CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS DEL INSTITUTO POLITÉCNICO NACIONAL

UNIDAD ZACATENCO

Programa de sistemas autónomos de navegación aérea y submarina

"Desarrollo de algoritmos para la identificación y seguimiento de objetivos en tierra usando un vehículo aéreo."

T E S I S

Que presenta

Ing. Cindy Angélica Herrera Carbajal

Para obtener el grado de

Maestra en ciencias
en sistemas autónomos de navegación aérea y submarina

Directores de la Tesis:

Dr. Eduardo Steed Espinoza Quesada

Dr. Sergio Rosario Salazar Cruz

Ciudad de México Diciembre, 2018

Agradecimientos

Agradezco a Dios por bendecirme a lo largo de mi vida y permitirme concluir este proyecto.

Les agradezco a mis padres, por el esfuerzo interminable que realizan a diario por mi hermano y por mi. Y a mi hermano, por enseñarme que todo objetivo se logra a base de esfuerzo, compromiso y dedicación.

A mi compañero Juan Manuel, que siempre me brinda su apoyo.

Agradezco especialmente a mis asesores de tesis, el Dr. Eduardo Steed Espinoza Quesada y el Dr. Sergio R. Salazar Cruz por asesorarme a lo largo de mi estancia en este grupo de investigación. También agradezco a mis compañeros de Maestría por el apoyo que me brindaron a lo largo de mi estancia. En especial a Jossué Cariño Escobar por su apoyo y paciencia durante algunas etapas del proyecto.

También le agradezco al Centro de Investigación y Estudios Avanzados del Instituto Politécnico Nacional (CINVESTAV), al programa de Sistemas Autónomos de Navegación Aérea y Submarina, al laboratorio UMI-LAFMIA y al Laboratorio Nacional en Vehículos Autónomos y Exoesqueltos junto con todos sus integrantes y docentes por el apoyo otorgado, durante el desarrollo de este trabajo de tesis.

Al Consejo Nacional de Ciencia y Tecnología (CONACYT) por el apoyo económico brindado mediante la Beca para Estudios de Posgrado (Maestría).

Dedicado a mi madre María de la Luz, mi padre Fausto y mi hermano Alejandro que gracias a su esfuerzo, su cariño y su paciencia, me han permitido concluir este proyecto

Tabla de Contenido

Agradecimientos **I**

Dedicatoria **III**

Lista de Figuras **VII**

Lista de Tablas **X**

Lista de Abreviaciones **XI**

Resumen **XIII**

Abstract **XV**

1. Introducción **1**

1.1. Motivación 2

1.2. Objetivo General 3

1.2.1. Objetivos Particulares 3

1.3. Justificación 4

1.4. Estado del Arte 5

1.4.1. Algoritmos de detección 5

1.4.2. Algoritmos de seguimiento 9

1.5. Alcance 15

1.6. Metodología de Diseño 17

1.7. Organización de la Tesis 20

2. Detección y Seguimiento del Objetivo **23**

2.1. ORB (Oriented FAST and Rotated BRIEF) 24

2.2. Detección por color 28

2.3. Flujo Óptico mediante el método de Lucas-Kanade 33

2.4. CMT: Consensus-based Matching and Tracking of Keypoints for Object Tracking 37

3. Arquitectura del sistema Cámara-Gimbal **45**

3.1. Integración de Hardware 46

3.2. Implementación de Software 47

3.3. Sistema de Control 57

4. Desarrollo del Prototípo **65**

4.1. Características Generales 66

4.2. Descripción del Sistema Embebido 72

5. Pruebas y Resultados Experimentales **77**

5.1. Pruebas de los Algoritmos de Detección y Seguimiento 77

5.2. Pruebas Experimentales del Prototipo 83

5.2.1. Pruebas de Vuelo 84

5.2.2. Control en el Sistema Cámara - Gimbal 84

6. Conclusiones y Trabajo Futuro **93**

6.1. Conclusiones 93

6.2. Trabajo Futuro 94

Apéndice A **97**

6.3. Conceptos Generales 97

Apéndice B **101**

6.4. Codigo del Sistema Embebido 101

Bibliografía **113**

Lista de Figuras

1.1. Diagrama jerárquico de la diferencia entre los algoritmos utilizados para seguimiento de objetivos. [2] 6

1.2. Diferenciación entre dos imágenes: (a) Imagen resultante de una sustracción. (b) Imagen clara y sin ruido que muestra el objeto detectado. [2]. 8

1.3. Sustracción de fondo: (a) Imagen resultante. (b) Resultado de la sustracción. (c) Imagen mostrando los objetos detectados. [2]. 9

1.4. Algoritmo *Mean-Shift*: (a)-(d) Seguimiento progresivo de un objeto en movimiento en un video de prueba realizado para el análisis del algoritmo. [2]. 14

1.5. Algoritmo *SIFT*: (a)-(c) Resultado del mapeo y ubicación de los puntos coincidentes. Análisis realizado en un video de prueba. [2]. 15

1.6. Metodología de diseño propuesta (Diagrama de Flujo) 19

2.1. Detector de puntos de interés FAST. [2]. 25

2.2. Cómo se compone una imagen en formato RGB. 29

2.3. Operaciones morfológicas. 31

2.4. Vehículo de pruebas. 32

2.5. Los vectores de votación se producen sólo en el primer cuadro. Los votos se escalan y se orientan de acuerdo con la constelación de los *keypoints* actuales. [18]. 40

2.6. Consenso en el comportamiento de votación. [18]. 41

3.1. Diagrama de controlamiento del sistema Cámara-Gimbal. 46

3.2. Archivo srv: tipo de mensaje utilizado por el servidor/cliente. 49
3.3. Diagrama de comunicación de los Nodos. 51
3.4. Diagrama de comunicación del sistema completo. 52
3.5. Parámetros de envío. 55
3.6. Recepción de las coordenadas x,y en el sistema de control. 56
3.7. Vehículo en el espacio 3D y su proyección en el plano de la imagen de la cámara. 58
3.8. Coordenadas del objetivo en el marco referencial de la imagen de la cámara. 59
3.9. Diagrama del control PID. 60
3.10. Derivación numérica y Diferenciador de Levant. 63

4.1. Hexarotor, prototipo utilizado. 66
4.2. Adaptaciones en la estructura. 67
4.3. Componentes de la plataforma. 69
4.4. Bateria 6S1P 69
4.5. a) Controlador de vuelo Pixhawk 1. b) GPS utilizado. 72
4.6. a) Computadora NUC 5i7RYH. b)Microcontrolador Arduino Mega 2560. 73
4.7. Gimbal TAROT ZYX FLIR y su Software de asistencia. 74
4.8. Diagrama de conexiones del Hardware. 75

5.1. Imágenes 60 y 71 de la secuencia originial de un video de pruebas, (a) Vectores de flujo óptico siguiendo al vehículo. (b) Los vectores de flujo óptico en una imágen borrosa. 78
5.2. Imágenes a escala de grises 03 y 45 de la secuencia de un video de pruebas. (a)Vehículo de referencia con mayor iluminación (b) Cuadro de la secuencia donde aparece el vehículo correspondiente al análisis con (a). Imágenes a escala de grises 07 y 22 de la secuencia 2. (c) Vehículo de referencia con menor iluminación. (d) Cuadro de la secuencia donde aparece el vehículo correspondiente al análisis con (c). 80

5.3. Imagen 56 de la secuencia de un video de pruebas. (a) Resultados de la substración en la imagen. (b) Imagen clara y sin ruido que muestra el vehículo detectado. 81
5.4. Imagen 110, 135 y 54 de la secuencia de un video de pruebas. (a), (b) y (c) muestran los resultados del algoritmo, detectando al vehículo utilizando un recuadro azul y además mostrando los *keypoints* del objeto con el mismo color. 82
5.5. Algoritmo CMT en exteriores (a) Automovil Sedán (b) Rover (c) Camioneta Pick Up. 86
5.6. Pruebas de vuelo sobre el hexarotor. (a) Automovil Sedán (b) Rover (c) Camioneta Pick Up. 87
5.7. Selección del objetivo. (a) Automovil Sedán (b) Rover (c) Camioneta Pick Up. 88
5.8. Camioneta Pick Up: Posición en el eje X en el marco referencial de la Imagen. 89
5.9. Camioneta Pick Up: Posición en el eje Y en el marco referencial de la Imagen. 89
5.10. Camioneta Pick Up: Velocidad en el eje X e Y en el marco referencial de la Imagen. 90
5.11. Rover: Posición en el eje X en el marco referencial de la Imagen. . . . 90
5.12. Rover : Posición en el eje Y en el marco referencial de la Imagen. . . 91
5.13. Rover: Velocidad en el eje X e Y en el marco referencial de la Imagen. 91

6.1. Diferentes tipos de segmentación de imagenes 100

Lista de Tablas

1.1. Problemas que se presentan en el seguimiento de objetivos. 12
1.2. Comparación de los algoritmos de detección y seguimiento de objetivos. 16

2.1. Comparación entre los algoritmos de visión de detección y seguimiento implementados. 44

3.1. Envío de datos al sistema de control 55

4.1. Especificaciones del motor Multistar 5008-340. 68
4.2. Especificaciones del Controlador de Velocidad Turnigy PLUSH-40A. . 68
4.3. Especificaciones del Controlador de Vuelo Pixhawk 1. 71
4.4. Especificaciones de la mini computadora NUC 73
4.5. Especificaciones del microcontrolador Arduino Mega 2560. 74
4.6. Especificaciones del gimbal TAROT ZYX FLIR. 74

Lista de Abreviaciones

- **AMIS** : Asociación Mexicana de Instituciones de Seguros
- **BRIEF** : Binary Robust Independent Elementary Features
- **CMOS** : Complementary Metal-Oxide-Semiconductor
- **CMT** : Consensus-based Matching and Tracking of Keypoints for Object Tracking
- **FAST** : Features from Accelerated Segment Test
- **GPS** : Global Positioning System
- **GPU** : Graphics Processing Unit
- **UMI** : Unidad de Medición Inercial
- **MIL** : Multiple Instance Learning
- **OpenCV** : Open Source Computer Vision Library
- **ORB** : Oriented FAST and Rotated BRIEF
- **PWM** : Pulse-Width Modulation
- **ROS** : Robot Operating System
- **SIFT** : Scale-Invariant Feature Transform
- **SURF** : Speeded-Up Robust Features
- **TLD** : Tracking-Learning-Detection
- **VANT** : Vehículo Aéreo No Tripulado
- **WLAN** : Wireless Local Area Network

Resumen

En el presente trabajo, se presenta el desarrollo de un sistema de seguimiento de vehículos basado en algoritmos de visión artificial integrado a una plataforma aérea no tripulada.

En la revisión realizada en la literatura actual con respecto a los algoritmos dedicados a la detección y seguimiento de vehículos, se encontraron ORB (Oriented FAST and Rotated BRIEF), Flujo Óptico por el método de Lucas-Kanade, la detección por Color y CMT (Consensus-based Matching and Tracking of Keypoints for Object Tracking). Seleccionando únicamente un algoritmo para evaluar, CMT, para el cual se realizaron las pruebas respectivas y la codificación para la implementación en un sistema embebido.
Los criterios utilizados para determinar un algoritmo se basan en un análisis cualitativo, realizado con tomas de imágenes en un banco de pruebas.

Se realizó la validación del algoritmo en tiempo real utilizando un vehículo aéreo no tripulado (VANT) mediante una interfaz en tierra que permitiera al usuario seleccionar un vehículo y observar su detección.

Abstract

The present work describes the development of a vehicule tracking system based on artificial vision algorithms integrated at an Unmanned Aerial Vehicle (UAV).

From the review realized in the current literature about the methods dedicated to vehicule detection, were found the following algorithms: ORB (Oriented FAST and Rotated BRIEF), Optical Flow by the Lucas-Kanade method, Color detection and CMT (Consensus-based Matching and Tracking of Keypoints for Object Tracking). Selecting only one algorithm to be evaluate, CMT, doing the corresponding tests and its codification in the embedded system.

The criteria used to determine the best algorithm was based on a qualitative analysis realized with frames from a video, taken on a test platform.

Real-time algorithm validation was performed by using an unmanned aerial vehicle (UAV) through a ground interface that allows the user to select a tracking vehicle and observe its detection.

Capítulo 1

Introducción

El problema de seguimiento de objetivos es un área importante de la visión artificial. Los algoritmos de seguimiento pueden ser utilizados en diversas aplicaciones tales como el análisis y control del tráfico terrestre, el reconocimiento de personas o vehículos, en operaciones de vigilancia y rastreo de autos, etc. A pesar de que distintos métodos de seguimiento de objetivos han sido estudiados y analizados por décadas y que en ese tiempo muchos algoritmos fueron desarrollados, el actual estado del arte está lejos de tener una solución perfecta y universal para cada aplicación.

Debido al enorme conjunto de variables, parámetros y entornos, hablando de características como el fondo, la iluminación, las características físicas de los objetos rastreados, etc., es casi imposible desarrollar un algoritmo de seguimiento universal. Por otra parte, la selección de un algoritmo en un principio no depende sólo del algoritmo en sí, sino también de su implementación. El lenguaje de programación, el compilador y la optimización manual pueden afectar significativamente el rendimiento y la solidez del algoritmo. Por este motivo, se ha decidido realizar un análisis que incluya a los algoritmos de seguimiento pre-implementados disponibles en las librerias de OpenCV.

Las librerias de OpenCV resultan ser muy conocidas en el tópico de visión artificial, que integra las estructuras y herramientas necesarias para los algoritmos de

visión artificial por computadora. Además, se pueden utilizar diferentes métodos de optimización que incluyen programación en paralelo, la utilización del GPU, etc., para ajustar el rendimiento del algoritmo seleccionado.

El presente trabajo ofrece una aportación en el creciente problema del robo de unidades de carga pesada que se presenta actualmente, realizando un análisis del funcionamiento e implementación de algoritmos de visión utilizados para el seguimiento de vehículos terrestres, utilizando la plataforma de un vehículo aéreo no tripulado mejor conocido como VANT, en la categoría de multirotor especificamente un hexarotor, realizando pruebas en interiores y exteriores.

1.1. Motivación

Durante el año 2017 el robo a tractocamiones asegurados creció 51 por ciento respecto al 2015, de acuerdo con las estadisticas de la Asociación Mexicana de Instituciones de Seguros (AMIS). En el 2015, la delincuencia se apropió de 6 mil 658 tractocamiones, al año siguiente de 8 mil 383 y en 2017 de 10 mil 64. Del total del año pasado, 2 mil 102 fueron de la marca Kenworth, es decir, la delincuencia robó 1.6 de cada 100 tractocamiones asegurados fabricados por la empresa. Hace tres años se hablaban de 4 mil unidades de equipo pesado robado y hoy se habla de más de 10 mil unidades según el director Carlos Jiménez de la AMIS.

Generalmente, el robo a tractocamiones sucede en las carreteras, cuando un grupo de automóviles u otros tractocamiones impiden el paso. El estado de México, Puebla, Jalisco y Veracruz, son las entidades donde se registran más robos.

Esto impacta directamente en los costos operativos de las empresas de transporte, ya que no solo se suman los gastos previstos para seguros, sino que hay que tener en cuenta los gastos de recuperación de los vehículos y los compromisos perdidos por el retraso de entregas, entre otros inconvenientes.

Afortunadamente, es posible tomar medidas preventivas para evitar robos a los camiones en plena ruta, o minimizar el impacto de los delitos cometidos hacia la pro-

piedad mediante la aplicación de tecnologías orientadas a la seguridad del vehículo. Empresas de logística ya trabajan buscando como proteger la carga de sus clientes. Tal es el caso de Grupo Logistics, que ha desarrollado estrategias de seguridad como rastreo GPS (*Sistema de Posicionamiento Global*) y seguimiento satelital para realizar un control de su flota y conocer la posición de cada vehículo en todo momento, software de seguimiento, sincronización de rutas y hasta drones.

Es por eso que surge la necesidad de desarrollar una primera aproximación en el extenso tema de la implementación de algoritmos de visión artificial enfocados en el seguimiento de vehículos en especial de vehículos de carga pesada mediante un VANT que ayuden a minimizar los delitos como el robo a tractocamiones mediante vigilancia y seguimiento aéreo en zonas de difícil acceso o en sitios que resulten muy complicados para acceder por personal autorizado salvaguardando su integridad física.

1.2. Objetivo General

Desarrollar e implementar algoritmos de procesamiento de imágenes que permitan realizar la identificación y seguimiento de un objetivo en tierra utilizando un vehículo aéreo.

1.2.1. Objetivos Particulares

- Realizar un análisis comparativo sobre los algoritmos de procesamiento de imágenes **ORB**, detección por **Color**, Flujo óptico por **Lucas-Kanade** y **CMT**.
- Identificar un objetivo en tierra utilizando un algoritmo de procesamiento de imágenes e implementarlo en una computadora embebida.
- Implementar leyes de control en un sistema gimbal-cámara para realizar el seguimiento del objetivo.

- Utilizar un Vehículo Aéreo No Tripulado (VANT) para la implementación del sistema de identificación de objetivos.
- Realizar pruebas de funcionamiento de los algoritmos que se utilizaron.

1.3. Justificación

La problemática de robo a tractocamiones alcanzó un registro a la alza que en épocas anteriores no se había visto en unidades aseguradas, lo que representa un incremento del 28 por ciento comparado con las cifras reportadas en el mismo periodo del 2017 según revela la Oficina Coordinadora de Riesgos Asegurados (OCRA).
La Asociación Nacional de Transporte Privado (ANTP), enfatizó que estos datos son referentes al sector asegurado, el cual representa solo al 35 por ciento del parque vehicular del país.

Existen métodos que se han ocupado para evitar o reducir las probabilidades de robo como la instalación de un GPS y el seguimiento de la unidad vía satélite, sin embargo, los delincuentes han logrado identificar los sistemas de protección y detección de los camiones, de tal manera que pueden desinstalarlos, o bien, bloquearlos.

La visión artificial se está enfatizando como una tecnología imprescindible en los sistemas de vigilancia gracias a las posibilidades que ofrece de obtener la máxima información.

Los algoritmos de detección y seguimiento pueden ser de gran utilidad en tareas como el seguimiento de un vehículo ya que este tipo de detección se basa en características que la percepción visual humana no podría distinguir logrando así no perder nunca el objetivo de la lente de la cámara.

El utilizar un vehículo aéreo no tripulado, sería de gran utilidad para seguir a un vehículo en lugares de dificil acceso o en una zona abierta. Además de ser una opor-

tunidad de evitar exponer a recursos humanos en este tipo de tareas.

A pesar de que este tipo de sistemas de detección de objetivos en movimiento con vehículos no tripulados ya han sido implementados en otros países lo que se busca es enfocarnos en desarrollar tecnologías similares en nuestro pais, que puedan ser construidas y adquiridas a un menor costo en comparación a otros sistemas importados y a su vez adaptarlos a las necesidades de la industria nacional.

1.4. Estado del Arte

La detección y seguimiento de un objeto en movimiento es un gran reto y una tarea esencial en sistemas de video vigilancia y es un punto de enfoque desde los años 60's. La literatura científica reporta un gran número de técnicas y algoritmos que sirven para el análisis de objetos en movimiento. Este análisis tiene dos partes principales, la detección y el seguimiento.
Para analizar el movimiento de un cuerpo, el primer paso es detectar el objeto de forma fiable y eficiente. Basado en esta detección el siguiente proceso es el seguimiento. Un diagrama jerárquico se muestra en la Figura 1.1, presentando sólo algunos del basto repertorio científico de algoritmos de detección y seguimiento.

1.4.1. Algoritmos de detección

Para detectar de manera eficiente y precisa los objetos en movimiento, se conocen dos métodos básicos. El método de diferenciación entre dos imágenes, y el método de sustracción de fondo.

Los detectores no son realmente seguidores. Éstos unicamente realizan el intento de detectar un objeto en específico en cada imagen ó cuadro individualmente. Digamos que trabajan de esta forma: suponiendo que se tienen dos imágenes, una es una

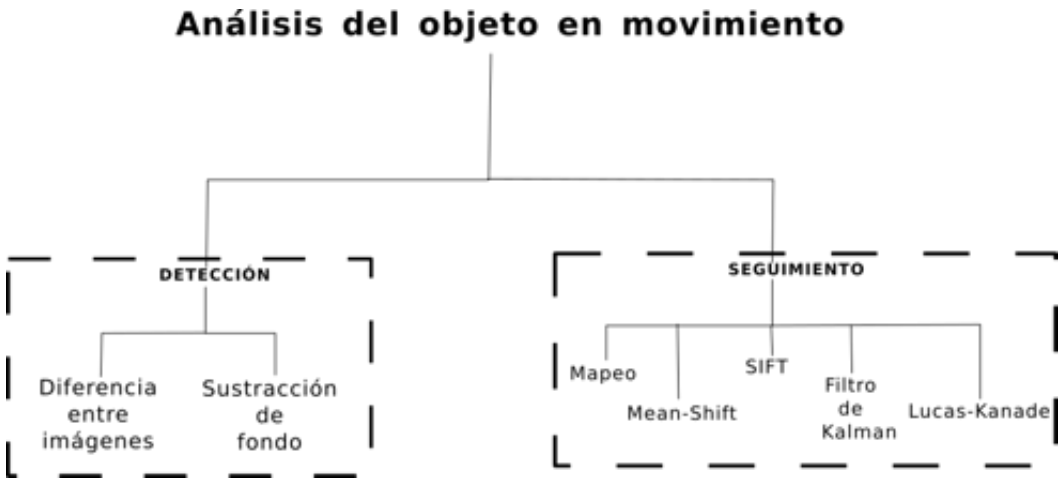


Figura 1.1: Diagrama jerárquico de la diferencia entre los algoritmos utilizados para seguimiento de objetivos. [2]

imagen de un objeto que se quiere seguir y la otra es el primer cuadro de un video ó la primer imagen obtenida desde una transmisión en tiempo real. Es posible obtener la primer imagen desde la segunda por medio de una selección rectangular.

Los detectores entonces intentan detectar algunas características en la imagen donde se encuentra el objeto para después intentar hallar el mejor mapeo de estas características en la imagen actual. Esto funciona muy bien si la imagen del objeto es lo suficientemente grande y el objeto en sí tiene características detectables, como bordes y textura. El objeto en cuestión puede girarse libremente en el plano del marco o girarse ligeramente en otros planos mientras mira aproximadamente en la misma dirección. En OpenCV [1], se utilizan las funciones de Homografía para encontrar la transformación entre los puntos clave emparejados y la función de transformación de perspectiva para mapear los puntos.

La razón por la que no se puede decir que los detectores de características son realmente seguidores es su inconsistencia entre imágenes. Los seguidores o rastreadores, siguen la trayectoria, los detectores sólo detectan la mejor coincidencia para dos imágenes, lo que hace que sean muy inestables, especialmente al confundir el objeto rastreado con algo en un preciso momento muy similar al objeto definído ori-

ginalmente.

Otro problema con los simples detectores podría ocurrir cuando el objeto rastreado es el mismo que uno o más objetos en la imagen o es parte de una estructura de repetición (por ejemplo, ventanas, cercas, etc.). Esto puede convertirse en un problema con el seguimiento en general, pero puede ayudarse enfocándose en la ubicación del objeto en la imágen. En este punto entran los seguidores.

El algoritmo de **diferenciación de imágenes** o cuadros se usa ampliamente para detectar objetivos en movimiento debido a la eficiencia de sus resultados. [2]. Este depende del cuadro de referencia y el cuadro actual, un umbral y los cambios de iluminación. Proporciona el centroide (coordenadas centrales) del objeto en movimiento y el área del objeto en movimiento como salida. De acuerdo con este algoritmo, en el caso de un video grabado o tomado en tiempo real, se toman dos imágenes consecutivas y luego se comparan entre sí dentro de cierto umbral. Los objetos que han cambiado su posición con respecto al cuadro anterior se tratan como objetos en movimiento. El reprocesamiento implica operaciones como el filtrado y operaciones morfológicas. En la Figura 1.2 podemos ver la aplicación de un filtro para remover el ruido y refinar la imagen realizando operaciones morfológicas.

El algoritmo de **sustracción del fondo** por otra parte es una técnica eficiente especialmente en casos donde la cámara y el fondo se encuentran estáticos. [2]. Este algoritmo tiene numerosas aplicaciones en análisis de movimiento, Interacción Hombre-Computadora (IHC) y sistemas de control inteligente. Su objetivo es separar el objeto en movimiento deseado del fondo del video y utilizarlo en operaciones avanzadas como la segmentación, el reconocimiento y el seguimiento. Este método se basa en un modelo dinámico del fondo y un umbral dinámico. Sin embargo es una técnica muy sensible a cambios en el exterior, como los efectos de variación de la luz.

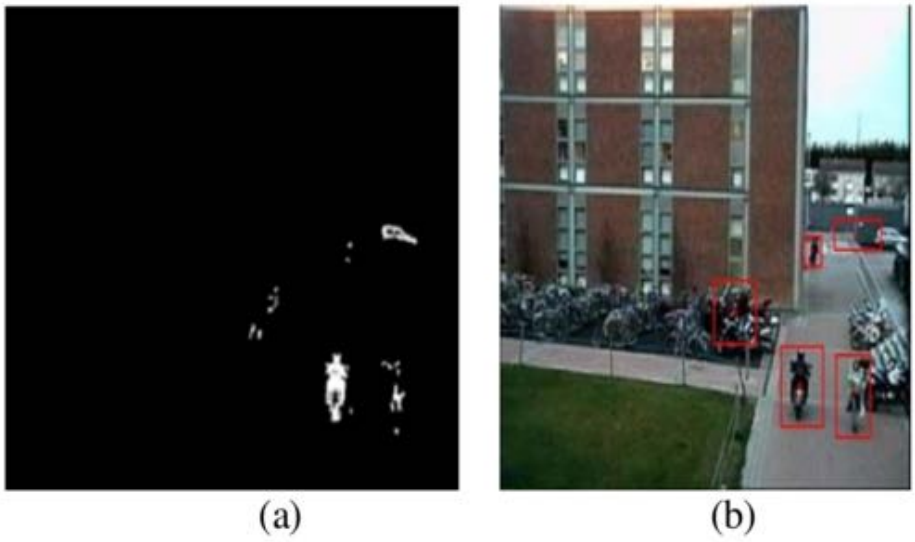


Figura 1.2: Diferenciación entre dos imágenes: (a) Imagen resultante de una sustracción. (b) Imagen clara y sin ruido que muestra el objeto detectado. [2].

El algoritmo utiliza la técnica anteriomente mencionada para la detección de objetos en movimiento (ver Figura 1.3). La imagen anterior con ciertas características modeladas dinámicamente actúa como un fondo para la imagen actual. Tomando la diferencia de las dos imágenes se logra indicar a los objetos en movimiento. Luego se aplican operaciones morfológicas para refinar la imagen y finalmente, para señalar el objetivo detectado.

En OpenCV, actualmente hay tres detectores de características útiles: **SURF** [3], **SIFT** [4] y **ORB** [5]. Los dos primeros utilizan números de punto flotante pero están patentados. El tercero usa números enteros y, por lo tanto, es menos preciso, pero es más rápido y tiene una licencia que facilita el uso del algoritmo por su sencillez en la implementación. Los conceptos básicos de los algoritmos SIFT y SURF y su uso con la implementación de OpenCV se pueden encontrar en [6].

Figura 1.3: Sustracción de fondo: (a) Imagen resultante. (b) Resultado de la sustracción. (c) Imagen mostrando los objetos detectados. [2].

1.4.2. Algoritmos de seguimiento

Después de una detección exitosa, el seguimiento de objetos en movimiento es una tarea importante en el análisis de movimiento.

Seguidores Puros

Para realizar una mejor distinción entre esta sección y la siguiente parte, se usará el término *seguidores "puros"* para los seguidores que únicamente realizan esta acción y nada más. Los seguidores están diseñados para seguir un objetivo siguiendo su trayectoria y prediciendo su posición en el futuro. Esto incluye la corrección de los errores en el proceso.

El gran problema con los *seguidores puros* es que cuando la cámara se mueve muy rápido o el objetivo de repente cambia su velocidad o dirección, los seguidores empienzan a tener dificultades. En estos casos, pueden intentar detectar el objetivo en una ubicación que resulte más predecible y encontrar algo similar en su lugar.

En OpenCV, existen los siguientes seguidores puros: **MIL** [7], **Boosting** [8] y **MedianFlow** [9].

Marcos de Seguimiento. Los marcos de seguimiento son algoritmos que intentan proporcionar la solución más compleja para el seguimiento paso por paso. Pueden ser considerados como un producto listo para usarse.

Están diseñados para superar los problemas más graves en el seguimiento adaptandose constantemente a nuevas condiciones y corrigiendo todos los errores que se presenten. Su desventaja es que requieren de más capacidad de memoria y consumo de energía en el procesamiento. Éste es el precio que hay que pagar por todas sus ventajas.

En OpenCV, se puede encontrar actualmente un marco de seguimiento: **TLD** [10]. Este algoritmo está dividido a grandes rasgos en tres partes independientes -*Tracking* -*Learning* -*Detecting*. El seguidor pretende seguir a un grupo de pixeles de una imagen a otra. El detector intenta encontrar objetos similares y corregir al seguidor si es necesario, y el aprendizaje, mejor conocido como *learning* en inglés, estima los errores de los detectores y se actualiza constantemente para eliminar esos errores en el futuro. Gracias a esto, TLD puede proporcionar un seguimiento estable a largo plazo. Adiconalmente, este marco es capaz de adaptarse a cambios en los conjuntos de pixeles.

Las tres partes básicas de TLD pueden correr simultaneamente, en un proceso diferente ó como una tarea separada. Además, cada una de ellas se puede optimizar (paralelamente) por separado debido al carácter que presenta cada una de ellas.

Métodos de medición

Realizar una comparación de los diferentes algoritmos de seguimiento puede ser muy complejo y difícil. La razón principal de esto es que el seguimiento en sí mismo puede ser usado para muchos própositos diferentes (seguimiento de rostros, vehículos, personas, etc.) en diversos entornos y situaciones (en un aeropuerto, dentro de un edificio, en caminos, en la naturaleza, en la niebla, de día o de noche, etc.) mientras que las cámaras que se utilizan pueden ser de baja o alta calidad, se pueden hacer tomas desde cerca o lejos, a color o en escala de grises, en movimiento o estáticas y podriamos seguir. Además, el seguimiento en sí consiste en superar diferentes tipos de problemas como lo son la rotación o la oclusión parcial de un objetivo, los cambios de iluminación las imágenes desenfocadas debido a movimientos rápidos de la cámara, etc. Esto significa que se necesita un conjunto de datos muy grande, para clasificar todos los problemas que presentan los seguidores. Afortunadamente en En [11] han logrado recopilar los videos de prueba más utilizados y ofrecerlos libremente con anotaciones. En ellos utilizan su base de datos para probar los algoritmos de seguimiento que se encuentran disponibles con su código fuente. Cada uno de los videos en la base de datos ha sido descrito enumerando los problemas principales a los que se han enfrentado en la implementación de cada algoritmo de seguimiento. En la tabla 1.1 podemos ver la recapitulación de los problemas a los que se enfrentan cada algoritmo.

Tabla 1.1: Problemas que se presentan en el seguimiento de objetivos.

Nombre	**Descripción**
Variación en la iluminación	La iluminación en la región del objetivo cambia significativamente
Variación en la escala	La relación de los recuadros que delimitan al objeto en el primer cuadro de un video y el cuadro actual están fuera de rango
Oclusión	El objeto esta parcialmente o completamente ocluido
Deformación	Deformación de un objeto no rígido
Difusión	La región del objeto es borrosa debido a movimientos del objeto en la cámara
Movimientos rápidos	El movimiento es tan rápido que se sale de los límites de la cámara
Rotación en el plano	El objeto gira en el plano de la imagen
Rotación fuera del plano	El objeto gira fuera del plano
Fuera de vista	Una porción del objeto está fuera de vista
Fondo	El fondo cercano al objeto tiene un color o texturas similares al objeto
Baja resolución	El número de pixeles dentro de la región del objeto es reducido.

Existen muchos algoritmos que realizan el seguimiento de manera eficiente sobre objetos en movimiento reportados en la literatura como hemos visto, y algunos de estos algoritmos incluyen **Template Matching** [12], **Mean Shift** [13], **SIFT** [4],

Lucas-Kanade [14], entre otros. Cada uno de ellos, tiene sus propias capacidades y limitaciones.

El algoritmo *Template Matching (TM)*, es simple pero muy eficiente para el seguimiento mejor conocido como *tracking*. El algoritmo puede ser implementado en diferentes formas, por correlación matemática, detección de bordes, análisis de texturas, etc.

El algoritmo TM funciona como un módulo de seguimiento. Puede ser utilizado para seguir objetos estáticos o en movimiento. El método puede reconocer y detectar el objeto de interés independientemente de su condición, ya sea en movimiento ó estático y también de la condición de los demas objetos que lo rodean.
Este método utiliza el proceso de mapeo en pixel por pixel. Entre sus limitaciones se incluyen el mapeo parcial de la imagen debido a límites en la misma y no es robusto a variaciones en la escala.

El algoritmo *Mean-Shift* es altamente utilizado en el seguimiento de objetivos, basado en cálculos estadisticos de color en el modelo del objeto y el modelo del posible candidato. Ver Figura 1.4. El algoritmo cambia iterativamente un punto de datos al valor promedio de los puntos de su vecindario. Utiliza la información del histograma de color y los cálculos estadísticos para localizar la nueva posición del objeto.

Otro algoritmo ampliamente utilizado es el algoritmo *SIFT*. Este algoritmo esta basado en la extracción y en el mapeo de puntos característicos. Extrae las características del objeto que son invariantes ante la iluminación y la orientación, y continuamente está detectando y mapeando estas características. Es importante mencionar que el modelo SIFT también calcula la posición de los puntos característicos y los mapea con la posición de los puntos característicos que son posibles candidatos en otra imagen basandose en la distancia Euclidiana. Los principales cálculos que realiza este algoritmo son los siguientes:

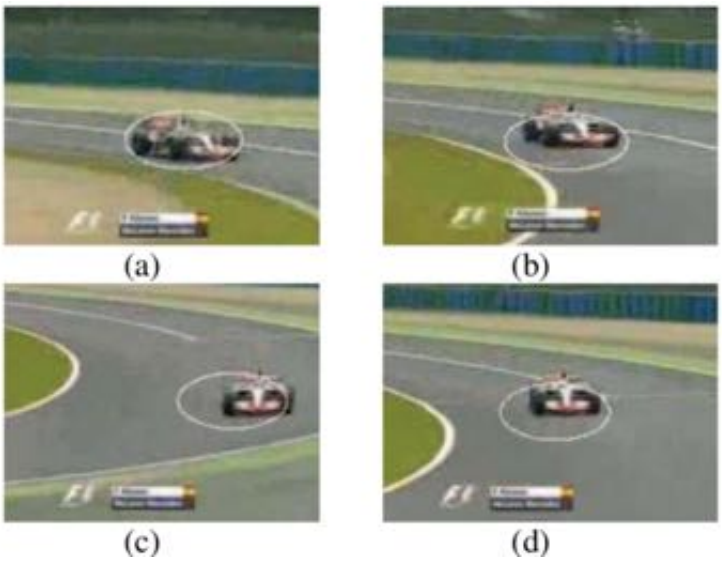


Figura 1.4: Algoritmo *Mean-Shift*: (a)-(d) Seguimiento progresivo de un objeto en movimiento en un video de prueba realizado para el análisis del algoritmo. [2].

1 . Detección de los extremos (escala-espacio)

2 . Localización de los puntos

3 . Asignación de la orientación

4 . Formulación de los vectores descriptores de los puntos

Es importante mencionar que para poder realizar el seguimiento del objeto es necesario realizar un proceso de mapeo entre los descriptores, como se puede ver en la Figura 1.5, y una ves que se realiza, seleccionar la posición de los puntos mapeados del objetivo para que con estos se pueda realizar el seguimiento.

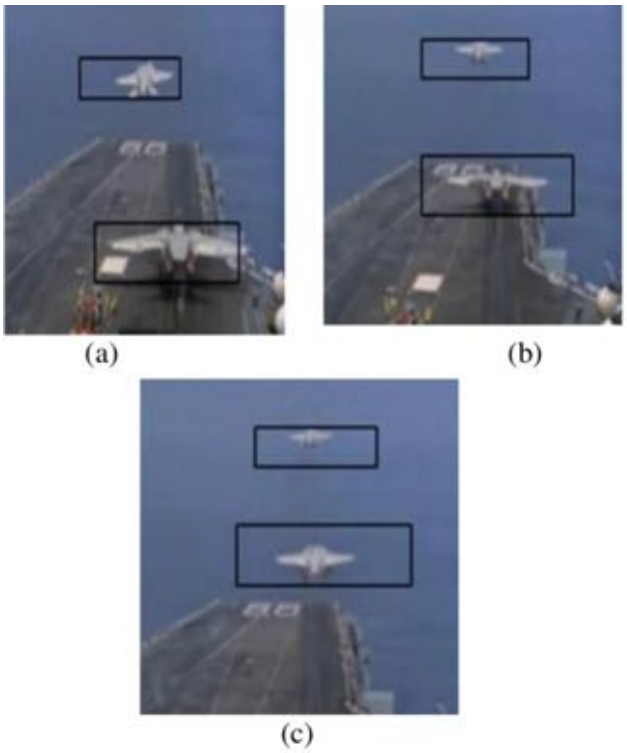


Figura 1.5: Algoritmo *SIFT*: (a)-(c) Resultado del mapeo y ubicación de los puntos coincidentes. Análisis realizado en un video de prueba. [2].

En esta sección, distintos algoritmos de detección y seguimiento fueron comparados con base en sus propiedades, eficiencia, especificaciones y limitaciones. La Tabla 1.2 presenta una visión general de los algoritmos mencionados anteriormente.

1.5. Alcance

En la presente investigación se pretende desarrollar un sistema de detección y seguimiento de un vehículo, seleccionando un algoritmo de visión artificial que permita realizar el seguimiento de forma satisfactoria a partir del estudio comparativo de diversos algoritmos que existen actualmente en la literatura. Seguido de la implementación en un VANT de tipo multirotor.

Con el fin de lograr estos objetivos, se han abordado las siguientes áreas de investigación.

Tabla 1.2: Comparación de los algoritmos de detección y seguimiento de objetivos.

No.	Algoritmo	Detección	Seguimiento	Dependencias	Limitaciones	Desempeño	Complejidad
1	Diferenciación de imagen	√	√	Imagen de referencia y actual, umbral, iluminación	Múltiples objetos, efectos de sombreado	Buena detección, Seguimiento pobre	Simple
2	Sustracción de fondo	√	X	Modelo dinámico del fondo,umbral	Cámara estática, cambios en el entorno	Detección eficiente, calcula la posición del objeto	Media
3	Mapeo entre imágenes	X	√	Umbral, correlación de imagenes	Cambios en la iluminación y forma	Seguimiento eficiente, dibuja la trayectoria del objeto	Simple
4	Mean-Shift	√	√	Color, medición de similitud, función Kernel	Resultado pobre de detección, sin información espacial	Seguimiento eficiente, Posición y trayectoria del objeto	Compleja
5	SIFT	X	√	Puntos característicos, distancia euclideana	Puntos del fondo	Seguimiento eficiente, invariante a cambios en el ambiente	Compleja
6	Filtro de Kalman	X	√	Estimación del pasado, presente y futuro	Cálculo lento, sensible a cambios bruscos	Buen seguimiento bajo condiciones no ideales	Compleja

La primera de ellas, es el área de **Visión Artificial**. Para lograr la detección del vehículo terrestre, primero, se requiere de un estudio científico extenso de los diferentes algoritmos de detección y seguimiento que se han desarrollado actualmente, permitiendo seleccionar el adecuado para esta aplicación y posteriormente profundizar en su implementación.

En segundo lugar, se encuentra el área de **Programación**. En necesario durante todo el proceso realizar la creación, desarrollo y análisis de programas en diversos lenguajes de programación que permitan realizar la implementación de los algoritmos de visión. Además de la realización de código para la implementación de una estación en tierra por medio de las librerias de ROS.

En tercer lugar, el desarrollo de una estrategia de **Control** que permita realizar el seguimiento del objetivo manteniéndolo dentro de los límites de la imagen a través de una plataforma gimbal en la cual se monta una cámara.

Por último, la selección, armado, implementación y adaptaciones de ser necesario de un vehículo multirotor en el cual se realizarán pruebas experimentales de los sistemas de visión artificial.

Destacando el procesamiento de visión artificial, el presente trabajo se limitará a realizar el seguimiento del vehículo utilizando una plataforma embebida de manera desacoplada con el multirotor, es decir, el control de seguimiento se implementará unicamente en la plataforma gimbal, realizando pruebas experimentales en exteriores sobre el VANT.

1.6. Metodología de Diseño

La metodología propuesta se describe en la Figura 1.6 donde se muestra un diagrama de flujo con los principales procesos que se siguen para cumplir con el objetivo principal. Considerando que ya se cuenta con un objetivo general claramente identificado como se muestra al inicio del diagrama de flujo 1.6.

En la **primera** etapa de la metodología, es necesario el diseñar y contruir una plataforma donde sea posible montar el sistema del gimbal y la cámara para su posterior utilización en las pruebas de los algoritmos de visión artificial dentro de un espacio habitual de estudio. También se realizan las adaptaciones necesarias para lograr la estabilización de la cámara sobre el gimbal.

En la **segunda** etapa, se realiza un estudio profundo del estado del arte de los algoritmos de visión artificial utilizados para la detección y seguimiento de objetivos, comparando sus limitaciones, las ventajas, la complejidad y sobre todo la funcionali-

dad de cada uno. Seguido de la selección de un algoritmo de detección y seguimiento que resulte adecuado para la aplicación del seguimiento de un vehículo terrestre.

En la **tercera** etapa, se desarrolla un esquema de control visual necesario sobre el sistema del gimbal, basado en la cámara para correlacionar el movimiento del vehículo en el plano de la imagen de la cámara con el movimiento del gimbal en dos ejes.

Para la **cuarta** etapa, una ves realizado el esquema de control y los algoritmos de visión, es necesario pensar en el diseño de una estación de control en tierra que permita la comunicación del sistema embarcado con un usuario en tierra para la manipulación del sistema de detección. En esta parte se realiza un análisis de cómo realizar la comunicación de manera eficiente, se diseña un protocolo que permita el enviar y recibir datos entre ambas plataformas (en tierra y embarcada).

En la **quinta** etapa, se hace uso de un sistema embebido para la implementación tanto del algoritmo de detección y seguimiento, como del control utilizado para el control del sistema cámara-gimbal y por último de los requerimientos necesarios para la etación en tierra.

Para la **sexta** etapa, debido a que el objetivo general es utilizar un vehículo aéreo para las pruebas del seguimiento del objetivo en tierra, se realiza la selección de un vehículo multirotor que permita realizar la aplicación de manera adecuada. Seguido del armado y configuración de la plataforma aérea y las adaptaciones que requiera para la implementación de los sistemas embarcados.

Por ultimo, se realizan las pruebas experimentales del prototipo completo, realizándolas en diferentes entornos al exterior. Y continuando con el análisis del comportamiento del sistema cámara-gimbal sobre el vehículo aéreo, junto con el análisis del algoritmo de detección y seguimiento en lugares abiertos.

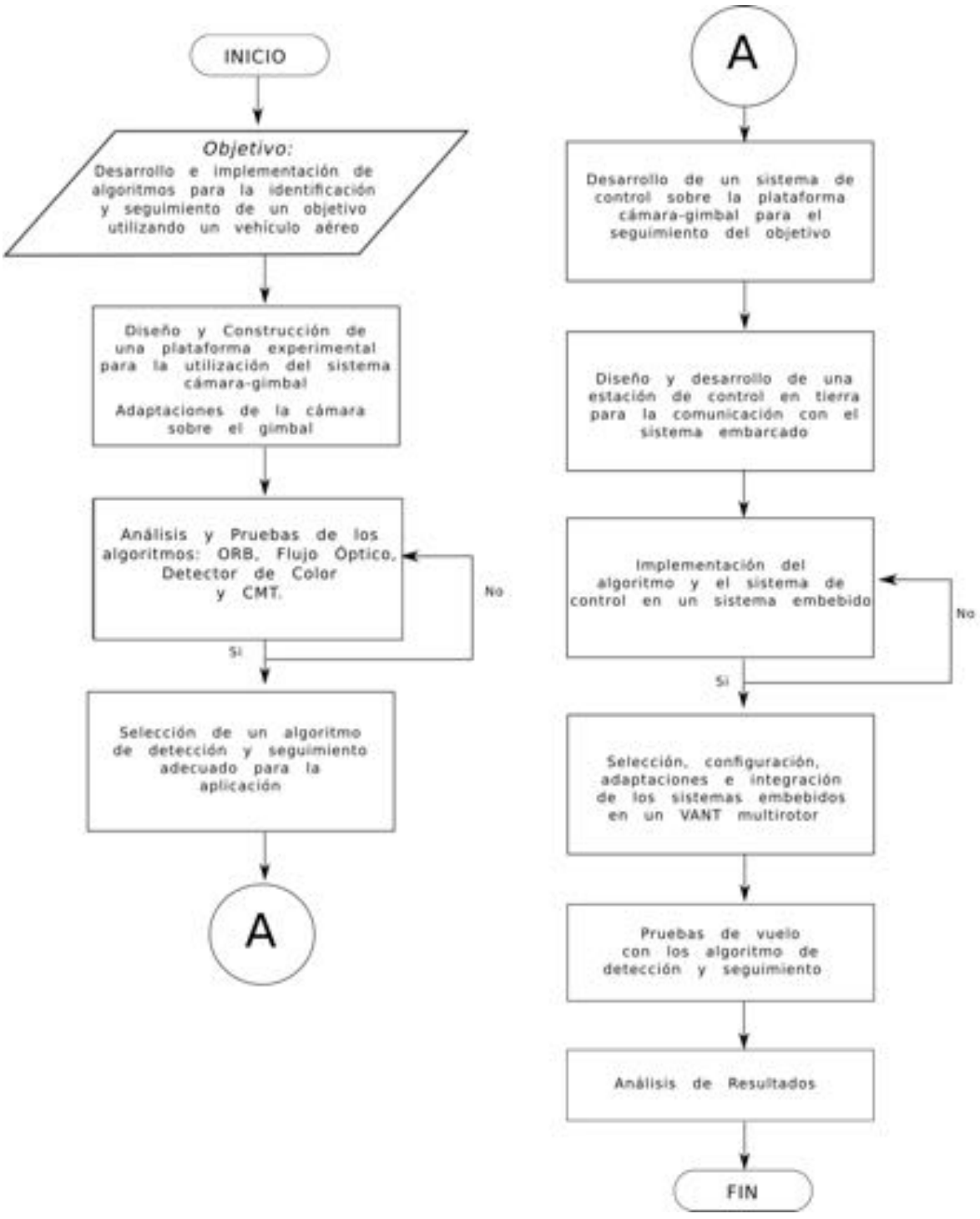


Figura 1.6: Metodología de diseño propuesta (Diagrama de Flujo)

1.7. Organización de la Tesis

El presente trabajo está estructurado en 6 capítulos que a continuación se describen.

En el **Capítulo 1**, se presentan los objetivos del proyecto. Se revisan los conceptos fundamentales en cuestión de la visión artificial relacionados con la detección y el seguimiento de objetivos en el estado del arte. Se discute la metodología utilizada para este trabajo, hablando de las áreas de estudio que se abordan a lo largo del proyecto en el alcance.

En el **Capítulo 2**, se discuten la implementación de los algoritmos que se probaron y se realiza un estudio comparativo entre los mismos. Por otro lado, se revisa de forma breve el desarrollo de cada uno de ellos, explicando de forma preliminar cada uno de ellos.

En el **Capitulo 3**, se presenta la arquitectura del sistema cámara-gimbal. Se explica la interacción del desarrollo de una estación de control en tierra y se discute el algoritmo de control visual utilizado basado en las imágenes del video. También se revisa de forma breve los conceptos de los repositorios de ROS y OpenCV.

En el **Capítulo 4**, se presenta la plataforma aérea utilizada, describiendo sus componentes, explicando las adaptaciones pertinentes que se realizaron sobre ella. Además se realiza la descripción del sistema embarcado, mostrando los sensores y conexiones del prototipo completo.

En el **Capítulo 5** se presentan las pruebas que se realizaron con los diferentes algoritmos en la plataforma de pruebas. Además se muestran los resultados de las pruebas experimentales realizadas con el VANT.

Finalmente, las conclusiones generales y futuro del proyecto se presentan en el

Capítulo 6.

En el **Apéndice A** se describen de forma detallada la implementación del algoritmo CMT utilizando el lenguaje de programación C++. Así como el esquema de comunicación utilizando la plataforma ROS, para la creación de una estación en tierra.

Capítulo 2

Detección y Seguimiento del Objetivo

A lo largo de este capítulo se van a describir con detalle las técnicas implementadas para la realización con éxito de la detección y seguimiento de un objetivo, particularmente de un vehículo, utilizando las librerias de OpenCV. Se van a explorar algunos métodos que se basan en la obtención de puntos característicos (traducción en inglés de *keypoints*, y que en lo sucesivo se le llamará de las dos maneras sin distinción) en imágenes consecutivas.

Las técnicas que se ponen en práctica son la detección de objetivos basandose en algoritmos de detección por **color** [15], el algorítmo de detección robusto **ORB** (*Oriented FAST and Rotated BRIEF*) [5] que se basa en la fusión de un detector **FAST** (*Features from Accelerated Segment Test*) [16] y el descriptor **BRIEF** (*Binary Robust Independent Elementary Features*) [17], el algoritmo de seguimiento de flujo óptico por el método **Lucas-Kanade** [14] y el algoritmo de detección y seguimiento **CMT** (*Consensus-based Matching and Tracking of Keypoints for Object Tracking*) [18].
Posteriormente, se realizará una comparativa entre los distintos métodos, utilizando como base, vídeos que se recrearon para este fín, haciendo un estudio sobre las ventajas y limitaciones que presentan.

Los términos *detección de objetivos* y *reconocimiento de objetivos* son comunmente

interpretados como sinónimos, aunque esto a veces depende de la aplicación y de quien escribe el programa. Otro término, *seguimiento de objetivos*, puede ser encontrado frecuentemente en compañía de algorítmos de detección y reconocimiento. Este trío puede trabajar de manera conjunta para realizar aplicaciones confiables a pesar de no saber en que punto se diferencían uno del otro y como se relacionan entre ellos. A continuación, se hace una clara distinción entre ellos, en primer instancia de manera descriptiva, seguido de la explicación de los algoritmos asociados a cada proceso.

Detección. En un algoritmo de detección se hace la pregunta: ¿Hay algo en la imagen?. Es el proceso de descubrir o encontrar algo, en este trabajo ese *algo* es un vehículo terrestre. Por lo tanto, el objetivo de la detección puede describirse como el descubrimiento de la presencia de un objeto en una imágen.

Seguimiento. Un algoritmo de seguimiento pretende saber hacia donde se dirige algo. El objetivo del seguimiento de un objeto es observarlo constantemente en las imágenes sucesivas de un video. El historial de la localización del objetivo (el seguimiento maneja constantemente las imágenes relacionadas entre sí) permite saber como cambia la posición del objetivo con el tiempo. Esto quiere decir que se puede tener un modelo del movimiento del objetivo (modelo predictivo). [19].

2.1. ORB (Oriented FAST and Rotated BRIEF)

El algoritmo **ORB** es basicamente una fusión del detector de puntos característicos **FAST** y el descriptor **BRIEF** con modificaciones para mejorar su rendimiento.

En primer lugar, este algoritmo utiliza el algoritmo **FAST** para encontrar puntos característicos en una imagen, utilizando este método como una alternativa para aplicaciones en tiempo real debido a la velocidad de procesamiento mayor a otros detectores. Básicamente, este algoritmo se centra en la detección de esquinas.

1. Se selecciona un pixel $\boldsymbol{p}$ en una imagen el cual puede ser o no ser identificado

como un punto de interés. Con una determinada intensidad I_p.

2. Se selecciona un umbral apropiado $\boldsymbol{t}$.

3. Se considera un circulo de 16 pixeles alrededor del pixel de interés. Como se muestra en la Figura 2.1.

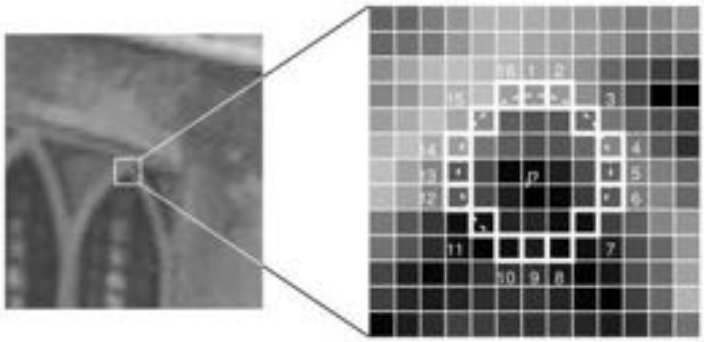


Figura 2.1: Detector de puntos de interés FAST. [2].

4. De tal manera que el pixel $\boldsymbol{p}$ es una esquina si existe un conjunto de $\boldsymbol{n}$ pixeles contiguos en el círculo (de 16 pixeles) el cual sea más brillante que $I_p + t$ ó mas oscuro que $I_p - t$. (El cual se marca con líneas punteadas sobre los pixeles en la Figura 2.1). $\boldsymbol{n}$ es igual a 12.

5. Una prueba de mayor velocidad fue propuesta para excluir un gran número de pixeles que no son esquinas. Esta prueba examina solamente 4 pixeles en las posiciones 1, 9, 5 y 13 (primero se prueban los pixeles en la posición 1 y 9, para saber sí estos pixeles son bastante brillantes u oscuros. Si es así, se prueban los pixeles 5 y 13). Sí $\boldsymbol{p}$ es una esquina, entonces al menos 3 de los pixeles deben ser más brillantes que $I_p + t$ ó más oscuros que $I_p - t$. Si esto no se cumple, entonces $\boldsymbol{p}$ no es una esquina. Este criterio de prueba de segmento completo puede ser aplicado en los candidatos aprobados examinando todos los pixeles del círculo.

Este detector se caracteriza por tener un alto rendimiento, sin embargo no es robusto ante altos niveles de ruido. Y depende de un umbral. [20].

Una vez realizada la detección de puntos característicos, se aplica el método de detección de esquinas de **Harris** para hallar un número $\boldsymbol{N}$ entre ellos. También se utilizan métodos de representaciones en piramide para producir puntos característicos multiescala.

Para la invariancia de rotación, se calcula la intensidad ponderada del centroide del parche con la esquina ubicada en el centro. La dirección del vector del punto de la esquina hacia el centroide proporciona la orientación. Para mejorar la invariancia de rotación, se calculan los momentos en x y y en una región circular de radio $\boldsymbol{r}$, donde $\boldsymbol{r}$ es el tamaño del parche.

Ahora, para los descriptores, **ORB** usa descriptores por el método **BRIEF**. Este método utiliza arreglos binarios sin encontrar los descriptores. Utiliza una región suavizada de la imágen y selecciona un grupo de n_d (x,y) pares de manera única [21]. Después, se comparan las intensidades de algunos pixeles de estos pares. Por ejemplo, sea el primer par p y q. Sí, $I(p) < I(q)$, esto resulta en un *1* de no ser así, el resultado es *0*. Esto se aplica para todo el conjunto n_d para obtener un arreglo de bits n_d-dimensional. El conjunto n_d puede ser de 128, 256 o 512.

El algoritmo **ORB** *direcciona* a **BRIEF** de acuerdo a la orientación de los puntos de interés. Para cualquier conjunto de características de $\boldsymbol{n}$ pruebas binarias en locaciones (x_i, y_i), se define una matriz de 2 x n, S es quien contiene las coordenadas de estos pixeles. Después, usando la orientación de ésta región o parche, θ, se halla la matriz de rotación y se rota el conjunto S para obtener la versión con rotación S_θ. **ORB** es mucho mas rápido computacionalmente que **SURF** y **SIFT** y además tiene un desempeño mejor al de **SURF**. En general, **ORB** es una buena elección en dispositivos donde no se requiere de un costo computacional alto. [5].

En general, el algoritmo de detección **ORB** determina la localización de las par-

tes que conforman el objetivo que se quiere seguir en otras imágenes.
Esta información es suficiente para hallar el objetivo en un video o secuencia de imágenes.

Implementación

En la implementación de este algorítmo se utiliza el video de un vehículo a escala 1:24 para fines ilustrativos, demostrando el comportamiento del algoritmo. El pseudocódigo de la implementación del algoritmo **ORB** se puede ver en el Algoritmo 1. Al inicio, se captura la primer imagen del video, en donde se asume que la imagen contiene al vehículo. Ya que es necesario obtener las características del vehículo,

Algoritmo 1 Descripción de la implementación del Algoritmo ORB

1: **Inicio**
2: **//Variables**
3: $I_1, ..., I_n, I_m, d_1, ..., d_n, H : Mat$
4: $k_1, ..., k_n : vector < Keypoint >$
5: $match, bmatch : vector < DMatch >$
6: $obj, escena : vector < Point2f >$
7: **//Entrada**
8: Leer I_1
9: **//Proceso**
10: $k_1 \leftarrow Detectar(I_1)$
11: $d_1 \leftarrow CalcularDescriptor(k_1)$
12: **Para** $t \leftarrow 2, ..., n$ **Hacer**
13: $k_t \leftarrow Detectar(I_t)$
14: $d_t \leftarrow CalcularDescriptor(k_t)$
15: $match \leftarrow Matching(d_t, d_1)$
16: $bmatch \leftarrow MatchingSelectivo(match)$
17: $I_m \leftarrow DibujarCoincidencias(I_1, I_t, k_1, k_t)$
18: $obj \leftarrow k_1\{bmatch[t = 0, ..., n]\}$
19: $escena \leftarrow k_t\{bmatch[t = 0, ..., n]\}$
20: $H \leftarrow CalcularHomografia(obj, escena)$
21: $r \leftarrow l_1, ..., l_4(I_1, I)$
22: **//Salida**
23: Mostrar(I_m)
24: **FinCiclo**
25: **Fin**

se realiza la detección de puntos característicos y a continuación el cálculo de los

vectores descriptores de los puntos. Una vez que se ha caracterizado al vehículo, el siguiente paso es repetir el proceso de detección y descripción sobre la secuencia de imágenes de dicho video para permitir conocer los puntos caracteristicos y los descriptores de cada imagen, y comparar sus descriptores con los correspondientes a la primer imagen mediante el proceso de *matching*. Esta técnica permite obtener un vector comparativo entre los descriptores correspondientes a cada imagen, con características como la distancia entre ambos descriptores. Discriminando elementos del conjunto mediante un umbral de distancia entre ellos. Por último, se calcula una matriz de transformación que determina la correspondencia de puntos característicos entre la secuencia de imágenes del video y la primera, cambiando la perspectiva de los puntos a una vista deseada calculando la homografía. Utilizando las esquinas del objeto detectado, el vehículo, se obtiene la posición de las mismas en las siguientes imágenes, dibujando con las localizaciones de las cuatro esquinas un rectangulo que delimite al vehículo, mostrándolo junto con la imagen I_t al final del proceso de cada ciclo.

2.2. Detección por color

La detección y segmentación de objetos es la tarea fundamental más importante y desafiante de la visión artificial. Es una de las partes críticas en muchas aplicaciones de detección, reconocimiento y seguimiento. Y aún, sigue siendo un problema abierto debido a la variedad y complejidad de las clases de objetos y fondos en una imagen.

El método de detección basado en color es una de las maneras más sencillas que sirven para detectar y segmentar un objeto en una imagen. El objeto y el fondo deben tener una diferencia de color significante para lograr con éxito segmentar el objeto usando este método.

Usualmente cuando se capturan imágenes ó videos por medio de las librerias de OpenCV éstas se obtienen en un formato de 8 bits sin signo y en formato **RGB** (*ro-*

jo, verde, azul). En otras palabras, las imágenes capturadas pueden ser consideradas como 3 matrices, *rojo*, *verde* y *azul* de allí el nombre **RGB**, con un valor entero entre un rango de 0 a 255.

La Figura 2.2 muestra como una imágen a color es representada usando estas tres matrices. En esta figura, cada rectangulo pequeño representa un pixel de la imagen. Aunque, en imágenes reales, estos píxeles son tan pequeños que el ojo humano no puede diferenciarlos.

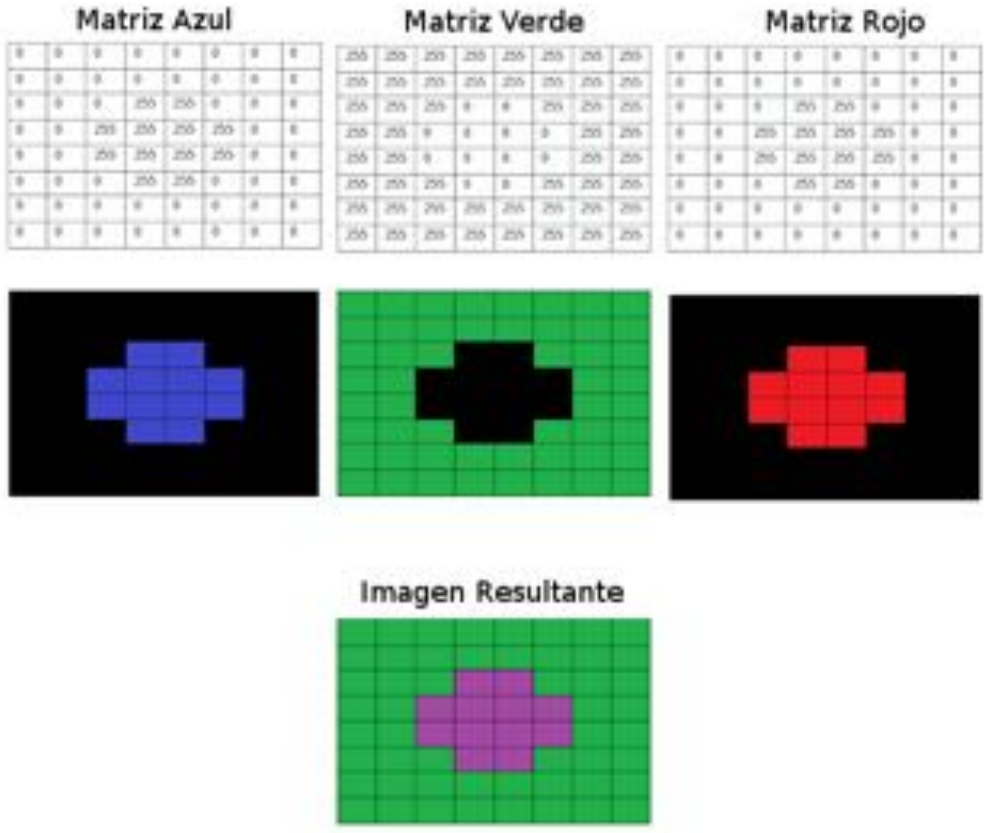


Figura 2.2: Cómo se compone una imagen en formato RGB.

Sin embargo, existe una representación alternativa al modelo de color **RGB** llamada **HSV**(*tono, saturación, valor*) que resulta ser más adecuada para la segmentación

basada en color.

Debido a esto, el algoritmo de detección por color utiliza una conversión del espacio de color de la imagen ó video original de **RGB** a **HSV**.
La representación de color en **HSV** se alínea estrechamente con la forma en que la visión humana percibe los atributos de creación de color. En este espacio al igual que en el espacio **RGB** se tienen tres matrices, una por cada elemento, es decir, una matriz de *tono*, otra de *saturación* y otra del *valor*, y en las librerias de OpenCV, los rangos de valores para el *tono*, la *saturación* y el *valor* van de $0 - 179$, $0 - 255$ y $0 - 255$ respectivamente.
El *tono* representa el color, la *saturación* representa la cantidad con la que el color se mezcla con el blanco y el *valor* representa la cantidad con la que el color se mezcla con el negro.

Una vez que se realiza la conversión de color entre **RGB** y **HSV**, se continúa con una segmentación de la imagen tranformandola a una representación binaria, utilizando como umbral el rango en la *tonalidad* en el que se encuentre el objeto en cuestión. Ya que el espacio de color **HSV** tiene la deseable propiedad que permite identificar un color en particular (usando un solo valor ó rango de valores) mediante el *tono* unicamente, en lugar de tres valores como sucede en **RGB**. [22].

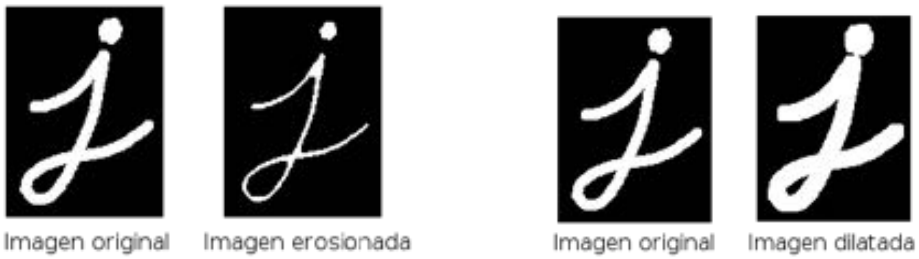
El siguiente proceso después de la segmentación, es el filtrado del ruido en la imagen para eliminar pequeños *lunares* aislados inecesarios que pueden haber aparecido porque tienen los mismos valores de *tonalidad* que el objetivo deseado.

Para la fase de filtración se utilizan transformaciones morfológicas en las imágenes. Las dos operaciones morfológicas básicas son la *erosión* y la *dilatación*.

Es necesario utilizar la imagen original y un *kernel* (ventana) el cual decide la naturaleza de la operación.

La erosión básicamente es un procedimiento similar a una convolución en 2D, es decir, se desliza un kernel de n x n dimensiones a través de la imagen segmentada (binarizada). Se toma un pixel en la imagen (1 ó 0), y se considera 1 sólo sí todos los pixeles del kernel son 1, de lo contrario se erosiona (se le asigna el valor 0). Ver Figura 2.3.

Por lo tanto el grosor o el tamaño del objeto en primer plano disminuye o simplemente la región blanca disminuye en la imágen. Es útil para eliminar pequeños ruidos *blancos*.



(a) Erosión (b) Dilatación

Figura 2.3: Operaciones morfológicas.

Por otra parte, la dilatación, es lo opuesto a la erosión. En este proceso, un píxel es 1, sí al menos un pixel del kernel es 1. De modo que este proceso incrementa la región blanca en la imagen o el tamaño del objeto. Normalmente, en casos donde está presente el ruido, la dilatación se implementa después de la erosión. De manera que, mientras la erosión elimina los ruidos blancos y al mismo tiempo *encoge* el objeto, la dilatación sirve para aumentar su tamaño, y dado que el ruido ha sido eliminado, al aumentar el área del objeto el ruido no vuele a aparecer [23].

Implementación

Las pruebas de este método se realizaron con la ayuda de un video en el cual se muestra el mismo vehículo a escala 1:24 de la implementación anterior con la inten-

Figura 2.4: Vehículo de pruebas.

ción de estudiar el comportamiento del algoritmo de forma sencilla. El pseudocódigo del algoritmo de color, puede verse en el Algoritmo 2. Al definir los rangos de color en **HSV** se utilizan dos rangos, el rango bajo y el rango alto, debido a que el valor del *tono* se distribuye de manera circular, el color del objeto se puede encontrar en dos partes distintas del círculo, tal como el *rojo*, color prominente sobre uno de los vehículos de pruebas que se utilizó (Ver Figura 2.4). Y para el cual se definen los rangos en HSV asumiendo los valores permitidos en las librerias de OpenCV. Continuando en el proceso, se captura cada imagen del video en formato **RGB** y se realiza la conversión al espacio de color **HSV** a fin de realizar la binarización de cada una, quedando el objeto en color blanco y el fondo en color negro. La obtención de contornos se realiza unicamente en los pixeles de color *rojo* utilizando la imagen previamente binarizada, haciendo la comparación de áreas delimitadas por los contornos hallados. Una vez que el área de mayor tamaño ha sido obtenida, se crea un rectángulo circundándola con información de las coordenadas (x, y) pertenientes al punto superior izquierdo del contorno y a las mediciones de ancho w y largo h del mismo, esto sobre cada imagen del video siempre y cuando el área exista. Mostrando al final, el vehículo detectado en cada imagen, delimitandolo con un rectángulo de color sobresaliente.

Algoritmo 2 Descripción de la implementación del Algoritmo de Detección por Color

Inicio
  //Variables
  $I_1, ...I_n, I_{1_{hsv}}, ...I_{n_{hsv}}, r_b, r_a : Mat$
  $rect : Rect$
  //Entrada
  $r_b \leftarrow DefinirRangoMinHSV()$
  $r_a \leftarrow DefinirRangoMaxHSV()$
  //Proceso
  Para $t \leftarrow 0, ..., n$ **hacer**
    Leer I_t
    $I_{t_{hsv}} \leftarrow ConvertirHSV(I_t)$
    $I_{t_{hsv}} \leftarrow ConvertirBN(I_{t_{hsv}})$
    $contornos \leftarrow HallarContornos(I_{t_{hsv}})$
    Crear $MaxArea \leftarrow 0$
    Para $cnt \leftarrow contornos[0], ..., contornos[n]$ **hacer**
    $area \leftarrow AreaContorno[cnt]$
    Sí $area \geq MaxArea$ **entonces**
      $bestCount \leftarrow cnt$
    FinCiclo
    $x, y, w, h \leftarrow delimitarRectangulo(bestCount)$
    $rect \leftarrow obtenerRectangulo(I_t, x, y, w, h)$
  //Salida
    Mostrar$(I_t, rect)$
  FinCiclo
Fin

2.3. Flujo Óptico mediante el método de Lucas-Kanade

Cuando se habla del seguimiento de puntos característicos muchos problemas se presentan, tales como el seguimiento de objetivos o la reconstrucción de escenarios por medio del movimiento que requieren del seguimiento de puntos de interés.

Existen diversos retos en el tema de seguimiento de puntos característicos, uno de los más importantes es conocer *cuales* puntos característicos se pueden seguir y lograr que el seguimiento sea eficiente en un video (a través de la secuencia de imágenes).

También es importante que el seguimiento se logre realizar a pesar de que algunos puntos característicos pudiesen cambiar su *apariencia* a través del tiempo (por cambios en su *rotación*, en la *iluminación*, etc.). Otro error que puede darse a menudo es la deriva, es decir, la acumulación de error durante la actualización del modelo. Y por último, uno de los retos más importantes es lograr agregar o eliminar puntos que han sido seguidos después de haber aparecido o desaparecido.

La metodología que se necesita para el seguimiento de objetivos utilizando el algoritmo de Lucas-Kanade se resume en dos pasos.

El primero de ellos se basa en utilizar un algoritmo de detección de puntos característicos adecuado para detectar características sobresalientes en una imagen.
El segundo paso consiste en el cálculo del flujo óptico utilizando el algoritmo de Lucas-Kanade. En esta sección hablaremos sobre el concepto del flujo óptico y su estimación por medio del método de Lucas-Kanade.

El flujo óptico es el patrón de movimiento aparente de un objeto entre dos *cuadros* (imagenes) consecutivas ocasionado por el movimiento del objeto ó la cámara. Es un campo vectorial en 2D (dos dimensiones) donde cada vector es un vector de desplazamiento que muestra el movimiento de los puntos desde el primer cuadro hasta el segundo.

Y funciona bajo dos suposiciones:

1 Las intensidades de los píxeles de un objeto no cambian entre cuadros consecutivos.

2 Los pixeles vecinos tienen un movimiento similar.

Considerando un pixel $I(x, y, t)$ en el primer cuadro, que se mueve una distancia (dx, dy) en el siguiente cuadro después de un tiempo dt. De tal forma que si esos

pixeles son los mismos y su intensidad no cambia, se puede decir,

$$I(x, y, t) = I(x + dx, y + dy, t + dt) \tag{2.1}$$

Mediante la aproximación de series de Taylor, se remueven algunos términos en común y dividiendo por dt se obtiene la siguiente ecuación:

$$f_x u + f_y v + f_t = 0$$

Donde

$$\begin{aligned} f_x = \frac{\partial f}{\partial x}; \quad f_y = \frac{\partial f}{\partial y} \\ u = \frac{dx}{dy}; \quad v = \frac{dy}{dt} \end{aligned} \tag{2.2}$$

La ecuación (2.2) es conocida como la ecuación de Flujo Óptico. Donde f_x y f_y son gradientes de la imágen. Similarmente f_t es el gradiente del tiempo. Pero (u, v) son variables desconocidas. Distintos métodos son propuestos para resolver estas dos incógnitas y uno de ellos es el método de Lucas Kanade.

El metodo de Lucas Kanade toma un vecindario de 3 x 3 pixeles alrededor del punto de interés, considerando que los píxeles vecinos tendrán un movimiento similar al punto de interés. Obteniendo de esta manera (f_x, f_y, f_t) para los nueve puntos.

La solución por este método es obtenida mediante el método de mínimos cuadrados. La ecuación (2.3) es la solución final, la cual esta compuesta de dos ecuaciones con dos incógnitas.

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \Sigma_i f_{xi}^2 & \Sigma_i f_{xi} f_{yi} \\ \Sigma_i f_{xi} f_{yi} & \Sigma_i f_{yi}^2 \end{bmatrix}^{-1} \begin{bmatrix} -\Sigma_i f_{xi} f_{ti} \\ -\Sigma_i f_{yi} f_{ti} \end{bmatrix} \tag{2.3}$$

De modo que, la idea es simple, se proporcionan algunos puntos para seguir, y se reciben los vectores de flujo óptico de esos puntos. Sin embargo, haciendo el proceso de esta forma, limita a trabajar con movimientos lentos. Por lo cual, si se utiliza el método de pirámide, los movimientos pequeños se eliminarán y los movimientos grandes se convertirán en movimientos pequeños. Entonces, al aplicar Lucas-Kanade

allí, se obtendrá flujo óptico junto con la escala [24].

Implementación

Se realizaron pruebas de este algoritmo mediante una toma de video directamente de una webcam. Los puntos característicos se eligieron mediante la detección de esquinas con el algoritmo de ShiTomasi. El pseudocódigo del algoritmo de flujo óptico se muestra en el Algoritmo 3 y fue realizado con las librerias de OpenCV y con librerias de Python. Al ingresar la primer imagen del video I_1, y tras hacer la conversión

Algoritmo 3 Descripción de la implementación del Algoritmo de Flujo Óptico

1: **Inicio**
2: //**Variables**
3: $I_1, ..., I_n, I_{1_{Gray}}, ..., I_{n_{Gray}}, I_{fo} : Mat$
4: $mask : Mat(zeros)$
5: $k_1, ..., k_n : vector < Keypoint >$
6: //**Entrada**
7: Leer I_1
8: //**Proceso**
9: $I_{1_{Gray}} \leftarrow ConvertirGrises(I_1)$
10: $k_1 \leftarrow Detectar(I_1)$
11: **Para** $t \leftarrow 2, ..., n$ **hacer**
12: Leer I_t
13: $I_{t_{Gray}} \leftarrow ConvertirGrises(I_t)$
14: $k_t \leftarrow CalcularFlujoOptico(I_{1_{Gray}}, I_{t_{Gray}}, k_1)$
15: $mask \leftarrow lines(k_1[t], k_t[t])$
16: $I_{fo} \leftarrow Suma(I_t, mask)$
17: Mostrar(I_{fo})
18: //**Salida**
19: $I_{1_{Gray}} \leftarrow I_{t_{Gray}}$
20: $k_1 \leftarrow k_t$
21: **FinCiclo**
22: **Fin**

del espacio de color **RGB** a escala de grises, se calculan los puntos característicos asumiendo que el vehículo a escala se encuentra contenido en la imagen. A continuación, se inicia la lectura de cada una de las imágenes del video, realizando el mismo proceso de conversión entre **RGB** y escala de grises al igual que en la primer imagen. Y mediante el cuadro *anterior* y los puntos de interés *anteriores*, se realiza

la estimación de los puntos de interés *actuales* en la imagen I_t *actual* por el método de Lucas-Kanade utilizando flujo óptico. Para lograr la visualización de la dinámica de los puntos característicos a lo largo del video, se crea una imagen con valor en **RGB** igual a cero, esto es, completamente oscura, llamada propiamente como una *máscara* porque unicamente se utiliza para la adición de las trayectorias de los puntos característicos a la imagen I_t. Donde se crean líneas de flujo por medio de dos puntos característicos pertenecientes al mismo índice, el *keypoint* anterior y el actual propiamente. Para la visualización se crea una imagen resultado de la adición anteriormente mencionada, entre la máscara y la imagen *actual*. Al final, se realiza una actualización del cuadro *actual* y los *keypoints actuales* convirtiendose en los *anteriores* en el próximo ciclo.

2.4. CMT: Consensus-based Matching and Tracking of Keypoints for Object Tracking

El algoritmo **CMT** fué desarrollado por Georg Nebehay y Roman Pflugfelder del Instituto Tecnológico de Austria. Ellos propusieron un método basado en la detección de puntos característicos para el seguimiento de objetos en un algoritmo que combina ambos procesos la *coincidencia* de puntos de interés mejor conocida como *matching* (y para efectos de un mejor y claro entendimiento se le llamará así a lo largo del presente) y el seguimiento de los mismos.

Mediante una secuencia de imágenes, I_1,...,I_n, y una *región* de interés b_1 en I_1 el objetivo en cada cuadro de la secuencia es reconstruir la posición del objeto de interés o indicar que el objeto es o no visible. Se estima la posición del objeto, junto con su centroide μ, su escala s y su rotación en grados en el plano α, donde s y α son estimados con respecto a la aparición inicial del objeto. Por simplicidad, la región donde se enmarca el objetivo en la inicialización del algoritmo está alineada con los ejes del plano.

En orden de localizar el objeto en cada cuadro, cada uno de los puntos característicos realiza un voto hacia el centro del objeto. Ya que los puntos característicos erróneos son difíciles de evitar, se emplea un esquema basado en un consenso para la detección fuera de objeto en el voteo. Para que computacionalmente este algoritmo sea factible, se propone no emplear un acumulador de espacio para los votos sino agrupar los votos directamente en el espacio de la imagen. El uso de la rápida detección de puntos característicos y de los descriptores binarios permite que la implementación pueda realizarce en tiempo real.

La primera parte del algoritmo se basa en la extracción de puntos de interés y el *matching* de los mismos. Dado un conjunto de *keypoints*

$$O = \{(r_i, f_i)\}_{i=1}^{N^O}, \tag{2.4}$$

donde cada *keypoint* se denota por una localización $r \in \Re^2$ en coordenadas en el plano y un descriptor f (los descriptores que utiliza el algoritmo son descriptores binarios, $f \in \{0,1\}^d$ con el objetivo de reducir el coste computacional). Se inicializa O por medio de los detectores y descriptores de *keypoints* en el cuadro de inicialización I_1 en la región de interés b_1 y para recuperar la posición del objeto en cada cuadro I_t con $t \geq 2$ se encuentra un conjunto de puntos de correspondencia

$$K_t = \{(a_i, m_i)\}_{i=1}^{N^{K_t}}, \tag{2.5}$$

donde a es la posición absoluta del *keypoint* en la imagen y m el índice del punto de correspondencia en O. Mediante ambos procesos el de *matching* y seguimiento de puntos característicos se obtiene al conjunto K_t.
Los detectores y descriptores de posibles candidatos

$$P = \{(a_i, f_i)\}_{i=1}^{N^P}, \tag{2.6}$$

en I_t son determinados por su posición absoluta a y sus descriptores f. Para cada uno de los candidatos, se calcula la distancia de Hamming de su descriptor con los descriptores de todos los *keypoints* encontrados en I_1 incluyendo los del fondo

$$d(f^1, f^2) = \sum_{i=1}^{d} XOR(f_i^1, f_i^2) \tag{2.7}$$

Se *emparejan* los puntos candidatos en P con los puntos en I_1 mediante la condición de que el punto vecino más cercano debe estar más cerca que el segundo punto vecino más cercano por un umbral ρ. El conjunto de puntos coincidentes M consiste después en el subconjunto de puntos en P que coinciden con O excluyendo a los candidatos que hicieron coincidencias con *keypoints* del fondo.

Para el proceso de seguimiento se calcula el desplazamiento de cada *keypoint* en K_{t-1} desde el cuadro I_{t-1} al cuadro I_t utilizando la estimación de flujo óptico por medio de la variante piramidal del método de Lucas-Kanade. Para un tiempo $t = 2$, K_1 se obtiene transformando O a coordenadas absolutas en la imagen.

El conjunto de *keypoints seguidos* T, se obtiene mediante la actualización de las localizaciones de los *keypoints* en K_{t-1} manteniendo su índice correspondiente. Los *keypoints* que llegan a fallar en el *seguimiento* o que terminan fuera de los límites de la imagen se descartan del conjunto T.

Una vez que el *matching* y el *seguimiento* se han calculado, se fusiona el conjunto de *keypoints* producidos por el algoritmo de seguimiento, T, y el conjunto de *keypoints* coincidentes de acuerdo a sus descriptores, M, en un conjunto K' de tamaño $N^{K'}$, descartando todos los *keypoints seguidos* que ya tienen un *keypoint coincidente* asociado con el mismo *keypoint* del modelo. Intuitivamente, los *keypoints coincidentes* son más robustos, debido a que no se basan en una estimación recursiva.

En orden de localizar el objeto de interés en cada cuadro, cada *keypoint* (a, m)

en K' realiza un voto $h(a, m) \rightarrow \Re^2$ hacia el centroide del objetivo, resultando en un conjunto de votos, como se muestra en la Figura 2.5

$$V = \{h(a_i, m_i)\}_{i=1}^{N^{K'}} \tag{2.8}$$

En su forma básica, se consideran unicamente cambios traslacionales del objeto,

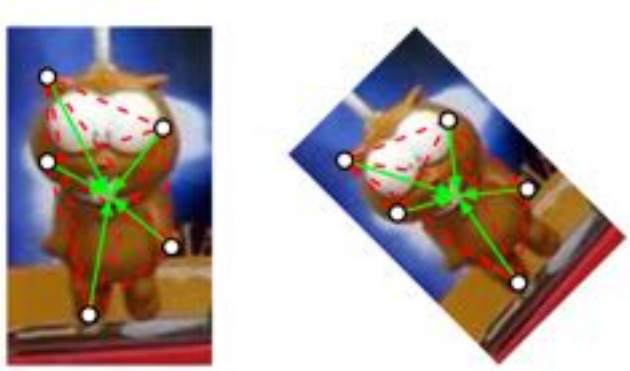


Figura 2.5: Los vectores de votación se producen sólo en el primer cuadro. Los votos se escalan y se orientan de acuerdo con la constelación de los *keypoints* actuales. [18].

$$h^T(a, m) = a - r_m, \tag{2.9}$$

donde r_m es la posición relativa del *keypoint* correspondiente en O. Para evitar que los votos sobrepasen o no alcanzen el centroide del objetivo debido a cambios en la escala del mismo, se escalan los votos mediante un factor de escala s, por lo que la ecuación (2.9) cambia a la forma

$$h^S(a, m) = a - s \cdot r_m, \tag{2.10}$$

Cuando el objeto está sujeto a una rotación en el plano, los votos deben orientarse de manera correspondiente para apuntar al centro del objeto. Por lo que la ecuación (2.10) cambia

$$h^R(a, m) = a - s \cdot Rr_m, \tag{2.11}$$

Donde R es una matriz de rotación en 2D,

$$R = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix} \tag{2.12}$$

Para estimar los cambios en escala s y en orientación α se analizan por pares los cambios entre los *keypoints* con respecto a su constelación inicial.

La última parte de este algoritmo se refiere a un proceso de *Consenso*. Ya sea que la ubicación a, o el índice correspondiente en el modelo m, de una entrada en K' sean incorrectos, los votos no apuntarán al centroide del objeto, sino que, apuntarán a un punto arbitrario en la imagen. Antes de localizar el centroide del objeto μ, se identifican y eliminan *keypoints* irregulares mediante un consenso en el comportamiento de la votación como se muestra en la Figura 2.6. Para calcular el centroide se realiza

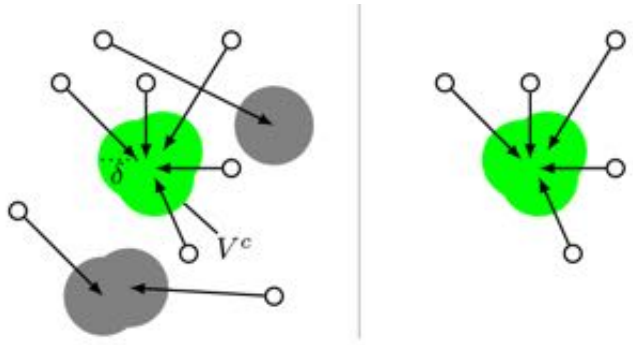


Figura 2.6: Consenso en el comportamiento de votación. [18].

un *agrupamiento jerárquico aglomerado* [25] en V basado en la distancia Euclideana como medida de disimilitud. En este tipo de agrupación, los datos son organizados en estructuras jerárquicas conforme a una matriz de proximidad, resultando en un dendograma[1] que se interrumpe en un cierto umbral δ. De manera que, el conjunto V es particionado en subconjuntos desarticulados $V^1, ..., V^m$. Considerando al conjunto que contenga el mayor número de elementos como el *grupo de consenso* V^c y

[1]Representación gráfica de datos en forma de árbol que muestra los grupos que se forman al crear conglomerados de observaciones en cada paso y sus niveles de similitud.

establece K_t en el subconjunto de K' que votó dentro de V^c.

Sí el conjunto V^c contiene menos de $\theta \cdot |O|$ elementos, se asume que el objeto no es visible. De lo contrario, se realiza una estimación del centro del objeto con los votos en el *grupo de consenso*

$$\mu = \frac{1}{n} \sum_{i=1}^{n} V_i^c, \tag{2.13}$$

Donde $n = |V^c|$. El centro del objeto junto con la escala a y la orientación α definen la posición del objeto de interés. Al final se calcula el rectángulo (que encierra al objetivo) no alineado a los ejes del plano, transformando las cuatro esquinas $c_1, ..., c_4$ por

$$c_i' = \mu + s \cdot Rc_i, \tag{2.14}$$

donde R es la matriz de rotación de la ecuación (2.12).
Una ventaja sobre otros métodos, es que este algoritmo no hace suposiciones sobre la planaridad de los objetos. En su lugar, se permite a los *keypoints* tener una ligera deriva de su posición original. El grado permitido de flexibilidad se basa en el parámetro δ [18].

Implementación

La implementación del algoritmo **CMT** se realizó mediante las librerias de OpenCV y con las librerias de C++. Las primeras pruebas se realizaron mediante el video de un vehículo a escala 1:24, cumpliendo con los requisítos del proyecto y facilitando el estudio de su comportamiento. El pseudocódigo de la implementación del algoritmo **CMT** se encuentra en el Algoritmo 4. La idea principal detrás de este algoritmo es que, el usuario tenga la habilidad de seleccionar el vehículo de su interés por medio de un objeto tipo rectángulo *rect* en la imagen de inicialización I_1 asumiendo que en ella está contenido el vehículo. Debido que la función de inicialización utiliza esta imagen a escala de grises, se realiza la conversion del formato **RGB** a escala

de grises. A continuación se realiza la captura del video dentro de un ciclo para n cuadros, siguiendo el mismo proceso de conversión de color a escalar de grises para la utilización de las imagenes en el procesamiento del algoritmo CMT, asumiendo que en el mismo se realizarán los procesos de *matching* y *seguimiento*. Por último, al final de cada ciclo se muestra I_t junto con los puntos caracteristicos correspondientes y el rectángulo que rodea al vehículo. **CMT**.

Algoritmo 4 Descripción de la implementación del Algoritmo CMT

Inicio
  //Variables
  $I_1, ..., I_n, I_{1_{Gray}}, ..., I_{n_{Gray}} : Mat$
  $cmt : CMT$
  $rect : Rect$
  //Entrada
  Leer I_1
  //Proceso
  $rect : obtenerRectangulo(I_1)$
  $I_{1_{Gray}} \leftarrow ConvertirGrises(I_1)$
  $cmt \leftarrow inicializarCMT(rect, I_{1_{Gray}})$
  Para $t \leftarrow 2, ..., n$ **hacer**
  Leer I_t
    $I_{t_{Gray}} \leftarrow ConvertirGrises(I_t)$
    $cmt \leftarrow procesarCMT(I_{t_{Gray}})$
  //Salida
    Mostrar(I_t, cmt)
  FinCiclo
Fin

A lo largo de este proyecto se realizaron pruebas con los algoritmos que se han mencionado anteriormente simulando la detección y seguimiento por medio de un carro de juguete. Sin embargo, es bien sabido que cada aplicación en particular tiene ciertos requisitos que pueden diferir entre una aplicación y otra. La aplicación de interés en este trabajo es lograr la detección y seguimiento de un vehículo particular a escala $1:1$. El hecho de discriminar otros vehículos, limita el método adecuado para lograrlo. A continuación se muestra en la Tabla 2.1 una comparativa de los algoritmos que se probaron.

Tabla 2.1: Comparación entre los algoritmos de visión de detección y seguimiento implementados.

	Limitaciones	**Ventajas**
ORB	Es un algoritmo limitado ante cambios de escala y rotación en comparación a otros algoritmos de detección de puntos característicos.	ORB puede ser una alternativa eficiente ya que trabaja con descriptores binarios lo que reduce significativamente el coste computacional.
FLUJO ÓPTICO	El algoritmo generalmente no actúa de forma correcta ante cambios abruptos de posición. Particularmente, no trabaja de manera correcta ante la oclusión en el movimiento de objetos en un video. Es susceptible a cambios de iluminación.	Es un método bastante preciso cuando los cambios en movimiento son lentos y en pequeñas distancias.
COLOR	Una de las principales limitaciones al utilizar éste método es que debido a las distintas condiciones de iluminación y saturaciones de color, los colores difícilmente lucen igual a los colores puros (rojo, azul, verde, etc.). Otra limitación es que al ser un detector de color, esta expuesto a detectar carros u objetos distintos de color rojo, es decir, puede detectar objetos distintos al de nuestro interés.	Su principal ventaja es que la implementación es simple resultando en un menor costo computacional y sencillez.
CMT	Debido a su estructura puede llegar a ser de alto coste computacional	Es un algoritmo de gran precisión ya que combina ambos procesos *detección* y *seguimiento*. Es invariante ante cambios de escala y orientación.

Capítulo 3

Arquitectura del sistema Cámara-Gimbal

A lo largo de este capítulo se presenta la interacción del sistema cámara-gimbal con un sistema de comunicación remota.
A continuación se describe la operación del sistema cámara-gimbal. Los componentes que conforman al sistema se describen posteriomente, así como el desarrollo de los algoritmos utilizados para la interacción remota.

En un escenario operativo típico, cuando el sistema cámara-gimbal se encuentra en operación a bordo de un VANT, el operador del sistema puede seleccionar un objetivo (*vehículo*) de interés en una pantalla desde una estación de control en tierra. Una vez que el operador identifica al objetivo, el algoritmo de procesamiento de imágenes para seguimiento de objetivos, calcula un rectángulo que enmarca al objetivo y proporciona la posicion del centroide del rectángulo en cada imagen contínua del video, tal video proviene de la cámara. Con el centroide del objetivo detectado, se calcula el cambio en pixeles necesario para posicionar al objetivo en el centro de la imagen. La diferencia de píxeles calculada se proporciona como entrada para un algoritmo de control integrado en el sistema cámara-gimbal. El algoritmo de control proporciona una señal **PWM** proporcional al error en píxeles en la imagen, hacia los servos en dos de sus tres ejes, *pan* y *tilt*, del gimbal, para mantener el objetivo en el centro en cada imagen del video. El diagrama para el control del sistema cámara-gimbal se muestra en la Figura 3.1.

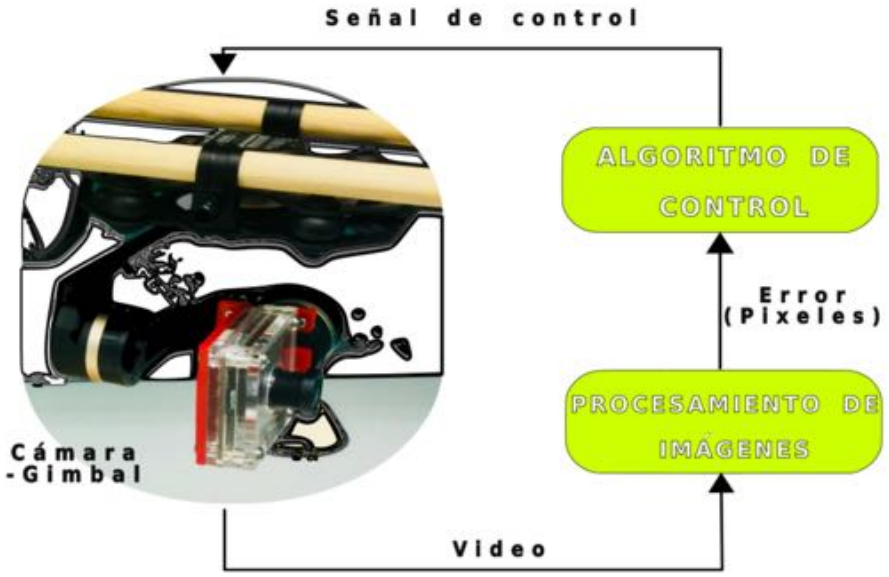


Figura 3.1: Diagrama de controlamiento del sistema Cámara-Gimbal.

3.1. Integración de Hardware

En un intento por mantener al objetivo siempre en el centro del marco de referencia, se utilizó una cámara montada sobre una plataforma de tipo gimbal de 3 ejes. Para realizar pruebas del prototipo en un inicio, no fue necesario volar un **VANT**. Así que con el hardware y software utilizado para la misión en vuelo, se realizó una configuración de laboratorio. A continuación se describe el hardware requerido para el banco de pruebas.

Cámara: Cámara oCam 5MP USB 3.0. Lente estándar M12 con una distancia focal de 3.6 mm. Cuenta con un rango de visión igual a 65°. Su sensor es de tipo **CMOS** (Complementary Metal-Oxide-Semiconductor) OmniVision OV5640. Control de brillo, contraste, tono, saturación y balance de blancos. Puede producir imágenes en formato YUV en una resolución máxima 1920x1080@15fps y en MPEG a 1920x1080@30fps máximo. Tiene un peso aproximado de 37 gramos y sus dimesiones son de 42 x 42 x 5 mm.

Gimbal: Gimbal TAROT ZYX FLIR de 3 ejes, lo que significa que puede hacer rotaciones en *Pan* (guiñada), *Tilt* (Cabeceo) y *Roll* (Alabeo). Puede ser montado en una basta variedad de **VANT's**. Cuenta con un módulo de control IMU (*Unidad de Medición Inercial*) y un módulo de servoaccionamiento especializado.

Microcontrolador: Tarjeta Arduino Mega. Posee un microcontrolador ATmega2560. Puede ser alimentado mediante puerto USB con un voltaje operativo de 5V. Cuenta con 54 pines digitales de entrada/salida de los cuales 15 proveen salida PWM, estas señales son utilizadas para operar los servos del gimbal.

Computadora: Computadora Intel NUC5i7RYH de 5ta generación. Con un procesador Intel® Core™ i7-5557U que permite realizar el procesamiento de imagenes del video.

3.2. Implementación de Software

Librerias como las de **OpenCV** y **ROS**(Robot Operating System) fueron utilizadas para la configuración del sistema. Las librerias de **ROS** se implementaron para el desarrollo de una estación de control en tierra. **OpenCV** como se mencionó anteriormente, fue usada para el procesamiento de imágenes. En cada imagen procesada por **OpenCV**, se calcula la posición del vehículo. Las librerias de **OpenCV**, asi como los repositorios de **ROS**, y la descripción de la implementación de la estación en tierra se presentan a continuación.

Repositorios

ROS *Kinetic Kame* es un sistema meta-operativo y además de código abierto que provee librerías y herramientas para ayudar a desarrollar software con aplicaciones en robótica. Proporciona los servicios de abstracción de hardware, controladores

de dispositivos de bajo nivel, librerías, herramientas de visualización, comunicación por mensajes y la administración de paquetes. También proporciona herramientas y bibliotecas para obtener, crear, escribir y ejecutar código en varios equipos.
El marco de **ROS** es fácil de implementar en cualquier lengujae de programación moderno, tales como Python, C++ y Lisp.
ROS se ejecuta en plataformas basadas en Unix. El software para **ROS** se prueba principalmente en sistemas Ubuntu, aunque actualmente ha estado contribuyendo con otras plataformas Linux [26].

OpenCV *3.1.0* es un conjunto de librerias de software de visión artificial y de aprendizaje automático de código abierto. Proporciona una infraestructura común para aplicaciones de visión artificial.

Al ser un producto con licencia BSD, **OpenCV** facilita que terceros utilicen y modifiquen el código. La biblioteca cuenta con más de 2500 algoritmos optimizados que se pueden utilizar para detección y reconocimiento de rostros, identificación de objetos, clasificación de acciones humanas en videos, rastreo, extracción de modelos en 3D, etc. **OpenCV** se inclina principalmente hacia aplicaciones de visión en tiempo real.
OpenCV tiene interfaces con C++, Python, Java y **MATLAB**(MATrix LABoratory) y se puede ejecutar en plataformas de Windows, Linux, Android y Mac OS [1].

Estación de control en tierra

Lo que se busca realizar es la operación remota del sistema cámara-gimbal a través de una arquitectura de comunicación implementada en el middleware **ROS** basado en mensajes, la incorporación de nuevos participantes en una red local, **WLAN**(Wireless Local Area Network) y la transmisión en tiempo real. Para lograr seleccionar el ob-

jetivo que se pretende seguir, es necesario tener una estación en tierra que permita seleccionar el vehículo y enviar los datos requeridos de nuevo al sistema embebido para su procesamiento .

En el proyecto se utilizó la comunicación de petición y respuesta que porporcionan los servicios en **ROS**. Un diagrama de la comunicación entre los nodos que se utilizaron se muestra en la Figura 3.3.
Para realizar la comunicación remota, el nodo *servidor* se mantiene a la espera de las solicitudes hasta que el nodo *cliente* realiza una solicitud. Es entonces cuando el *servidor* realiza un procesamiento y puede responder al cliente.

Desde la estación en tierra, vista como el nodo *cliente* se manda a llamar al servicio con dos mensajes diferentes: imagen donde se selecciona al vehículo y el cuadro que lo encierra. La libreria de ROS en la que se incluyen los tipos de mensaje existente *sensor_msgs* no contiene ningún tipo de mensaje en el que se engloben dos diferentes, pues solo se permite el envío de un tipo de mensaje a la vez. Por lo cual se requirió la creación de un tipo de mensaje de servicio modificado que permitiera dicha tarea. Este tipo de mensaje se caracteriza por tener dos partes: una petición y

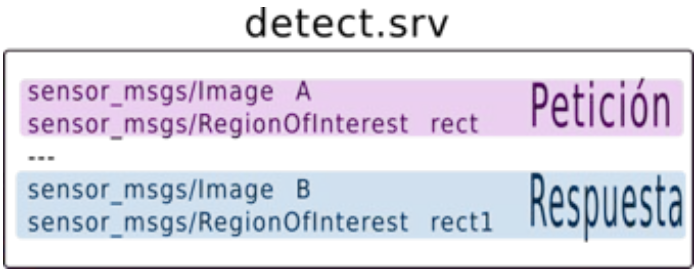


Figura 3.2: Archivo srv: tipo de mensaje utilizado por el servidor/cliente.

una respuesta (ver Figura 3.2). Donde la petición al igual que la respuesta contienen dos variables correspondientes a la imagen y recuadro donde se selecciona al vehículo
.

ROS_Master

Cuando uno de los dos nodos arranca en el entorno de **ROS**, este comienza a buscar al ROS Master y registra el nombre del nodo en él. De esta forma, el master tiene los detalles del nodo que está corriendo en el sistema actualmente, una vez obtenidos los detalles de cada nodo, el master interconecta a los nodos mediante el protocolo TCPROS basado en la arquitectura TCP/IP.

Para realizar la comunicación de forma remota, fue necesario utilizar el entorno **ROS_MASTER_URI**. Esta variable de entorno contiene la IP y el puerto del ROS Master, de modo que, al utilizar este entorno, los nodos 1 y 2 pudieron localizar al máster independientemente, es decir, ejecutandose en otra máquina. De tal forma, que mediante una red distribuida, se ejecutaron ambos nodos en diferentes máquinas, el nodo donde se encuentra el servidor se realizó en la computadora embebida, mientras que el nodo donde se encuentra el cliente se corrió en la computadora de la estación en tierra. Es importante mencionar que la defición de la variable **ROS_MASTER_URI** debe ser correcta, ya que solo de esta manera, los nodos remotos son capaces de encontrar el master y, así, poder comunicarse.

Implementación

La implementación del sistema, está dividido en dos subsistemas, nodos, donde cada uno de ellos se ejecuta en una máquina distinta como se mencionó anteriomente. Un esquema que facilitará el entendimiento se muestra en la Figura 3.4. El pseudocódigo de la implementación del *nodo 2* se encuentra en el Algoritmo 5. El *nodo 2* se encarga de dos tareas principales *publicar* las imágenes consecutivamente del video que se toma a través de la cámara montada en el gimbal y lo segundo es utilizar los datos provenientes del nodo cliente para el procesamiento de imagenes.
En este programa se utilizan los repositorios de OpenCV, C++ y ROS. Primero, se inicializa ROS. Esto permite la utilización de comandos bajo sus librerias, además de especificar el nombre del nodo 2. Lo siguiente a realizar es decirle al *master* que

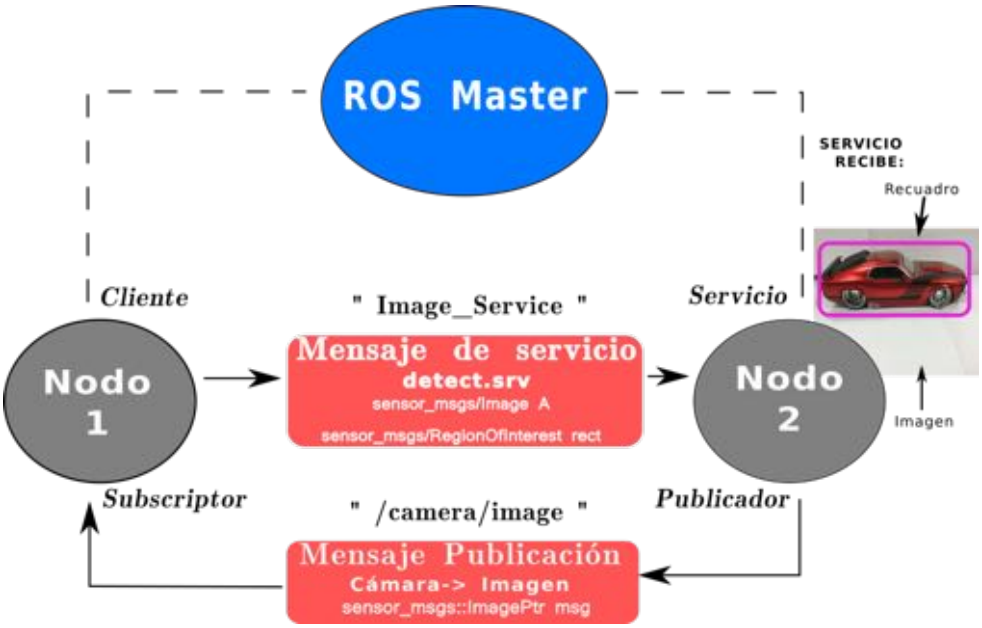


Figura 3.3: Diagrama de comunicación de los Nodos.

se van a publicar imágenes en el tópico *"/camera/image"*. A continuación se crea el servicio, caracterizado como *"imageService"* mandando a llamar a la función *add* por medio de la cual se obtienen los datos de la imagen y el rectángulo a través de los tipos *Request* y *Response* definidos en el archivo srv. Dentro del ciclo de trabajo se capturan constantemente las imágenes del video de la cámara conviertiendo cada una a un mensaje compatible que se pueda publicar con ROS posteriormente. Para saber si se han recibido los datos del cliente, se realiza una condición donde al resultar cierto, se prosigue a realizar el procesamiento de imágenes con el algoritmo CMT. Por último con los datos del rectángulo, se calcula el centroide del mismo en cada imagen del video y se envían las coordenadas de éste por comunicación serial hacia el microcontrolador ATMega2560.

En la implementación del *Nodo 1* se utilizaron librerias de OpenCV y de ROS mediante el lenguaje de programación C++. Al igual que en el nodo de servicio, se requiere iniciar el entorno de ROS bajo el nombre *"Nodo_Cliente"*. Este nodo tiene por finalidad, recibir el video proveniente del sistema embebido y transmitirlo en

Figura 3.4: Diagrama de comunicación del sistema completo.

la estación en tierra, de allí la creación de un *subscriptor*. De esta forma es como se le indica a ROS que se quieren recibir mensajes -imágenes, en el tópico llamado *"/camera/image"*. Existen dos motivos por los que se reciben las imágenes a través del subscriptor, el primero es para hacer la selección del vehículo en las imágenes. De esta forma, se puede hacer uso del video que se transmite en la cámara en el sistema embebido. Para ello, se crea un *cliente* para el servicio *"imageService"*. Durante el ciclo de trabajo, se comprueba si se reciben las imágenes del video a través del subscriptor, de resultar cierto, se prosigue a seleccionar el objetivo en una imagen mediante un rectángulo, acto seguido se envían los datos que solicita el servicio, datos del tipo que se indica en el archivo *detect.srv*, es decir, la imagen y las características del rectangulo, coordenadas de la esquina superior izquierda y dimensiones del mismo.

El segundo motivo es el recibir constantemente a una velocidad de $20Hz$ imagenes directamente de la cámara montada en el sistema embebido en la estación de control en tierra, para captar en cada momento al vehículo de interés y saber su ubicación con respecto a la cámara.

Algoritmo 5 NODO 2 - Servicio

Inicio
//Variables
$a, a_{Gray} : Mat$
Función add(req : Request, res : Response) : bool {
$res.ImagenB \leftarrow req.ImagenA$
$rect \leftarrow Rect(req.x, req.y, req.width, req.height)$
$a \leftarrow res.ImagenB$ }
//Entrada
$inicializarROS()$
$pub : Publisher \rightarrow advertise("/camera/image")$
$service : ServiceServer \rightarrow advertise("imageService", add)$
//Proceso
Mientras *true* hacer
Leer I_n
Publicar($I_n \rightarrow msg$)
Sí $!a.data$ **entonces**
"Solicitar al Cliente"
Sino
$a_{Gris} \leftarrow ConvertirGrises(a)$
$InicializarCMT(a_{Gris})$
Mientras *true* hacer
Leer I_n
Publicar($I_n \rightarrow msg$)
$I_{Gris} \leftarrow ConvertirGrises(I_n)$
$ProcesarCMT(I_{Gris})$
$EnvíoCoordenadasSerial(x_{rect}, y_{rect})$
FinBucle
FinCondición
FinBucle
Fin

Comunicación Serial

Hasta ahora, en el presente trabajo se ha mostrado la obtención y la manipulación de los diferentes algoritmos, de manera que el objetivo principal de este proyecto se cumpla. Sin embargo, el tema de la comunicación serial entre el sistema de visión artificial implementado en la computadora embebida con el sistema de control implementado en el microcontrolador, juega un papel importante. Por lo que a continuación se habla sobre la interacción entre ambos sistemas.

Algoritmo 6 NODO 1 - Cliente

Inicio
//Variables
Función imageCallback(msg : Image) : void {
$im0 \leftarrow cvBridge :: ToCvCopy(msg) \rightarrow image$ } El motivo por el
//Entrada
$inicializarROS()$
$sub : Subscriber \rightarrow subscribe("/camera/image", imageCallback)$
$cliente : ServiceClient \rightarrow serviceClient("imageService")$
$srv : carpetaSRV :: detect$
//Proceso
Mientras *true* hacer
Sí $!im0.data$ **entonces**
"No se tiene imagen del subscriptor"
Sino
$msg \leftarrow im0.toImageMsg$
$srv.req.ImagenA \leftarrow *msg$
$rect \leftarrow seleccionarRectangulo(im0)$
$srv.req.x \leftarrow rect.x$
$srv.req.y \leftarrow rect.y$
$srv.req.w \leftarrow rect.w$
$srv.req.h \leftarrow rect.h$
FinCondición
FinBucle
Fin

Parámetros de envío

Debido a que uno de los objetivos del proyecto es lograr el seguimiento del vehículo por medio de una plataforma que direccione a la cámara en todo momento, es decir utilizando un gimbal, fue necesario tener un control sobre éste. Dicho esto, lo que se busca es realizar un control que utilice la desviación de la posición del vehículo en la imagen con respecto al centro de la misma. Y de acuerdo al valor de la desviación, realizar el cálculo para el control de los servos en el gimbal. Ya que se tiene un gimbal con 2 ejes que se pueden manipular, el motor en *pan* y el motor en *tilt* corresponderán a las coordenadas en x y y del vehículo en la imagen respectivamente.

Teniendo las coordenadas (x_c, y_c) como el centro de la imagen, calculadas a partir de la altura y el ancho de la imagen. Y donde (x, y) corresponden a las coordenadas

de la posición del vehículo en la imagen, calculadas a partir de las propiedades del rectángulo que encierra al vehículo, se puede calcular el error en posición, a partir del cual se realiza un control para el gimbal. Como se ve en la Figura 3.5, d_x, d_y son los errores respectivamente en las coordenadas x y y respectivamente.

De manera general, las variables que son requeridas para poder controlar la rotación de los servos del gimbal, son (x, y) referentes a la posición del vehículo en el marco referencial de la imagen.

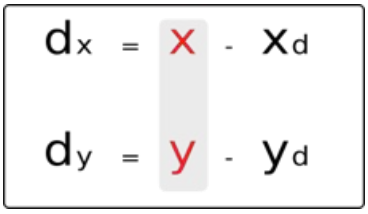


Figura 3.5: Parámetros de envío.

Protocolo de comunicación serial

Una vez que se han identificado los datos que se requieren en el microcontrolador ATMega2560 para calcular el control del gimbal, es necesario mostrar el protocolo empleado para el envío de las coordenadas.

Para el envío de datos, se utilizó la libreria *Serial.h* anexando sus archivos y directorios en el archivo de compilación y creación del ejecutable *CMakeLists.txt* ya que esta librería no se encuentra en los repositorios de OpenCV ni existe una libreria similar como es el caso de la libreria *PySerial* en el lenguaje de programación Python.

El envío de coordenadas utiliza una secuencia, la cual se muestra en la Tabla 3.1

Tabla 3.1: Envío de datos al sistema de control

Parámetro	**Sistema decimal**	**Sistema hexadecimal**
Header	1000	03e8
Coordenada X	x	byte1-byte2
Coordenada Y	y	byte1-byte2

Para el envío se incluyó un valor inicial nombrado *Header*, este dato tiene como finalidad asegurar que lo que se envía después, son las coordenadas (x, y), ya que se supone es un valor que ninguna de las dos coordenadas puede adquirir en ningún momento si el programa opera de manera correcta. En la Tabla 3.1 se muestra su valor en conversión hexadecimal, con la finalidad de mostrar que la trama se envía en dos bytes, es decir, como por ejemplo el *Header* se enviará primero un byte asignado a *e8*, seguido del siguiente byte con asignación *03*, y así sucesivamente.

Para hacer la lectura de las coordenadas en el sistema de control desarrollado en la IDE de Arduino, a una velocidad de transmisión de 9600 Bauds por segundo, se implementó una máquina de estados de tipo Mealy, este tipo de máquina es una máquina de estados finitos que genera una salida basándose en su estado actual y una entrada.
A continuación se presenta el diagrama de la recepción de los bytes correspondientes a la trama de envío.

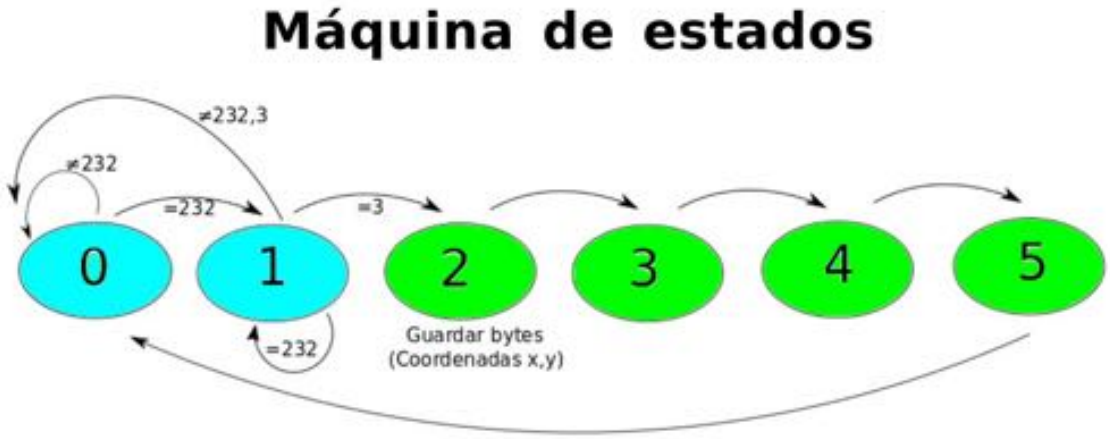


Figura 3.6: Recepción de las coordenadas x,y en el sistema de control.

Para la recepción de las coordenadas se envía un orden de bytes. Como se mencionó anteriormente, cada parámetro se envía en dos bytes, dando un total de 6 bytes por trama. Es importante mencionar que los bytes se reciben convirtiendolos de su valor en hexadecimal a decimal, es por eso que en la Figura 3.6 vemos un valor inicial igual a 232 que corresponde a la conversión $e8 \rightarrow 232$ del primer byte del *Header*,

realizando el mismo procedimiento para los demás bytes.
La forma en la que se envían los datos y reciben se muestra en el anexo del código fuente de arduino.

3.3. Sistema de Control

Durante la transmisión del video emitido por la cámara montada en el gimbal, cada vez que el gimbal gira en Pan (guiñada) / Tilt (cabeceo) el vehículo cambia su posición original (x_0, y_0) en el marco referencial de la imagen a una nueva posición (x_1, y_1). En esta sección se presenta un esquema de control visual basado en las imágenes del video para correlacionar el movimiento del vehículo en el plano de la imagen del sensor de la cámara con el movimiento del gimbal en Pan y Tilt.

Características de la imagen en movimiento

Con la finalidad de realizar una estimación en el cambio de orientación de la cámara basada en la posición actual del vehículo en el plano de la imagen del sensor de la misma como se muestra en la figura 3.7, se introduce un modelo que relaciona el movimiento del vehículo y el movimiento de la cámara. La figura 3.7 muestra la relación entre un punto en el espacio en 3 dimensiones siendo éste el vehículo y su punto característico en el plano de la imágen del sensor de la cámara. El punto característico p_1 es definido como $p1 = [p_{x_1}, p_{y_1}]^T$ como una proyección del vehículo en el marco de la imagen, donde el punto $C_p = [x_c, y_c]$ es el punto central en la imagen y el origen del marco de referencia de la imagen se indica mediante el punto O_p.

Control PID y Esquema de Control

En esta sección se presenta una estrategia de control que permite que el carro de prueba sea seguido continuamente. El sistema de la cámara esta montada sobre el gimbal de 3 ejes desacoplado del **VANT**, de manera que los rangos de movimiento

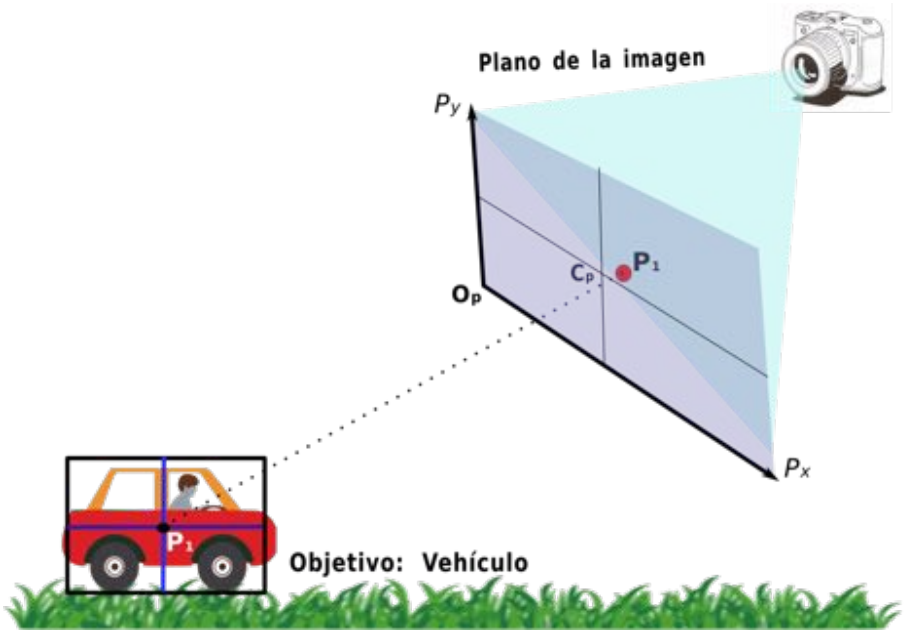


Figura 3.7: Vehículo en el espacio 3D y su proyección en el plano de la imagen de la cámara.

en las rotaciones en Pan (90°) y Tilt (30°) se mantienen constantes durante el movimiento de traslación.

En el presente trabajo se implementó un control de tipo Proporcional-Derivativo-Integral (PID) en dos ejes del gimbal, Pan y Tilt, correspondientes a las coordenadas en los ejes P_x y P_y en la imagen respectivamente.

En cada cuadro del video, el algoritmo de seguimiento de CMT intenta predecir la posición del centro del recuadro que contiene al objetivo. El error en pixeles en los ejes x e y entre el centro del recuadro que encierra al objetivo actual y el centro del cuadro de video se usa como entrada para el controlador PID. La salida se traduce en dos señales PWM dirigidas hacia el movimiento en Pan y Tilt en el gimbal. La corrección se produce a la velocidad de la cámara (30 fps), ajustando incrementalmente la posición del gimbal hasta que el objetivo se centra en el campo de visión de la cámara, es decir, el error en los ejes x e y se acerca a cero.
El uso de un control PID para estabilizar al gimbal permite un cálculo relativamente

rápido, extensibilidad hacia otras plataformas del mismo tipo y robustez sin necesidad de utilizar el modelo mátematico que describe al sistema.

Teniendo a $C_p = [x_c, y_c]$ como las coordenadas del centro de la imagen, en el sensor de la cámara, y $P = [x, y]$ como las coordenadas de la posición del carro en la imagen. Se definen los errores en posición en pixeles en el marco referencial de la imagen en la cámara (ver Figura 3.8)

$$\begin{aligned} e_x &= x_c - x \\ e_y &= y_c - y \end{aligned} \tag{3.1}$$

Donde e_x representa el error en posición del centro de la imagen en el eje x con respecto al centro del recuadro que encierra al vehículo en el eje x, en el marco referencial de la imagen de la cámara. Y e_y representa el error en el eje y, del centro de la imagen con respecto al centro del rectangulo que encierra al vehículo en el marco referencial de la imagen de la cámara.

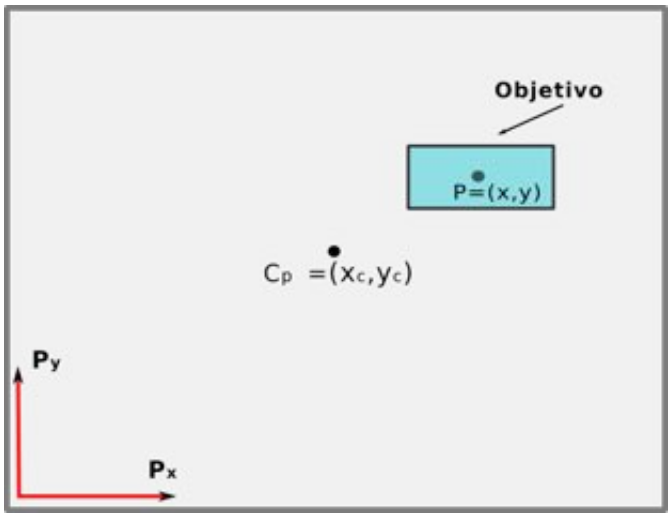


Figura 3.8: Coordenadas del objetivo en el marco referencial de la imagen de la cámara.

De la documentación existente sobre sistemas de control, se destaca la siguiente

ecuación que define el comportamiento del control PID en tiempo contínuo:

$$u(t) = K_p e(t) + \frac{K_p}{T_i} \int_0^t e(t)dt + K_p T_d \frac{de(t)}{dt} \tag{3.2}$$

De la ecuación 3.2 , podemos hacer las siguientes afirmaciones:

- $e(t)$ es el error de la señal de posición del carro en el marco referencial de la imágen
- $u(t)$ salida del controlador, señal pwm hacia el gimbal
- K_p ganancia proporcional
- T_i constante de tiempo integral
- T_d constante de tiempo derivativa

Para tener una idea más clara, el siguiente diagrama en la figura 3.9, describe el funcionamiento del control PID que se implementó en el proyecto. Podemos ver que la atención se focaliza a la salida del algoritmo CMT, ya que como se menciona en la ecuación 3.1 el error en posición se determina por el centro del recuadro (x, y) a la salida del algoritmo de visión y de acuerdo al mismo, el controlador proporcionará el PWM necesario para cada motor.

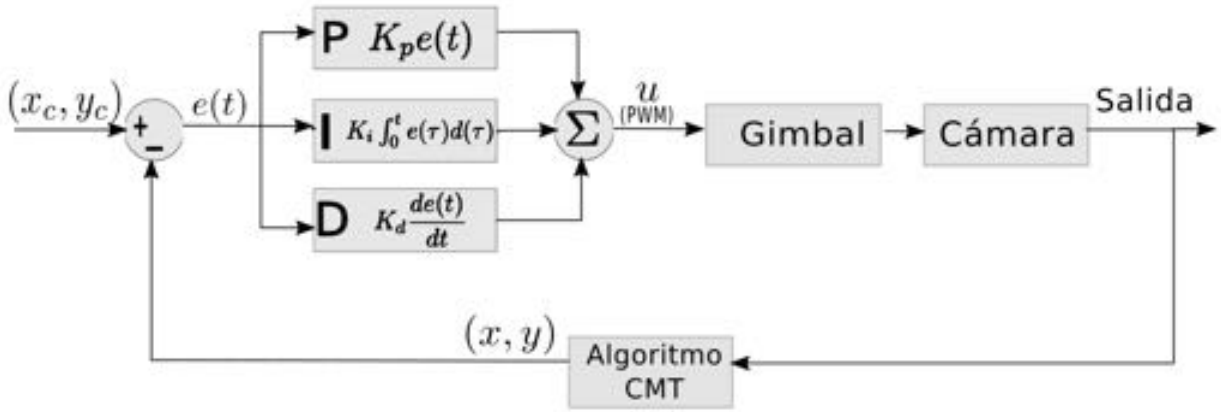


Figura 3.9: Diagrama del control PID.

A manera de implementar el control PID se realizó la estimación de la razón de cambio de la posición del carro con respecto al tiempo en el marco referencial de la

imágen, para esto, se ocuparon dos métodos distintos y se compararon sus resultados.

La derivación numérica es una técnica de análisis numérico para calcular una aproximación a la derivada de una función en un punto utilizando los valores y propiedades de la misma. [28]

Por definición la derivada de una función *f(x)* es:

$$\dot{f}(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \tag{3.3}$$

Utilizando ésta técnica se realizó la derivación numérica de la posición en x e y en el sistema de control con un tiempo de muestreo $\delta t = 100\ ms$.

$$\dot{f}(x_0) \approx \frac{f(x_0 + \delta t) - f(x_0)}{\delta t} \tag{3.4}$$

El segundo resultado se obtuvo mediante la implementación del *Diferenciador de Levant* para la estimación de la velocidad de las lecturas de posición. El diferenciador de Levant es un control por modos deslizantes basado en un algorítmo de diferenciación en tiempo real propuesto como un estimador de velocidad. [29]

Levant propone una técnica de diferenciación robusta basada en un algoritmo de modos deslizantes para señales con un límite superior dado en la constante de Lipschitz de su derivada. Es decir, dada una señal de entrada $f_0(t)$, la constante de Lipschitz de su derivada es una constante C la cual satisface

$$|\dot{f}_0(t_1) - \dot{f}_0(t_2)| \leq C|t_1 - t_2|$$

Sea entonces la señal $f_0(t)$ definida en el intervalo $[0, \infty)$ se considera una ecuación auxiliar

$$\dot{z_0} = v \tag{3.5}$$

Donde se define la variable de deslizamiento como $\sigma = z_0 - f_0(t)$ procurando mantener $\sigma = 0$. En éste caso sucederá que $\sigma_0 = \dot{\sigma_0} = 0$ cuando se alcance la superficie de deslizamiento, lo que significa que $z_0 = f_0(t)$ y $\dot{f}_0(t) = \dot{z}_0 = v$. Por lo que el sistema

puede ser reescrito como

$$\dot{\sigma} = -\dot{f}_0(t) + v,\ |\ddot{f}_0| \leq L \tag{3.6}$$

La función $\dot{f}_0$ puede no ser suave, pero debe de ser Lipschitz y su derivada debe existir.

Por lo que, siendo éste el caso, se puede utilizar una modificación del controlador Super-Twisting

$$\dot{z}_0 = v = -\lambda_1 |\sigma_0|^{\frac{1}{2}} sign(\sigma_0) + z$$

$$\dot{z} = -\lambda_2 sign(\sigma_0)$$

Quedando

$$\begin{aligned} \dot{z}_0 = v = -\lambda_1 |z_0 - f_0(t)|^{\frac{1}{2}} sign(z_0 - f_0(t)) + z \\ \dot{z} = -\lambda_2 sign(z_0 - f_0(t)), \end{aligned} \tag{3.7}$$

de donde tanto v como z pueden ser usadas como salidas del diferenciador. [30] Las ganancias λ_1 y λ_2 son estrictamente constantes positivas las cuales determinan la precisión de la diferenciación, y deben ser escogidas propiamente para asegurar la convergencia. Levant propone una condición *suficiente* para la convergencia de v a $\dot{f}_0(t)$ dada por

$$\lambda_2 > C, \lambda_1^2 \geq 4C \frac{\lambda_2 + C}{\lambda_2 - C} \tag{3.8}$$

Una manera sencilla de escoger los parametros $lambda_1$ y $lambda_2$ es

$$\lambda_2 = 1.1C, \lambda_1 = C^{\frac{1}{2}} \tag{3.9}$$

Aunque el resultado de estas condiciones, proporciona una estimación cruda. [29]

En las pruebas que se realizaron dentro del laboratorio, se probó el diferenciador

de Levant de tercer orden, y posteriormente se utilizó un filtro promediador de orden 7 y 10.

En la figura 3.10 se presenta una prueba realizada en un vehículo a escala en movimiento dentro del laboratorio, las señales que se muestran corresponden a la posición en pixeles en el eje x y la velocidad estimada en pixeles en el mismo eje en el marco referencial de la imagen, realizada por la derivación numérica y por el diferenciador de Levant.

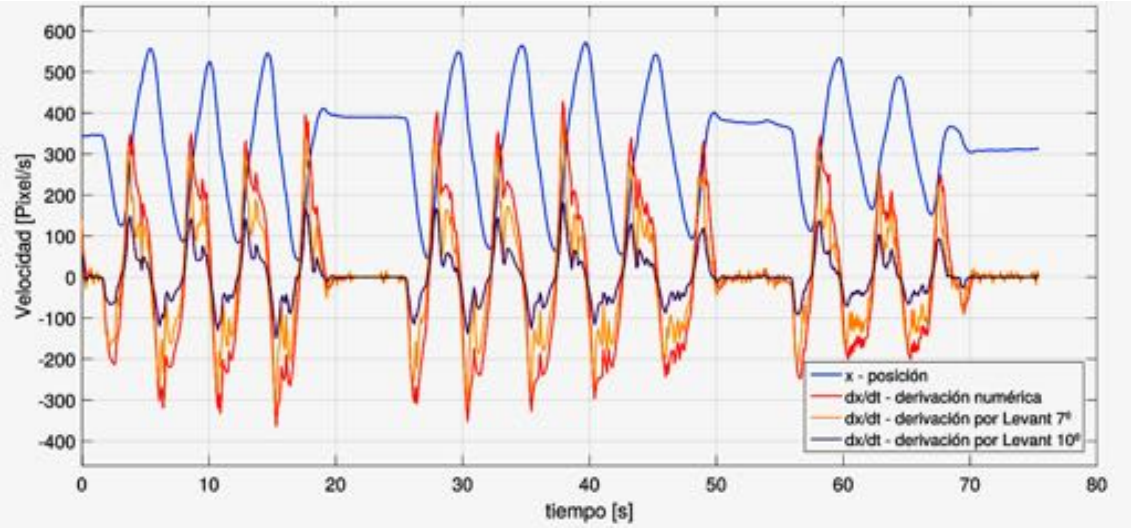


Figura 3.10: Derivación numérica y Diferenciador de Levant.

Es importante mencionar que al implementar este control, se presentó un fenomeno llamado *WindUp* que se presenta cuando existe un error muy grande durante un tiempo prolongado. Esto provoca que el término integral aumente de forma considerable para reducir el error. Pero debido a que el actuador que se usó es limitado, es decir los motores utilizaban una tensión entre 0 a 5V (0-255, pwm de 8 bits), la salida de control se saturaba y el termino integral seguia creciendo. Por lo que la respuesta del sistema se manifestaba en retrasos extraños.

Para mitigar el efecto del WindUp, se acotó tanto al controlador PID como al término integral, una ves que se alcanzaba el límite el PID detenía el funcionamiento del término integral.

Capítulo 4

Desarrollo del Prototípo

La selección de un prototipo adecuado para realizar las pruebas de los algoritmos de visión artificial se presenta en este capítulo. El hexarotor es una plataforma que se utiliza en un gran número de aplicaciones de seguridad, busqueda y rescate, debido a que presenta gran estabilidad para montar cámaras o instrumentos de medición. El hexarotor fue seleccionado para este proyecto principalmente porque es una plataforma que resulta ser muy estable en comparación a otros multirotores y principalmente porque es una plataforma que ofrece una alta capacidad de carga útil, en la cual es posible implementar el sistema del gimbal y la cámara además de otros sistemas embebidos utilizados.

Las caracteristicas generales de la plataforma utilizada asi como las adaptaciones que se realizaron en ella, la descripción de las piezas que la componen y y su función para la generación de una plataforma automatizada en la detección y seguimiento de objetivos son presentadas a continuación.

4.1. Características Generales

La plataforma que se utiliza fue seleccionada considerando que fuera capaz de levantar el peso del sistema cámara - gimbal y la computadora embebida.
Los componentes del hexarotor se presentan a continuación en la figura 4.1.

Figura 4.1: Hexarotor, prototipo utilizado.

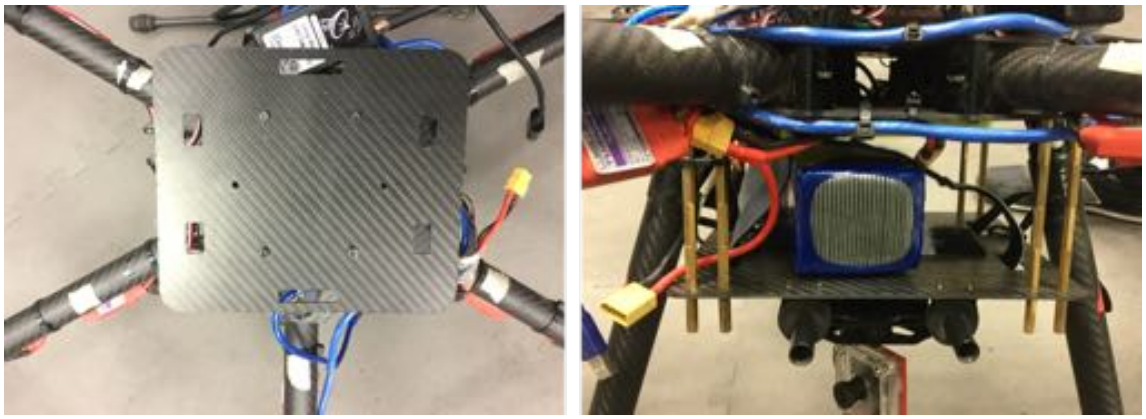
Componentes

Estructura. La estructura tiene la característica de ser plegable para facilitar el transporte y almacenamiento. Consiste en una cubierta unida a 6 tubos de 25mm de diametro, todo hecho de fibra de carbono Toray 3K. Los tubos cuentan con sujeciones a los extremos para los motores elaboradas de aluminio.
Todo mecanizado en CNC cumpliendo con altos estándares, además todas las piezas son unidas mediante tornillos de acero. Tiene un peso de 1020gr.
El tren de aterrizaje al igual que la estructura esta hecho de fibra de carbono y mecanizado en CNC unido a la estructura mediante piezas de aluminio. Tiene un peso de 300gr.

Adaptaciones

Debido a que en la estructra se tenía pensado colocar componentes electrónicos adicionales, fue necesario diseñar dos cubiertas más las cuales también se mecanizaron en CNC utilizando fibra de carbono para su creación. Se colocaron en los extremos de las cubiertas originales del hexarotor, y fueron utilizadas para la colocación de la computadora embebida y la fuente de energía de la plataforma.

(a) Estructura original



(b) Estructura modificada

Figura 4.2: Adaptaciones en la estructura.

Motores, hélices y controladores de velocidad. Los motores utilizados en el proyecto fueron motores sin escobillas Turnigy Multistar. Sus características se

pueden encontrar en la tabla 4.1 . Los rotores utilizados tienen un diametro de 15 pulgadas y un ángulo de paso de 5.5 pulgadas.

Tabla 4.1: Especificaciones del motor Multistar 5008-340.

Parámetro	**Valor**
KV	340 RPM/V
Celulas LiPo	4-6s
Potencia Máxima	700 W
Corriente Máxima	40 A
Sin corriente de carga	0.7 A
Resistencia Interna	0.130 Ω
Número de polos	12
Dimensiones	(Dia.xL) 59 x 28 mm
Eje del motor	4 mm
Peso	163 g

Los controladores de velocidad son componentes esenciales dentro del multirotor. Estos dispositivos electrónicos tiene como propósito controlar la velocidad de los motores junto con el sentido de giro. Las especificaciones técnicas se pueden ver en la tabla 4.2.

Fuente de energía. Se utilizó una bateria de polímero de litio (LiPo), com-

Tabla 4.2: Especificaciones del Controlador de Velocidad Turnigy PLUSH-40A.

Parámetro	**Valor**
Consumo de Corriente	40 A Continua
Corriente de Desborde	55 A
Modo BEC	Lineal
BEC	5V/3A
Dimensiones	55 x 28 x 13 mm
Peso	33 g

puesta de 6 celdas en paralelo, 6S1P, con un voltaje nominal igual a 22.2V, con una capacidad de corriente de 8400 mAh, una taza constante de descarga de 25C. Su

Figura 4.3: Componentes de la plataforma.

peso es de 1120 gr.

La duración de vuelo estacionario sin interrupción con esta bateria y sin la computadora embebida y el sistema de cámara-gimbal fue de 12 minutos.

Figura 4.4: Bateria 6S1P

Controlador de vuelo. El controlador de vuelo es una tarjeta electrónica la cual tiene como principal función estabilizar al hexarotor. Es basicamente el cerebro del multirotor. El controlador contiene algoritmos para controlar la altitud y posición del multirotor en vuelo, además de contar con algoritmos de navegación y control. El software que lo controla conecta con distintos sensores, extensiones de telemetría y otros accesorios periféricos. Además tiene la característica de ser un hardware libre, por lo que es de facil acceso, bajo costo y alta calidad para proyectos académicos e industriales.

El hardware que se utilizó pertenece a la serie de Pixhawk, en especifico se identifica como el Pixhawk 1, el software que lo controla se llama Dronecode y el controlador utiliza el Firmware PX4 que se puede compilar es sistemas operativos NuttX.

El controlador de vuelo además incluye diversos sensores para realizar la estabilización del multirotor, tales como un giroscopio, un acelerómetro, un magnetómetro y un barómetro.

Una interesante opción de Pixhawk es su comunicación por telemetría con computadoras o tablets que permite un completo, avanzado y de bajo costo control de vuelo con el software QGroundControl, el cual se utilizó en el proyecto.

Y debido a la compatibilidad con tecnologías Spectrum DSM-X® se utilizó un sistema de transmisor DX8 MD2 con receptor AR8000 en una banda de comunicación igual a 2.4 GHz. Las especificaciones técnicas se presentan en la tabla 4.3.

GPS. El módulo del GPS que se incorpora en la plataforma tiene una brújula digital IST8310, que proporciona un método conveniente para montar la brujula lejos de fuentes de interferencia que puedan estar presentes en los confines del vehículo.

Tabla 4.3: Especificaciones del Controlador de Vuelo Pixhawk 1.

Parámetro	**Valor**
Microncontrolador	STM32F427
CPU	180 MHz ARM® Cortex®
RAM	256 KB
Interruptor de seguridad	Externo
Giroscopio	ST L3GD20H 3-ejes 16 bit
Acelerómetro/Magnetómetro	ST LSM303D 6-EJES 14 bit
Acelerómetro/Giroscopio	MPU 6000 3-ejes
Barómetro	MEAS MS5611
Alimentación	4.8V-5.4V
Puertos Serial	5x
Puertos CAN	2x
Compatibilidad	Spektrum, Futaba
Puertos	PPM, RSSI, 12C, SPI, 3.3-6.6V ADC
Puertos microUSB	1-interno, 1-externo
Dimensiones	50 x 80 x 13 mm
Peso	80 g

Tiene un módulo UBLOX M7N asi como un indicador led. Este modulo se envía a una velocidad de transmision de 38400 Bauds, 5Hz.

Computadora. La computadora embebida es usada en conjunto con el controlador de vuelo, de forma desacoplada. Mientras que el controlador de vuelo se encarga principalmente del control del hexarotor, la mini computadora es usada para el procesamiento de los algoritmos de detección y seguimiento sin interrumpir las funciones del controlador. La computadora que se utilizó es la NUC5i7RYH de Intel®. Las especificaciones técnicas se muestran en la Tabla 4.4.

Microcontrolador. La referencia de este dispositivo se hace en la sección *Integración de Hardware* en el capítulo anterior. Las especificaciones específicas se muestran en la tabla 4.5.

Gimbal. La mención de este dispositivo electrónico se hace en el capítulo anterior, sin embargo forma parte de la plataforma. Las especificaciones técnicas se presentan

Figura 4.5: a) Controlador de vuelo Pixhawk 1. b) GPS utilizado.

en la tabla 4.6.

4.2. Descripción del Sistema Embebido

El sistema embebido presenta dos procesos principales. El microprocesador del controlador de vuelo se encarga del control y la estabilidad del hexarotor. La computadora embebida, por otro lado, es la encargada del procesamiento del algoritmo de seguimiento de objetivos.

Por medio de una red WLAN y a través del Nodo Maestro de la plataforma de ROS, la computadora embebida se comunica con una PC en tierra, de esta forma se coordina la información del procesamiento entre ambos sistemas. Por medio de la red, se envia video continuamente a la PC en tierra, y esta a la ves envía datos hacia la computadora embebida de la cual depende el algoritmo de procesamiento. Ambos sistemas están desacoplados, de esta manera las operaciones que realizan no los afectan entre sí. En la Figura 4.8 se muestra un diagrama mostrando los sensores y las conexiones en la plataforma.

Tabla 4.4: Especificaciones de la mini computadora NUC

Procesamiento	
Procesador	Intel® Core i7-5557U 5ta generación 3.4GHz
Graficos	
Graficos	Intel® Iris^{TM} 6100
Monitor	1 puerto UHD 4K, 1 puerto Mini HDMI
Memoria del sistema	
RAM	8GB (2 ranuras DDR SO-DIMM, hasta 16GB)
Capacidad de almacenamiento	
Almacenamiento	1 ranura tipo M.2 que admite SATA(22x42, 22x60, 22x80) o PCIe SSDs
Periféricos	
USB	4 puertos usb 3.0 (alta velocidad), 2 puertos usb 2.0 (internos)
Red	Conexión inalámbrica de doble banda AC 7265, 802.11ac, hasta 867 Mbps
LAN	Integrada Intel® Gigabit
Bluetooth	Modo dual 4.0
Requerimientos de energía	
Adaptador	19V, 65W AC/DC , multiregión (conector IEC tipo A, C, G, I)
Dimensiones	115 x 111 x 48.7 mm
Peso	600 g

Figura 4.6: a) Computadora NUC 5i7RYH. b)Microcontrolador Arduino Mega 2560.

Tabla 4.5: Especificaciones del microcontrolador Arduino Mega 2560.

Microcontrolador	ATmega2560
Voltaje Operativo	5V
Voltaje de entrada	7V
Voltaje máximo	20V
Pines digitales de entrada/salida	54 (15-PWM)
Pines análogos de entrada	16
Memoria flash	256 KB
SRAM	8 KB
EEPROM	4 KB
Frecuencia de reloj	16 MHz

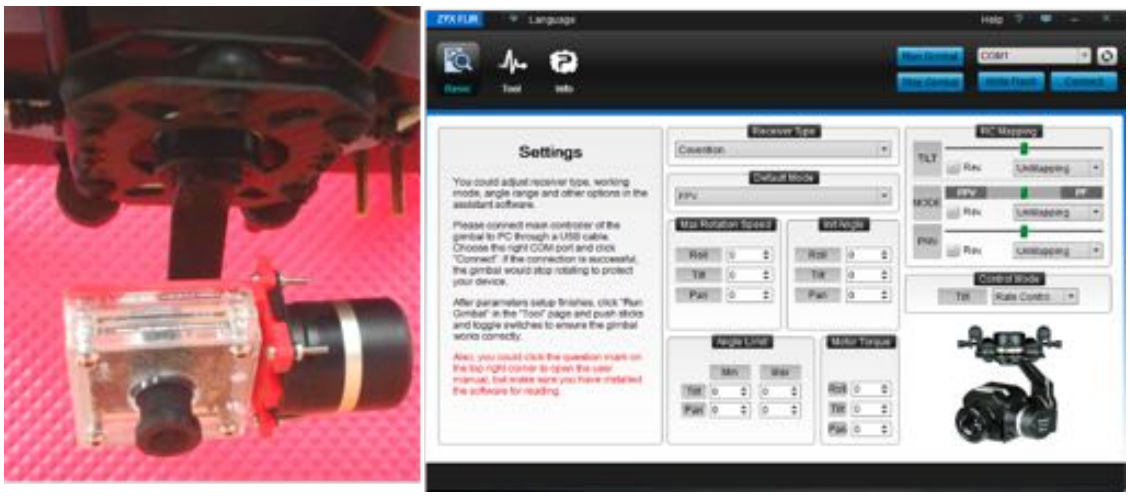


Figura 4.7: Gimbal TAROT ZYX FLIR y su Software de asistencia.

Tabla 4.6: Especificaciones del gimbal TAROT ZYX FLIR.

Módulo de control	TAROT ZYX-M
Corriente Operativa	30mA (@25V)
Alimentación de entrada	3S-6S Li (11V-26V)
Corriente de desborde	350mA (@25V)
Temperatura Operativa	-20°C a 50°C
Peso	162g
Dimensiones	79 x 95 x 95.5mm
Velocidad de rotación máxima	Tilt (Cabeceo): $\pm 200°/s$ Roll (Alabeo): $\pm 200°/s$ Pan (Guiñada): $\pm 200°/s$
Precisión de control de orientación	$\pm 0.02°$
Plataformas Software Asistencia	Windows XP/VISTA/7/8/10

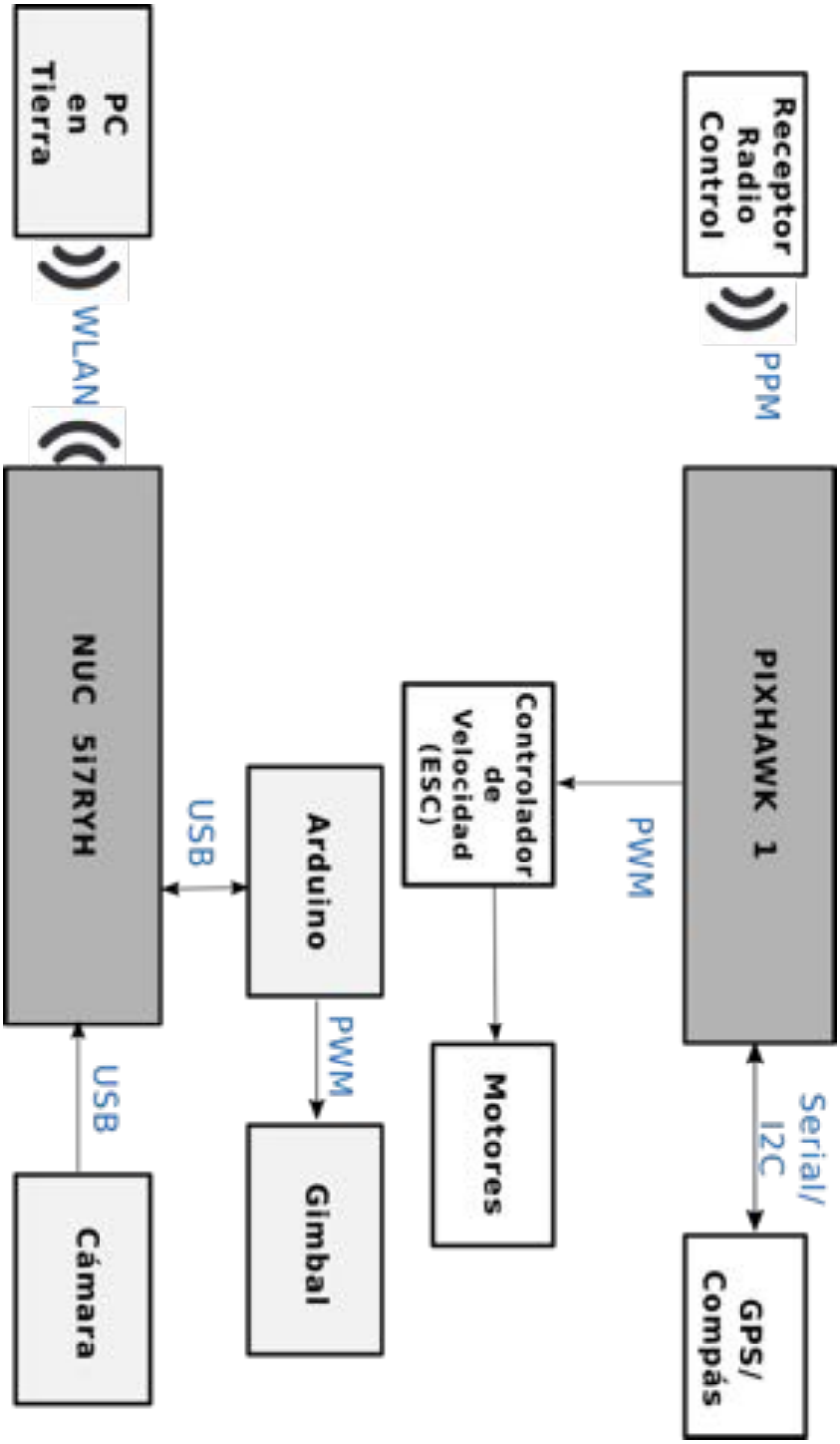


Figura 4.8: Diagrama de conexiones del Hardware.

Capítulo 5

Pruebas y Resultados Experimentales

En este capítulo se presentan los resultados obtenidos de las pruebas que se realizaron con los diferentes algorítmos de detección y seguimiento de objetivos en un espacio cerrado en la plataforma de pruebas presentada en el capitulo 3, y el control de seguimiento que se implementó en el gimbal utilizando el algoritmo CMT sobre el hexarotor en el exterior.

5.1. Pruebas de los Algoritmos de Detección y Seguimiento

Método de Lucas-Kanade utilizando Flujo Óptico

En esta sección mostramos los resultados experimentales, utilizando tomas de video de un vehículo de pruebas a escala en un espacio cerrado para evaluar el sistema. En la figura 5.1 se presentan los resultados de la utilización de este algoritmo.

La estimación del flujo óptico independientemente de la movilidad, da como resultado puntos de alto constraste en los que presenta principal atención. Como se muestra en la figura 5.1, además de las partes móviles, hay algunos vectores, posibles candi-

datos a movimiento en la esquina superior derecha, los cuales pueden ser eliminados mediante operaciones morfológicas en función del tamaño y características como el movimiento. Antes de aplicar la estimación del flujo óptico en cada cuadro del video, el formato de la imágen se convirtió del formato RGB a escala de grises, debido a que las mediciones de intensidad actúan mejor en marcos de escala de grises.

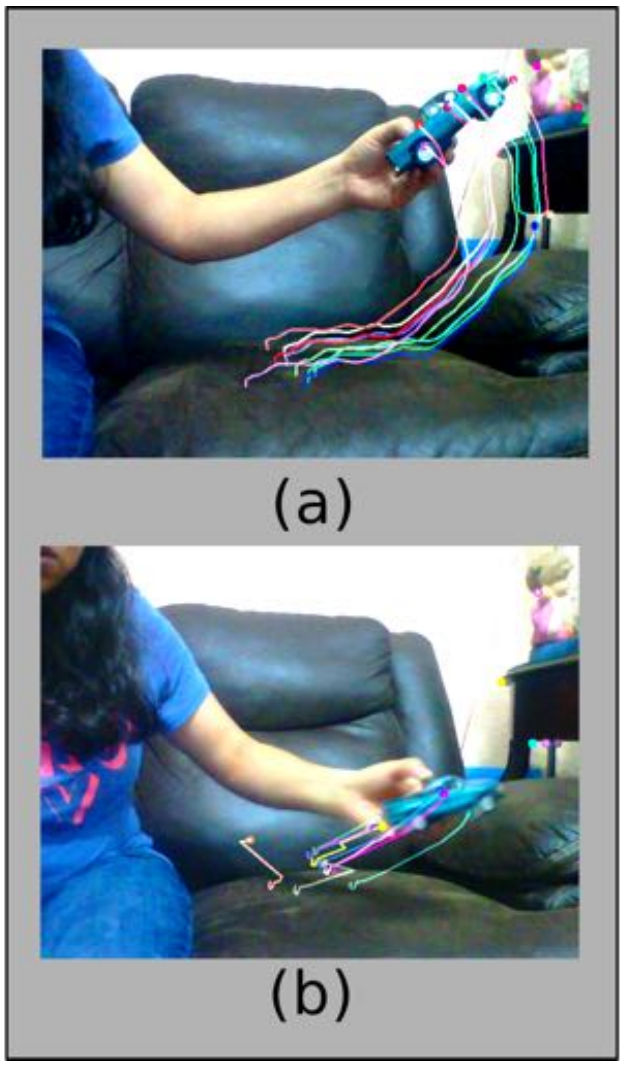


Figura 5.1: Imágenes 60 y 71 de la secuencia originial de un video de pruebas, (a) Vectores de flujo óptico siguiendo al vehículo. (b) Los vectores de flujo óptico en una imágen borrosa.

Como se puede ver, el algoritmo funciona mejor si el movimiento del vehículo en movimiento entre cuadros consecutivos es lento. Por el contrario, si el movimiento es grande, el algoritmo falla.

Los puntos vecinos en la escena 5.1 (a) tipicamente pertenecen al vehículo en sí y por lo tanto presentan un movimiento similar. De tal forma que existe coherencia espacial en el flujo de la imágen sin embargo esto no sucede en la escena (b) por que el movimiento es muy rápido y los vectores de movimiento pierden la secuencia.

Como se mencionó en el capítulo 3, debido al número relativamente grande de iteraciones, el método de Lucas-Kanade es computacionalmente lento pero tiene buena precisión. Este costo computacional es debido al cálculo de los vectores de movimiento.

Algoritmo ORB

En el presente trabajo también se realizó la implementación del algoritmo de detección ORB y en la figura 5.2 podemos ver los resultados del algoritmo utilizando un vehículo a escala para realizar las pruebas. Estas pruebas se llevaron a cabo en el laboratorio de la UMI-LAFMIA en las instalaciones del CINVESTAV unidad Zacatenco.
En 5.2 se presentan dos resultados de forma independiente. El vehículo de referencia se muestra en (a) y (c) utilizando distintos grados de iluminación y por lo tanto obteniendo un vector de puntos diferente uno del otro, por lo cual se tratarán de forma independiente. En 5.2 (b) y (d) se muestran dos cuadros de la secuencia del video que se realizó.
En las figuras (a) y (b) al igual que en el par (c) y (d) podemos ver el mapeo entre los puntos candidatos en la secuencia con el vehículo de referencia.

En los resultados en (a) y (b), podemos ver que un grupo de puntos que pertenecen al fondo de la imagen (b) realizan coincidencias con la imágen de referencia (a). Esto causa que el código devuelva un mapeo inexacto, y se debe principalmente al ruido que se presenta en el fondo.

Tambien se puede ver que el vehículo en la imagen (b) tiene una orientación di-

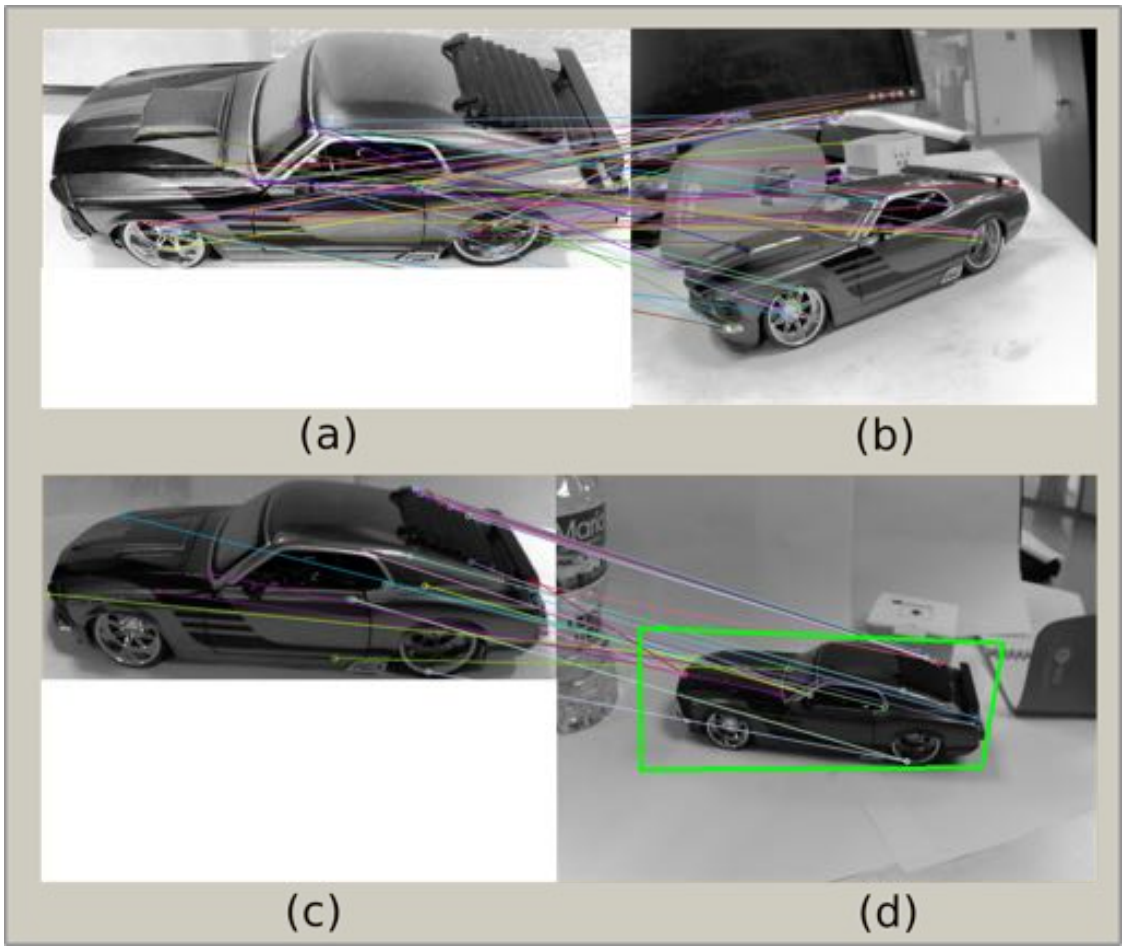


Figura 5.2: Imágenes a escala de grises 03 y 45 de la secuencia de un video de pruebas. (a)Vehículo de referencia con mayor iluminación (b) Cuadro de la secuencia donde aparece el vehículo correspondiente al análisis con (a). Imágenes a escala de grises 07 y 22 de la secuencia 2. (c) Vehículo de referencia con menor iluminación. (d) Cuadro de la secuencia donde aparece el vehículo correspondiente al análisis con (c).

ferente al vehículo de referencia y aunque el algoritmo ORB es invariante ante los cambios de rotación, está limitado a formas que no cambian tanto en la rotación.

En (c) y (d), se puede notar que mediante la localización del objeto por medio de los puntos coincidentes, se realizó una detección exitosa en (d), aunque a diferencia del par anterior aqui el grado de orientación del vehículo en la secuencia con respecto al vehículo de referencia es mínimo.

Algoritmo de detección por Color

A continuación se presentan los resultados del algoritmo de detección por color. Las pruebas se realizaron en el laboratorio de la UMI-LAFMIA en las instalaciones del CINVESTAV unidad Zacatenco al igual que las pruebas del algoritmo anterior. Se realizaron distintos videos utilizando vehículos a escala como se muestra en la figura 5.3.

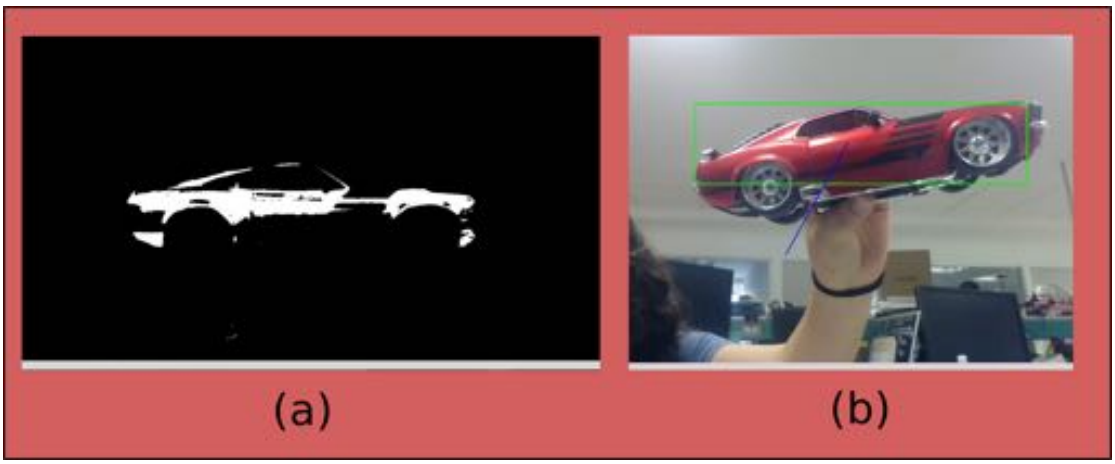


Figura 5.3: Imagen 56 de la secuencia de un video de pruebas. (a) Resultados de la substración en la imagen. (b) Imagen clara y sin ruido que muestra el vehículo detectado.

El seguimiento que realiza este algoritmo utilizando el color como su principal dependencia fue exitoso. En la imagen 5.3(a) se muestra el resultado de la sustracción del cuadro, es decir la imagen se convirtió a una imagen binaria como parte del proceso de detección por éste método y como se puede notar se logró obtener una imagen refinada al aplicar las operaciones morfológicas y de filtrado.

La imagen 5.3(b) es el resultado de separar correctamente el vehículo del fondo de las imagenes en el video, la detección se realiza mediante un recuadro alrededor del vehículo y un vector de seguimiento que utiliza el centroide del mismo recuadro.

Algoritmo CMT

En la figura 5.4 se muestran los resultados de las pruebas que se hicieron con el algoritmo CMT, en un espacio cerrado en el laboratorio de la UMI-LAFMIA. Estas pruebas se realizaron con un vehículo a escala para fines prácticos, sin embargo, con éste algoritmo en particular también se realizaron pruebas en el exterior, mas adelante se hablarán sobre ellas.

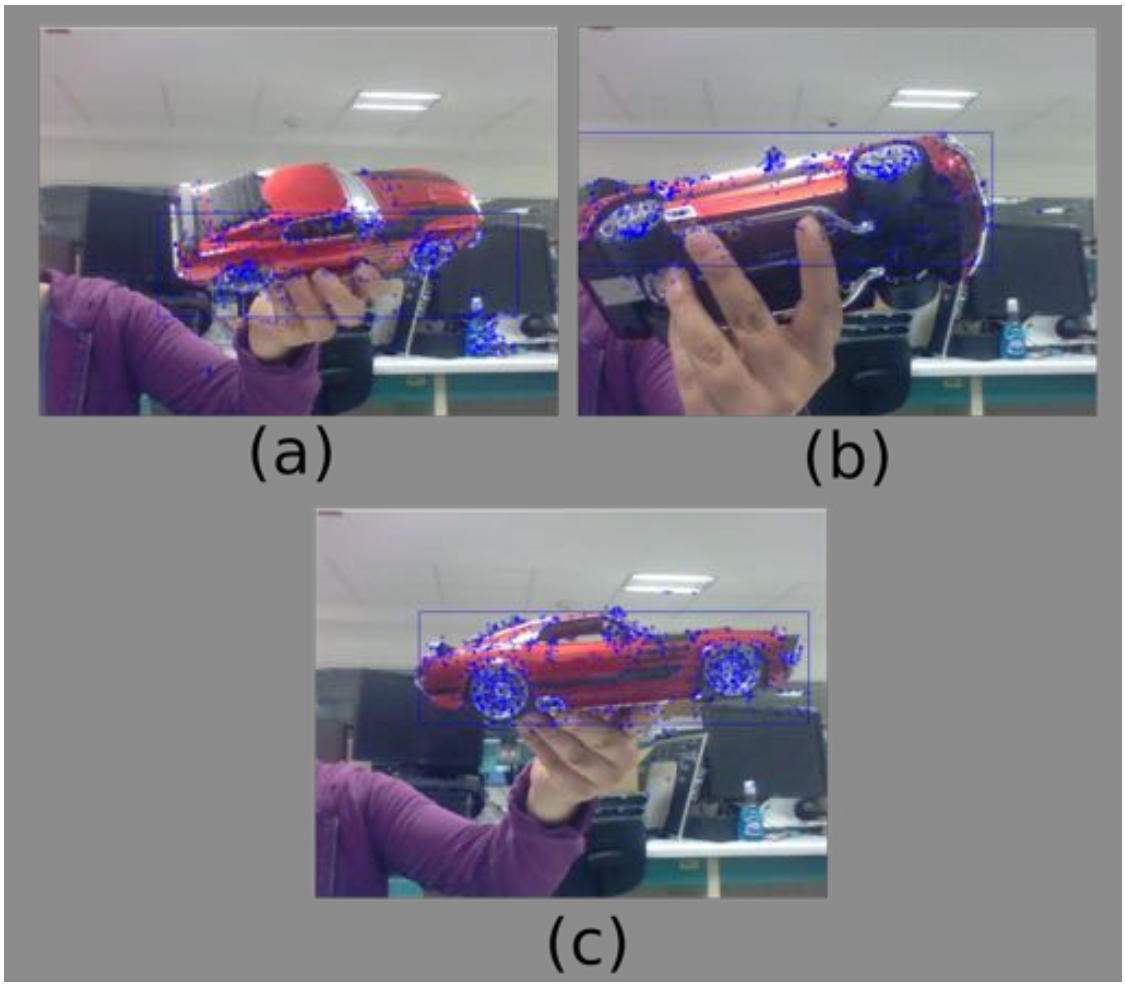


Figura 5.4: Imagen 110, 135 y 54 de la secuencia de un video de pruebas. (a), (b) y (c) muestran los resultados del algoritmo, detectando al vehículo utilizando un recuadro azul y además mostrando los *keypoints* del objeto con el mismo color.

Como se puede ver en la figura 5.4 el algoritmo logra realizar la detección y el seguimiento del vehículo enmarcándolo con un recuadro de color azul y mostrando los puntos característicos resultantes en cada cuadro.

En las imágenes 5.4 (a), (b) y (c) se observa que el algoritmo es robusto ante cambios en la orientación, en la escala, en la traslación e incluso ante cambios de iluminación. Además en (b) se puede notar que también es robusto ante oclusiones parciales, considerando que la imagen del vehículo inicial que se utilizó para el procesamiento fue similar a (c), tomando solo un costado del vehículo.

Por otro lado, en (a), (b) y (c) los recuadros que enmarcan al vehículo cambian en sus dimensiones, esto se debe a que las dimensiones del rectángulo dependen del voteo realizado en el proceso de consenso en cada imagen del video para elegir el centroide del objeto, de tal forma que las dimensiones cambian dependiendo de la ubicación del nuevo centroide.

Es preciso señalar que debido a las características de robustez ante cambios en la escala y orientación, además de utilizar ambos procesos, *detección* y *seguimiento*,y realizando un voteo para el centroide del objetivo, usando suposiciones sobre la constelación geométrica de los puntos característicos de tal forma que el procesamiento disminuya y al mismo tiempo haciendo robusta la detección y seguimiento, se utilizó posteriormente en las pruebas experimentales el algoritmo CMT, para realizar la detección y seguimiento de los vehículos de prueba.

5.2. Pruebas Experimentales del Prototipo

En esta sección se presentan los resultados de las pruebas que se realizaron con la plataforma de vuelo y el sistema gimbal-cámara incluyendo una estación de control en tierra para la comunicación entre ambos sistemas, realizadas en lugares abiertos a una altura de 15 m aproximadamente y con diferentes vehículos reales.

5.2.1. Pruebas de Vuelo

Durante las pruebas experimentales, se utilizaron diferentes vehículos y escenarios. Durante los experimentos se implementaron los mismos procedimientos en cada uno de ellos.

En la Figura 5.5 (a), (b) y (c) se pueden ver los resultados de la implementación del algoritmo CMT. Como se puede visualizar el control sobre la plataforma del gimbal intenta mantener al objetivo en el centro de la imagen. Por otro lado, también podemos observar que a pesar de los cambios en la iluminación el algoritmo CMT funciona de forma eficiente.

En la Figura 5.6 (a), (b) y (c) se pueden ver las demostraciones del vuelo del hexarotor corriendo el algoritmo de detección y seguimiento en el sistema embebido, sobre los diferentes vehículos.

En la Figura 5.7 (a), (b) y (c) se muestra la utilización de la estación en tierra, realizando el proceso de selección del vehículo correspondiente a cada insciso, para posteriormente enviar esta instrucción al sistema embarcado en el hexarotor.

5.2.2. Control en el Sistema Cámara - Gimbal

En esta sección se presentan los resultados gráficos del control implementado en el gimbal en Pan y Tilt en las pruebas experimentales sobre la camioneta Pick Up y Rover.

Como se mencionó anteriormente, se realizó un control PID para el seguimiento del *centroide* del rectángulo que rodea al vehículo que se quiere seguir, utilizando el marco referencial de la **imagen de la cámara**. Por lo tanto en los resultados mostrados a continuación se puede ver el comportamiento de la posición del centroide del vehículo (Camioneta Pick Up y Rover) con respecto al centro de la imagen de la

cámara el cual tiene por coordenadas (**320, 240**) pixeles , en el eje x e y respectivamente.

En las Figuras 5.8, 5.9 y 5.10 se presentan las gráficas de la posición y velocidad en los ejes x y y del centroide del rectángulo que rodea a la camioneta Pick Up en el marco referencial de la imagen.

Mientras que en 5.11, 5.12 y 5.13 se presentan las gráficas de la posición y velocidad en los ejes x y y del centroide del rectángulo que rodea al Rover.

Es importante mencionar que los vehículos se encuentran en movimiento durante las pruebas mientras que el hexarotor permanece en una posición fija, siendo el gimbal el encargado de seguir a los vehículos utilizando los ejes de Pan/Tilt e intentando posicionarlos en el centro de la imágen todo el tiempo. La altura a la que se encontraba el hexarotor, es de 15 m como se menciona al inicio de la sección.

A continuación se muestra el link del video de la prueba de detección y seguimiento del vehículo ROVER. `https://youtu.be/1QN2-w-UMrU`.

Figura 5.5: Algoritmo CMT en exteriores (a) Automovil Sedán (b) Rover (c) Camioneta Pick Up.

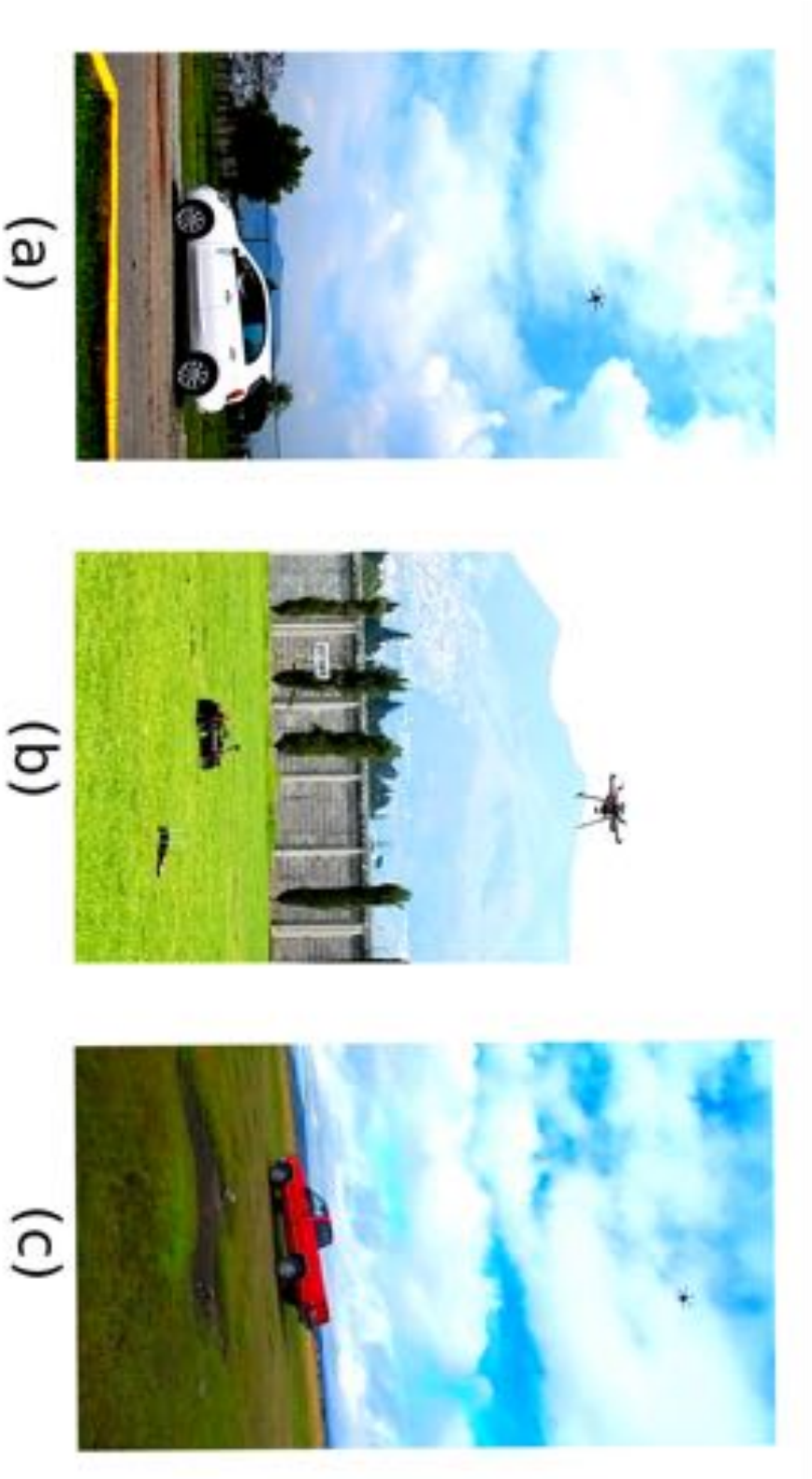


Figura 5.6: Pruebas de vuelo sobre el hexarotor. (a) Automovil Sedán (b) Rover (c) Camioneta Pick Up.

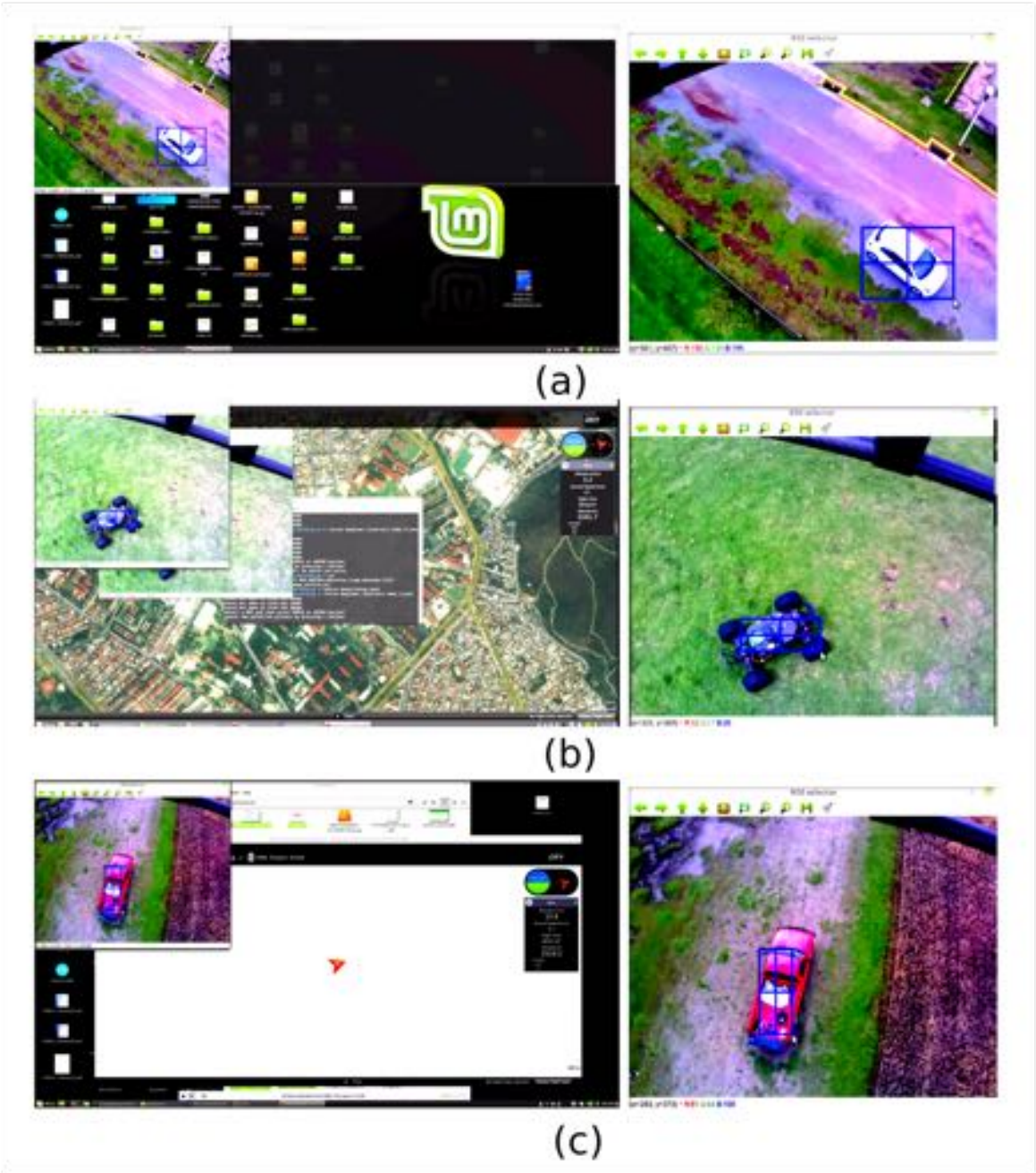


Figura 5.7: Selección del objetivo. (a) Automovil Sedán (b) Rover (c) Camioneta Pick Up.

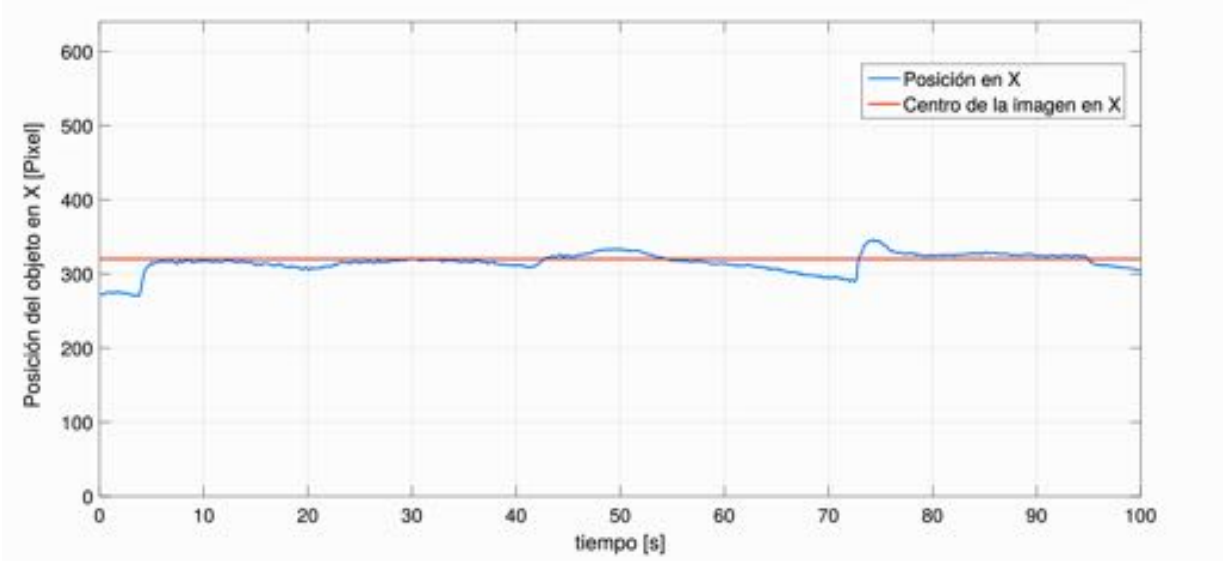


Figura 5.8: Camioneta Pick Up: Posición en el eje X en el marco referencial de la Imagen.

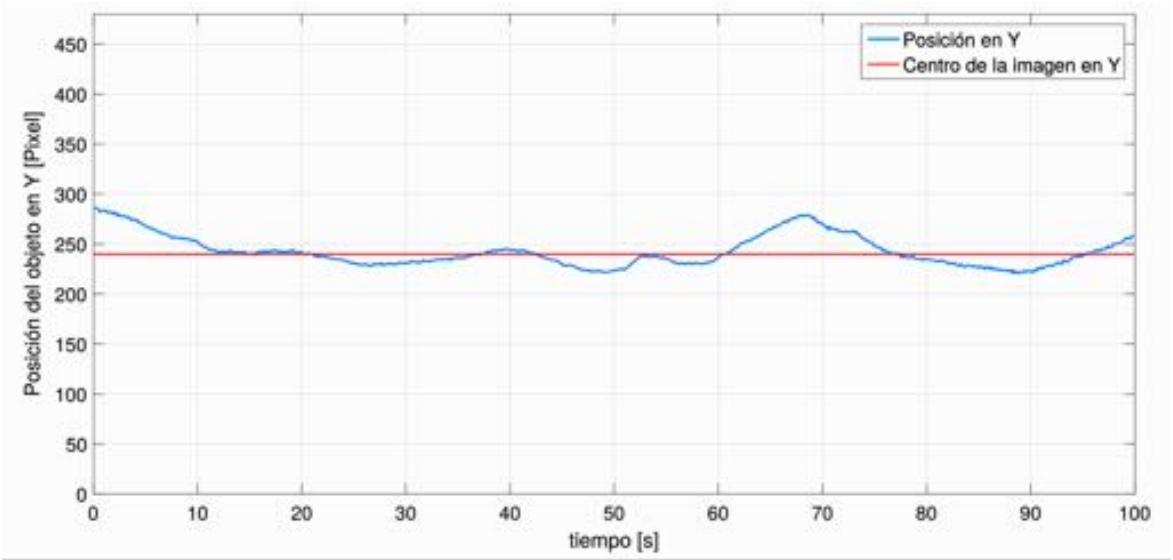


Figura 5.9: Camioneta Pick Up: Posición en el eje Y en el marco referencial de la Imagen.

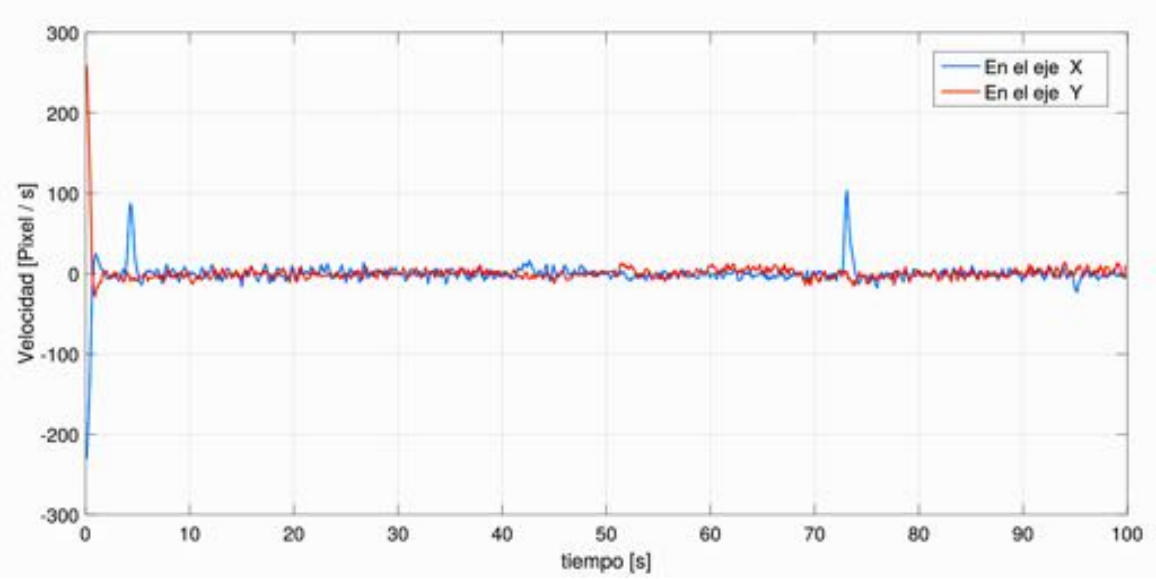


Figura 5.10: Camioneta Pick Up: Velocidad en el eje X e Y en el marco referencial de la Imagen.

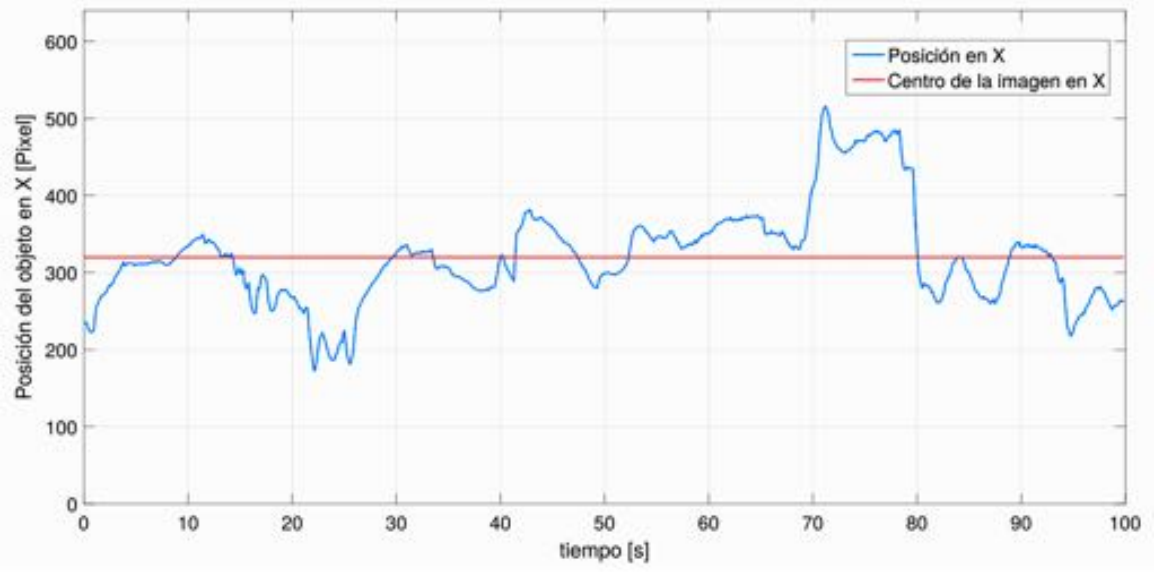


Figura 5.11: Rover: Posición en el eje X en el marco referencial de la Imagen.

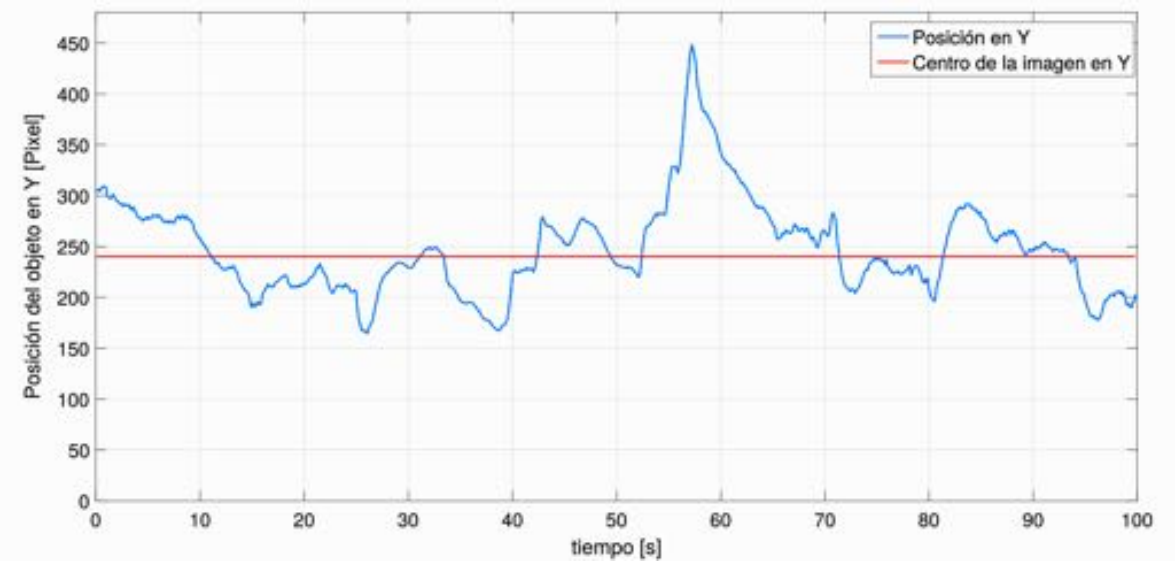


Figura 5.12: Rover : Posición en el eje Y en el marco referencial de la Imagen.

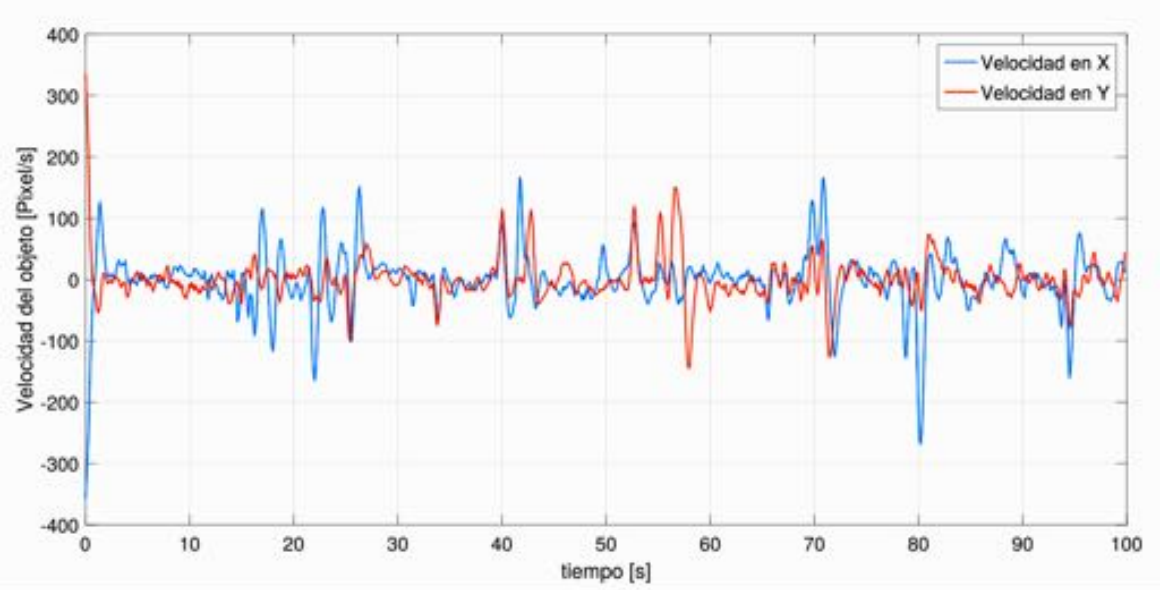


Figura 5.13: Rover: Velocidad en el eje X e Y en el marco referencial de la Imagen.

Capítulo 6

Conclusiones y Trabajo Futuro

6.1. Conclusiones

Mediante el estudio e implementación de diversos algoritmos de detección y seguimiento, se analizaron las ventajas así como las limitaciones de cada uno de ellos, y logrando seleccionar e implementar el algoritmo de detección y seguimiento de objetivos adecuado.

El algoritmo ORB tiene por ventajas ser eficiente al detectar un número grande de características sobre un objeto y el mapeo es relativamente rápido en comparación a otros detectores, sin embargo, presenta limitaciones ante la iluminación y también en la escala, por lo cual, este algoritmo no fue viable para el seguimiento de un vehículo utilizando un VANT.

Por otra parte en el algoritmo de seguimiento de flujo óptico mediante el método de Lucas-Kanade la implementación fué sencilla en comparación a otros métodos y el tiempo de procesamiento rápido, pero presentó limitaciones ante movimientos muy rápidos, la oclusión y la inestabilidad en el video.

El algoritmo de detección por Color, fué el más sencillo en cuanto a la implementación, además de tener un rendimiento aceptable en tiempo real, sin embargo, es un

algoritmo selectivo que utiliza la tonalidad como factor para el seguimiento.

El algoritmo CMT combina el proceso de la detección y el seguimiento. Trabaja con precisión ante cambios en la traslación, escala y orientación en distintos objetos, además es robusto ante la oclusión parcial e incluso total en los objetos así como a los cambios en la apariencia. Debido a esto, este fué el algoritmo que se implementó en la plataforma experimental con éxito. Es importante mencionar que no está disponible en las librerias de OpenCV.

La implementación de una ley de control en el sistema cámara-gimbal para lograr el seguimiento de un vehículo terrestre se logró con éxito.
El control realizado sobre la plataforma del gimbal se realizó utilizando un control PID, aprovechando los datos de posición del objetivo en el marco referencial de la imágen en la cámara, estimando la velocidad y sintonizando el control. Sin embargo, es necesario mejorar físicamente esta plataforma para optimizar el desempeño del control PID.

Por último, se utilizó un hexarotor para la implementación de todo el sistema, debido a la capacidad de carga con la que cuenta. Y se realizaron con éxito las pruebas de funcionamiento del algoritmo de detección y seguimiento sobre el vehículo aéreo, utilizando diversos entornos y diferentes vehículos reales en un campo abierto como se observa en el capítulo anterior, comprobando que el algoritmo de visión tiene la capacidad de detectar múltiples vehículos.

6.2. Trabajo Futuro

En el trabajo a futuro, se podrá abordar la cuestión de realizar mejoras sobre el algoritmo de detección y seguimiento CMT, empleando diferentes métodos para la detección y descripción de puntos clave. Además, podría ser interesante aplicar el método para el reconocimiento por categorias de vehículos terrestres.

Entre las mejoras a la parte de la detección y seguimiento del objetivo, sería conveniente mejorar la robustez del algoritmo en cuanto a la escala, y permitir realizar vuelos a mayor altura con el vehículo aéreo.

Otra posible mejora, podría ser la inclusión del vehículo aéreo en el seguimiento del objetivo, realizando un control sobre el VANT que permita seguir al objetivo en coordinación con el gimbal.

Por último, la selección de un VANT de ala fija podría ser una opción para aumentar la autonomía de vuelo y realizar el seguimiento por un tiempo prolongado, diseñando un sistema de planificación de trayectorias que trabajen en coordinación con el seguimiento del objetivo.

Apendice A

6.3. Conceptos Generales

En esta sección se presentan algunos conceptos generales necesarios para el entendimiento del trabajo.

Distancia de Hamming

La distancia de Hamming es el número de bits en que difieren dos palabras del código. Sean x_i y x_j dos secuencias binarias de la misma longitud $i, j = 1, ..., K$, la distancia Hamming entre ellas es el número de símbolos en que difiere. $d_{ij} = W(x_i \oplus x_j)$.

Gimbal

Un gimbal es una plataforma motorizada y controlada mediante una placa con varios sensores, generalmente acelerómetros y compás magnético que se encarga mediante el uso de algoritmos de control y PIDs de mantener un objeto, normalmente una cámara estabilizada, de modo que independientemente del movimiento que realice el portador de la misma, ésta quede estable permitiendo tomar buenas capturas.

Matching

Concepto que sirve para realizar la coincidencia de puntos característicos entre dos imagenes, llamado en inglés **Matching**. Esta técnica permite comparar los puntos de interés entre dos imagenes y obtener un vector comparativo entre los descriptores correspondientes a cada imagen, con características como la distancia entre ambos

descriptores (entre mas corta, es mejor), el índice de los descriptores en la *imagen de consulta* y la *imagen de prueba* y el índice de la *imagen de prueba*.
Brute-Force Matcher es un tipo de *matching* simple en las librerias de OpenCV. Se basa en utilizar un punto de interés de la *imagen de consulta* y compararlo con todos los puntos de interés en la *imagen de prueba* calculando la distancia entre ellos, regresando el punto con la distancia más corta, utilizando como medición la distancia de Hamming en los descriptores de tipo binario.

Mensaje (ROS)

Los nodos se comunican entre sí pasando mensajes. Un mensaje es simplemente una estructura de datos, que comprende campos escritos. Se admiten los tipos primitivos estándar (entero, punto flotante, booleano, etc.), al igual que las matrices de tipos primitivos. Los mensajes pueden incluir estructuras y matrices arbitrariamente anidadas (al igual que las estructuras C).

Nodo (ROS)

Los nodos son procesos que realizan computación. ROS está diseñado para ser modular en una escala de grano fino; Un sistema de control de robot usualmente comprende muchos nodos.

Servicio (ROS)

El modelo de publicación/suscripción es un paradigma de comunicación muy flexible, pero su transporte unidireccional de muchos a muchos no es apropiado para las interacciones de solicitud/respuesta, que a menudo se requieren en un sistema distribuido. La solicitud/respuesta se realiza a través de servicios, que se definen mediante un par de estructuras de mensajes: una para la solicitud y otra para la respuesta. Un nodo proveedor ofrece un servicio con un nombre y un cliente utiliza el servicio enviando el mensaje de solicitud y esperando la respuesta. Las bibliotecas cliente de ROS generalmente presentan esta interacción al programador como si fuera una

llamada a un procedimiento remoto.

Tópico (ROS)

Los mensajes se enrutan a través de un sistema de transporte con semántica de publicación / suscripción. Un nodo envía un mensaje publicándolo en un tema determinado. El topico es un nombre que se utiliza para identificar el contenido del mensaje. Un nodo que esté interesado en cierto tipo de datos se suscribirá al tópico apropiado. Puede haber múltiples editores y suscriptores concurrentes para un solo tema, y un solo nodo puede publicar y / o suscribirse a múltiples temas. En general, los editores y suscriptores no son conscientes de la existencia de los demás. La idea es desacoplar la producción de información de su consumo. Lógicamente, uno puede pensar en un tópico como un bus de mensajes fuertemente tipado. Cada bus tiene un nombre, y cualquiera puede conectarse al bus para enviar o recibir mensajes siempre que sean del tipo correcto.

Píxel

Un píxel, es la menor unidad homogénea en color(siendo la escala de grises una gama de color monocromática) que forma parte de una imágen digital. Las imágenes se forma como una sucesión de píxeles. La sucesión marca la coherencia de la información presentada, siendo su conjunto una matriz coherente de información para el uso digital.

Puntos de interés

Los puntos característicos o de interés de una imagen, también llamados *keypoints* en inglés, son aquellos puntos que son fácilmente diferenciables en una imagen. Algunos de ellos se conocen como **bordes (edges)**, **esquinas (corners)** y **manchas (blobs)**.

Segmentación de una imagen

Dividir una imagen en regiones (continuas) ó en un conjunto de pixeles. [27] Las principales segmentaciones están basadas en:

1 Regiones

2 Contornos o fronteras

3 Bordes

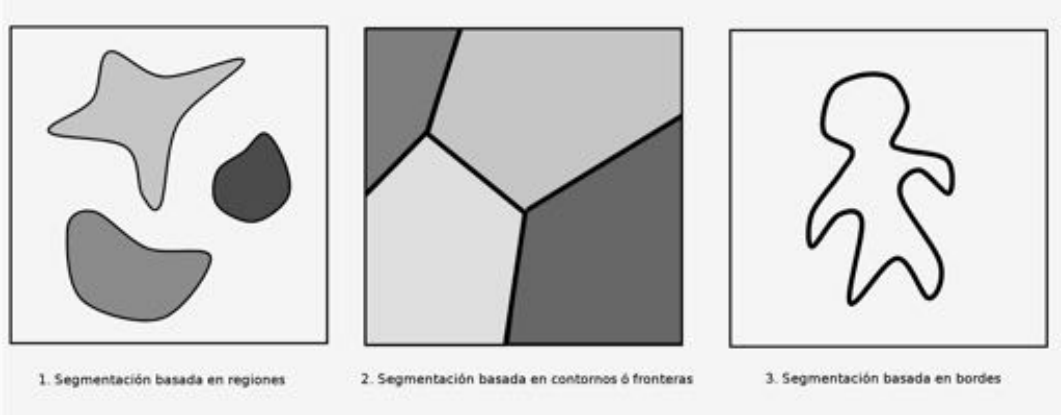


Figura 6.1: Diferentes tipos de segmentación de imagenes

Apendice B

6.4. Codigo del Sistema Embebido

Servidor

El siguiente código fue utilizado para generar el procesamiento del algoritmo implementado. Este archivo se utilizó para producir el nodo servidor en ROS:

```
#include "ros/ros.h"
#include <image_transport/image_transport.h>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/core/core.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <cv_bridge/cv_bridge.h>
#include <iostream>
#include <fstream>
#include <sstream>
#include <cstdio>
#include <cstddef>
#include <time.h>
#include "beginner_tutorials/detect.h"
#include "beginner_tutorials/Rect.h"
#include "serial/serial.h"
#include "CMT.h"
#include "gui.h"

```

```
19
20
21  using std::endl;
22  using cmt::CMT;
23  using namespace cv;
24  using namespace std;
25
26  //***algoritmo CMT**///
27  union Int{
28   uint16_t m_int;
29   uint8_t m_bytes[sizeof(uint16_t)];
30  };
31
32
33  int display(Mat im, CMT & cmt)
34  {
35
36      //Visualize the output
37      //It is ok to draw on im itself, as CMT only uses the
            grayscale image
38      for(size_t i = 0; i < cmt.points_active.size(); i++)
39      {
40          circle(im, cmt.points_active[i], 2, Scalar(255,0,0));
41      }
42
43      Point2f vertices[4];
44      cmt.bb_rot.points(vertices);
45      for (int i = 0; i < 4; i++)
46      {
47          line(im, vertices[i], vertices[(i+1)%4], Scalar
                (255,0,0));
48      }
```

```
49
50      imshow("CMT", im);
51      return waitKey(5);
52  }
53  //***algoritmo CMT**///
54
55
56  Mat a, im0_gray;
57  Rect rect;
58  bool add(beginner_tutorials::detect::Request &req,
59           beginner_tutorials::detect::Response &res)
60  {
61    res.B = req.A;
62    rect = Rect(req.rec.x_offset, req.rec.y_offset, req.rec.
        width, req.rec.height);
63    cv_bridge::CvImageConstPtr cv_ptr;
64    cv_ptr = cv_bridge::toCvCopy(res.B, sensor_msgs::
        image_encodings::BGR8);
65    a = cv_ptr->image;
66    ROS_INFO("La imagen ha sido utilizada. ");
67    return true;
68  }
69
70  int main(int argc, char **argv)
71  {
72  //Initialization Serial Communication //"/dev/ttvACM0"
73
74          if(argc != 2) return -2;
75
76          serial::Serial ser( argv[1] , 9600, serial::Timeout::
              simpleTimeout(1000));
77         cout << "Is the serial port open?";
```

```
78          /*if(ser.isOpen())
79          cout << " Yes." << endl;
80          else
81          cout << "No." << endl;*/
82  //--------->
83  //Variables of serial communication
84          Int Header;
85          Int pntx;
86          Int pnty;
87          Int pntmin;
88          Int pntmaxx;
89          Int pntmaxy;
90          Point2f centro;
91          uint16_t x,y;
92          int header = 1000;
93          int max_x = 620;
94          int max_y = 480;
95          int min = 20;
96         uint8_t checksum;//Variable checksum
97  //--------->
98
99          ros::init(argc, argv, "node_service");
100         ros::NodeHandle n;
101        Mat frame;
102         sensor_msgs::ImagePtr msg;
103        CMT cmt;
104        image_transport::ImageTransport it(n);
105        image_transport::Publisher pub = it.advertise("camera/
             image", 1);
106 /*
107 // Convert the passed as command line parameter index for the
    video device to an integer
```

```
108          std::istringstream video_sourceCmd(argv[1]);
109          int video_source;
110  // Check if it is indeed a number
111         if(!(video_sourceCmd >> video_source)) return 1; */
112          cv::VideoCapture cap;
113          cap.open(0);
114  // Check if video device can be opened with the given index
115          if(!cap.isOpened()) return 1;
116  // Check if grabbed frame is actually full with some content
117          ros::ServiceServer service = n.advertiseService("
                image_service", add);
118          ROS_INFO("Ready to add one image and Rect");
119          Mat im,im_gray;
120         ros::Rate loop_rate(25);
121         while(ros::ok())
122         {
123            cap >> frame;
124            msg = cv_bridge::CvImage(std_msgs::Header(), "bgr8",
                   frame).toImageMsg();
125            pub.publish(msg);
126            ros::spinOnce();
127
128          if(!a.data )   // Check for invalid input
129         {
130            cout <<  "Waiting" << std::endl ;
131         }
132
133         else
134         {
135            cvtColor(a, im0_gray, CV_BGR2GRAY);
136            normalize(im0_gray,im0_gray,0,1,NORM_MINMAX);
137            equalizeHist(im0_gray,im0_gray);
```

```
138             cmt.initialize(im0_gray,rect);
139             while(true)
140             {
141
142                   cap >> im;
143                   msg = cv_bridge::CvImage(std_msgs::Header(), "
                         bgr8", im).toImageMsg();
144                   pub.publish(msg);
145                   cvtColor(im, im_gray, CV_BGR2GRAY);
146                 normalize(im_gray,im_gray,0,1,NORM_MINMAX);
147                 equalizeHist(im_gray,im_gray);
148                   cmt.processFrame(im_gray);
149                   centro = cmt.bb_rot.center;
150                   x = round(centro.x);
151                   y = round(centro.y);
152                   Header.m_int = header;
153                   pntx.m_int = x;
154                   pnty.m_int = y;
155                   pntmin.m_int = min;
156                   pntmaxx.m_int = max_x;
157                   pntmaxy.m_int = max_y;
158                   checksum=Header.m_bytes[0]+pntx.m_bytes[0]+
                         pnty.m_bytes[0];
159                 if(ser.isOpen())
160                   {
161                      cout << " Yes." << endl;
162                      ser.write(Header.m_bytes, sizeof(uint16_t))
                            ;
163
164                      if (pntx.m_int < min){
165                      ser.write(pntmin.m_bytes, sizeof(uint16_t))
                            ;
```

```
166                     }
167                     else if (pntx.m_int > max_x){
168                     ser.write(pntmaxx.m_bytes, sizeof(uint16_t)
                           );
169                     }
170                     else {
171                     ser.write(pntx.m_bytes, sizeof(uint16_t));
172                     }
173
174
175                     if (pnty.m_int < min){
176                     ser.write(pntmin.m_bytes, sizeof(uint16_t))
                           ;
177                     }
178                     else if (pnty.m_int > max_y){
179                     ser.write(pntmaxy.m_bytes, sizeof(uint16_t)
                           );
180                     }
181                     else {
182                     ser.write(pnty.m_bytes, sizeof(uint16_t));
183                     }
184                 }
185                 else{
186                 cout << "No." << endl;
187                 }
188                 char key = display(im, cmt);
189                 im.release();
190                 im_gray.release();
191                 if(key == 'q') break;
192             }
193         }
194
```

```
195             loop_rate.sleep();
196         }
197         return 0;
198 }
```

Cliente

El siguiente código fue utilizado para la selección del objetivo en el procesamiento. Tambien se utilizó para la creación del nodo Cliente.

```
1  #include "ros/ros.h"
2  #include <image_transport/image_transport.h>
3  #include <cv_bridge/cv_bridge.h>
4  #include <opencv2/highgui/highgui.hpp>
5  #include <opencv2/tracking.hpp>  //selectROI
6  #include "beginner_tutorials/detect.h"
7  #include "beginner_tutorials/Rect.h"
8  #include <sstream>
9
10 using namespace cv;
11 using namespace std;
12
13 //Variables globales
14 Mat im0, im0_gray;
15 Rect rect;
16
17
18 //Funcion callback usada por el subscriptor
19 void imageCallback(const sensor_msgs::ImageConstPtr& msg)
20 {
21         try
22          {
```

```
23            im0= cv_bridge::toCvCopy(msg, "bgr8")->image;
24          }
25          catch (cv_bridge::Exception& e)
26          {
27            ROS_ERROR("No se puede convertir de '%s' a 'bgr8'.",
                  msg->encoding.c_str());
28          }
29  }
30
31
32  int main(int argc, char **argv)
33  {
34      ros::init(argc, argv, "Node_Client");
35
36      ros::NodeHandle n;
37
38      image_transport::ImageTransport it(n); //be used to
           publish and subscribe to images
39
40      ros::Rate loop_rate(10);
41
42      image_transport::Subscriber sub = it.subscribe("/camera/
           image", 1, imageCallback);
43
44      ros::ServiceClient client = n.serviceClient<
           beginner_tutorials::detect>("image_service"); //se crea
            el cliente
45
46      beginner_tutorials::detect srv;
47
48      sensor_msgs::ImagePtr msg;
49      //int count = 0;
```

```
50
51      while(ros::ok())
52      {
53         ros::spinOnce();
54
55         if(!im0.data )   // Check for invalid input
56         {
57            cout <<  "Could not open or find the image" << std::
                  endl ;
58         }
59
60         else
61         {
62            cv::imshow("video", im0);
63            char k = waitKey(5);
64            if (k != -1)
65            {
66                  cvtColor(im0, im0_gray, CV_BGR2GRAY);
67
68                  msg = cv_bridge::CvImage(std_msgs::Header(), "
                        bgr8",  im0).toImageMsg();
69
70                  srv.request.A = *msg;
71
72                  rect = selectROI(im0, "video");
73
74                  waitKey(50);
75
76                  destroyAllWindows();
77
78                  srv.request.rec.x_offset = rect.x;
79
```

```
80                  srv.request.rec.y_offset = rect.y;
81
82                  srv.request.rec.height = rect.height;
83
84                  srv.request.rec.width = rect.width;
85
86                  if (client.call(srv))
87                  {
88                     ROS_INFO("Se envio con exito. ");
89                  }
90                  else
91           {
92                     ROS_ERROR("Failed to call service
                          image_service");
93                  }
94
95                  k=-1;
96              }
97           }
98
99           loop_rate.sleep();
100     }
101     return 0;
102  }
```

Bibliografía

[1] OpenCV Library. (2018) `https://www.opencv.org/about.html`. 17 de Noviembre de 2018.

[2] Javed Iqbal, Mustafa Pasha, Riaz-un-Nabi, Hamza Khan y Jamshed Iqbal. *Real-Time Target Detection and Tracking: A Comparative In-depth Review of Strategies.* Life Science Journal (2013).

[3] Bay H.,Tuytelaars T. *SURF: Speeded Up Robust Features.* 9th European Conference on Computer Vision, 2006.

[4] Lowe, D. G. *Distinctive Image Features from Scale-Invariant Keypoints.* International Journal of Com- puter Vision, 60, 2, pp. 91-110, 2004.

[5] Ethan Rublee, Vincent Rabaud, Kurt Konolige, Gary R. Bradski. *ORB:An efficient alternative to SIFT or SURF.* ICCV 2011:2564-2571.

[6] Introduction to SURF (Speeded-Up Robust Features). (2016) *http* : *//docs.opencv.org/*. 24 de Octubre de 2018.

[7] B. Babenko, M-H. Yang y S. Belongie *Visual Tracking with Online Multiple Instance Learning.* CVPR (2009).

[8] Friedman, J. H., Hastie, T. y Tibshirani, R. *.Additive Logistic Regression: a Statistical View of Boosting* Technical Report, Dept. of Statistics, Stanford University (1998).

[9] Z. Kalal, K. Mikolajczyk, y J. Matas. *Forward-Backward Error: Automatic Detection of Tracking Failures.* International Conference on Pattern Recognition (2010).

[10] Z. Kalal, K. Mikolajczyk, y J. Matas. *Tracking-Learning-Detection.* IEEE (2010).

[11] Ye Wu et al. *Online Object Tracking: A Benchmark* CVPTR (2013).

[12] N. Prabhakar, AP. Sharma. *Object tracking using Frame differencing and template matching.* Research Journal of Applied Sciences, Engineering and Technology (2012).

[13] K. Du, Y. Ju, Y. Jin, y G. Li. *MeanShift tracking algorithm with adaptive block color histogram.* CECNet (2012).

[14] Svoboda, T. *Kanade–Lucas–Tomasi Tracking.* Research Report, Center for Machine Perception, Czech Technical University, Prague (2007).

[15] Vladimir Kravtchenko. *Tracking Color Objects in Real Time.* Diploma in Computer Engineering, I.M. Gubkin State Oil and Gas Academy, (1992).

[16] Deepak Geetha Viswanathan. *Features from Accelerated Segment Test.* (2011).

[17] Stefan Leutenegger, Margarita Chli and Roland Y. Siegwart. *BRISK: Binary Robust Invariant Scalable Keypoints.* Autonomous Systems Lab. (2013).

[18] Georg Nebehay, Roman Pflugfelder. *Consensus-based Matching and Tracking of Keypoints for Object Tracking.* CCVPR, 2015.

[19] Grevelink Evelyn. *A Closer Look at Object Detection, Recognition and Tracking.* Intel (Diciembre 18, 2017).

[20] Edward Rosten and Tom Drummond. *Machine learning for high speed corner detection.* 9th European Conference on Computer Vision, vol. 1, 2006.

[21] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. *BRIEF: Binary Robust Independent Elementary Features.* 11th European Conference on Computer Vision (ECCV), Heraklion, Crete. LNCS Springer. (Septiembre, 2010).

[22] Solarian Programmer. (2008) `https://solarianprogrammer.com/2015/05/08/detect-red-circles-image-using-opencv/`. 25 de Octubre de 2018.

[23] Morphological Operations. (2003) `http://homepages.inf.ed.ac.uk/rbf/HIPR2/morops.htm`. 25 de Octubre de 2018.

[24] Flujo Óptico. (2018) `https://docs.opencv.org`. 25 de Octubre de 2018.

[25] R. Xu, D. Wunsh. *Survey of clustering algorithms.* TNN, 16(3), 2005.

[26] ROS. (2018) `http://wiki.ros.org/ROS/Introduction`. 17 de Noviembre de 2018.

[27] Hagit Hel-Or. (2018) `http://cs.haifa.ac.il/hagit/courses/ip/Lectures/`. 25 de Octubre de 2018.

[28] Weisstein, Eric W. Rev. *Numerical Differentiation MathWorld.* Wolfram Research.

[29] Chawda Vinay, Celik Ozkan, O'Malley Marcia K. *Application of Levant's Differentiator for Velocity Estimation and Increased Z-Width in Haptic Interfaces.* IEEE World Haptics Conference 2011.

[30] Martinez Fuentes Carlos Arturo. *Diferenciador de Ganancias Variables Aplicado a un motor de DC.* Universidad Nacional Autónoma de México (2014).

[31] Dhara Patel, Saurabh Upadhyay *Optical Flow Measurement using Lucas Kanade Method.* International Journal of Computer Applications(0975-8887). Enero,2013.

[32] OpenCV tutorial c++. (2010) https://www.opencv-srf.com/2010/09/object-detection-using-color-seperation.html. 24 de Octubre de 2018.