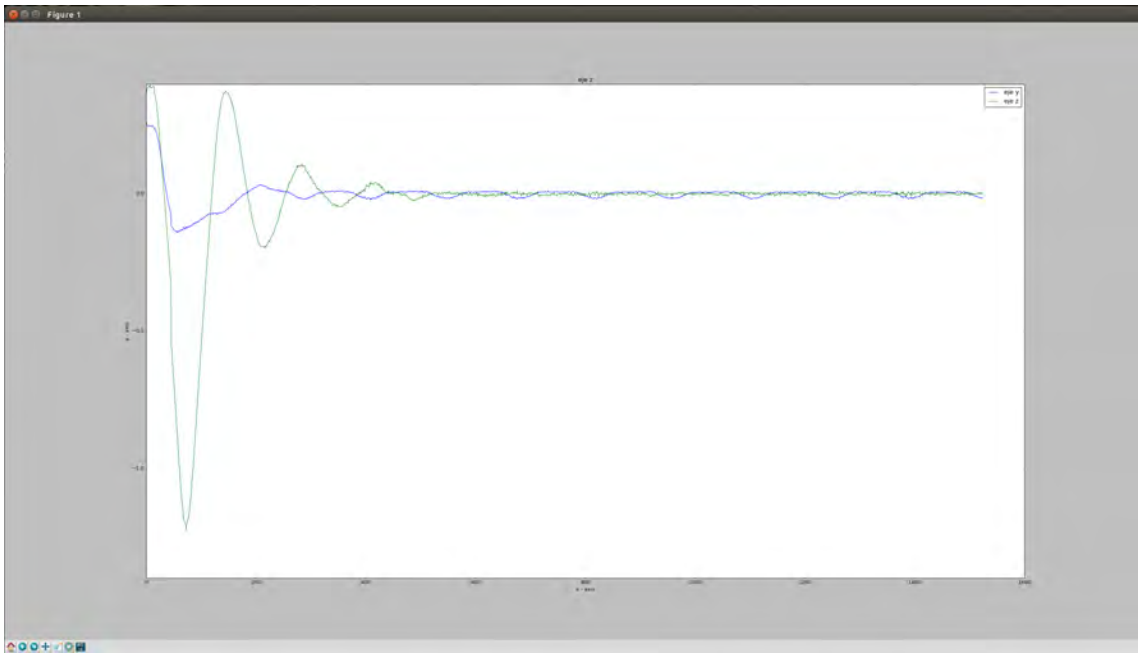

Figura 4.13: Respuesta en los ejes Z y Y para el movimiento de la cámara.

4.4. Resultados preliminares de seguimiento con Bebop 2 Drone.

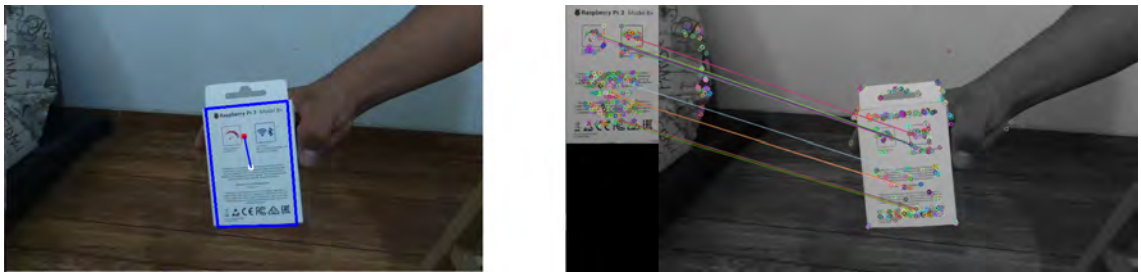

Figura 4.14: Seguimiento de imagen mediante envío de mensajes a través del tópico camera/control de ROS, imagen de seguimiento A.

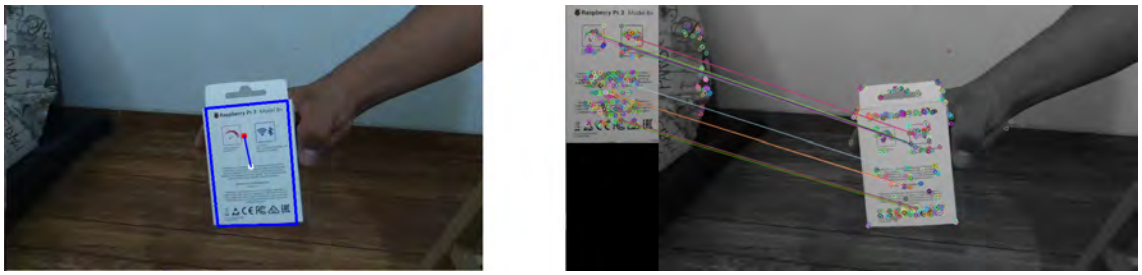

Figura 4.15: Seguimiento de imagen mediante envío de mensajes a través del tópico camera/control de ROS, imagen de seguimiento B.

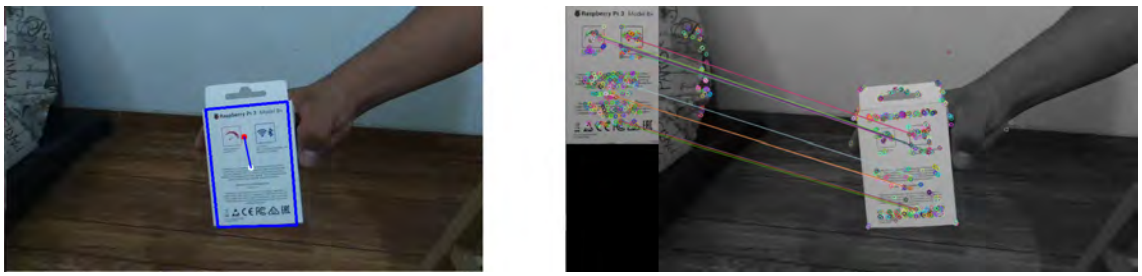

Figura 4.16: Seguimiento de imagen mediante envío de mensajes a través del tópico camera/control de ROS, imagen de seguimiento C.

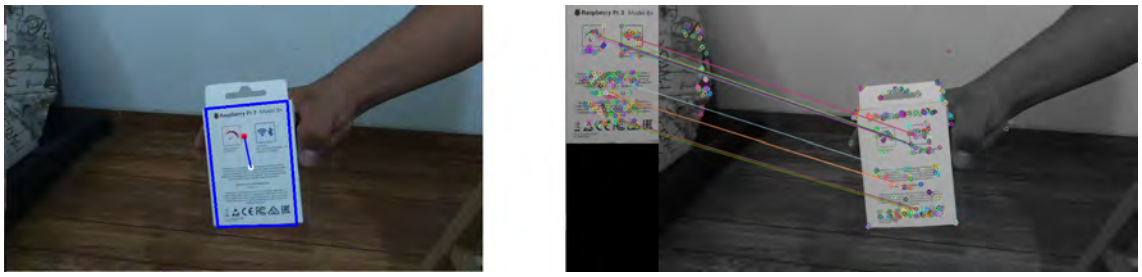

Figura 4.17: Seguimiento de imagen mediante envío de mensajes a través del tópico camera/control de ROS, imagen de seguimiento D.

En las figuras 4.14, 4.15, 4.16, 4.17, es posible notar el seguimiento que se realiza en conjunto con el control de orientación de la cámara, cabe destacar que la localización de los descriptores se modifico a un algoritmo diferente denominado *Oriented FAST and Rotated BRIEF*, el cual tiene un costo computacional bastante inferior al algoritmo SIFT, por lo que se tomó la decisión de hacer la implementación, con lo que se obtuvieron buenos resultados, y se elimino el retraso en la visualización de la imagen, por otra parte cumple con todos los requerimientos, para realizar el trabajo planeado,

que consiste en seguir una plataforma de aterrizaje para posteriormente una vez detenida realizar una maniobra de aterrizaje.

4.5. Calculo de proximidad por visión

Para realizar el calculo de proximidad de la imagen, se realiza por medio del método de mínimos cuadrados utilizando un orden 5 que según pruebas fue la mejor aproximación a los datos ingresados, que es una técnica de análisis numérico enmarcada dentro de la optimizacion matemática, en la que, dados un conjunto de pares ordenados, y una familia de funciones se intenta encontrar una función continua, que mejor se aproxime a los datos, de acuerdo con el criterio de mínimo error cuadrático.

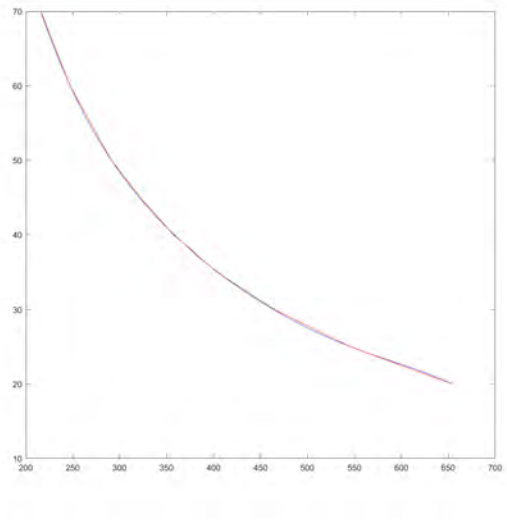

Figura 4.18: Interpolación, para calculo de distancia

$$
\begin{aligned}
1.0e+02* & \\
& -0.000000000000087x^5 + 0.000000000207031x^4 \\
& -0.000000197673710x^3 + 0.000096347572002x^2 \\
& -0.025030342204099x + 3.193388042512088
\end{aligned}
$$

Bibliografía

[1] Oualid Araar, Nabil Aouf, and Ivan Vitanov. Vision based autonomous landing of multirotor uav on moving platform. *Journal of Intelligent & Robotic Systems*, 85(2):369–384, 2017.

[2] Alexandre Borowczyk, Duc-Tien Nguyen, André Phu-Van Nguyen, Dang Quang Nguyen, David Saussié, and Jerome Le Ny. Autonomous landing of a multirotor micro air vehicle on a high velocity ground vehicle. *IFAC-PapersOnLine*, 50(1):10488–10494, 2017.

[3] Huizhong Chen, Sam S Tsai, Georg Schroth, David M Chen, Radek Grzeszczuk, and Bernd Girod. Robust text detection in natural images with edge-enhanced maximally stable extremal regions. In *2011 18th IEEE International Conference on Image Processing*, pages 2609–2612. IEEE, 2011.

[4] Dorin Comaniciu and Peter Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (5):603–619, 2002.

[5] Dorin Comaniciu, Visvanathan Ramesh, and Peter Meer. Real-time tracking of non-rigid objects using mean shift. In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition. CVPR 2000 (Cat. No. PR00662)*, volume 2, pages 142–149. IEEE, 2000.

[6] Rita Cucchiara, Costantino Grana, Massimo Piccardi, Andrea Prati, and Stefano Sirotti. Improving shadow suppression in moving object detection with hsv color information. In *ITSC 2001. 2001 IEEE Intelligent Transportation Systems. Proceedings (Cat. No. 01TH8585)*, pages 334–339. IEEE, 2001.

[7] Ioannis Daramouskas, Isidoros Perikos, and Ioannis Hatzilygeroudis. A method for performing efficient real-time object tracing for drones. *Advances in Smart Systems Research*, 6(2):55, 2012.

[8] Keinosuke Fukunaga and Larry Hostetler. The estimation of the gradient of a density function, with applications in pattern recognition. *IEEE Transactions on information theory*, 21(1):32–40, 1975.

[9] Nicolas Gallardo, Karthik Pai, Berat A Erol, Patrick Benavidez, and Mo Jamshidi. Formation control implementation using kobuki turtlebots and parrot bebop drone. In *World Automation Congress (WAC), 2016*, pages 1–6. IEEE, 2016.

[10] Bruno Herisse, Francois-Xavier Russotto, Tarek Hamel, and Robert Mahony. Hovering flight and vertical landing control of a vtol unmanned aerial vehicle using optical flow. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 801–806. IEEE, 2008.

[11] Prateek Kumar, Steven Henikoff, and Pauline C Ng. Predicting the effects of coding non-synonymous variants on protein function using the sift algorithm. *Nature protocols*, 4(7):1073, 2009.

[12] Sven Lange, Niko Sunderhauf, and Peter Protzel. A vision based onboard approach for landing and position control of an autonomous multirotor uav in gps-denied environments. In *Advanced Robotics, 2009. ICAR 2009. International Conference on*, pages 1–6. IEEE, 2009.

[13] Kevin Ling. Precision landing of a quadrotor uav on a moving target using low-cost sensors. Master's thesis, University of Waterloo, 2014.

[14] Ce Liu, Jenny Yuen, Antonio Torralba, Josef Sivic, and William T Freeman. Sift flow: Dense correspondence across different scenes. In *European conference on computer vision*, pages 28–42. Springer, 2008.

[15] Ritanjali Majhi, Ganapati Panda, and Gadadhar Sahoo. Development and performance evaluation of flann based model for forecasting of stock markets. *Expert systems with applications*, 36(3):6800–6808, 2009.

[16] Tin Muskardin, Georg Balmer, Sven Wlach, Konstantin Kondak, Maximilian Laiacker, and Anibal Ollero. Landing of a fixed-wing uav on a mobile ground vehicle. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 1237–1242. IEEE, 2016.

[17] Tobias Nägeli, Lukas Meier, Alexander Domahidi, Javier Alonso-Mora, and Otmar Hilliges. Real-time planning for automated multi-view drone cinematography. *ACM Transactions on Graphics (TOG)*, 36(4):132, 2017.

[18] Salcedo Peña and Vinicio Stalin. Aterrizaje automático de un vehículo aéreo no tripulado basado en seguimiento de puntos de interés para superficies móviles. B.S. thesis, Universidad de las Fuerzas Armadas ESPE. Carrera de Ingeniería en Electrónica, Automatización y Control., 2018.

[19] Patrick Pérez, Carine Hue, Jaco Vermaak, and Michel Gangnet. Color-based probabilistic tracking. In *European Conference on Computer Vision*, pages 661–675. Springer, 2002.

[20] Giovanni L Sicuranza and Alberto Carini. A generalized flann filter for nonlinear active noise control. *IEEE Transactions on Audio, Speech, and Language Processing*, 19(8):2412–2417, 2011.

[21] Joao Silva, Ricardo Mendonça, Francisco Marques, Paulo Rodrigues, Pedro Santana, and José Barata. Saliency-based cooperative landing of a multirotor aerial vehicle on an autonomous surface vehicle. In *Robotics and Biomimetics (ROBIO), 2014 IEEE International Conference on*, pages 1523–1530. IEEE, 2014.

[22] Andrews Sobral. Bgslibrary: An opencv c++ background subtraction library. In *IX Workshop de Visao Computacional*, volume 2, page 7, 2013.

[23] Markus Andreas Stricker and Markus Orengo. Similarity of color images. In *Storage and Retrieval for Image and Video Databases III*, volume 2420, pages 381–393. International Society for Optics and Photonics, 1995.

[24] Shaharyar Ahmed Khan Tareen and Zahra Saleem. A comparative analysis of sift, surf, kaze, akaze, orb, and brisk. In *2018 International Conference on Computing, Mathematics and Engineering Technologies (iCoMET)*, pages 1–10. IEEE, 2018.

[25] Tomasz Trzcinski, Mario Christoudias, Pascal Fua, and Vincent Lepetit. Boosting binary keypoint descriptors. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2874–2881, 2013.

[26] Koen Van De Sande, Theo Gevers, and Cees Snoek. Evaluating color descriptors for object and scene recognition. *IEEE transactions on pattern analysis and machine intelligence*, 32(9):1582–1596, 2010.

[27] Fadel Adib Yunfei Ma, Nicholas Selby. Drone relays for battery-free networks. *ACM Transactions on Graphics*, pages 335–347, 2017.

[28] Guyue Zhou, Lu Fang, Ketan Tang, Honghui Zhang, Kai Wang, and Kang Yang. Guidance: A visual sensing platform for robotic applications. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 9–14, 2015.

.1. Ubuntu 16.04 LTS 64 bits

Ubuntu es un sistema operativo completo basado en GNU/Linux. Su desarrollo está orientado tanto al ámbito hogareño como al profesional, brindando las herramientas necesarias para cada tipo de usuario. Ubuntu tiene promesas que mantiene y ha mantenido a lo largo de los años las cuales son:

- *Ubuntu* Siempre ser
  a gratuito, incluyendo versiones empresariales y actualizaciones de seguridad.
- *Ubuntu* Viene con soporte completo de Canonical y cientos de compañías en todo el mundo.
- *Ubuntu* Incluye las mejores infraestructuras de traduccín y accesibilidad que el software libre tiene para ofrecer.
- Los CDs de Ubuntu contienen mayoritariamente aplicaciones de software libre.

Se destaca que Ubuntu puede ejecutarse en arquitecturas de Intel, AMD, ARM. Su patrocinador, Canonical, es una compañía británica propiedad del empresario sudafricano *Mark Shuttleworth*. Ofrece de manera gratuita, y se financia por medio de servicios vinculados al sistema operativo y vendiendo soporte técnico. Ademas al mantenerlo libre y gratuito la empresa es capaz de aprovechar los desarrolladores de la comunidad para mejorar los componentes de su sistema operativo. Extraoficialmente la comunidad de desarrolladores proporciona soporte para otras derivaciones de Ubuntu, con otros entornos gráficos como Kubuntu, Xubuntu, ubuntu MATE, Edubuntu, Ubuntu Studio, Mythbuntu, Ubuntu Gnome y Lubuntu. Cada 6 meses se publica una version de ubuntu, esta recibe soporte por parte de canonical durante 9 meses por medio de actualizaciones de seguridad, parches para bugs críticos y actualizaciones menores de programas. Las versiones LTS(Long term Support), que se libera cada dos años, reciben soporte durante cinco años en los sistemas de escritorio y de servidor.

Figura .19: Logo de Ubuntu

.2. Instalación de ROS

1. Configurar los repositorios de Ubuntu

   a) Configurar los repositorios para permitir *restricted*, *universe* y *multiverse*

2. Configura tu sources.list

   a) Configurar para permitir *software* de *packages.ros.org*.
   sudo sh -c 'echo "deb http //packages.ros.org/ros/ubuntu $(lsb_release -sc) main" >/etc/apt/sources.list.d/ros-latest.list'

3. Coloca las llaves del servidor.

   a) llaves para el servidor sudo apt-key adv –keyserver hkp //ha.pool.sks-keyservers.net 80 –recv-key 421C365BD9FF1F717815A3895523BAEEB01FA116

4. Instalación: ROS contiene 4 configuraciones distintas, se instalar a la versión completa para contar con todas los paquetes necesarios.

   a) Actualizamos los repositorios
   sudo apt-get update

   b) Para seleccionar la instalación completa se ejecuta el siguiente script sobre la terminal de ubuntu. sudo apt-get install ros-kinetic-desktop-full

5. Inicializacion de ROSdep: Es necesario activar *rosdep* para realizar una fácil instalación de las dependencias necesarias para la compilación de los archivos fuente.

   sudo rosdep init rosdep update

6. Configuración del entorno de desarrollo: Para lograr que las variables de entorno se agreguen automáticamente cada vez que se inicia una terminal es necesario ejecutar el siguiente comando.

   echo "source /opt/ros/kinetic/setup.bash" >> /.bashrc.
   source /.bashrc

7. Instalando ROSinstall: Permite descargar fácilmente paquetes

   sudo apt-get install python-rosinstall

.3. Instalación del paquete bebop autonomy

Para la instalación de ROS es necesario reunir los requerimientos previos que son:

- Distribucion de ROS
- Paquetes de Ubuntu *build-esstential*, *python-rosdep*, *python-catkin-tools*
- Nociones básicas para la construcción de paquetes de ROS.

Para su instalación es necesario clonar el código fuente en un espacio de trabajo *Catkin* existente o crear uno "nuevo", para posteriormente compilarlo usando el comando *catkin build*, es factible su instalación siguiendo la siguiente serie de comandos.

```
# Crea e inicializa el espacio de trabajo.
$ mkdir -p  /catkin_ws/src && cd  /catkin_ws
$ catkin init
$ git clone https://github.com/AutonomyLab/bebop_autonomy.git src/bebop_autonomy
# Actualiza la base de datos e instala dependencias.
$ rosdep update
$ rosdep install –from-paths src -i
# Construye el espacio de trabajo.
$ catkin build
```

.4. Librería de código abierto para visión por computadora OpenCV

Open Source Computer Vision Library(OpenCV) es una librería que fue diseñada para eficiencia computacional y con un fuerte enfoque en aplicaciones de tiempo real, escrito en C/C++, la librería puede tomar la ventaja de un procesador multinúcleo, con ayuda del OpenCL puede tomar la ventaja de la aceleración por *hardware*, puede correr bajo arquitecturas Linux, Windows, y MacOS, se desarrolla activamente bajo Python, Ruby, MatLab y otros lenguajes, el objetivo de OpenCV es proveer una infraestructura *SimpleToUse(simple de usar)*, para ayudar a los desarrolladores a generar algoritmos de visión de manera más rápida [22].