



ESPE
UNIVERSIDAD DE LAS FUERZAS ARMADAS
INNOVACIÓN PARA LA EXCELENCIA

**Reconocimiento y seguimiento de plataformas para el aterrizaje automático de un
vehículo aéreo no tripulado basado en inteligencia artificial y odometría visual**

Chasillacta Canencia, Milton Nicolay

Departamento de Eléctrica, Electrónica y Telecomunicaciones

Carrera de Ingeniería en Electrónica, Automatización y Control

Trabajo de titulación, previo a la obtención del título de ingeniero en Electrónica,
Automatización y Control

PhD. Aguilar Castillo, Wilbert Geovanny

20 de agosto del 2020

Urkund Analysis Result

Analysed Document: Chasillacta_Milton_TESIS_Urkund.pdf (D78216144)
Submitted: 8/27/2020 4:33:00 AM
Submitted By: mnchasillacta@espe.edu.ec
Significance: 3 %

Sources included in the report:

TrabajoTitulaciónVinicioSalcedo.pdf (D35374643)
tesis_calderon_merizalde.pdf (D75308789)
Tesis_Camino_Rojas_V_12_V1.pdf (D60824540)
Tesis-Alvarez.pdf (D44942716)
tesis_grijalva_santiago.pdf (D57099779)
TESIS_CHAUCA_BRYAN_urkund.pdf (D78121896)
<https://docplayer.es/92890108-La-version-digital-de-esta-tesis-esta-protegida-por-la-ley-de-derechos-de-autor-del-ecuador.html>
https://www.researchgate.net/profile/Wilbert_Aguilar/publication/265842947_Compensacion_de_los_efectos_generados_en_la_imagen_por_el_control_de_navegacion_del_robot_Aibo_ERS_7/links/541c7e100cf2218008c9df84.pdf?origin=publication_detail

Instances where selected sources appear:

33



Dr. Wilbert G. Aguilar
Director de Tesis



DEPARTAMENTO DE ELÉCTRICA, ELECTRÓNICA Y TELECOMUNICACIONES
CARRERA DE INGENIERÍA EN ELECTRÓNICA, AUTOMATIZACIÓN Y CONTROL

CERTIFICACIÓN

Certifico que el trabajo de titulación “**Reconocimiento y seguimiento de plataformas para el aterrizaje automático de un vehículo aéreo no tripulado basado en inteligencia artificial y odometría visual**” fue realizado por el señor **Chasillacta Canencia, Milton Nicolay**, el cual ha sido revisado y analizado en su totalidad por la herramienta de verificación de similitud de contenido; por lo tanto cumple con los requisitos legales, teóricos, científicos, técnicos y metodológicos establecidos por la Universidad de las Fuerzas Armadas ESPE, razón por la cual me permito autorizar para que lo sustente públicamente.

Sangolquí, 27 de agosto de 2020

Firma:



PhD. Aguilar Castillo, Wilbert Geovanny

C.C: 0703844696



ESPE
UNIVERSIDAD DE LAS FUERZAS ARMADAS
INNOVACIÓN PARA LA EXCELENCIA

DEPARTAMENTO DE ELÉCTRICA, ELECTRÓNICA Y TELECOMUNICACIONES
CARRERA DE INGENIERÍA EN ELECTRÓNICA, AUTOMATIZACIÓN Y CONTROL

RESPONSABILIDAD DE AUTORÍA

Yo, Chasillacta Canencia, Milton Nicolay con CI: 171900729-4, declaro que el contenido, ideas y criterios del trabajo de titulación: **"Reconocimiento y seguimiento de plataformas para el aterrizaje automático de un vehículo aéreo no tripulado basado en inteligencia artificial y odometría visual"** es de mi autoría y responsabilidad, cumpliendo con los requisitos legales, teóricos, científicos, técnicos y metodológicos establecidos por la Universidad de las Fuerzas Armadas ESPE, respetando los derechos intelectuales de terceros y referenciando las citas bibliográficas.

Sangolquí, 28 de agosto de 2020

Firma:

Milton Nicolay Chasillacta Canencia

C.C: 171900729-4



DEPARTAMENTO DE ELÉCTRICA, ELECTRÓNICA Y TELECOMUNICACIONES
CARRERA DE INGENIERÍA EN ELECTRÓNICA, AUTOMATIZACIÓN Y CONTROL

AUTORIZACIÓN DE PUBLICACIÓN

Yo, Chasillacta Canencia, Milton Nicolay con CI: 171900729-4, autorizo a la Universidad de las Fuerzas Armadas ESPE publicar el trabajo de titulación: **“Reconocimiento y seguimiento de plataformas para el aterrizaje automático de un vehículo aéreo no tripulado basado en inteligencia artificial y odometría visual”** en el Repositorio Institucional, cuyo contenido, ideas y criterios son de mi responsabilidad.

Sangolquí, 28 de agosto de 2020

Firma:

Milton Nicolay Chasillacta Canencia

C.C: 171900729-4

Dedicatoria

*A mi madre Nelly y a mi padre Milton,
sin su esfuerzo nada de esto sería posible.*

*A mi hermano Diego,
por todo permanente ayuda y la paciencia que me brinda.*

*A mi familia y amigos,
que son parte de mi vida.*

*Por último, al trabajo duro y honesto,
capaz de conquistar colosales metas.*

Milton Nicolay Chasillacta Canencia

Agradecimiento

Agradezco enormemente a mis padres por su ayuda constante, por el apoyo que nunca me ha faltado, por permanecer siempre a mi lado, por la confianza que me tienen, por nunca dejar que me rinda y siempre incentivarme a dar todo de mí.

Agradezco de manera muy especial a mi hermano que constantemente me ha brindado su ayuda desinteresada sin importar el día o la hora. Por todas las experiencias vividas que han puesto en evidencia todo el inmenso amor que nos tenemos.

Agradezco a Dios por la oportunidad de estudiar en esta maravillosa universidad, por todas las ocasiones que me brindó la fuerza de seguir adelante, de nunca desmayar ante los problemas que aparentaban eran insuperables.

Agradezco a la Universidad de las Fuerzas Armadas ESPE y a los profesores con una verdadera vocación por la enseñanza, por todo el esfuerzo hecho para compartir sus conocimientos. De manera especial a mi tutor de tesis el Doctor Wilbert por toda la ayuda, consejos y motivación de participar cada vez más en el mundo de la investigación.

Agradezco a toda mi familia, siempre presta a obsequiarme su colaboración. A mis amigos, con los que hemos podido superar toda clase de desafíos, especialmente a Paolo y Bryan.

Milton Nicolay Chasillacta Canencia

Índice de Contenidos

Urkund Report.....	2
Certificación	3
Responsabilidad de Autoría	4
Autorización de Publicación	5
Dedicatoria	6
Agradecimiento	7
Índice de Contenidos	8
Índice de Tablas.....	14
Índice de Figuras	16
Resumen.....	20
Abstract	21
Capítulo I.....	22
Descripción del proyecto de investigación	22
Antecedentes	22
Justificación e importancia.....	26
Alcance del proyecto	28
Objetivos	30
Objetivo general	30
Objetivos específicos.....	30

	9
Capitulo II	32
Marco teórico	32
Vehículos aéreos no tripulados (<i>UAV</i>)	32
Dinámica del movimiento de un cuadricóptero	33
Control de AUVs	36
Control PID.....	36
Control Difuso	38
Inteligencia Artificial (IA)	39
Machine learning.....	41
Redes Neuronales	42
Algoritmos de seguimiento visual	42
Odometría visual.....	46
Estimación de movimiento.....	47
Formación de imágenes	48
Caracterización de imágenes	49
Características globales	50
Características locales.....	51
Puntos de interés	52
Detección de puntos de interés	52
Descripción de puntos de interés	53
Matching de puntos de interés	58

	10
Transformada geométrica.....	60
Transformada de traslación	62
Transformación afín	63
Transformada de similitud	64
Transformada de perspectiva.....	65
Capítulo III.....	66
Generalidades del sistema	66
Hardware del sistema	66
Estación en tierra	66
Estación aérea (<i>UAV</i>).....	67
Especificaciones generales	68
Conectividad con el dron	69
Características de la cámara.....	70
Características de movimiento	72
Plataforma móvil terrestre.....	73
Target en la plataforma móvil de aterrizaje	75
Software del sistema.....	76
Robot Operating System (<i>ROS</i>).....	77
Conceptos <i>ROS</i>	78
Comunicación <i>ROS</i>	81
Comunicación entre <i>ROS</i> y <i>Bebop 2</i>	82

	11
Publicadores de clase.....	85
Suscriptores de clase	88
Capítulo IV	90
Detección de la plataforma de aterrizaje.....	90
Sistema de detección <i>YOLO</i>	90
Comparación de <i>YOLOv3</i> con otros detectores	92
Predicción del cuadro delimitador de <i>YOLOv3</i>	97
Vista del target desde el <i>UAV</i>	99
Detección del target con <i>YOLO</i>	101
¿Cómo se usa <i>YOLO</i> dentro del algoritmo?	102
Primer proceso de detección	106
Segundo proceso de detección	107
Capítulo V	110
Seguimiento y aterrizaje automáticos del <i>UAV</i>	110
Descripción del algoritmo global de aterrizaje	110
Estimación del movimiento del <i>UAV</i>	117
Modelamiento servo visual	119
Detección y descripción de puntos de interés.....	119
Obtención del modelo del movimiento acumulado	122
Obtención de la función de transferencias para el eje <i>X</i>	128
Obtención de la función de transferencias para el eje <i>Y</i>	130

	12
Seguimiento de la plataforma móvil terrestre.....	132
Diseño del Controlador PID	134
Controlador PID para el eje <i>X</i> (Pitch).....	134
Controlador PID para el eje <i>Y</i> (Roll).....	136
Diseño del Controlador Fuzzy	137
Variable lingüística del error de posición en el eje <i>Y</i> (Roll).....	138
Variable lingüística de la velocidad del <i>UAV</i> en eje <i>Y</i> (Roll).....	140
Reglas de control para el seguimiento en el eje <i>Y</i> (Roll).....	143
Variable lingüística del error de posición en el eje <i>X</i> (Pitch)	145
Variable lingüística de la velocidad del <i>UAV</i> en eje <i>X</i> (Pitch)....	147
Reglas de control para el seguimiento en el eje <i>X</i> (Pitch).....	149
Aterrizaje automático del <i>UAV</i> sobre la plataforma móvil.....	152
Condiciones para el descenso final.....	152
Condición 1: Por el ancho y alto del target	154
Condición 2: Por la no detección del target	155
Condición 3: Por la altura del dron e	156
Capítulo VI	157
Pruebas y resultados	157
Posicionamiento y seguimiento de objetos detectados	157
Seguimiento del target respecto a la altura del <i>UAV</i>	161
Sistema de control para el seguimiento de la plataforma.....	163

	13
Prueba del control de seguimiento en el eje <i>X</i> positivo	164
Prueba del control de seguimiento en el eje <i>X</i> negativo.....	166
Prueba del control de seguimiento en el eje <i>Y</i> positivo	168
Prueba del control de seguimiento en el eje <i>Y</i> negativo.....	170
Tiempo y posición final del <i>UAV</i> sobre la plataforma de aterrizaje	171
Prueba del algoritmo completo.....	179
Capítulo VII	182
Conclusiones y recomendaciones.....	182
Conclusiones.....	182
Recomendaciones	183
REFERENCIAS BIBLIOGRÁFICAS.....	185

Índice de Tablas

Tabla 1 <i>Configuración de la velocidad de los rotores de un cuadricóptero</i>	35
Tabla 2 <i>Descripción de los recursos de la estación en tierra</i>	66
Tabla 3 <i>Especificaciones generales del drone Bebop 2</i>	68
Tabla 4 <i>Comunicación inalámbrica del UAV</i>	70
Tabla 5 <i>Características de la cámara del Bebop 2</i>	71
Tabla 6 <i>Descripción de Parámetros configurables para el Bebop 2</i>	84
Tabla 7 <i>Descripción de los publicadores utilizados en el algoritmo de control</i>	86
Tabla 8 <i>Descripción de los suscriptores utilizados para algoritmo de control</i>	88
Tabla 9 <i>Banco de clases de COCO</i>	95
Tabla 10 <i>Datos de salida de YOLO para dos detecciones de diferentes clases</i>	105
Tabla 11 <i>Información de detección del vehículo</i>	106
Tabla 12 <i>Información de salida para la detección del nuevo target</i>	109
Tabla 13 <i>Movimiento en los ejes X, Y y Z vinculados a acciones de control</i>	118
Tabla 14 <i>Condiciones de vuelo y video del UAV para su modelamiento</i>	124
Tabla 15 <i>Parámetros para la función de estimación de movimiento en el eje X</i>	128
Tabla 16 <i>Parámetros para la función de estimación de movimiento en el eje Y</i>	130
Tabla 17 <i>Definición de la variable lingüística del error de posición en el eje Y</i>	139
Tabla 18 <i>Definición de la variable lingüística de la velocidad del UAV en el eje Y</i>	141
Tabla 19 <i>Reglas de control para el seguimiento del target en el eje Y (Roll)</i>	143
Tabla 20 <i>Definición de la variable lingüística del error de posición en el eje X</i>	146
Tabla 21 <i>Definición de la variable lingüística de la velocidad del UAV en el eje X</i>	148
Tabla 22 <i>Reglas de control para el seguimiento del target en el eje X (Pitch)</i>	149
Tabla 23 <i>Tiempos de detección y descenso con el controlador PID</i>	172

Tabla 24 <i>Tiempos de detección y descenso con el controlador Difuso.....</i>	173
Tabla 25 <i>Error de posición del UAV sobre la plataforma con controlador PID.....</i>	175
Tabla 26 <i>Error de posición del UAV sobre la plataforma con controlador difuso</i>	177

Índice de Figuras

Figura 1 <i>Esquema general de reconocimiento y aterrizaje automático del UAV</i>	29
Figura 2 <i>Ejes de referencia inerciales y corporales de un cuadricóptero</i>	33
Figura 3 <i>Diagrama para el controlador PID</i>	37
Figura 4 <i>Esquema del controlador de lógica difusa</i>	39
Figura 5 <i>Comparación entre algoritmos de seguimiento</i>	46
Figura 6 <i>Modelo Pin-Hole para la formación de imágenes</i>	49
Figura 7 <i>Características de una imagen</i>	51
Figura 8 <i>Representación esquemática del descriptor SIFT</i>	54
Figura 9 <i>Aproximación de derivadas gaussianas de segundo orden</i>	56
Figura 10 <i>Secuencia piramidal de imágenes escalas por el método ORB</i>	58
Figura 11 <i>Matching entre dos imágenes</i>	59
Figura 12 <i>Imágenes modificadas resultantes por el cambio del dominio</i>	61
Figura 13 <i>Imagen distorsionada de un piso de cerámica tras una transformación</i>	62
Figura 14 <i>Aplicación de una transformada geométrica afin</i>	64
Figura 15 <i>UAV Bebop 2 de la marca Parrot</i>	67
Figura 16 <i>Dimensiones de drone Bebop 2 de Parrot</i>	69
Figura 17 <i>Rango de movilidad digital de la cámara del drone Bebop 2</i>	71
Figura 18 <i>Movimiento lineal y angular del Bebop 2</i>	72
Figura 19 <i>Área de visión de la escena respecto a la altura</i>	73
Figura 20 <i>Vista cenital del vehículo a diferentes alturas</i>	74
Figura 21 <i>Vista cenital del nuevo target a diferentes alturas</i>	75
Figura 22 <i>Esquema de comunicación entre la PC y la estación aérea</i>	76
Figura 23 <i>Representación del flujo de información en ROS</i>	77
Figura 24 <i>Comunicación entre Bebop 2 y la estación en tierra</i>	80

Figura 25 <i>Esquema interno del t3pico para el control de velocidad</i>	81
Figura 26 <i>Directorio del archivo ejecutable del controlador bebop_autonomy</i>	83
Figura 27 <i>Conversi3n entre mensaje ROS a im3genes OpenCV</i>	89
Figura 28 <i>Sistema de detecci3n de YOLO</i>	91
Figura 29 <i>Comparaci3n de YOLO con otros detectores</i>	92
Figura 30 <i>Intersecci3n entre Verdad Fundamental y la predicci3n</i>	94
Figura 31 <i>Creaci3n de Bounding boxes</i>	98
Figura 32 <i>3rea de visi3n de la escena respecto a la altura del UAV</i>	100
Figura 33 <i>Visi3n de la plataforma de aterrizaje respecto a su altura</i>	101
Figura 34 <i>Detecci3n de objetos con YOLO</i>	102
Figura 35 <i>Posici3n de p3xeles dentro de la imagen</i>	103
Figura 36 <i>Cuadros delimitadores entregados por YOLO</i>	104
Figura 37 <i>Detecci3n del veh3culo real con YOLO</i>	105
Figura 38 <i>Vista cenital de la primea y segunda detecci3n a diferentes alturas</i>	107
Figura 39 <i>Detecci3n real con YOLO del nuevo target</i>	109
Figura 40 <i>Diagrama de flujo del algoritmo global de aterrizaje autom3tico</i>	111
Figura 41 <i>Despegue y posicionamiento virtual de la c3mara del UAV</i>	112
Figura 42 <i>Detecciones seguidas del veh3culo con YOLO</i>	113
Figura 43 <i>Cuadro de seguimiento del objeto de inter3s con algoritmo KCF</i>	114
Figura 44 <i>B3squeda del nuevo target</i>	115
Figura 45 <i>Cuadro de seguimiento del nuevo target de inter3s con algoritmo KCF</i>	115
Figura 46 <i>Cuadro de detecci3n y seguimiento del objeto antes del aterrizaje</i>	116
Figura 47 <i>Cantidad de puntos de inter3s detectados</i>	120
Figura 48 <i>Relaci3n detecci3n y tiempo de puntos de inter3s</i>	121
Figura 49 <i>Tren de pulsos para las secuencias de vuelo del UAV</i>	123

Figura 50 <i>Estimación de movimiento acumulado en el eje X</i>	127
Figura 51 <i>Estimación de movimiento acumulado en el eje Y</i>	128
Figura 52 <i>Modelo estimado con el medido para el eje X</i>	130
Figura 53 <i>Modelo estimado con el medido para el eje Y</i>	132
Figura 54 <i>Lazo de control para los desplazamientos en Roll y Pitch del UAV</i>	133
Figura 55 <i>Respuesta del controlador al escalón unitario del movimiento en el eje X</i> ..	135
Figura 56 <i>Respuesta del controlador al escalón unitario del movimiento en el eje Y</i> ..	136
Figura 57 <i>Variables de entradas y salida para el control de posición en roll</i>	137
Figura 58 <i>Variables de entradas y salida para el control de posición en pitch</i>	138
Figura 59 <i>Definición de la variable lingüística para el error de posición en el eje Y</i>	140
Figura 60 <i>Definición de la variable lingüística para la velocidad del drone en el eje Y</i>	142
Figura 61 <i>Posición del target respecto al movimiento lateral del UAV</i>	144
Figura 62 <i>Curva de control para el seguimiento en el eje Y</i>	145
Figura 63 <i>Definición de la variable lingüística para el error de posición en el eje X</i>	147
Figura 64 <i>Definición de la variable lingüística para el error de posición en el eje X</i>	149
Figura 65 <i>Posición del target respecto al movimiento frontal y trasero del UAV</i>	150
Figura 66 <i>Curva de control para el seguimiento en el eje X</i>	151
Figura 67 <i>Estado 5 del diagrama de flujo general del aterrizaje automático</i>	153
Figura 68 <i>Alto y ancho del cuadro delimitador de la detección de objetos</i>	154
Figura 69 <i>Condición para finalizar el aterrizaje con la no identificación del target</i>	155
Figura 70 <i>Detección de objetos en imágenes del UAV</i>	158
Figura 71 <i>Cuadro delimitador del algoritmo de seguimiento KCF con imagen inicial</i> ..	159
Figura 72 <i>Algoritmo de seguimiento KCF con imagen desplazada</i>	160
Figura 73 <i>Detección y señalamiento del objeto para el seguimiento</i>	161

Figura 74 <i>Error de escala con KCF</i>	162
Figura 75 <i>Comportamiento de YOLO a diferentes escalas</i>	163
Figura 76 <i>Distancia entre los centros de la imagen y del cuadro del target</i>	164
Figura 77 <i>Desplazamiento hacia delante del UAV</i>	165
Figura 78 <i>Error de posición en eje X con perturbación del UAV hacia delante</i>	166
Figura 79 <i>Desplazamiento hacia atrás del UAV</i>	167
Figura 80 <i>Error de posición en eje X con perturbación del UAV hacia atrás</i>	167
Figura 81 <i>Desplazamiento hacia la izquierda del UAV</i>	168
Figura 82 <i>Error de posición en eje Y con perturbación del UAV hacia la izquierda</i>	169
Figura 83 <i>Desplazamiento hacia la derecha del UAV</i>	170
Figura 84 <i>Error de posición en eje Y con perturbación del UAV hacia la derecha</i>	171
Figura 85 <i>Tiempo total de aterrizaje entre el controlador PID y Fuzzy</i>	173
Figura 86 <i>Tiempo de descenso entre el controlador PID y Fuzzy</i>	174
Figura 87 <i>Error de posición del UAV sobre la plataforma móvil al aterrizar</i>	179
Figura 88 <i>Funcionamiento del algoritmo de aterrizaje con el controlador PID</i>	180

Resumen

Esta investigación propone la creación e implementación de un sistema de detección y seguimiento de plataformas móviles aptas para un aterrizaje automático de un *UAV*. El proyecto consta de PC portátil que mantiene una comunicación inalámbrica *Wi-Fi* con el *UAV* y es la encargada del procesar el streaming de video usando visión por computadora. El primer paso es la detección de la plataforma de aterrizaje usando funciones del sistema de detección de objetos llamado *YOLO (You Only Look Once)*, los datos de posicionamiento del objeto son usados en el algoritmo *KCF (Kernelized Correlation Filter)* de seguimiento, permitiendo el cálculo del error en el posicionamiento del *UAV* en función de la plataforma en cada fotograma. Para el control de desplazamiento en el eje *X* e *Y* se diseñan un controlador Difuso y un PID, el modelo dinámico del *UAV* se obtiene estimando su desplazamiento entre fotogramas sucesivos usando transformaciones geométricas que determinan las características locales de los fotogramas. La implementación de los controladores produce el control de seguimiento y gradual descenso automáticos del *UAV* hacia la plataforma de aterrizaje en movimiento. Las pruebas se enfocan en la evaluación del sistema de detección y comparación de parámetros de tiempo y posición final tras el aterrizaje al implementar ambos controladores.

PALABRAS CLAVE:

- **DETECCIÓN DE OBJETOS**
- **TRANSFORMACIÓN GEOMÉTRICA**
- **ESTIMACIÓN DE MOVIMIENTO**
- **SEGUIMIENTO Y ATERRIZAJES AUTOMÁTICOS**
- **ERROR DE POSICIÓN**

Abstract

This research proposes the creation and implementation of a detection and tracking system for mobile platforms suitable for an automatic landing of a *UAV*. The project consists of a portable PC that maintains *Wi-Fi* wireless communication with the *UAV* and is in charge of processing the video streaming using computer vision. The first step is the detection of the landing pad using functions of the object detection system called *YOLO* (*You Only Look Once*), the object positioning data is used in the tracking algorithm *KCF* (*Kernelized Correlation Filter*), allowing the calculation of the error in the positioning of the *UAV* depending on the platform in each frame. For the displacement control in the X and Y axis, a Diffuse controller and a PID are designed, the dynamic model of the *UAV* is obtained by estimating its displacement between successive frames using geometric transformations that determine the local characteristics of the frames. The implementation of the controllers produces the automatic tracking and descent control of the *UAV* towards the moving landing pad. The tests are focused on the evaluation of the detection system and comparison of parameters of time and final position after landing by implementing both controllers.

KEYWORDS:

- **OBJECT DETECTION**
- **GEOMETRIC TRANSFORMATION**
- **MOTION ESTIMATION**
- **AUTOMATIC TRACKING AND LANDING**
- **POSITION ERROR**

Capítulo I

Descripción del proyecto de investigación

Antecedentes

En las últimas décadas, el campo de los Vehículos Aéreos No Tripulados *UAV* (del inglés *Unmanned Aerial Vehicle*) ha logrado un notable desarrollo en tecnologías de autonomía lo que ha provocado un número creciente de aplicaciones, donde la investigación militar (Andrea, Byron, Jorge, Inti, & Aguilar, 2018), (Pardo, Aguilar, & Toulkeridis, 2017), (Jara-Olmedo A. , Medina-Pazmiño, Tozer, Aguilar, & Pardo, 2018), (Jara-Olmedo A. , et al., 2018), (Orbea, et al., Vertical take off and landing with fixed rotor, 2017), (Amaguaña, Collaguazo, Tituaña, & Aguilar, 2018) se ha destacado (Kendoul, 2012). En la actualidad, existe un mercado significativo en distintos campos como fumigación, vigilancia, agricultura de precisión, transporte de suministros médicos, operaciones de búsqueda y rescate, monitoreo de infraestructura de difícil acceso, entre otros. Lo que ha generado considerables ahorros de costos (Otto, Agatz, Campbell, Golden, & Pesch, 2018). También se han realizado investigaciones para el desarrollo de soluciones frente a desastres naturales e inspecciones periódicas de infraestructuras. En (Murphy, et al., 2008) utilizaron conjuntamente vehículos no tripulados de superficie marítima y micro aéreos después del huracán Wilma en 2008, en (Campoy, et al., 2008) discutieron las aplicaciones en el campo de visión por computador relacionadas con tareas civiles, en las cuales se pueden utilizar *UAV*.

El guiado, la navegación y el control (GNC) son los campos de investigación de interés sobre *UAV* ampliamente presente en la literatura desde la década de los noventa (Kendoul, 2012). Se han realizado varias investigaciones sobre algoritmos para el control automático de *UAV* utilizando teorías de control modernas (Belkacem & Ghazzawi, 2011).

Los controladores PID han sido integrados con éxito en el control en tiempo real y en sistemas de navegación para *UAV* debido a su estructura simple, fácil implementación y adecuado desempeño. Sin embargo, para lograr una operación con mejor performance es necesario un procedimiento de ajuste para obtener valores óptimos de los parámetros del controlador, considerado como un desafío crítico (Turkoglu, Ozdemir, Nikbay, & Jafarov, 2008) y (Turkoglu & Jafarov, 2006). Con un modelo matemático preciso se lograría un rendimiento óptimo, pero existen incertidumbres que afectan al sistema como alteraciones aerodinámicas, cambios en el entorno o ruido en los sensores y actuadores (Sarabakha, Fu, Kayacan, & Kumbasar, 2017). Una alternativa es el uso de controladores inteligentes, usando lógica difusa (Kayacan & Maslim, 2017), (Cervantes & Castillo, 2015) que han proporcionado soluciones robustas y que no necesitan de un modelo matemático preciso del sistema (Kayacan & Maslim, 2017). El controlador basado en lógica difusa tipo 1 (T1-FLC) es uno de los más conocidos y utilizados (Sarabakha, Fu, Kayacan, & Kumbasar, 2017).

En (Clough, 2002) y (Merz, Duranti, & Conte, 2004) se ha establecido una distinción entre sistemas automáticos, autónomos e inteligentes. Un sistema automático hará exactamente lo programado y no tiene capacidad de razonamiento o planificación. Un sistema autónomo cuenta con la habilidad de elegir su solución y planificar sus tareas culminando la misión asignada. Y un sistema inteligente tiene las capacidades de un sistema autónomo además de la posibilidad de generar sus propios objetivos sin ninguna instrucción o influencia externa.

El diseño de *UAVs* con niveles más altos de autonomía requieren un control de vuelo y un sistema de navegación para la abstracción del ambiente y conciencia situacional. La detección es el primer componente en la navegación que involucra un

dispositivo sensorial real , como una *IMU* (del inglés *Inertial Measurement Unit*), una cámara o un *GPS* (del inglés *Global Positioning System*) para proporcionar al sistema de control de vuelo información de altitud y posición (Aguilar W. G., Morales, Ruiz, & Abad, 2017), (Aguilar & Morales, 3D Environment Mapping Using the Kinect V2 and Path Planning Based on RRT Algorithms, 2016). La navegación basada en *GPS* está limitada al acceso de señales de satélites, creando una necesidad de sistemas de respaldo en instantes donde el *GPS* no esté disponible, y cumplir con las exigencias de precisión en aplicaciones civiles (Kendoul, 2012). La visión artificial se puede utilizar como parte del sistema de estimación de estado para el control de vuelo o en sistemas de percepción para la detección de obstáculos y objetivos. Se mencionan seis clases principales de sistemas de estimación de estado basados en visión (Kendoul, 2012).

- Visión en tierra (OGV)
- Odometría visual (VO)
- Navegación relativa al objetivo (TGRN)
- Navegación relativa del terreno (TRN)
- Estimación concurrente de movimiento y estructura (SFM y SLAM)
- Navegación de flujo óptico bio-inspirada (BIOFN).

Las técnicas clásicas de Odometría Visual o en inglés *Visual Odometry* utilizan datos del movimiento de los actuadores para estimar la posición y orientación relativa de un vehículo analizando una secuencia de imágenes adquiridas por un sistema de visión a bordo del mismo (Andrea, Byron, Jorge, Inti, & Aguilar, 2018), (Pardo, Aguilar, & Toulkeridis, 2017), (Jara-Olmedo A. , Medina-Pazmiño, Tozer, Aguilar, & Pardo, 2018), (Jara-Olmedo A. , et al., 2018), (Orbea, et al., Vertical take off and landing with fixed rotor, 2017), (Amaguaña, Collaguazo, Tituaña, & Aguilar, 2018), (Kendoul, 2012) y (Strydom,

Thurrowgood, & Srinivasan, 2014). La Odometría basada en visión monocular trabaja con una sola cámara, lo que representa un reto debido a la ambigüedad del factor de escala dificultando conocer la profundidad de las imágenes (Kendoul, 2012). En (Nourani-Vatani & Koerich Borges, 2011) presentan un esquema de VO para vehículos terrestres que mide el flujo óptico en una secuencia de imágenes obtenidas de una cámara con la mira hacia abajo. Estima el movimiento utilizando un algoritmo de coincidencia de puntos a medida que la imagen cambia de cuadro a cuadro demostrando que el posicionamiento de la cámara a una altura fija sobre el suelo simplifica el problema de cálculo de odometría para vehículos terrestres. En (Maimone, Cheng, & Matthies, 2007) se determinó que los Mars Exploration Rovers luego de un recorrido sobre rocas grandes, pendientes pronunciadas o terreno arenoso necesitaban de odometría visual para corregir los errores que surgen de movimientos bruscos en 3D y deslizamiento de ruedas. Investigadores de Draper Laboratory y MIT han desarrollado un sistema de navegación asistida por visión para UAV utilizando imágenes monoculares (Madison, Andrews, DeBitetto, Rasmussen, & Bottkol, 2007). El sistema permite la estimación de la posición absoluta y velocidad del UAV cuando pierde la señal GPS. Esto se ha logrado al estimar las posiciones 3D de los puntos de referencia cuando el GPS está disponible, y luego usarlos junto con las mediciones de la línea de visión de estos puntos para continuar la estimación de trayectorias del vehículo después de perder tanto la señal GPS como los datos de las características que se localizaron mediante el mismo.

Existe un estudio del aterrizaje basado en visión de un objetivo, aunque no se han realizado pruebas automáticas en exteriores debido a la incertidumbre del viento (Lee, Jung, & Shim, 2016) y (Ling, 2014). Algunas investigaciones enfocadas en el aterrizaje automático que utilizan visión por computadora para pruebas en interiores han utilizado

un *IPS* (del inglés *Indoor Positioning System*) (Jung, Lee, & Bang, 2015) y (Gomes Carreira, 2013). En (Kim, Jung, Lee, & Shim, 2014) habla del aterrizaje automático en un vehículo en movimiento mediante geolocalización estimada del objetivo con un procesamiento de imágenes realizado en un PC en tierra y no en el *UAV*. El trabajo presentado en (Saripalli & Sukhatme, 2003) combina la visión por computadora y la señal *GPS* logrando un descenso automático sobre un objetivo conocido. Es decir, no se considera un sistema de aterrizaje basado únicamente en visión, sino un sistema de percepción que trabaja con visión por computadora. Los primeros resultados experimentales sobre el aterrizaje automático con este método (sin ayuda de *GPS*) de un *UAV* fueron publicados en (Merz, Duranti, & Conte, 2004). Su sistema de aterrizaje basado en visión utilizó una sola cámara con giro e inclinación y una plataforma blanca con círculos negros.

Justificación e importancia

Recientes proyectos desarrollados en el Centro de Investigación de Aplicaciones Militares CICTE han enfocado su investigación para la creación de sistemas de control para Vehículos Aéreos No Tripulados *UAV* para diversas operaciones militares. La finalidad de las investigaciones en los vehículos es proveer al personal militar de instrumentos de ingeniería para mitigar su exposición a distintos peligros.

La automatización de *UAVs* ha evolucionado en una de las áreas de estudio más desarrollados por el abaratamiento de sus costos añadiendo su fácil despliegue si lo relacionamos con aviones tripulados (Ling, 2014). A pesar del continuo progreso en su automatismo, aún existen algunas limitaciones en este campo (King, 2017). Los *UAVs* dependen de la observación del operador o del *GPS*, además de otros sensores a bordo como soporte en los sistemas de control de posicionamiento y navegación. La restricción

de una buena señal *GPS* que limita el uso amplio y confiable, es un desafío que los investigadores están abordando para eliminar la dependencia de señales para la comunicación con el piloto o teleoperador. El uso de la odometría visual es la alternativa que presenta este trabajo para estimar la posición y orientación relativa del dron, analizando una secuencia e imágenes adquiridas por el sistema de visión a bordo.

Una de las temáticas más innovadoras en los sistemas de navegación autónomos es el aterrizaje automático cuando existe fallas de comunicación con el *GPS* (Vázquez Ruano, 2017). Esta investigación desarrolló un sistema de detección de plataformas terrestres móviles junto a un algoritmo de seguimiento para el aterrizaje automático basado en inteligencia artificial y odometría visual proporcionando una solución a esta problemática. En (Salcedo Peña, 2018) se elaboró un algoritmo para el reconocimiento de una plataforma de aterrizaje utilizando una imagen específica sobre el target. Esta investigación busca generalizar el proceso de reconocimiento a distintos objetos. Además, realizará una comparativa del desempeño en el aterrizaje automático con la implementación de dos tipos de controladores, un PID y un difuso.

CICTE ha desarrollado varios proyectos utilizando *UAVs*, como el “Sistema de aterrizaje automático sobre plataformas en movimiento para micro vehículos aéreos no tripulados de ala rotativa basado en estimación de movimiento de puntos de interés”. Al trabajar en líneas de investigación semejantes, con un enfoque en el aterrizaje automático, el presente proyecto de investigación impulsará el desarrollo de este tipo de proyectos, aportando nuevas herramientas como el reconocimiento y seguimiento de plataformas utilizando inteligencia artificial y la estimación de posición y orientación usando odometría visual, y así alcanzar un prototipo funcional de mejores condiciones y capacidades operativas. Los algoritmos de estas investigaciones serán complementos a

proyectos spin off como “Smart Drone” aprobado por el consejo de carrera de la Universidad de las Fuerzas Armadas ESPE. Este trabajo de investigación también impulsará el diseño de nuevos prototipos militares en busca de la soberanía y paz del país, como el proyecto “*Sistemas inteligentes de monitoreo y aterrizaje para UAV tácticos y mutirotores Fase 1 - SmartDrone1*” desarrollado por CICTE y CIDFAE.

Alcance del proyecto

En esta investigación se plantea desarrollar un algoritmo para el reconocimiento y seguimiento de plataformas aptas para el aterrizaje automático de micro vehículos aéreos no tripulados multirotor. La plataforma de aterrizaje será la parte superior de un automóvil en movimiento que seguirá una trayectoria recta entre dos puntos A y B a una velocidad que no supere los 5 km/h que será monitoreada con el velocímetro del vehículo o utilizando una aplicación móvil. El seguimiento y aterrizaje del dron estarán limitados a esta velocidad para precautelar la seguridad del mismo. Además, se respetará las restricciones de la guía de vuelo del *UAV* como: no volar en condiciones meteorológicas adversas (lluvia, nieve, niebla, vientos fuertes ni durante la noche). El proyecto será desarrollado en dos etapas:

- **Reconocimiento del objeto de interés**

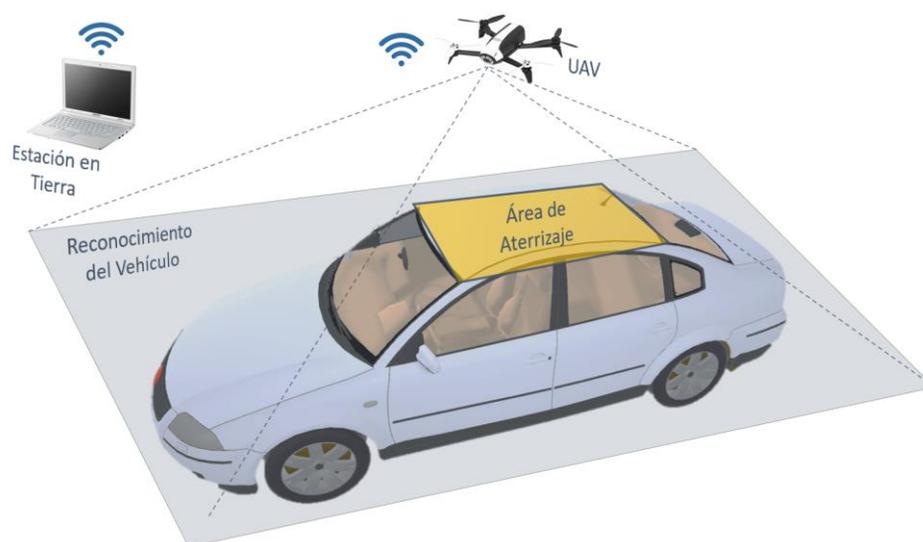
El primer paso es analizar la altura adecuada para obtener una vista útil de los objetos de interés dentro de la escena capturada en los fotogramas de la cámara monocular del *UAV*, luego se utilizará un sistema de detección de objetos llamado *YOLO* (*You Only Look Once*) para reconocer y posicionar únicamente al automóvil en movimiento del resto de objetos detectados durante el streaming de video enviado por el *UAV*.

- **Seguimiento y aterrizaje automáticos del UAV**

Se realizará un modelamiento servo visual utilizando odometría visual, que se apoyará en detectar y describir puntos de interés de un objeto específico con el método *ORB (Oriented FAST and Rotated BRIEF)* mientras el UAV realiza un desplazamiento dictado por una señal formada por un tren de impulsos. Con una transformada geométrica aplicada en imágenes sucesivas del streaming de video se logrará estimar la posición del UAV respecto al objeto específico. Antes de comenzar con el control de aterrizaje, se obtendrá el modelo servo visual del UAV.

Figura 1

Esquema general de reconocimiento y aterrizaje automático del UAV



Nota. El gráfico representa los elementos que intervienen en el reconocimiento y aterrizaje automático de UAV sobre el área superior del automóvil.

Se realizará una comparativa entre dos controladores: PID y Fuzzy implementados en el algoritmo para el seguimiento y descenso automáticos. El controlador se encargará que el UAV mantenga la misma ruta del automóvil en

movimiento en los ejes X e Y por un tiempo determinado antes de comenzar con el procedimiento de descenso gradual. El controlador que presente mejores resultados bajo los mismos parámetros de evaluación será implementado en la PC con el Sistema Operativo del Robot *ROS* (del inglés *Robot Operating System*) que maneja una comunicación a distancia por *Wi-Fi* entre un *UAV* y el tele operador (ver Figura 1). La evaluación del desempeño del control de aterrizaje estará basada en el seguimiento de la plataforma y un descenso adecuado.

Objetivos

Objetivo general

Desarrollar un sistema de reconocimiento y seguimiento de plataformas móviles terrestres para un aterrizaje automático de un vehículo aéreo no tripulado basado en inteligencia artificial y odometría visual.

Objetivos específicos

- Realizar el estudio del estado del arte referente a machine learning, caracterización de imágenes, estimación de movimiento y control de aterrizaje de *UAV*.
- Desarrollar un sistema de detección, discriminación y seguimiento de plataformas aptas para el aterrizaje.
- Desarrollar un sistema de control para el aterrizaje automático de un vehículo aéreo no tripulado en las plataformas previamente detectadas y discriminadas.
- Evaluar el sistema de reconocimiento de plataformas mediante la identificación general de objetos.

- Evaluar el desempeño del control de aterrizaje automático basado en la localización concreta de la plataforma y un descenso controlado hasta terminar la tarea.
- Validación del correcto funcionamiento del sistema de reconocimiento y seguimiento de plataformas trabajando en conjunto con el control de aterrizaje automático del *UAV*.

Capítulo II

Marco teórico

Vehículos aéreos no tripulados (UAV)

Los Vehículos Aéreos No Tripulados *UAV (Unmanned Aerial Vehicle)* se refieren a aeronaves capaces mantener un vuelo estable sin la necesidad de un piloto a bordo. Son capaces de ser pilotados a distancia o incluso realizar tareas de forma automática (Quan, 2017). El desarrollo de su tecnología ha alcanzado un número creciente de aplicaciones, donde ha destacado la investigación militar (Kendoul, 2012).

Actualmente, el empleo de los *UAVs* está presente en muchas áreas, por ejemplo: en inspecciones periódicas a infraestructuras, la agricultura de precisión, transporte y suministro de implementos médicos, reconocimiento (Aguilar W. G., et al., 2017), (Aguilar, et al., Real-Time Detection and Simulation of Abnormal Crowd Behavior, 2017), (Aguilar & Angulo, Estabilización de vídeo en micro vehículos aéreos y su aplicación en la detección de caras, 2014), (Aguilar, et al., 2017), (Galindo, Aguilar, & Reyes Ch, 2019) operaciones de búsqueda y rescate, entre otras que han generado considerables ahorros de costos (Otto, Agatz, Campbell, Golden, & Pesch, 2018). Incluso, la producción de *UAVs* es relativamente barata y son fáciles de usar, por lo que han ganado popularidad en gran cantidad de aplicaciones técnicas, así como para fines recreativos (Prates, y otros, 2018).

En (Kendoul, 2012) se propone una clasificación de los *UAVs* considerando particularidades como sus dimensiones, aplicación, diseño, tolerancia de carga durante el vuelo, nivel de autonomía, etc. Tomando en cuenta únicamente el tamaño y la carga útil que soporta el vehículo, se puede catalogar al drone como un vehículo aéreo de

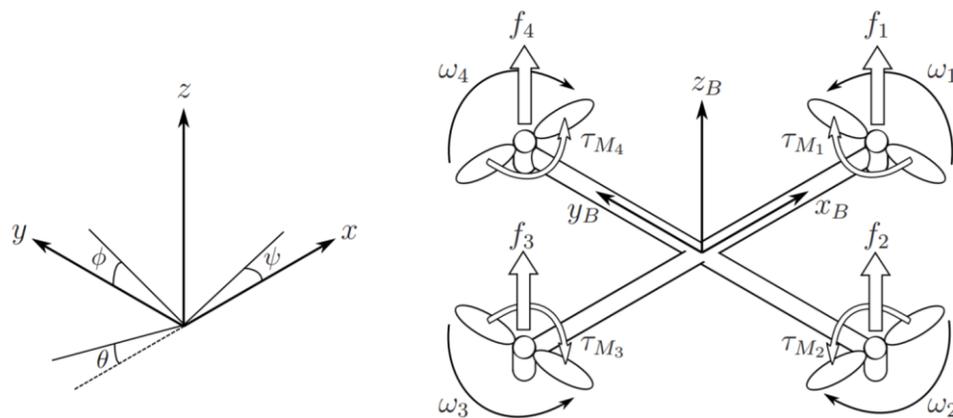
microescala MAVs, si su peso total no sobrepasa los cientos de gramos. Estos pueden ser diseñados de dos maneras: con ala fija o con rotores, este último más comúnmente conocido como cuadricóptero, es con el que se realizó este proyecto de investigación.

Dinámica del movimiento de un cuadricóptero

El cuadricóptero es un cuerpo tridimensional que está sometido a varias fuerzas y momentos, donde diferencias específicas en el accionar de sus rotores producen su ascenso, descenso, traslación y rotación. La estructura del cuadricóptero está representada en la Figura 2, mostrando las velocidades angulares, torques y fuerzas correspondientes generadas por los cuatro rotores responsables del movimiento dentro de los seis grados de libertad del cuadricóptero.

Figura 2

Ejes de referencia inerciales y corporales de un cuadricóptero



Nota. El gráfico representa los marcos de referencia necesarios para definir la posición lineal de un cuadricóptero. Tomado de “*Modelling and control of quadcopter*”, (Luukkonen, 2011).

La posición lineal absoluta del cuadricóptero está definida en el marco de referencia inercial con los ejes x, y, z representados con ξ . Su posición angular está definida en el marco de referencia inercial con los tres ángulos de Euler η . El ángulo *pitch* θ determina la rotación del cuadricóptero alrededor del eje y . El ángulo *roll* ϕ establece la rotación sobre el eje x , y el ángulo *yaw* ψ establece la rotación alrededor del eje z . El vector q contiene los vectores posición lineal y angular del cuadricóptero (Luukkonen, 2011).

$$\xi = \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \quad \eta = \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix}, \quad q = \begin{bmatrix} \xi \\ \eta \end{bmatrix} \quad (1)$$

La matriz rotacional establece la relación de los sistemas del cuadricóptero y de coordenadas en Tierra expresada en (2):

$$R = \begin{bmatrix} C_\psi C_\theta & C_\psi S_\theta S_\phi - S_\psi C_\theta & C_\psi S_\theta C_\phi + S_\psi S_\phi \\ S_\psi C_\theta & S_\psi S_\theta S_\phi + C_\psi C_\theta & S_\psi S_\theta C_\phi - C_\psi S_\phi \\ -S_\theta & C_\theta C_\phi & C_\theta S_\phi \end{bmatrix} \quad (2)$$

Donde C_x y S_x son notaciones de $\cos(x)$ y $\sin(x)$ respectivamente.

Se supone que el cuadricóptero es simétrico al tener los cuatro brazos alineados con los ejes X e Y del cuerpo. Entonces, se puede definir la matriz de inercia como una matriz diagonal I en la que $I_{xx} = I_{yy}$.

$$I = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \quad (3)$$

La velocidad angular del rotor i , se la representa por ω_i , esta crea una fuerza f_i en la dirección del eje del rotor. La aceleración, junto a la velocidad angular del rotor producen un torque τ_{Mi} alrededor del eje del rotor.

$$f_i = k\omega_i^2, \quad \tau_{Mi} = b\omega_i^2 + I_M\dot{\omega}_i \quad (4)$$

Donde la constante de elevación es k , la constante de arrastre es b y el momento de inercia del motor es I_M . A menudo el efecto de $\dot{\omega}_i$ es muy pequeño, por lo que se lo puede omitir.

Para lograr una rotación en el eje x , se manipula las velocidades de los rotores 2 y 4, y el par de rotores 1 y 3 permanecen invariantes en velocidad, al igual que su torque total, produciendo un movimiento de derecha a izquierda. En cambio, para la rotación en el eje y , las velocidades de los rotores 1 y 3 son manipuladas y el par de rotores 2 y 4 permanecen invariantes en su velocidad, al igual que su torque total, generando un movimiento para adelante y atrás. Finalmente, para una rotación en el eje z , las velocidades de los rotores 1 y 3 deben aumentar y en los rotores 2 y 4 debe disminuir, causando un desbalanceo aerodinámico de los torques permitiendo girar al cuadricóptero sobre su propio eje.

La dinámica necesaria para producir los diferentes movimientos de traslación y rotación a partir de las variaciones de las velocidades y los torques de cada rotor se detallan en la Tabla 1 basada de (Salcedo Peña, 2018) y (Grijalva Caisachana, 2019) donde la velocidad angular de cada rotor se la describe cómo ω_i y a la variación de igual proporción es descrita como Δ .

Tabla 1

Configuración de la velocidad de los rotores de un cuadricóptero

Movimiento	Rotor 1	Rotor 2	Rotor 3	Rotor 4
Arriba	$\omega_i + \Delta$	$\omega_i + \Delta$	$\omega_i + \Delta$	$\omega_i + \Delta$

Movimiento		Rotor 1	Rotor 2	Rotor 3	Rotor 4
Abajo		$\omega_i - \Delta$	$\omega_i - \Delta$	$\omega_i - \Delta$	$\omega_i - \Delta$
Roll ϕ	Derecha	$\omega_i - \Delta$	$\omega_i - \Delta$	ω_i	ω_i
	Izquierda	ω_i	ω_i	$\omega_i - \Delta$	$\omega_i - \Delta$
Pitch θ	Adelante	$\omega_i - \Delta$	ω_i	ω_i	$\omega_i - \Delta$
	Atrás	ω_i	$\omega_i - \Delta$	$\omega_i - \Delta$	ω_i
Yaw ψ	Horario	$\omega_i - \Delta$	$\omega_i + \Delta$	$\omega_i - \Delta$	$\omega_i + \Delta$
	Anti-Horario	$\omega_i + \Delta$	$\omega_i - \Delta$	$\omega_i + \Delta$	$\omega_i - \Delta$

Nota. Esta tabla muestra la configuración de las velocidades de los cuatro rotores del cuadricóptero para generar movimientos de traslación y rotación.

Control de AUVs

Se ha desarrollado una gran cantidad de algoritmos para sistemas de navegación y control a bordo de UAVs. La mayoría de estos sistemas incluyen algunos términos no lineales, algoritmos evolutivos o técnicas de optimización (Kendoul, 2012). A pesar de su éxito (Aguilar, Álvarez, Grijalva, & Rojas, 2019), el número de implementaciones no es tan alto como los sistemas con un control clásico debido a su complejidad, naturaleza no lineal y costo de cálculo.

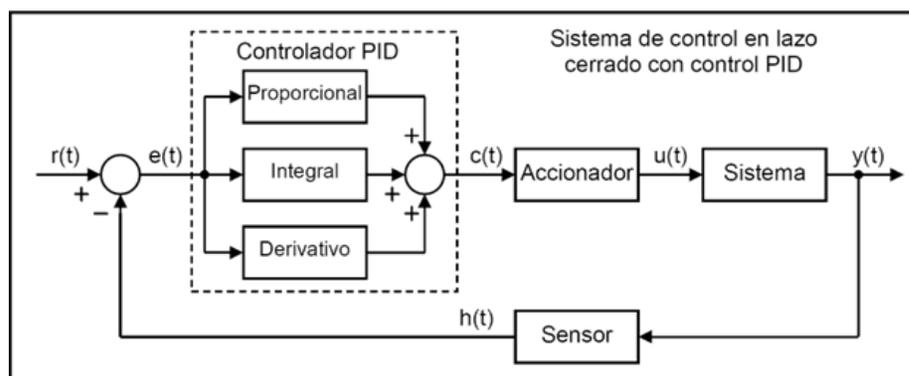
Control PID

La técnica de control PID se han integrado con éxito como control en tiempo real y sistemas de navegación en línea para UAVs (Aguilar, Casaliglla, & Pólit, Obstacle Avoidance Based-Visual Navigation for Micro Aerial Vehicles, 2017), (Aguilar, Álvarez, Grijalva, & Rojas, 2019), (Aguilar, et al., 2018), (Aguilar W. G., Manosalvas, Guillén, &

Collaguazo, 2018), (Aguilar, Angulo, & Costa-Castello, Autonomous Navigation Control for Quadrotors in Trajectories Tracking, 2017). Esto, no solo se debe a su estructura simple y fácil implementación, sino también a su adecuado desempeño. Una correcta implementación de dichos controladores, y sin requerir un desarrollo matemático complejo, se necesita un ajuste de parámetros o un procedimiento de sintonización para lograr un rendimiento mejorado. El proceso de adecuación, para obtener los valores óptimos de los parámetros del controlador, es un desafío crítico (Kada & Ghazzawi, 2011).

Figura 3

Diagrama para el controlador PID



Nota. Este gráfico muestra un diagrama de bloques del sistema de control en lazo cerrado con el controlador PID. Tomado de <https://www.picuino.com/es/arduprog/control-pid.html>, (Pardo, 2020)

El controlador PID, mostrado en la Figura 3 está conformado por tres acciones que son:

- **Acción proporcional:** Consiste en el producto de la señal del error (entre el valor medido y el valor deseado) y la constante proporcional para lograr que el error en estado estacionario se aproxime a cero.

- **Acción integral:** Se la utiliza para disminuir y eliminar el error en estado estacionario. La acción integral actúa cuando existe una diferencia entre la variable medida y el valor establecido, integrando la diferencia en el tiempo y sumándola a la acción proporcional.
- **Acción derivativa:** Mantiene el error al mínimo corrigiéndolo proporcionalmente gracias a que traza su crecimiento generando una salida del controlador para adelantarse a los grandes cambios con la misma velocidad que se producen.

Control Difuso

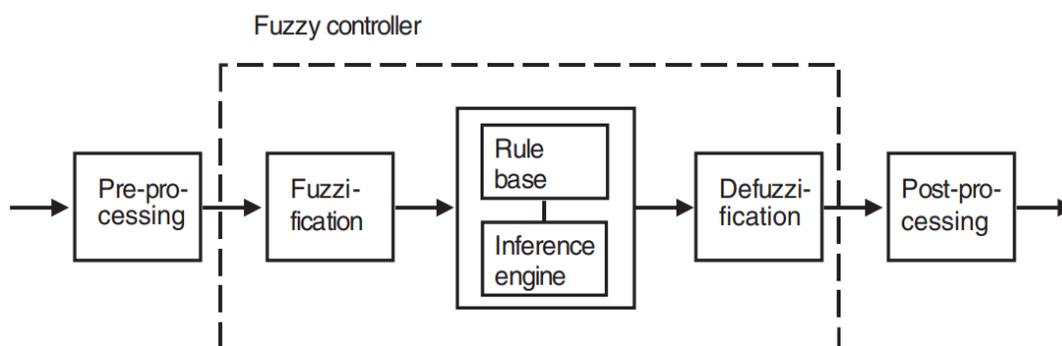
La lógica difusa se ha aplicado en muchos campos, desde la teoría del control hasta la inteligencia artificial. En la lógica booleana, los valores de verdad de las variables solo pueden ser los enteros 0 o 1. La lógica difusa es empleada para manejar el concepto de verdad parcial, donde el valor de verdad puede variar entre completamente verdadero y completamente falso. Además, cuando se utilizan variables lingüísticas difusas, los niveles de pertenencia pueden gestionarse mediante funciones específicas (Zadeh, 1996).

La lógica difusa es útil para representar el conocimiento humano en un dominio específico de aplicaciones y para razonar con ese conocimiento, generando inferencias o acciones útiles. En particular, la lógica difusa puede emplearse para representar, como un conjunto de reglas difusas, el conocimiento de un humano que controla un sistema. Entonces, una regla de inferencia en la lógica difusa puede usarse de acuerdo con esta base de conocimiento difuso, para tomar decisiones de control para un conjunto dado de observaciones de diferentes sistemas. Esta tarea se refiere al procesamiento del conocimiento. En este sentido, la lógica difusa dentro del control inteligente sirve para

representar y procesar el conocimiento de control de un humano en un sistema determinado (Liu, 2018) (ver Figura 4).

Figura 4

Esquema del controlador de lógica difusa



Nota. Este gráfico muestra un diagrama de bloques del sistema de control difuso. Tomado de “*Foundations of Fuzzy Control*”, (Jantzen, 2007).

Inteligencia Artificial (IA)

Definir de manera precisa la palabra inteligencia y sobre todo de inteligencia artificial, es algo aún por resolver. Las definiciones han variado con el paso del tiempo, debido a al rápido desarrollo de la tecnología (Kok, Boers, Kusters, & Van der Putten, 2009). Algunas definiciones incluyen “copiar la conducta humano inteligente”. Existe muchas formas en que una máquina puede responder con una conducta inteligente, pero eso no las convierte en más capaces que nosotros. Al trabajar con cualquiera de estas inteligencias artificiales que sobresalen en un dominio muy específico y se intenta que realicen otra tarea, el resultado obtenido es desastroso.

Esta capacidad de lograr realizar múltiples tareas al mismo tiempo como: pensar, ver, andar y hablar, es una característica muy codiciada, pero en la actualidad se sigue

investigando en los departamentos de inteligencia artificial. Con esta idea se puede definir dos tipos de IAs:

- **Inteligencia artificial débil.** Se dice que una inteligencia artificial es débil para referirse a aquellos sistemas que únicamente pueden cumplir con un conjunto muy limitado de tareas (Aguilar & Morales, 3D Environment Mapping Using the Kinect V2 and Path Planning Based on RRT Algorithms, 2016), (Cabras, Rosell, Pérez, Aguilar, & Rosell, 2011), (Aguilar, Abad, Ruiz, Aguilar, & Aguilar-Castillo, 2017), (Aguilar, Morales, Ruiz, & Abad, RRT* GL Based Path Planning for Virtual Aerial Navigation, 2017), (Aguilar, Sandoval, Limaico, Villegas-Pico, & Asimbaya, 2019), por ejemplo: Si se ha enseñado a un robot a caminar, al intentar que haga algo diferente como patear una pelota es muy probable que el resultado sea desastroso.
- **Inteligencia artificial fuerte.** Son aquellas IAs que con la capacidad de dar solución a varios tipos de problemáticas de dominio diferentes, pero hasta la fecha, todas las IAs todavía se clasifican como débiles.

La idea de imitar comportamientos inteligentes no representa algo cognitivo, es decir, se puede programar de manera clásica los movimientos del brazo robótico para que siempre realicen un mismo movimiento, no tiene un comportamiento inteligente porque la lógica del movimiento se ha programado. Sin embargo, esto encaja dentro de la definición antes descrita, porque en apariencia la máquina realiza un comportamiento inteligente.

La capacidad de moverse y adaptarse al entorno hace referencia al campo de la robótica o la capacidad de entender el lenguaje está sujeta al campo del *Natural Language Processing*. Todas estas capacidades conforman un campo de estudio propio

dentro del mundo de la inteligencia artificial sobre todo al hablar de la destreza para aprender, o dicho en otras palabras, el machine learning.

Machine learning

El machine learning o aprendizaje automático es parte de la inteligencia artificial y busca la manera de proporcionar a máquinas con la destreza del aprendizaje, es decir, que generalice conceptos mediante varias experiencias. Este aprendizaje puede dividirse en 2 grupos diferentes:

- **Aprendizaje supervisado.** En este tipo de aprendizaje, los algoritmos trabajan con datos “etiquetados” para el entrenamiento, intentando determinar una función a partir de las variables de entrada (*input data*) para asignar una correcta etiqueta de salida. El algoritmo trabaja con el conjunto de datos proporcionados para aprender a designar una etiqueta de salida adecuada a un nuevo valor, predecir el valor de salida (Simeone, 2018).
- **Aprendizaje No supervisado.** En este aprendizaje, los datos usados durante el entrenamiento del modelo no cuentan con un identificador o etiqueta, no se conoce la correspondencia entre los datos de entrada y de salida.

Dentro del machine learning se puede encontrar un nuevo mundo donde existen diferentes técnicas que sirven para cubrir diferentes tipos de aplicaciones, por ejemplo, técnicas como los árboles de decisión, modelos de regresión, modelos de clasificación, entre otro. Pero de manera especial las redes neuronales han contribuido el campo del machine learning durante la última década (Simeone, 2018).

Redes Neuronales

Tienen la capacidad para aprender en “forma jerarquizada”, la información se asimila por niveles donde las primeras capas aprenden conceptos muy concretos, por ejemplo:

- ¿Qué es un tornillo?
- ¿Qué es un espejo?
- ¿Qué es una rueda?

En las capas posteriores se usa la información aprendida previamente para entender conceptos más abstractos como un auto, un piano o una persona. A medida que se añade más capas la información aprendida es cada vez más abstracta. El número de capas que se puede usar no tiene límite y la tendencia general es que cada vez se aumenten más, produciendo algoritmos muy complejos. Este incremento del número de capas y de la complejidad son conocidos como algoritmos de *Deep learning*.

Algoritmos de seguimiento visual

La detección de objetos en imágenes fijas y el seguimiento de estos en secuencias de video son aplicaciones recurrentes en el campo de la visión por computadora. Se ha presenciado muchos avances en el posicionamiento y reconocimiento de objetos aprovechando los progresos en *deep learning*, a menudo superando la precisión del nivel humano (He, Zhang, Ren, & Sun, 2016). Si bien es posible hacer un seguimiento de objetos al volver a detectarlos en un conjunto de fotogramas (seguimiento por detección), este enfoque tiene varios inconvenientes:

- El seguimiento se limita a las categorías utilizadas durante el entrenamiento.

- El detector puede perder el rastro del objeto con poca iluminación, cambios de posición, etc.
- Lo más importante, el reconocimiento de objetos utilizando *deep learning* demanda grandes recursos computacionales como una GPU.

A menudo es ventajoso abordar el problema desde el ángulo tradicional del seguimiento de objetos genérico.

Los algoritmos de seguimiento suelen diferir en la manera de detectar los cambios en la apariencia del objeto a lo largo del tiempo. Los algoritmos de seguimiento suelen implementar dos funciones: añadir (*add*) y actualizar (*update*), la primera se inicializa cada vez que aparece un nuevo objeto en la escena y la segunda se aplica posteriormente para cada fotograma.

Las variaciones entre los algoritmos están en el paso de actualización, donde cada método aprende esencialmente la apariencia más reciente de cada objetivo. Algunos algoritmos pueden aprovechar la información histórica sobre las trayectorias del movimiento de los objetos, por ejemplo, para limitar la región de búsqueda. Los algoritmos de seguimiento tienden a tener una complejidad computacional menor que la que tendría un enfoque de detección a gran escala.

A continuación, se describe brevemente cinco algoritmos de seguimiento disponibles en la librería de Open Computer Vision (*OpenCV*).

- ***On-Line Boosting***. Es un algoritmo de seguimiento que considera esta acción como una clasificación binaria. En cada paso del seguimiento, el algoritmo actualiza el modelo de objetos utilizando *AdaBoost* para entrenar una colección de clasificadores débiles. La etapa de entrenamiento usa la región objetivo como

un ejemplo positivo y muestra parches de los alrededores de la región del objeto actualmente seguida. La ubicación del objeto en el siguiente cuadro se estima aplicando el clasificador binario en dicho cuadro y eligiendo la ubicación más probable según lo predicho por el clasificador *AdaBoost*.

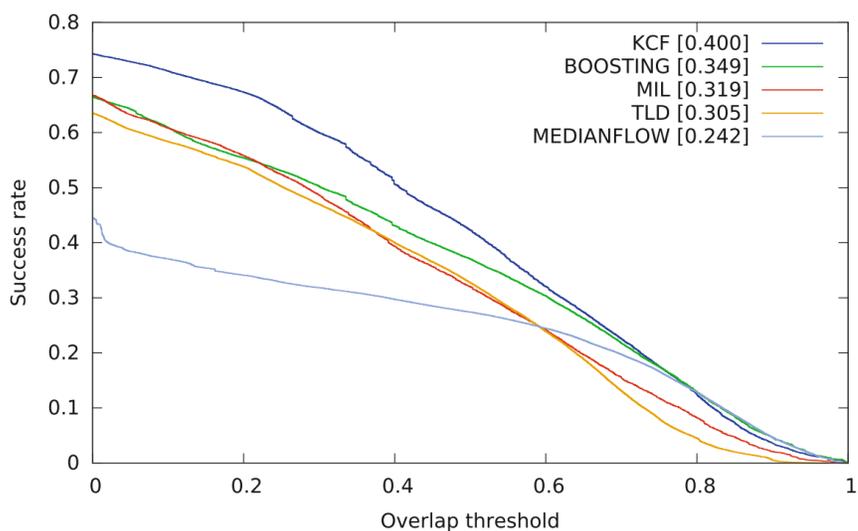
- **Multiple Instance Learning (MIL).** Este algoritmo también plantea el problema del seguimiento como una tarea de clasificación. El método utiliza el enfoque de aprendizaje de instancias múltiples, que considera *bags* de objetos agrupando muestras similares en bolsas, donde cada una se considera una muestra positiva general si al menos una de las muestras individuales que contiene es positiva y caso contrario será negativa. Esto intenta evitar confundir al clasificador con las muestras subóptimas que se etiquetan como positivas, en lugar de darle al clasificador una comprensión más vaga de cómo deberían ser las muestras positivas.
- **Median Flow.** Los autores (Kalal, Mikolajczyk, & Matas, 2010) presentan una medida de error de seguimiento, donde los puntos de cada objeto son seguidos hacia adelante y atrás en el tiempo y se comparan las trayectorias resultantes. Suponiendo que un método de seguimiento correcto produciría la misma trayectoria, pero opuesta con una entrada de tiempo invertido, cualquier divergencia de las trayectorias de seguimiento hacia adelante y atrás indica un error de seguimiento. Los autores utilizan esta medida de error para proponer un método de seguimiento en el que los puntos dentro de un cuadro delimitador se rastrean y miden el error. Los valores atípicos se filtran y el movimiento del cuadro delimitador se estima en función de los valores internos.

- ***Tracking-Learning-Detection (TLD)***. Es un método para tareas de seguimiento a largo plazo (Lehtola, Huttunen, Christophe, & Mikkonen, 2017). El objetivo del método es mejorar la robustez del seguimiento al deshabilitar el aprendizaje *on-line* si el objeto está fuera del fotograma o completamente ocluido por otros objetos, evitando así el aprendizaje de información errónea. El componente de detección permite que el método vuelva a detectar el objeto, en caso que aparezca nuevamente en el video.
- ***Kernelized Correlation Filter (KCF)***. El método *KCF* (Henriques, Caseiro, Martins, & Batista, 2015) emplea la propiedad de invariancia de desplazamiento de la transformada de Fourier para diseñar un algoritmo rápido para el emparejamiento basado en filtros de correlación. La transformada de Fourier simplifica el cálculo de correlación para hacerlo extremadamente rápido. Además, el método se generaliza aplicando el truco del kernel para permitir medidas de correlación no lineales. Los autores señalan que uno de los principales desafíos en el seguimiento es la incapacidad de utilizar una cantidad suficientemente grande de datos de entrenamiento disponibles de cada fotograma de entrada debido a la necesidad de considerables recursos computacionales. Este problema se evita con *KCF* debido a su ligera implementación.

La precisión de los algoritmos representados por la valoración de índices de Jaccard promediados (medida del grado de similitud entre dos conjuntos) y el desempeño midiendo su velocidad entre fotogramas se exhibe a continuación (ver Figura 5).

Figura 5

Comparación entre algoritmos de seguimiento



Nota. Este gráfico muestra una comparación entre 5 algoritmos de seguimiento visual en función del índice de Jaccard. Tomado de “*Evaluation of Visual Tracking Algorithms for Embedded Devices*”, (Lehtola, Huttunen, Christophe, & Mikkonen, 2017).

Los resultados muestran que *Median Flow* es el más rápido, pero su precisión es la más baja. Por otro lado, (Lehtola, Huttunen, Christophe, & Mikkonen, 2017) recomienda el algoritmo *KCF*, ya que es el segundo más rápido y el más preciso.

Odometría visual

En los últimos años, la odometría visual (VO) ha ganado importancia en aplicaciones de robótica, como vehículos terrestres que se mueven en terrenos irregulares o vehículos aéreos no tripulados (UAV) (Gomez-Ojeda & Gonzalez-Jimenez, 2016). La VO es un proceso para estimar la posición de un cuerpo usando solo la entrada de una o varias cámaras conectadas a él. VO opera estimando gradualmente la posición del vehículo analizando las variaciones que el movimiento induce en los fotogramas de sus cámaras a bordo. Para que VO funcione de manera efectiva, debe haber suficiente

iluminación en el entorno y una escena estática con suficiente textura para permitir la extracción del movimiento aparente. Además, se deben capturar fotogramas consecutivos asegurándose que tengan una superposición de escena suficiente (Scaramuzza & Fraundorfer, 2011).

Estimación de movimiento

Es la etapa de cálculo central realizado para cada imagen en un sistema de VO. En esta etapa, se realiza un cálculo del desplazamiento de la cámara usando la imagen actual y la imagen anterior. Mediante la concatenación de todos estos movimientos individuales, se puede recuperar la trayectoria completa de la cámara y el cuerpo (suponiendo que la cámara esté montada rígidamente). Esta sección explica cómo la transformación T_k entre dos imágenes I_{k-1} e I_k puede calcularse a partir de dos conjuntos de características correspondientes f_{k-1} , f_k en instantes de tiempo $k-1$ y k , respectivamente (Scaramuzza & Fraundorfer, 2011). Considerando que las correspondencias de características se especifican en dos o tres dimensiones, existen tres métodos diferentes:

- **2-D-to-2-D:** Tanto f_{k-1} como f_k se especifican en coordenadas de imagen 2-D.
- **3-D-to-3-D:** Tanto f_{k-1} como f_k se especifican en coordenadas 3-D. Es necesario triangular puntos tridimensionales por cada pequeño avance en el tiempo.
- **3-D-to-2-D:** En este caso, f_{k-1} se especifican en 3-D y f_k son sus correspondientes proyecciones 2-D en la imagen I_k . En el caso monocular, la estructura tridimensional debe triangularse a partir de dos vistas de cámara adyacentes I_{k-2} e I_{k-1} y luego coincidir con las características de imagen 2-D en

una tercera vista I_k . En el esquema monocular, se necesitan coincidencias en al menos tres vistas.

Formación de imágenes

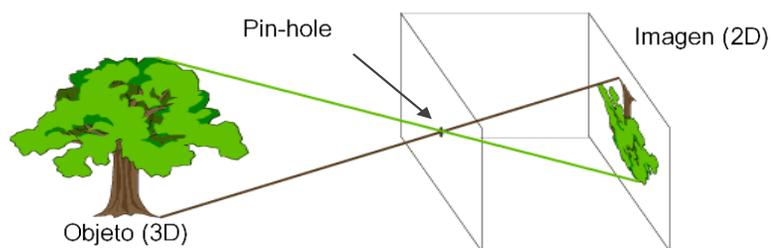
Para formar una imagen de una escena en tres dimensiones a una superficie en dos dimensiones se utiliza un sistema de caja cerrada. Este sistema simula dos pantallas paralelas, una frente a la otra. Considerando la presencia de un minúsculo orificio en la primera pantalla de la caja donde pasaría un fotón de luz permitiendo que los rayos de luz emitidos o reflejados por un objeto de la escena, atraviesen la primera pantalla y formen una imagen invertida en la segunda pantalla (ver Figura 6).

La representación del objeto en 2D necesita de la excitación de un elemento fotosensible en la otra pantalla. Se debe considerar que el minúsculo orificio en la primera pantalla, impide el paso de suficiente luz para captura de la imagen. Una solución es colocar lentes en la apertura para enfocar el conjunto de rayos que parten de una zona específica de la escena en tres dimensiones a su correspondiente zona dentro de los límites de la imagen (Viala, 2006).

El modelo *Pin-Hole* de una cámara se refiere a la relación matemática existente entre el posicionamiento de un punto en tres dimensiones y su proyección dentro de los límites de la imagen. Considerando como un punto infinitesimalmente minúsculo a la apertura de la cámara, llamado centro óptico y que no utiliza ninguna clase de lente para enfocar los rayos de luz. Es decir, que este modelo es únicamente utilizado como una aproximación de primer orden en la generación de una imagen 2D a partir de una escena 3D (Jiménez Camacho , 2009).

Figura 6

Modelo Pin-Hole para la formación de imágenes



Nota. Este gráfico representa el funcionamiento del modelo Pin-Hole para representar un objeto en tres dimensiones a una imagen. Tomado de *“Caracterización y optimización del proceso de calibrado de cámaras basado en plantilla bidimensional”*, (Viala, 2006).

Caracterización de imágenes

La capacidad cognitiva de los humanos permite que con un pequeño vistazo se extraiga toda la información de una imagen. En cambio, la visión por computadora y el procesamiento de imágenes necesita que la imagen sea representada por su extracción de características (Aguilar & Angulo, Robust video stabilization based on motion intention for low-cost micro aerial vehicles, 2014), (Aguilar, Angulo, & Pardo, Motion intention optimization for multirrotor robust video stabilization, 2017). Para llevar a cabo este objetivo, los detectores y extractores de características deben cumplir importantes propiedades, como:

- **Robustez:** ser capaz de detectar la misma ubicación de características halladas en los píxeles que forman la imagen independientemente de la escala, rotación, desplazamiento y ruido.
- **Repetibilidad:** ser capaz de detectar las mismas características en la imagen sometida a diferentes condiciones de visualización.

- **Precisión:** localizar con precisión la misma ubicación de características halladas en los píxeles que forman la imagen para tareas de semejanza entre estas.
- **Generalidad:** ser capaz de detectar características que se pueden usar en diferentes aplicaciones.
- **Eficiencia:** el algoritmo de detección de características debería poder detectar características en nuevas imágenes rápidamente para admitir aplicaciones en tiempo real.
- **Cantidad:** capacidad de detección de la mayoría o la totalidad de características en los píxeles de la imagen reflejando su contenido para obtener una representación compacta de la imagen.

Existe principalmente dos métodos para representar imágenes, en características globales y en características locales.

Características globales

En la representación global de una imagen se crea un vector con valores que cuantifican diferentes particularidades de la imagen como la textura, el color o su forma. Este vector es multidimensional y detalla la información de toda la imagen. De cada imagen se puede obtener un único vector de características, esto permite la comparación entre imágenes utilizando solamente dichos vectores. Por ejemplo, para hacer la distinción entre imágenes del cielo despejado (azul) y un bosque (verde), el descriptor global de color de cada imagen crearía vectores muy diferentes.

Las características globales se las puede definir como una propiedad de la imagen obtenida de la información de todos sus píxeles, tales como histogramas de color, bordes, textura o descriptores específicos extraídos de algunos filtros aplicados a la imagen

(Hassaballah, Abdelmgeid, & Alshazly, 2016). Son más rápidas y fáciles de calcular. Requieren pequeñas cantidades de memoria. Preferentemente utilizadas en aplicaciones donde esté disponible una segmentación aproximada del objeto de interés. Son sensibles a la oclusión y el desorden.

Características locales

Estas características representan de forma más distintiva y estable a la imagen, trabaja con regiones sobresalientes sin dejar de ser invariable para la iluminación y alteraciones en el punto de vista. Estas utilizan un grupo de descriptores de características locales extraídos de zonas de interés de la imagen llamados *keypoints*. La Figura 7 ejemplifica lo mencionado.

Figura 7

Características de una imagen



Nota. Esta imagen representa las características locales y globales que definen sus particularidades. Tomado de “*Image Features Detection, Description and Matching*”, (Hassaballah, Abdelmgeid, & Alshazly, 2016).

El uso de características locales para la búsqueda de imágenes a gran escala tiene un rendimiento superior que las características globales (Bianco, Mazzini, Pau, & Schettini, 2015). Como la imagen puede llegar a tener cientos de características locales

(Hassaballah, Abdelmgeid, & Alshazly, 2016) necesitan de una gran cantidad de memoria, así mismo.

Puntos de interés

Procuran representar características principales de diferentes regiones dentro de una imagen utilizando un grupo pequeño de información. Generalmente, se los utiliza en procesos de comparación de imágenes (Aguilar & Angulo, 2014). Esta tarea está definida por tres procesos:

- Detección de puntos de interés
- Descripción de puntos de interés
- Matching de puntos de interés

Detección de puntos de interés

Este proceso debe realizarse minuciosamente para que la localización de los *keypoints* logre características invariantes en variaciones de visualización o la existencia de ruido, las características extraídas de estos puntos de interés deben ser lo suficientemente distintivas para facilitar una coincidencia más precisa (Mian, Bennamoun, & Owens, 2007). Estos pueden clasificarse en tres categorías: detectores de escala única, detectores de escala múltiple y detectores invariantes afines. Entre los algoritmos para los detectores de escala única se tiene a:

- **Detector Harris:** Es un algoritmo básico que se utiliza como base para algunos métodos de detección más modernos. Se encarga de la localización de bordes y esquinas.

- **Detector *FAST*:** Es un algoritmo presentado en (Rosten & Drummond, 2006) de detección binaria de *keypoints* utilizado en el método *ORB* para la descripción de características.
- **Detector de la matriz Hessiana:** Es un algoritmo con alta velocidad de detección utilizado por el descriptor de características *SURF*.

En cambio, uno de los algoritmos más utilizados en los detectores de escala múltiple es:

- **Diferencia de Gaussianas (*DoG*):** Es un algoritmo caracterizado por lograr una alta invarianza en el escalado. Utilizado en el método *SIFT* para la descripción de características.

Descripción de puntos de interés

Tras la detección de un grupo de *keypoints*, ubicados en $p(x, y)$, con escala s y orientación θ , su estructura en una vecindad de p , debe codificarse en un descriptor apropiado para la coincidencia selectiva e insensibilidad a las variaciones locales de la escena. Este descriptor debe estar alineado a θ y ser proporcional a la escala s . Existen varios descriptores de características de imágenes, entre los más utilizados están:

- ***SIFT (Scale Invariant Feature Transform)***

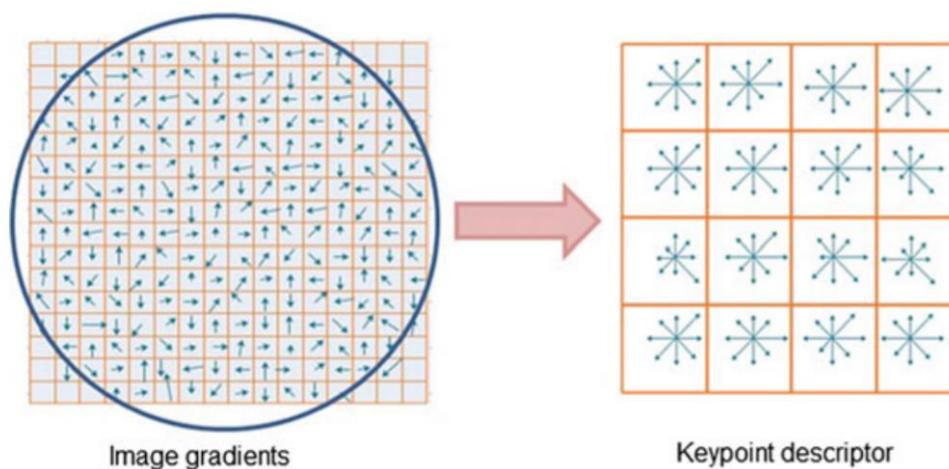
Este algoritmo realiza 4 pasos básicos. El primero es estimar un extremo espacial de escala usando la Diferencia de Gauss (*DoG*). El siguiente paso es realizar una exploración de *keypoints* considerando que los candidatos se localizan y refinan eliminando aquellos de bajo contraste. El tercer paso, es asignar una orientación a los *keypoints* basada en el gradiente de la imagen local y, por último, un generador de

descriptores para el cálculo del descriptor de la escena local para cada *keypoint* respecto a la magnitud y orientación del gradiente de la escena.

El proceso de descripción del algoritmo *SIFT* comienza muestreando las magnitudes y orientaciones del gradiente de la escena en una región de 16×16 píxeles que rodea a cada *keypoint* utilizando su escala para seleccionar el grado de desenfoque gaussiano para la imagen. Luego, se produce un grupo de histogramas de orientación donde cada uno incluye muestras de una subregión 4×4 píxeles del vecindario original y tiene ocho contenedores de orientaciones en cada uno.

Figura 8

Representación esquemática del descriptor SIFT



Nota. Este gráfico representa el esquema del descriptor *SIFT* para un cuadro de 16×16 píxeles y una matriz de descriptores de 4×4 . Tomado de “*Image Features Detection, Description and Matching*”, (Hassaballah, Abdelmgeid, & Alshazly, 2016).

Se utiliza una función de ponderación gaussiana con σ igual a la mitad del tamaño de la región para asignar peso a la magnitud de cada punto de muestra otorgando pesos más altos a los gradientes más cercanos al centro de la región. La Figura 8 es una

representación esquemática del algoritmo *SIFT*; donde las orientaciones de gradiente y las magnitudes se calculan en cada píxel y luego se ponderan mediante una caída de Gauss (indicada por un círculo superpuesto). Luego se calcula un histograma de orientación de gradiente ponderado para cada subregión.

La etapa de descripción del algoritmo *SIFT* comienza muestreando las magnitudes y orientaciones del gradiente de la imagen en una región de 16×16 píxeles alrededor de cada punto de interés utilizando su escala para seleccionar el nivel de desenfoque gaussiano para la imagen. Luego, se crea un conjunto de histogramas de orientación donde cada uno contiene muestras de una subregión 4×4 píxeles de la región del vecindario original y tiene ocho contenedores de orientaciones en cada uno.

Se utiliza una función de ponderación gaussiana con σ igual a la mitad del tamaño de la región para asignar peso a la magnitud de cada punto de muestra otorgando pesos más altos a los gradientes más cercanos al centro de la región. La Figura 8 es una representación esquemática del algoritmo *SIFT*; donde las orientaciones de gradiente y las magnitudes se calculan en cada píxel y luego se ponderan mediante una caída de Gauss (indicada por un círculo superpuesto). Luego se produce un histograma de orientación de gradiente ponderado para cada subregión.

- ***SURF (Speeded-Up Robust Features Descriptor)***

Este algoritmo fue desarrollado por (Bay, Tuytelaars, & Van Gool, 2006) con una notable rapidez, mayor robustez y eficiencia que *SIFT*. La localización de *keypoints* se basa en sencillas cajas 2D como filtros que utilizan detectores de *blobs* (manchas, peculiaridades de mayor tamaño que esquinas y bordes) invariantes a la escala basados en el determinante de la matriz de Hesse utilizados para definir la escala y la posición de

keypoints. Trabaja con una aproximación de derivadas gaussianas de segundo orden utilizando un conjunto de filtros de caja en imágenes integrales. Las aproximaciones se las define como D_{xx} , D_{yy} y D_{xy} donde el determinante aproximado de la matriz Hessiana se lo representaría en la expresión (5):

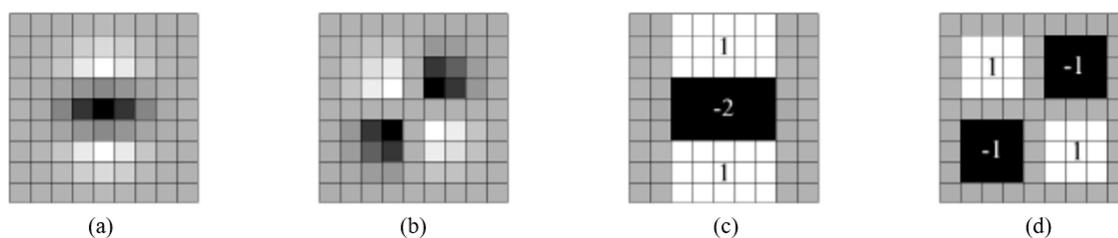
$$\det(H_{approx}) = D_{xx}D_{yy} - (wD_{xy})^2 \quad (5)$$

El término w se utiliza para equilibrar la expresión para el determinante Hessiano. Es decir, es un peso relativo para la respuesta del filtro.

Los filtros de caja de 9×9 representan la escala más baja para calcular los mapas de respuesta de *blobs* que son equivalentes a derivadas Gaussianas de escala $\sigma = 1.2$ (ver Figura 9).

Figura 9

Aproximación de derivadas gaussianas de segundo orden



Nota. Este gráfico representa en (a) y (b) derivadas gaussianas de segundo orden en y , xy . En (c) y (d) aproximaciones con filtros de caja donde las regiones grises son cero. Tomado de “*SURF: Speeded Up Robust Features*”, (Bay, Tuytelaars, & Van Gool, 2006).

Los determinantes resultantes se almacenan en un mapa de respuesta de *blobs*, luego se detectan y refinan los máximos locales con una interpolación cuadrática. Para

terminar, se obtienen *keypoints* estables realizando una eliminación no máxima en un vecindario $3 \times 3 \times 3$.

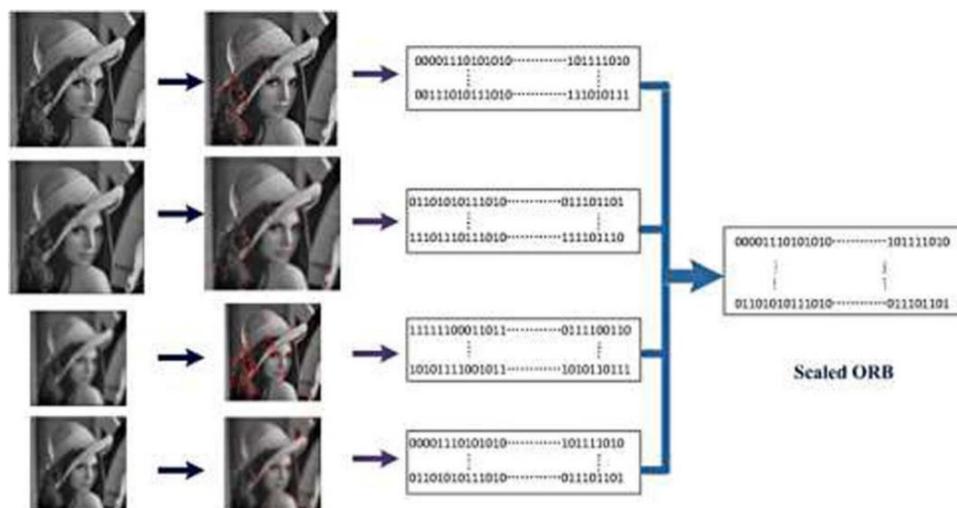
- ***ORB (Oriented FAST and Rotated BRIEF)***

Este algoritmo está abierto al público, fue presentado por (Rublee, Rabaud, Konolige, & Bradski, 2011) como una mejor alternativa en eficiencia y rendimiento en algoritmos para detectar y describir *keypoints* para el matching entre imágenes. Este método con respecto a los algoritmos *SIFT* y *SURF* arroja mejores resultados en la rapidez de operación por su bajo coste computacional resultando en un menor tiempo de procesamiento.

ORB es una versión más desarrollada del detector *FAST* (se agrega un componente de orientación al que se denomina *oFAST*), el método se modifica para estimar la orientación de un punto de interés utilizando el proceso modificado de descripción de *BRIEF* (agregar un componente de invarianza rotacional denominado *rBRIEF*) para proporcionar robustez a la rotación (Salcedo Peña, 2018) y (Zhu, Shen, & Chen, 2015). Este método funciona generando una pirámide de imágenes a diferentes escalas, esta sucesión de imágenes se representa a sí misma, pero a menores resoluciones (ver Figura 10), donde la primera columna representa el espacio de escala piramidal, la segunda y tercera columnas son *keypoints oFAST* y descriptores *ORB* de cada espacio, se descartan determinados *keypoints* con el método de esquinas de Harris y la última columna es el escalado *ORB*. Luego de la detección y descripción de *keypoints*, se asigna una orientación a cada uno dependiendo de la variación por el grado de intensidad alrededor de ese punto.

Figura 10

Secuencia piramidal de imágenes escalas por el método ORB



Nota. Esta gráfica representa una pirámide de imágenes a diferentes escalas de menor resolución. Tomado de “*ORB: an efficient alternative to SIFT or SURF*”, (Rublee, Rabaud, Konolige, & Bradski, 2011).

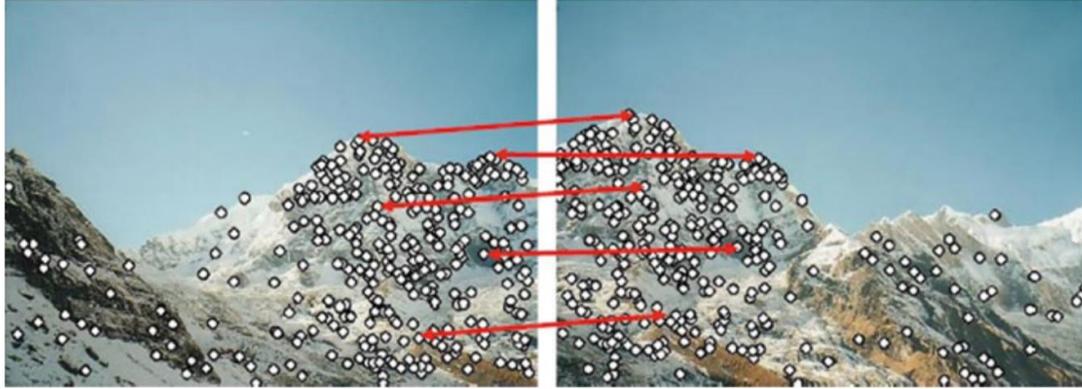
En los trabajos de investigación de tesis anteriores (Grijalva Caisachana, 2019) y (Salcedo Peña, 2018) se concluyó que el algoritmo *ORB* tienen un mejor desempeño para realizar el modelamiento servo visual procesando imágenes sucesivas del dron.

Matching de puntos de interés

Una parte importante dentro de las aplicaciones de visión artificial como el registro de imágenes o la detección de objetos, es la tarea de establecer el matching o correspondencias entre dos imágenes con un mismo objeto o escena. Este procedimiento entre fotogramas se refiere a la detección de un grupo de *keypoints*, asociado con sus descriptores de imagen. Una vez lograda la detección y descripción de *keypoints*, se debe establecer algunas coincidencias de características preliminares entre estos fotogramas (ver Figura 11).

Figura 11

Matching entre dos imágenes



Nota. Este gráfico representa el emparejamiento de puntos de interés entre dos imágenes en función de su descriptor de características local. Tomado de “*Image Features Detection, Description and Matching*”, (Hassaballah, Abdelmgeid, & Alshazly, 2016).

El proceso para la coincidencia de puntos de interés entre imágenes se puede formular considerando que p es un punto de interés localizado por un detector en una imagen, asociado con un descriptor.

$$\Phi(p) = \{\phi_k(p) \mid k = 1, 2, \dots, K\} \quad (6)$$

donde, para todo K , el vector de características entregado por el descriptor k -ésimo es:

$$\phi_k(p) = (f_{1p}^k, f_{2p}^k, f_{3p}^k, \dots, f_{n_{kp}}^k) \quad (7)$$

Se debe encontrar la mejor correspondencia q en otra imagen del conjunto de N puntos de interés representado por: $Q = \{q_1, q_2, \dots, q_N\}$ comparando el vector de características $\phi_k(p)$ con los puntos en el conjunto Q . Se hace una medición de la separación entre los dos descriptores de *keypoints* $\phi_k(p)$ y $\phi_k(q)$ definidas por:

$$d_k(p, q) = |\phi_k(p) - \phi_k(q)| \quad (8)$$

Los puntos de Q son ordenados de manera ascendente, respecto a la distancia d_k de forma independiente para cada descriptor que produce los conjuntos.

$$\Psi(p, Q) = \{\psi_k(p, Q) \mid k = 1, 2, \dots, k\} \quad (9)$$

Tal que:

$$\psi_k(p, Q) = \{(\psi_k^1, \psi_k^2, \dots, \psi_k^N) \in Q \mid d_k(p, \psi_k^i) \leq d_k(p, \psi_k^j), \forall i > j\} \quad (10)$$

Se acepta una coincidencia entre el par de puntos de interés (p, q) si se cumple con estas consideraciones:

- Si p tiene la coincidencia más alta para q considerando a todo el resto de puntos en la primera imagen.
- Si q tiene la coincidencia más alta para p considerando a todo el resto de puntos en la segunda imagen.

Según (Hassaballah, Abdelmgeid, & Alshazly, 2016), tiene mucha importancia el diseño de algoritmos eficientes para que el desarrollo de correspondencia logre una alta velocidad.

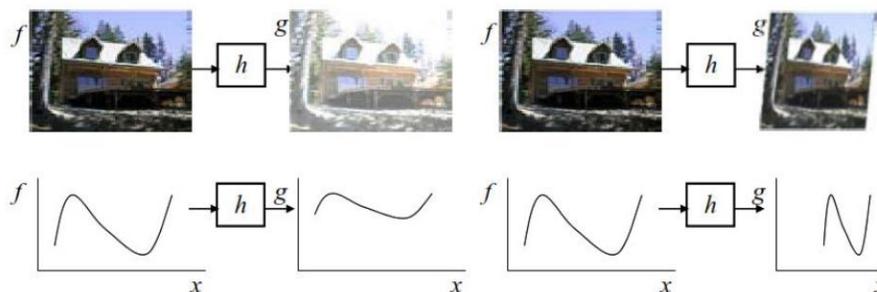
Transformada geométrica

Una transformación geométrica (Aguilar & Angulo, Real-time video stabilization without phantom movements for micro aerial vehicles, 2014), (Aguilar & Angulo, Real-Time Model-Based Video Stabilization for Microaerial Vehicles, 2016), (Aguilar & Angulo, Estabilización robusta de vídeo basada en diferencia de nivel de gris, 2013), (Aguilar & Angulo, Compensación y aprendizaje de efectos generados en la imagen durante el

desplazamiento de un robot, 2012), (Aguilar & Angulo, Compensación de los efectos generados en la imagen por el control de navegación del robot Aibo ERS 7, 2012) es una función que mapea todos los puntos de una imagen (representados por P) hacia otro dominio (representados por P'). (Grijalva Caisachana, 2019), la definición anterior puede representarse por la expresión: $g(x) = f(h(x))$ haciendo referencia a la Figura 12.

Figura 12

Imágenes modificadas resultantes por el cambio del dominio



Nota. Esta gráfica muestra el resultado visual tras la aplicación de una transformada geométrica sobre una imagen. Tomado de “*Sistema de navegación autónomo basado en visión para pistas de carreras de drones con marcas de superficie*”, (Grijalva Caisachana, 2019).

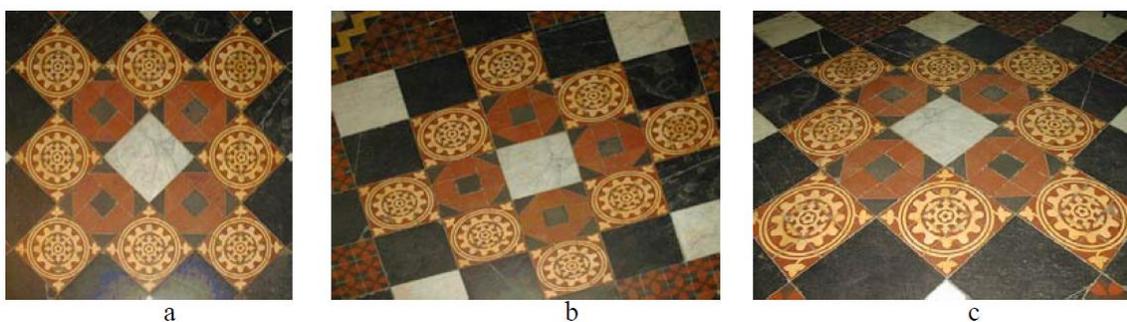
Algunos de estos cambios de dominio pueden ser por escalamiento, traslación o rotación donde sus modelos de transformaciones, acorde a la complejidad son: transformación de traslación, transformación de semejanza, transformación afín y transformación proyectiva. La Figura 13 exhibe la implementación de transformación de:

- **Similitud:** el patrón circular del piso se representa como un círculo, en cambio el mosaico cuadrado se representa como un cuadrado y las líneas que son paralelas o perpendiculares tienen la misma orientación relativa en la imagen.

- **Afín:** los círculos del piso se representan como elipses, sus líneas ortogonales dejan de serlo. Pero los lados cuadrados del mosaico, que son paralelos en la realidad, siguen siendo paralelos en la imagen.
- **Proyectivo:** las líneas paralelas en la realidad se representan como líneas convergentes, es decir que los mosaicos más cercanos a la cámara tienen una mayor dimensión que aquellos que están más alejados.

Figura 13

Imagen distorsionada de un piso de cerámica tras una transformación



Nota. Esta gráfica muestra la imagen de un piso de cerámica distorsionada por una transformación geométrica (a) de Similitud, (b) Afín y (c) Proyectiva. Tomado de “*Computer Vision: Algorithms and Applications, 1st ed.*”, (Szeliski, 2010).

Transformada de traslación

A esta transformación se la puede representar por $x' = x + t$, o como su forma matricial representada en la ecuación (11):

$$x' = [I \quad t]\bar{x} \quad (11)$$

Donde:

- I representa la matriz identidad de dimensión 2×2

- t representa al vector de traslación de dimensión 2×1

A pesar de tener una representación compacta, imposibilita el trabajo con operaciones matriciales. Una solución es el uso de coordenadas homogéneas con el fin de representar la transformación en el plano como una matriz de la forma $[0^T \quad 1]$ (Szeliski, 2010), donde:

- 0^T representa un vector de ceros de dimensión 1×2
- 1 representa una constante

Teniendo como expresión final a la ecuación (12):

$$x' = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \quad (12)$$

Transformación afín

Para definir la transformación afín hay que hablar de las transformaciones euclidianas. Las transformaciones euclidianas son un tipo de transformaciones geométricas que mantienen el valor del ángulo y la longitud. Es decir, si se toma una figura geométrica y se la aplica una transformación euclidiana, su estructura básica se conservará. Se puede manipular giros, desplazamientos, etc. en otras palabras las líneas seguirán siendo líneas, los cuadrados seguirán siendo cuadrados, los círculos seguirán siendo círculos y los planos seguirán siendo planos (Joshi, 2015).

Entonces una transformación afín se la puede definir como un caso general de las transformaciones euclidianas. Continuando con la analogía, en las transformaciones afines, los cuadrados pueden convertirse en rectángulos o paralelogramos, es decir, no se conserva longitudes ni ángulos (ver Figura 14).

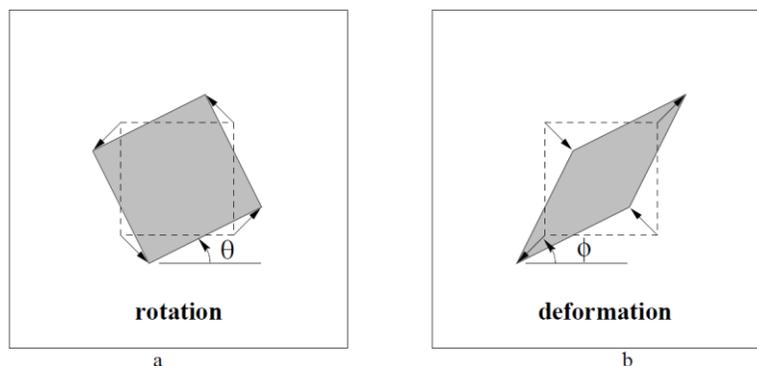
Esta transformación también es conocida como afinidad, es una transformación lineal no singular seguida de una transformación de traslación (Hartley & Zisserman, 2003). La representación matricial se expone en la expresión (13):

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \quad (13)$$

Una transformación plana afín tiene seis grados de libertad correspondientes a los seis elementos de la matriz. La transformación se puede calcular a partir de correspondencias de tres puntos.

Figura 14

Aplicación de una transformada geométrica afín



Nota. Esta gráfica representa las modificaciones a la imagen aplicando una transformación afín plana. Para (a): Rotación por $R(\theta)$ y (b): Deformación por $R(\varphi)$. Tomado de “*Multiple View Geometry in computer vision*”, (Hartley & Zisserman, 2003).

Transformada de similitud

La transformación de similitud o también denominada como similitud o rotación escalada, es una transformación del plano \mathbb{R}^2 que conserva las distancias euclidianas, compuesta con una escala isotrópica. Una transformación de similitud plana cuanta con

cuatro grados de libertad donde el factor de escala se lo considera como un grado más de libertad respecto a una transformación euclidiana. La representación matricial se expone en la expresión (14):

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{bmatrix} s \cdot \cos \theta & -s \cdot \sin \theta & t_x \\ s \cdot \sin \theta & s \cdot \cos \theta & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \quad (14)$$

Una representación más simple, en forma de bloque está dada por:

$$x' = H_s x = \begin{bmatrix} sR & t \\ 0^T & 1 \end{bmatrix} x \quad (15)$$

Donde s representa la escala isotrópica.

Transformada de perspectiva

Es una transformación lineal general, también denominada como transformación de perspectiva u homografía que trabaja con coordenadas homogéneas. En esta transformación, las líneas rectas permanecen rectas (Szeliski, 2010). Se la representa por:

$$\tilde{x}' = \tilde{H} \tilde{x} \quad (16)$$

Donde \tilde{H} representa una matriz homogénea de tres filas y tres columnas, es posible generar matrices equivalentes a \tilde{H} , si se agrega un factor de escala a la matriz. El resultado de las coordenadas homogéneas \tilde{x}' deben ser normalizadas para lograr un resultado no homogéneo denominado x , mostrado en la expresión (17):

$$x' = \frac{h_{00}x + h_{01}y + h_{02}}{h_{20}x + h_{21}y + h_{22}} \quad \wedge \quad y' = \frac{h_{10}x + h_{11}y + h_{12}}{h_{20}x + h_{21}y + h_{22}} \quad (17)$$

Capítulo III

Generalidades del sistema

Hardware del sistema

El hardware del sistema de este proyecto de investigación tiene tres componentes. El primero es un computador portátil que representa la estación en tierra. El segundo componente es el *UAV* que se comunica inalámbricamente por *Wi-Fi* con la estación en tierra a través de una red generada por el propio *UAV*. Y el último componente del sistema es la plataforma móvil terrestre para el aterrizaje del *UAV*.

Estación en tierra

Hace referencia a un PC portátil de considerables características. Se trabajó con este equipo porque sus recursos computacionales tienen la capacidad de cubrir las necesidades de procesamiento de imágenes del *UAV* en tiempo real. La Tabla 2 expone las características del equipo.

Tabla 2

Descripción de los recursos de la estación en tierra

CARACTERÍSTICAS	DESCRIPCIÓN
Marca	MSI
Modelo	GS65 Stealth 8SE
Procesador	Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz
Tipo de sistema	Sistema operativo de 64 bits, procesador x64
Memoria RAM	16,0 GB
Tarjeta gráfica	NVIDIA GeForce RTX 2060

Nota. En esta tabla se describen las características más importantes de equipo utilizado con la estación en tierra.

Estación aérea (UAV)

La estación aérea de esta investigación se refiere al UAV de código abierto llamado Bebop 2 de la marca francesa Parrot (ver Figura 15) con un controlador de vuelo que tiene una *CPU* dual-core basada en Linux con una *GPU* quad-core. Cuenta con 8 GB de memoria flash para almacenamiento de video y fotos. Cuenta con una batería *plug and play* con un tiempo de carga de 120 minutos. El UAV es de fácil transporte y sencillo montaje, con un peso de 500 gramos y una autonomía de vuelo de 25 minutos.

Figura 15

UAV Bebop 2 de la marca Parrot



Nota. En esta gráfica se muestra el aspecto físico del drone usado en esta investigación. Tomado de <https://www.parrot.com/es/drones/parrot-bebop-2>, (Parrot Bebop 2 Drone, 2020).

Puede filmar en full HD 1080p con un lente gran angular de 14 megapíxeles con un sistema de estabilización de imagen digital. Su lente de 180° permite que la imagen no pierda la orientación de su vista sin importar que el drone se incline o sufra de perturbaciones. Logra velocidades de hasta 60 *km/h* en horizontal y de casi 22 *km/h* en vertical sin afectar la calidad de la imagen.

La empresa Parrot publicó un *SDK (Software Development Kit)*, que dió paso a la creación del driver llamado “*bebop_autonomy*” el cual permite interactuar con el dron a mediante Linux, utilizando *ROS (Robot Operating System)*.

Especificaciones generales

En la Tabla 3 y en la Figura 16 se encuentra un resumen de las especificaciones generales de dron Bebob 2.

Tabla 3

Especificaciones generales del dron Bebob 2

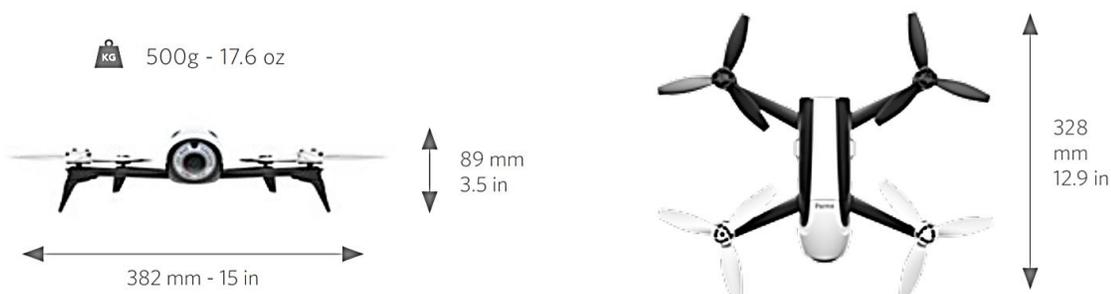
Estabilización de imagen	Estabilización digital (sistema Parrot)
Batería	25 minutos de tiempo de vuelo (batería Li-Po de 2700 mAh)
Sensores	GPS + IMU
Almacenamiento	Sistema de almacenamiento flash de 8 GB
Controlador	Procesador dual core con una GPU quad core
Motores	4 motores de jaula giratoria sin escobillas. Máximo 12000 rpm
Peso	500 gramos (cuerpo dron) + 192 gramos (batería)

Nota. Esta tabla se describe las características generales del UAV usado. Tomado de <https://www.parrot.com/es/drones/parrot-bebob-2>, (Parrot Bebob 2 Dron, 2020).

Las dimensiones junto al peso del dron, hacen a este vehículo de fácil transporte, sus hélices son flexibles y están programadas para bloquearse en caso de contacto o colisión.

Figura 16

Dimensiones de drone Bebop 2 de Parrot



Nota. Este gráfico señala las dimensiones del largo, alto y ancho de drone Bebop 2. Tomado de <https://www.parrot.com/es/drones/parrot-bebop-2>, (Parrot Bebop 2 Drone, 2020).

Conectividad con el drone

La conexión y pilotaje del *UAV* se la puede hacer de tres maneras, tomando en cuenta que todas ellas utilizan una conexión inalámbrica *Wi-Fi* con la propia red generada por el drone Bebop 2.

- Usando la aplicación gratuita de la empresa Parrot llamada *Freeflight* instalada en un dispositivo móvil inteligente (*Android* o *IOS*) con un alcance máximo de 300 metros con línea de vista.
- Usando el control remoto llamado Parrot Skycontroller 2 que integra una antena para aumentar el alcance máximo hasta 2 km con línea de vista.
- Usando el driver llamado "*bebop_autonomy*" creado por el *SDK (Software Development Kit)* de Parrot. El alcance de este método está limitado la antena de la estación en tierra.

En la Tabla 4 se muestra algunas características adicionales del *UAV* relacionadas con su conexión a la estación en tierra o al equipo móvil que lo controlará.

Tabla 4

Comunicación inalámbrica del UAV

CARACTERÍSTICA	DESCRIPCIÓN
Tipo de Conexión inalámbrica	Wi-Fi IEEE 802.11 a /b /g /n /ac
Antenas Wi-Fi	Antenas dipolares duales de 2.4 y 5 GHz
Network	MIMO Dual band
Rango de la señal	300 m con dispositivo móvil 2 km con control remoto Parrot Skycontroller 2

Nota. La tabla describe especificaciones de conexión entre el *UAV* y el dispositivo de control. Tomado de <https://www.parrot.com/es/drones/parrot-bebop-2>, (Parrot Bebop 2 Drone, 2020).

Características de la cámara

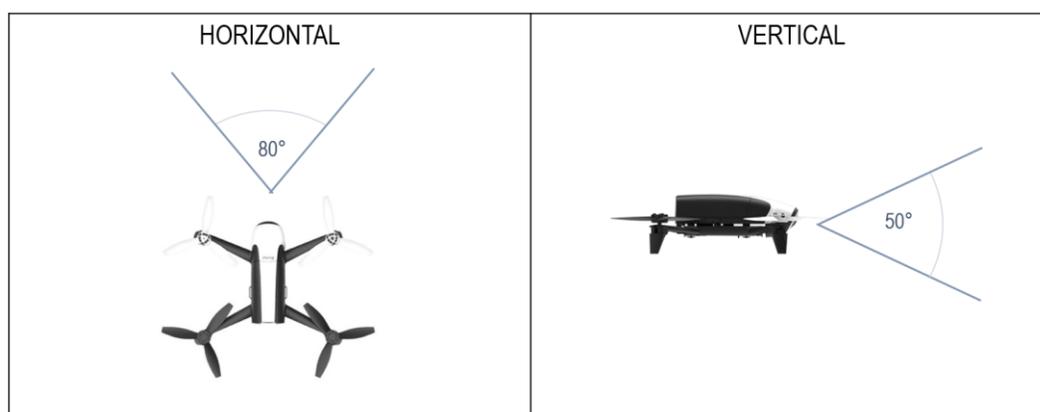
La cámara del dron puede generar una vista cuasi cenital de la escena sobre la que está volando, esto permite una observación con un campo de visión de 180° de su posición. Pero Parrot no permite el acceso a la imagen global de la cámara durante el streaming de video porque lo utiliza para la estabilización digital de video. La Figura 17 expone el rango permitido de movimiento digital de la cámara del *UAV*.

En la Tabla 5 se resume las características técnicas de la cámara del Bebop 2, tanto para captura de videos y fotografías.

Tabla 5*Características de la cámara del Bebop 2*

CARACTERÍSTICAS	DESCRIPCIÓN
Cámara	14 mega pixeles con sensor CMOS
Estabilización de video	Sistema digital de 3 ejes
Resolución de video	1920 × 1080p a (24, 25 o 30 fps)
Codificación de video	H264
Resolución de fotografía	4096 × 3072 pixeles
Formato de fotografía	JPEG, RAW, DNG

Nota. Esta tabla describe las características de la cámara y de la captura de fotografías y videos de Bebop 2.

Figura 17*Rango de movilidad digital de la cámara del drone Bebop 2*

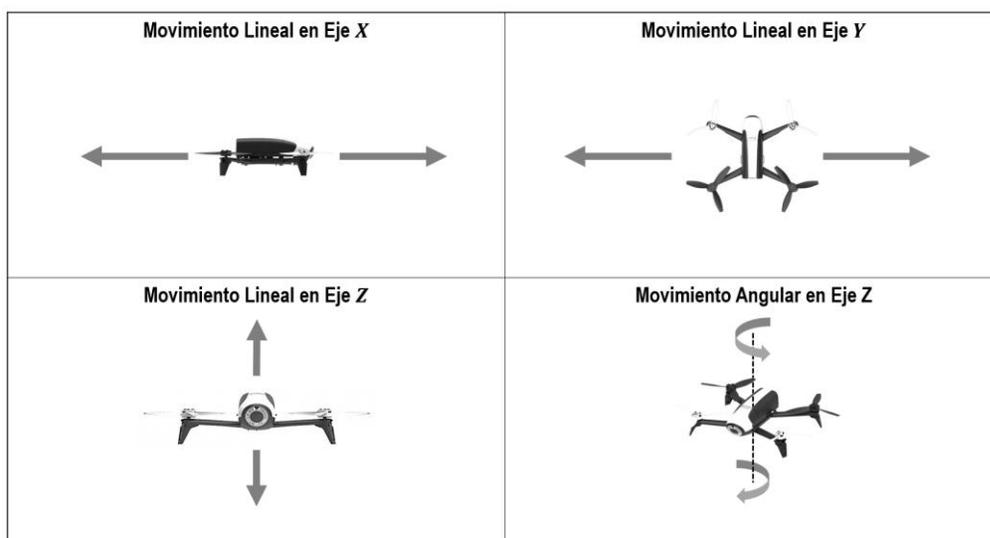
Nota. Esta gráfica muestra el rango de movimiento de la cámara del UAV expresado en grados.

Características de movimiento

Un cuadricóptero tiene seis grados de libertad, tres movimientos de traslación y tres movimientos de rotación. Los tres primeros permiten desplazar linealmente al dron de: atrás hacia adelante (eje X), de izquierda a derecha (eje Y) y en ascenso – descenso (eje Z).

Figura 18

Movimiento lineal y angular del Bebop 2



Nota. Esta gráfica representa visualmente el movimiento lineal y angular que puede ejecutar el UAV.

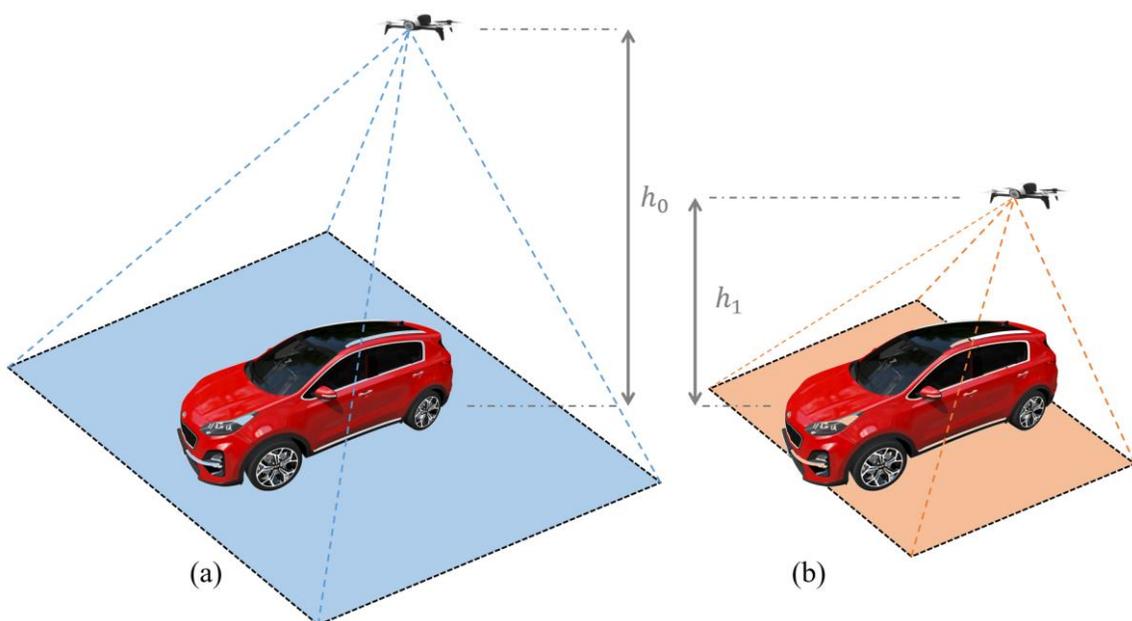
En cambio, los movimientos de rotación permiten al dron rotar: alrededor del eje X (*Pitch*), alrededor de eje Y (*Roll*) y alrededor del eje Z (*Yaw*). El control de movimientos del Bebop 2 permite controlar solamente cuatro grados de libertad, los tres movimientos lineales de traslación y uno angular en el eje Z (*Yaw*) (ver Figura 18).

Plataforma móvil terrestre

Utilizando visión por computadora y algoritmos de control, el *UAV* hace un seguimiento del vehículo y progresivamente desciende hasta aterrizar sobre la parte superior del chasis del mismo. El vehículo mantendrá una trayectoria rectilínea limitando su velocidad máxima a 5 km/h .

Figura 19

Área de visión de la escena respecto a la altura



Nota. La gráfica muestra la visión del vehículo para una altura (a) h_0 y (b) h_1 del UAV.

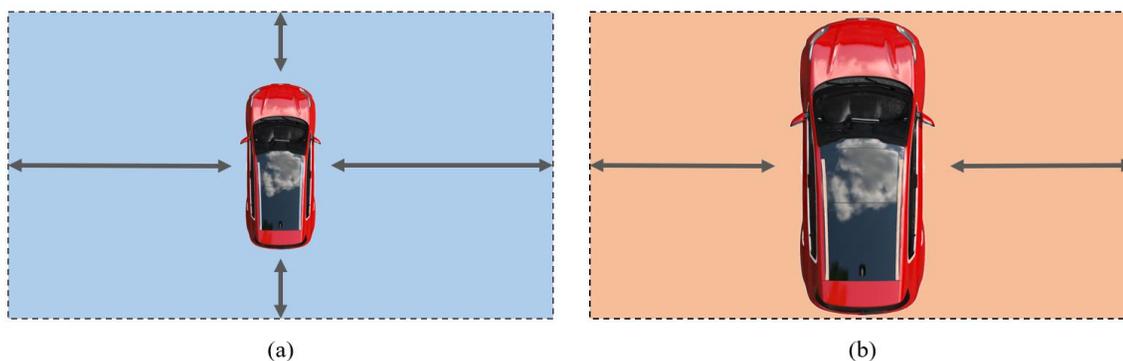
La visión del *UAV* durante el vuelo captura y envía imágenes cuasi perpendiculares al piso, esto facilita el seguimiento de la plataforma móvil, pero ocasiona importantes problemas de visión para la detección de objetos en un rango de alturas menor a un valor referencial (h_1). En la Figura 19 se ejemplifica el área de visión de la cámara del *UAV*, donde h_1 representa la altura mínima desde el suelo hasta el dron, para que la imagen pueda incluir toda la vista superior del vehículo. Si la imagen no

incluye completamente el contorno de la plataforma de aterrizaje, la detección no podrá ser realizada eficazmente, ocasionando problemas que afectan directamente el rendimiento del algoritmo de seguimiento y control del UAV.

En la Figura 19 (a), el área de visión de la cámara para una altura del drone h_0 , es suficientemente amplia para detectar al vehículo con una separación adecuada a los límites superior e inferior del marco de la imagen, esto facilita la identificación de la plataforma de aterrizaje del resto de la escena. En cambio, la Figura 18 (b) muestra una altura h_1 donde el área de visión también cubre completamente el contorno del vehículo pero sus bordes se encuentran muy cerca, tanto del límite superior como del inferior del marco de la imagen (ver Figura 20). Es decir, la altura h_1 es el valor crítico antes de comenzar a perder parcialmente la vista superior completa del vehículo.

Figura 20

Vista cenital del vehículo a diferentes alturas



Nota. Esta gráfica muestra el área de visión del vehículo con una vista cenital del UAV para una altura de: (a) h_0 y (b) h_1

En este escenario, la detección del objeto está directamente comprometida, provocando que los algoritmos de seguimiento y aterrizaje no cuenten con la información

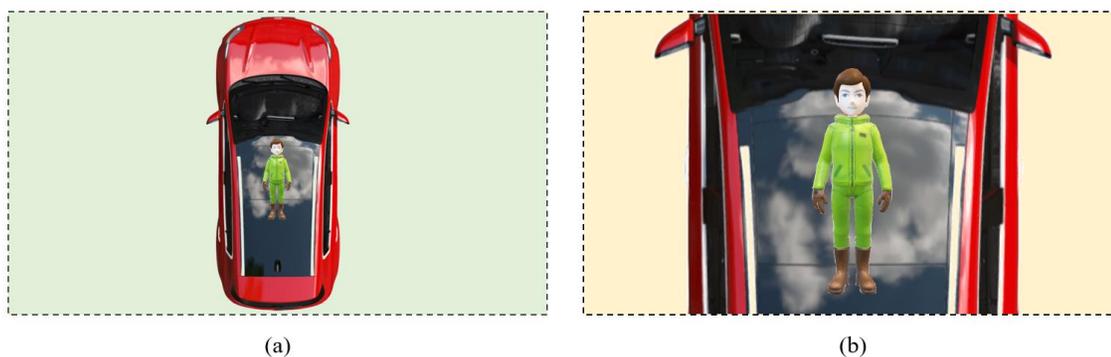
de entrada adecuada, porque la detección del objeto habrá generando un target erróneo o incluso no será generado, privando a los algoritmos de información real de la posición del *UAV* respecto a la plataforma móvil.

Una de las opciones probadas para dar solución a este problema fue obligar al *UAV* a aterrizar en el instante que el algoritmo de detección haya perdido de vista a la plataforma o, dicho de otra manera, que el *UAV* haya descendido bajo la altura h_1 (alrededor de 5 metros sobre el piso). Pero los resultados no fueron satisfactorios, la distancia entre el techo del vehículo y el dron era muy amplia provocando una descoordinación muy grande para un posicionamiento aceptable del *UAV* sobre del techo del vehículo.

Target en la plataforma móvil de aterrizaje

Figura 21

Vista cenital del nuevo target a diferentes alturas



Nota. Este gráfico representa la visualización del target con una cenital del *UAV* para: (a) altura h_0 y (b) altura menor a h_1 .

La solución de pérdida parcial y gradual de la visión del contorno del vehículo durante el descenso del *UAV* a un valor inferior a la referencia (h_1), fue colocar una

imagen en la parte superior del vehículo, que será considerada como el nuevo objetivo de seguimiento (*target*) para la etapa final del aterrizaje. La Figura 21 muestra la plataforma móvil con el nuevo *target* en la parte superior del chasis. De esta manera se puede asegurar un descenso bastante cercano a la parte superior del vehículo sin perder de vista el objetivo de seguimiento.

Software del sistema

Esta investigación trabaja con un Sistema Operativo (SO) libre y de código abierto de la distribución *Linux* denominado *Ubuntu 16.04 LTS*.

Figura 22

Esquema de comunicación entre la PC y la estación aérea



Nota. Esta gráfica representa el sistema operativo y softwares necesarios para el desarrollo de la investigación.

Este sistema tiene compatibilidad con diferentes softwares, lenguajes de programación y *frameworks* para el diseño de algoritmos de control mediante la comunicación entre el UAV (Bebop 2 de Parrot) y la estación en tierra (computador

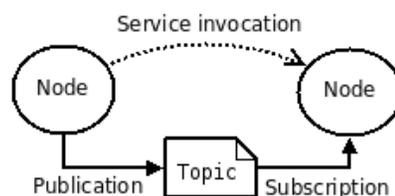
portátil) trabajando principalmente con el procesamiento de imágenes (ver Figura 22). El entorno de comunicación que se utilizó en esta investigación es ROS (*Robot Operating System*). Algo muy importante, es la compatibilidad de *Ubuntu* para trabajar con los drivers de la tarjeta de video (NVIDIA GEFORCE RTX 2060) de la computadora portátil para el procesamiento de imágenes en tiempo real. El algoritmo de control necesita conocer la posición inmediata del *UAV* respecto al automóvil en movimiento para la rápida acción al corregir el error de posición durante el seguimiento y aterrizaje.

Robot Operating System (ROS)

ROS es un entorno de trabajo (*framework*) desarrollado para la creación de software para robots, está basado en grupos de librerías, convenciones y herramientas que facilitan el desarrollo de software robusto para sistemas robotizados. Proporciona servicios similares a un SO como implementación de funcionalidades frecuentemente utilizadas para robots, transmisión de mensajes entre procedimientos, control de equipos de bajo nivel, entre otros.

Figura 23

Representación del flujo de información en ROS



Nota. Este gráfico representa la comunicación entre nodos usando suscriptores y publicadores. Tomado de <http://wiki.ros.org/ROS/Concepts>, (ROS wiki AaronMR, 2014).

Está basado en arquitectura de grafos que trabaja con multiplexación de información; su procesamiento se realiza en “nodos” independientes, estos pueden

publicar y suscribirse a mensaje de otros nodos a través de tópicos (ver Figura 23), llamar o proporcionar servicios a otros nodos, provenientes de estados, planificaciones, señales de control, sensores, etc. De esta manera fomenta una comunidad de diversos paquetes creados por usuarios ayudando al desarrollo de soluciones, desde implementación de pruebas de concepto hasta algoritmos con capacidades y calidades industriales.

Conceptos ROS

ROS divide sus conceptos en tres niveles: nivel de sistema de archivos, nivel de cómputo gráfico y nivel de comunidad. Estos niveles y conceptos están claramente detallados en (ROS wiki AaronMR, 2014).

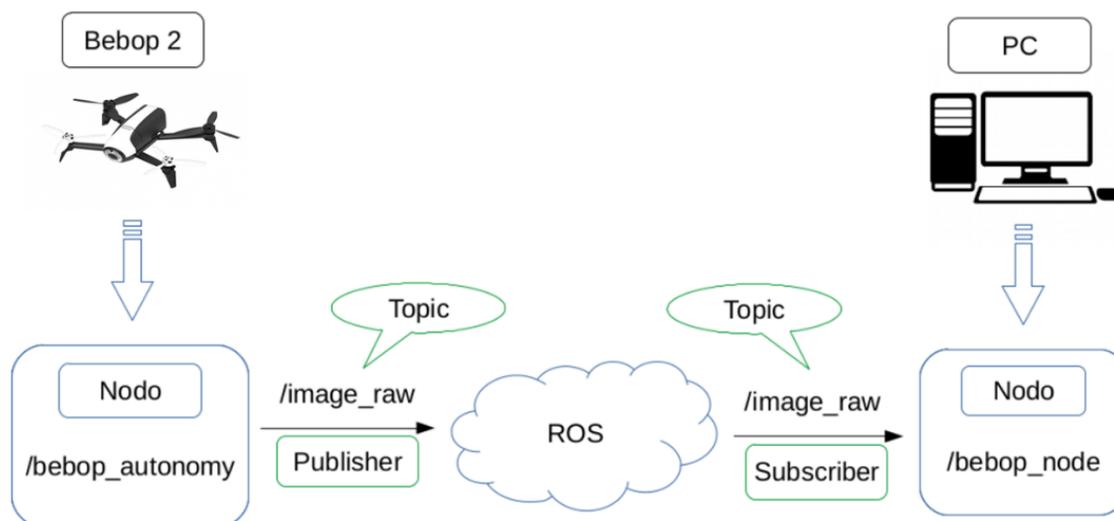
- **Nivel de sistema de archivos:** Sus conceptos cubren principalmente los recursos ROS localizados en el disco, como:
 - **Paquetes:** Es el mecanismo principal de organización del software en ROS. Un paquete ROS puede integrar procesos, librerías dependientes de ROS, grupos de archivos de configuración, datos o diferentes elementos que estén organizados de manera útil.
 - **Meta paquetes:** Son paquetes especializados con el único objetivo de representar un conjunto de distintos paquetes relacionados.
 - **Manifiestos de paquetes (package.xml):** Proporcionan metadatos acerca de un paquete, incluido su nombre, descripción, versión, dependencia, información de licencia y otra información meta como paquetes exportados.
 - **Repositorios:** Es una recopilación de paquetes que comunican un Sistema de Control de Versiones (VCS) común. Es posible que los repositorios puedan contener un solo paquete.

- **Tipos de mensaje (msg):** Son organizaciones de datos usados para el envío de mensajes en ROS.
- **Tipos de servicio (srv):** Son organizaciones de datos de respuesta y solicitud usados para los servicios en ROS.
- **Nivel de cómputo gráfico:** El cómputo gráfico es la red de procesos ROS que trabajan con datos punto a punto. Esta red está conformada por algunos conceptos básicos de cómputo gráfico descritos a continuación:
 - **Nodo (Node):** Son procesos encargados del cálculo o computación, por ejemplo, el control de motores, planificación de rutas, etc.
 - **Maestro (Master):** O nodo principal, proporciona un registro de nombres y búsquedas al resto del cómputo gráfico. Sin el maestro, los nodos no podrían encontrarse entre sí, intercambiar mensajes o invocar servicios.
 - **Servidor de parámetros (Parameter Server):** Es parte del nodo Maestro para almacenar datos en una ubicación central.
 - **Mensajes (Messages):** Es el medio de comunicación de los nodos. Un mensaje es básicamente una organización de datos formada por diferentes campos de datos (punto flotante, entero, booleano, entre otros).
 - **Tópico (Topic):** Es una etiqueta o identificador del contenido del mensaje enrutados mediante un sistema de transporte como publicadores o suscriptores. El nodo despacha un mensaje publicándolo en un tópico determinado y un nodo interesado a un tipo determinado de dato se suscribirá al tópico adecuado. Existen varios suscriptores y publicadores simultáneos para un solo tópico, pero un solo nodo puede publicar suscribirse o publicar en varios tópicos. La Figura

24 ejemplifica la utilización de tópicos en la comunicación entre el UAV y la estación en tierra.

Figura 24

Comunicación entre Bebop 2 y la estación en tierra



Nota. Este gráfico representa el proceso y elementos de comunicación de entre el UAV y la estación en tierra usando ROS. Tomado de “*Desarrollo de aplicaciones basadas en visión con entornos ROS para el Drone Bebop 2*”, (Valero Lavid, 2017).

- **Servicios (Services):** Establecen una comunicación de tipo síncrona entre nodos. Conformados por mensajes, uno de Request (solicitud) y otro de Reply (respuesta), refiriéndose a un nodo que pide llamar a un servicio, tiene que hacerlo y esperar la respuesta procedente del otro nodo.
- **Bolsa (Bag):** Es un componente especial para guardar datos necesarios para el diseño e implementación de algoritmos.
- **Nivel de comunidad ROS:** El concepto de la comunidad ROS se refieren a compartir recursos, conocimientos y software ROS entre comunidades

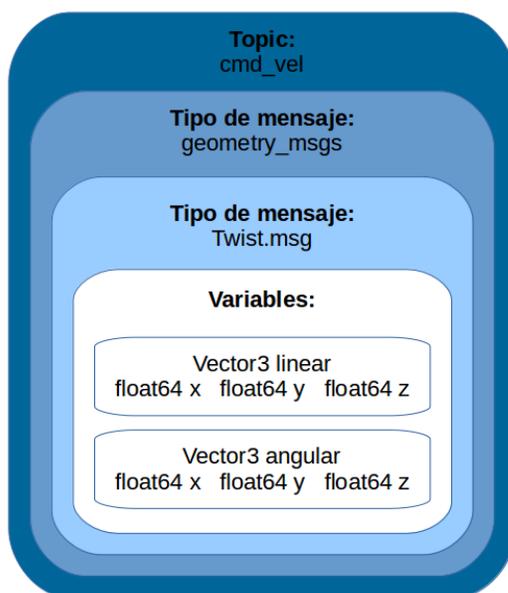
separadas. Estos recursos incluyen: Distribuidores, Repositorios, The ROS Wiki, Sistemas de entrada de errores, Listas de correo, ROS Answers y Blogs.

Comunicación ROS

La comunicación de ROS trabaja con una arquitectura Maestro-Esclavo (Master-Slave) donde primero se debe definir el equipo Maestro que entregue datos de comunicación a nodos para que todos los dispositivos que necesiten ejecutarlos (Esclavos) establezcan una conexión con el Maestro.

Figura 25

Esquema interno del tópico para el control de velocidad



Nota. Este gráfico representa el tópico de ROS para definir la velocidad del UAV. Tomado de “Desarrollo de aplicaciones basadas en visión con entornos ROS para el Drone Bebop 2”, (Valero Lavid, 2017).

El envío de mensajes de nodo a nodo como parámetros en streaming está dada por rutas de información denominados tópicos. Los nodos deben publicar o suscribirse a un tópico

para completar el canal del mensaje (comunicación unidireccional). Por el contrario, si se necesita un tipo de comunicación sincrónica bidireccional se debe utilizar los servicios con un formato de Solicitud-Respuesta entre nodos. Al operar a bajo nivel, las sentencias de control de sistemas sofisticados con robos estarán compuestos por varios nodos.

Hay que recordar que se puede transmitir diferentes tipos de mensajes, pero se debe utilizar varios tópicos para enviar tipos específicos de información. Por ejemplo, al utilizar el tópico *cmd_vel* en para publicar las velocidades deseadas para el vuelo del UAV, se especifica un mensaje de tipo *geometry_msgs/Twist.msg* que está conformados por dos vectores, uno para la comunicar la velocidad lineal y el otro para la velocidad angular en los ejes *X,Y,Z*. La Figura 25 representa visualmente el ejemplo antes mencionado.

Comunicación entre ROS y Bebop 2

El controlador de ROS llamado “*bebop_autonomy*” fue desarrollado para los drones (cuadricópteros) de Parrot Bebop 1.0 y Bebop 2.0 basados en el SDK oficial ARDroneSDK3 de Parrot. Fue desarrollado en el Autonomy Lab de la Universidad Simon Fraser por Mani Monajjemi y otros participantes. Los encargados del soporte para este software son *Autonomy Lab, Dynamic Systems Lab, University of Toronto* y *Advanced Interactive Technologies Lab*. Una descripción detallada de este controlador se encuentra en la página oficial de ROS (Monajjemi, 2015).

La implementación del controlador “*bebop_autonomy*” generará una serie de tópicos para entablar la comunicación entre el cuadricóptero (Bebop 2) y la estación en tierra, esto permitirá la creación de publicadores y suscriptores con la finalidad de enviar instrucciones al dron y recibir información del mismo respectivamente.

El primer paso es acceder al directorio de trabajo dentro de *Ubuntu* donde se haya instalado *ROS*, la carpeta se llamada *bebop* contiene al controlador “*bebop_autonomy*”. El archivo ejecutable es de tipo *.launch* está dentro de la carpeta *launch* (ver Figura 26).

Figura 26

Directorio del archivo ejecutable del controlador bebop_autonomy



Nota. Este gráfico representa la disposición de carpetas generadas por el framework *ROS* donde se encuentran carpetas de configuración del *UAV*.

Antes de ejecutar el archivo *launch*, la computadora portátil debe estar conectada a la red *Wi-Fi* del *Bebop 2*, luego se debe abrir una terminal en *Ubuntu* para ingresar el comando *roslaunch* seguido del nombre del paquete más el nombre del archivo *launch*, de esta forma:

```
roslaunch bebop_driver bebop_node.launch
```

Al haber ejecutado el controlador, se crean varios tópicos para conocer diferentes aspectos del dron e o enviar señales de control al mismo. Una descripción detallada de todas las acciones que presenta este driver se encuentra en la página oficial “*bebop_autonomy - ROS Driver for Parrot Bebop Drone (quadrocopter) 1.0 & 2.0*” (Monajjemi, 2015).

En la Figura 26 se muestra los archivos generados tras la instalación del controlador, dentro de *bebop_autonomy* se encuentra la carpeta *bebop_driver* que contiene varios archivos, entre estos existen dos específicos (*cfg*) para realizar la configuración de los parámetros de vuelo, desplazamiento, etc. del Bebop 2. La Tabla 6 basada en (Grijalva Caisachana, 2019), (Valero Lavid, 2017) y (Salcedo Peña, 2018) describe estos parámetros:

Tabla 6

Descripción de Parámetros configurables para el Bebop 2

Tipo de parámetro	Descripción y tipo de variable	Nombre del parámetro
Pilotaje	Altitud máxima en [m] (tipo Double)	<i>PilotingSettingsMax</i> <i>AltitudeCurrent</i>
	Ángulo máximo de inclinación en [°] (tipo Double)	<i>PilotingSettingsMax</i> <i>TiltCurrent</i>
	Distancia máxima de separación desde el punto de partida en [m] (tipo Double)	<i>PilotingSettingsMax</i> <i>DistanceValue</i>

Tipo de parámetro	Descripción y tipo de variable	Nombre del parámetro
	Altitud mínima en [m] (tipo Double)	<i>PilotingSettingsMinAltitudeCurrent</i>
Movimiento	Velocidad máxima vertical en [m/s] (tipo Double)	<i>SpeedSettingsMaxVerticalSpeedCurrent</i>
	Velocidad de rotación máxima en [°/s] (tipo Double)	<i>SpeedSettingsMaxRotationSpeedCurrent</i>
	Velocidad de rotación máxima en pitch y roll en [°/s] (tipo Double)	<i>SpeedSettingsMaxPitchRollRotationSpeedCurrent</i>
Red	Tipo de red Wi-Fi (tipo Int, donde 0: Selección automática y 1: Selección manual)	<i>NetworkSettingsWifiSelectionType</i>
Imagen	Número de fotogramas por segundo (tipo Int, donde 0: 23.967 fps, 1: 25 fps y 2: 29.97fps)	<i>PictureSettingsVideoFramerateFramerate</i>
	Resolución del video (tipo Int, donde 0: Grabación a 1080p – Streaming a 480p y 1: Grabación a 720p – Streaming a 720p)	<i>PictureSettingsVideoResolutionsType</i>

Nota. Esta tabla describe el tipo de variables para configurar los parámetros de pilotaje, movimiento, red e imagen.

Publicadores de clase

Para el diseño del algoritmo de control es necesario conocer ciertos parámetros del dron, así como también enviar mensaje de control de desplazamiento y rotación dependiendo de las circunstancias en las que se encuentre. Para lograr esta

comunicación se creó una clase que contiene a los suscriptores y publicadores. La creación de los publicadores dentro de la clase necesita especificar tres argumentos:

- Tópico en donde se va a publicar el mensaje.
- Tipo de mensaje.
- Tiempo de espera máximo antes de publicar el mensaje, argumento etiquetado con el nombre *queue_size*.

En la Tabla 7 se describen los publicadores utilizados en este proyecto.

Tabla 7

Descripción de los publicadores utilizados en el algoritmo de control

Nombre del Publicador	Descripción	Tópico (T) Mensaje (M)
Pub_Twist	<p>Se encarga de controlar los movimientos del dron. Trabaja con los siguientes argumentos:</p> <ul style="list-style-type: none"> • Linear.x: valores entre $[-1.0 a 1.0]$ <ul style="list-style-type: none"> ○ (+) Adelante. ○ (-) Atrás. • Linear.y: valores entre $[-1.0 a 1.0]$ <ul style="list-style-type: none"> ○ (+) Izquierda. ○ (-) Derecha. • Linear.z: valores entre $[-1.0 a 1.0]$ <ul style="list-style-type: none"> ○ (+) Abajo. ○ (-) Arriba. • Angular.x: valores entre $[-1.0 a 1.0]$ <ul style="list-style-type: none"> ○ (+) Inclinación en Roll horario. 	<p>T: /bebop/cmd_vel M: geometry_msgs.msg.Twist</p>

Nombre del Publicador	Descripción	Tópico (T) Mensaje (M)
	<ul style="list-style-type: none"> ○ (–) Inclinación en Roll anti horario. • Angular.y: valores entre [–1.0 a 1.0] ○ (+) <i>Inclinación en Pitch horario.</i> ○ (–) <i>Inclinación en Pitch anti horario.</i> • Angular.z: valores entre [–1.0 a 1.0] ○ (+) Giro en el propio eje horario. ○ (–) Giro en el propio eje anti horario. 	
Pub_Takeoff	Se encarga del despegue del drone.	T: /bebop/takeoff M: std_msgs.msg.Empty
Pub_Land	Se encarga del aterrizaje del drone.	T: /bebop/land M: std_msgs.msg.Empty
	Se encarga del controlar la posición virtual de la cámara frontal del UAV. Trabaja con los siguientes argumentos:	
Pub_Cam	<ul style="list-style-type: none"> • Angular.y: valores entre [–80 a 80] ○ (+) tilt abajo. ○ (–) tilt arriba. • Angular.z: valores entre [–80 a 80] ○ (+) panning derecha. ○ (–) panning izquierda. 	T: /bebop/camera_control M: geometry_msgs.msg.Twist
Pub_Record	Se encarga de realizar grabaciones en video del vuelo del drone en primera persona.	T: /bebop/record M: std_msgs.msg.Bool

Nota. Esta tabla muestra los tipos de publicadores para generar acciones de control sobre el UAV y la configuración de las variables utilizadas.

Suscriptores de clase

Los suscriptores también proporcionan información necesaria para la generación automática de los movimientos del dron para su seguimiento y aterrizaje. Su trabajo es la recepción de la información de posición y de las imágenes provenientes del dron. La creación de suscriptores dentro de la clase necesita especificar tres argumentos:

- El tópico al que se va a suscribir
- El tipo de mensaje
- La función que se ejecutará en un hilo al crear el suscriptor, llamada *callback*.

En la Tabla 8 se describen los suscriptores utilizados en este proyecto.

Tabla 8

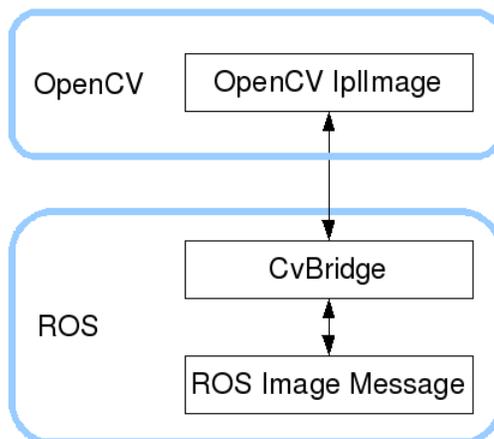
Descripción de los suscriptores utilizados para algoritmo de control

Nombre del suscriptor	Descripción	Tópico (T) Mensaje (M)
Sub_Odom	Obtiene los fotogramas de odometría. El valor de interés es la altura actual del dron.	T: /bebop/odom M: nav_msgs.msg.Odometry
Sub_Image	Obtiene las capturas de la cámara principal del dron. Estas imágenes de adquieren en mensajes de ROS que necesitan ser interpretados desde Python usando la librería <i>cv_bridge</i> transformado el mensaje a un array de <i>NumPy</i> (ver Figura 27).	T: /bebop/image_raw M: sensor_msgs.msg.Image

Nota. Esta tabla muestra los suscriptores de ROS para obtener los datos de odometría del UAV y las imágenes de la cámara del mismo.

Figura 27

Conversión entre mensaje ROS a imágenes OpenCV



Nota. Esta gráfica muestra un diagrama de bloques para transformar los mensajes tipo ROS de las imágenes enviadas por el UAV a imágenes compatibles y manipulables con OpenCV. Tomado de “*Sistema de navegación autónomo basado en visión para pistas de carreras de drones con marcas de superficie*”, (Grijalva Caisachana, 2019).

Capítulo IV

Detección de la plataforma de aterrizaje

Se describe el sistema de reconocimiento de la plataforma móvil de aterrizaje terrestre usando los fotogramas del streaming de video del *UAV* procesadas en la estación en tierra. Este sistema de detección de objetos llamado *YOLO* (You Only Look Once) genera cuadros (*boxes*) para señalar las detecciones y mostrar su posición dentro de las imágenes procesadas. Además, se presenta la implementación de algunas herramientas de *YOLO* dentro del algoritmo de seguimiento y aterrizaje automático del *UAV* sobre el target.

Sistema de detección *YOLO*

YOLO es un sistema de detección de objetos en tiempo real de última generación de libre acceso relativamente simple. Trabaja con una sola red convolucional que divide la imagen en regiones y predice simultáneamente múltiples cuadros delimitadores (*bounding box*) y probabilidades de coincidencia para cada región. Los cuadros delimitadores están ponderados por las probabilidades predichas.

El proceso de detección de objetos de *YOLO* está compuesto por tres tareas (ver Figura 28). Estas son:

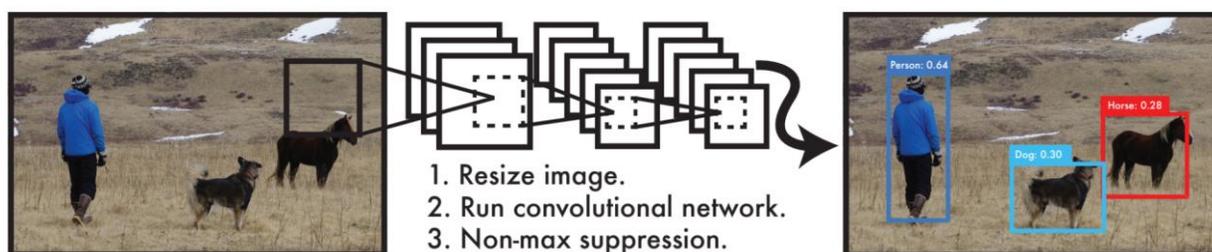
1. Redimensionamiento del fotograma de entrada.
2. Ejecutar la única red convolucional sobre el fotograma de entrada.
3. Filtrar el total de las detecciones resultantes dependiendo del nivel de confianza del modelo.

El entrenamiento de *YOLO* se lo realiza en imágenes completas, optimizando directamente el rendimiento de detección. Su modelo unificado tiene muchas ventajas

frente a sistemas diseñados con clasificadores (Redmon, Divvala Santosh, Girshick, & Farhadi, 2016) y (Redmon & Farhadi, 2018). Es extremadamente rápido dado que etiqueta el reconocimiento de objetos como un problema de regresión. Simplemente se ejecuta la red neuronal en una nueva imagen al momento de la prueba para predecir las detecciones, logrando más del doble de la precisión media promedio de otros sistemas en tiempo real. Actúa sobre una escena completa al instante de ejecutar la prueba para que sus predicciones estén informadas por el contexto global de la escena, es decir, trabaja sobre la imagen completa al hacer predicciones, las observa enteras a lo largo del entrenamiento y pruebas, codificando implícitamente información contextual sobre las clases y su apariencia.

Figura 28

Sistema de detección de YOLO



Nota. Esta gráfica representa el proceso de detección de imágenes que utiliza YOLO.

Tomado de “*You Only Look Once: Unified, Real-Time Object Detection*”, (Redmon, Divvala Santosh, Girshick, & Farhadi, 2016).

Tiene la capacidad de aprender representaciones generalizables de objetos, cuando se entrena en imágenes naturales y se prueba en ilustraciones. Hace predicciones con una sola evaluación de la red al contrario de sistemas como *Region-based Convolutional Network* (R-CNN) que necesitan miles para una sola escena. En su versión *YOLOv3* logra resultados considerablemente rápidos, más de mil veces más

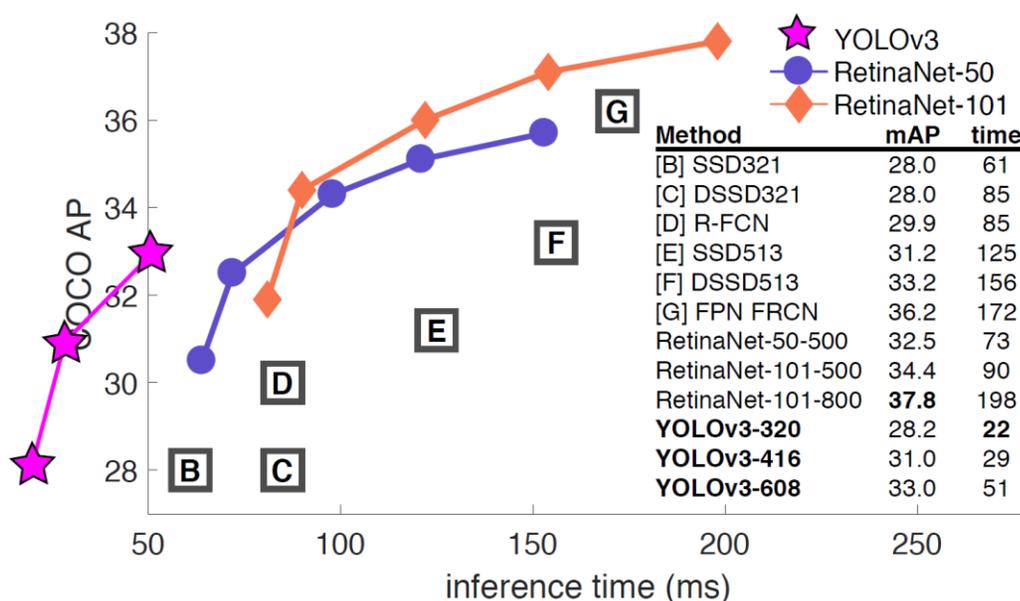
rápido que R-CNN y cien veces más rápido que *Fast Region-based Convolutional Network (Fast R-CNN)*. Al ser un sistema altamente generalizable tiene una menor probabilidad de error al aplicarlos a nuevos dominios o entradas inesperadas. A pesar de lograr identificar rápidamente objetos en diferentes escenarios, tiene dificultades para localizar con precisión algunos objetivos, especialmente los pequeños.

Comparación de YOLOv3 con otros detectores

YOLOv3 es extremadamente rápido y preciso. En mAP medido a 0.5 (ver Figura 29), este sistema está al mismo nivel de *Focal Loss* pero aproximadamente 4 veces más rápido. Además, es fácil intercambiar entre velocidad y precisión en la detección simplemente modificando el tamaño del modelo (no se requiere reentrenamiento).

Figura 29

Comparación de YOLO con otros detectores



Nota. Esta gráfica representa el desempeño del sistema de detección de objetos en tiempo real YOLO frente a otros detectores. Tomado de “YOLOv3: An incremental improvement”, (Redmon & Farhadi, 2018).

- **AP (Average Precision)**

Es una métrica popular para medir la precisión de detectores de objetos como *Faster R-CNN*, *SSD*, etc. La precisión promedio (*average precision*) calcula el valor de precisión promedio para obtener una magnitud (*recall*) entre 0 a 1.

- **Precisión**

Dicta cuán precisas son las predicciones. Es decir, qué porcentaje de las predicciones son correctas.

$$\text{Precisión} = \frac{TP}{TP + FP} = \frac{TP}{\text{Todas las detecciones}} \quad (18)$$

Donde:

- $TP = \text{Verdadero Positivo}$
- $FP = \text{Falso Positivo}$

- **Recall**

Mide qué tan buenas se encuentran las detecciones positivas. Por ejemplo, se puede encontrar un 80% de los posibles casos positivos en la predicción principal.

$$\text{Recall} = \frac{TP}{TP + FN} = \frac{TP}{\text{Toda verdad fundamental}} \quad (19)$$

Donde:

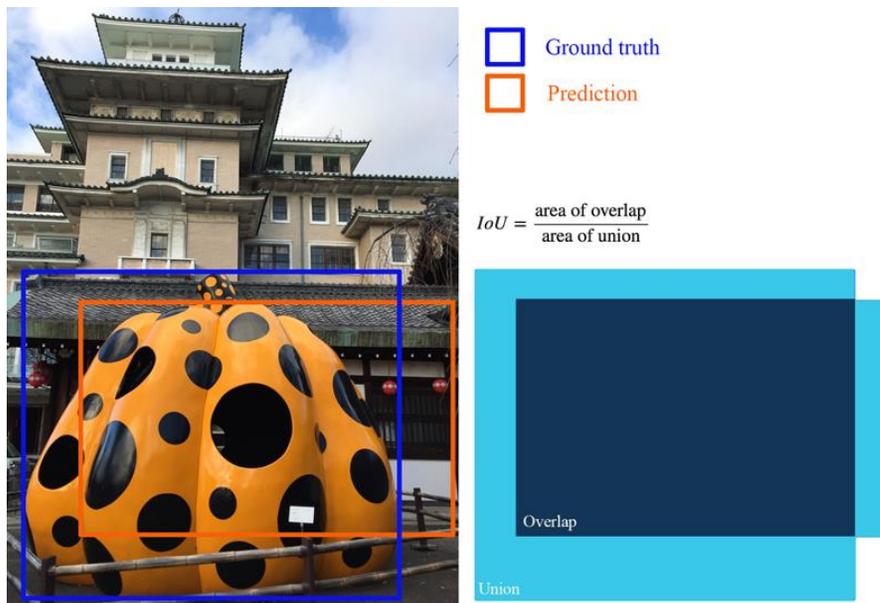
- $TP = \text{Verdadero Positivo}$
- $FN = \text{Falso Negativo}$

- **IoU (Intersection Over Union)**

Mide la superposición entre 2 límites, cuánto se superpone el límite predicho con la verdad fundamental (*ground truth*, el límite del objeto real). En algunos conjuntos de datos, se predefine un umbral de *IoU*, por ejemplo 0.5 para clasificar si la predicción es un verdadero positivo o un falso positivo (Ver Figura 30).

Figura 30

Intersección entre Verdad Fundamental y la predicción



Nota. Esta gráfica representa la intersección entre el cuadro delimitador verdadero y el generado por el sistema de detección. Tomado de https://medium.com/@jonathan_hui/map-mean-average-precision, (Hui, 2018)

- **mAP (Mean Average Precision)**

Es una métrica popular utilizada para determinar el desempeño de modelos que realizan tareas como: detección de objetos o recuperación de documentos e información.

- **COCO**

Es un conjunto de datos utilizados para la detección, segmentación y subtitulación de objetos a gran escala. COCO tiene varias características, como:

- Segmentación de objetos
- Reconocimiento de contexto
- Segmentación de super píxeles
- 330K imágenes (> 200K etiquetadas)
- 1.5 millones de instancias de objetos
- 80 categorías de objetos (clases)
- 91 categorías de cosas
- 5 subtítulos por imagen
- 250,000 personas con *keypoints*

Para este proyecto de investigación se trabajó con el conjunto de datos utilizados en la detección de objetos. Estas 80 categorías de objetos o clases se enumeran en la siguiente la Tabla 9.

Tabla 9

Banco de clases de COCO

N	Clase	N	Clase	N	Clase
1	Persona	2	Bicicleta	3	Automóvil
4	Motocicleta	5	Aeroplano	6	Bus
7	Tren	8	Tractor	9	Barco
10	Semáforo	11	Hidrante	12	Señal de paro

N	Clase	N	Clase	N	Clase
13	Parquímetro	14	Banco	15	Pájaro
16	Gato	17	Perro	18	Caballo
19	Oveja	20	Vaca	21	Elefante
22	Oso	23	Cebra	24	Jirafa
25	Mochila	26	Paraguas	27	Bolso
28	Corbata	29	Valija	30	Frisbee
31	Esquí	32	Snowboard	33	Pelota deportiva
34	Cometa	35	Bate de béisbol	36	Guante de béisbol
37	Patineta	38	Tabla hawaiana	39	Raqueta de tenis
40	Botella	41	Copa de vino	42	Taza
43	Tenedor	44	Cuchillo	45	Cuchara
46	Tazón	47	Plátano	48	Manzana
49	Sándwich	50	Naranja	51	Brócoli
52	Zanahoria	53	Hot dog	54	Pizza
55	Donut	56	Pastel	57	Silla
58	Sofá	59	Planta en maceta	60	Cama
61	Mesa de comedor	62	Inodoro	63	Monitor de TV
64	Portátil	65	Ratón	66	Remoto
67	Teclado	68	Teléfono celular	69	Microondas
70	Horno	71	Tostadora	72	Fregadero

N	Clase	N	Clase	N	Clase
73	Refrigerador	74	Libro	75	Reloj
76	Florero	77	Tijeras	78	Oso de peluche
79	Secadora de pelo	80	Cepillo de dientes		

Nota. Esta tabla muestra las clases de COCO utilizadas para evaluar la detección de objetos con sistemas de detección como YOLO.

Predicción del cuadro delimitador de YOLOv3

YOLOv3 predice cuadros delimitadores utilizando grupos de dimensiones como cuadros (*boxes*) de anclaje. La red predice 4 coordenadas para cada cuadro delimitador t_x, t_y, t_w, t_h . Si la celda está desplazada desde la esquina superior izquierda (píxel inicial) de la imagen por (c_x, c_y) y el cuadro delimitador tiene ancho y alto p_w, p_h , entonces las predicciones del cuadro delimitador corresponden a:

$$b_x = \sigma(t_x) + c_x \quad (20)$$

$$b_y = \sigma(t_y) + c_y \quad (21)$$

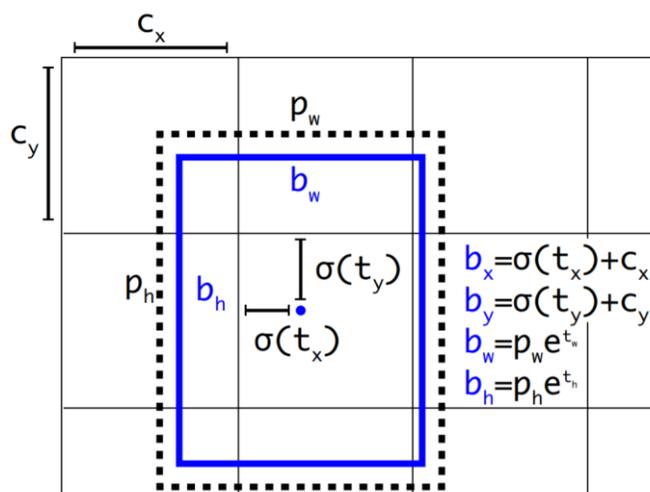
$$b_w = p_w e^{t_w} \quad (22)$$

$$b_h = p_h e^{t_h} \quad (23)$$

Se predice el ancho y la altura del box como compensación (*offsets*) de los centroides del grupo. Se pronostica las coordenadas centrales del box en relación a la ubicación de la aplicación del filtro, utilizando una función sigmoide (σ) mostrada en la Figura 31.

Figura 31

Creación de Bounding boxes



Nota. Esta gráfica representa la creación de los cuadros de limitadores de las imágenes detectadas con dimensiones previas y predicción de ubicación. Tomado de “YOLOv3: An incremental improvement”, (Redmon & Farhadi, 2018).

Durante el entrenamiento se usa la suma de la pérdida de error al cuadrado considerando que la precisión de la clasificación del conjunto de entrenamiento para las técnicas de aprendizaje supervisado (verdad fundamental o *ground truth*) para alguna predicción de coordenadas es \hat{t}^* , y el gradiente lo define el valor de la verdad fundamental (calculado a partir de la verdad fundamental del box) menos la predicción: $\hat{t}^* - t^*$. Esta magnitud de verdad fundamental puede calcularse invirtiendo las ecuaciones anteriores.

El sistema pronostica una puntuación de objetividad para cada cuadro delimitador mediante regresión logística. Esto debería ser 1 si el cuadro delimitador anterior se superpone a un objeto de verdad fundamental más que cualquier otro cuadro delimitador anterior. Por otro lado, si el cuadro delimitador anterior no es el mejor, pero se superpone

a un objeto de verdad fundamental por más de un umbral, se ignora la predicción. El sistema asigna antes un solo cuadro delimitador para cada objeto de verdad fundamental y si no es asignado, no incurre en pérdida de coordenadas o predicciones de clase. (Redmon & Farhadi, 2018).

Vista del target desde el UAV

Es importante recordar que la visión del *UAV* es perpendicular al piso (vista cenital) con el objetivo de facilitar el aterrizaje y seguimiento automático de la plataforma de aterrizaje móvil. Como se mencionó en secciones anteriores, la detección de la plataforma se realiza en dos procesos diferentes que dependen directamente del porcentaje de visión del target dentro de las imágenes enviadas por el *UAV*. La reducción de visualización de la escena se ve afectada por el descenso del dron; las imágenes captadas abarcan una menor área de visión del escenario debido a la gradual cercanía del *UAV* al piso (ver Figura 32).

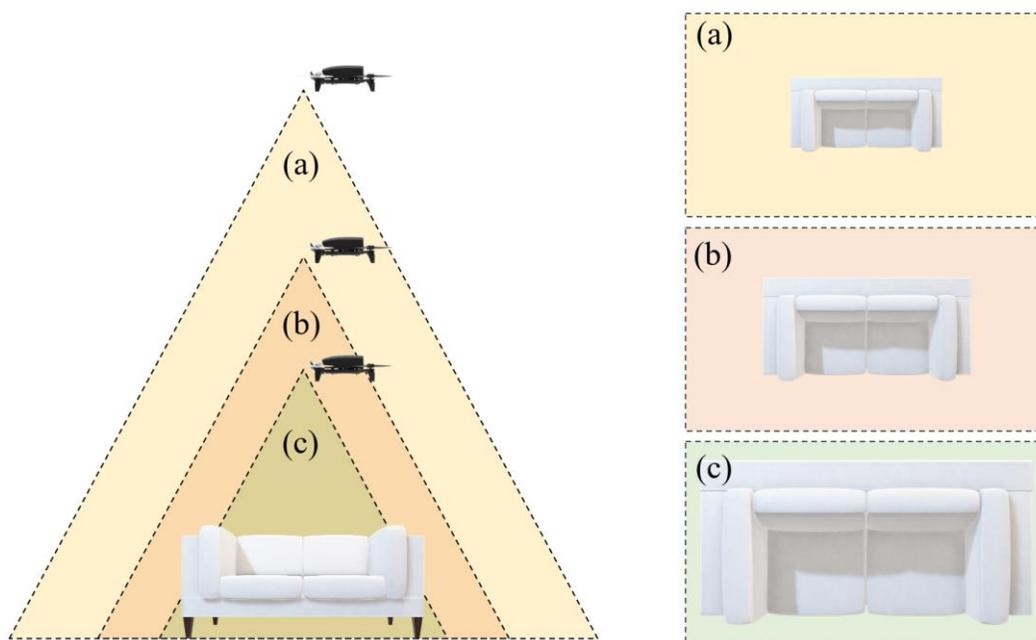
Por esta razón, el algoritmo principal de este proyecto de investigación contempla dos procesos de detección diferentes, no por el método utilizado sino por los problemas de visualización originados por la disminución de la altura del dron. El primero proceso está enfocado en el reconocimiento del automóvil en movimiento a grandes alturas (superiores a 5 metros), en este caso un vehículo en movimiento. Y el segundo, se encarga del reconocimiento de un objeto específico en el techo del automóvil siendo este el nuevo target y lugar de descenso final del *UAV*.

El objetivo inicial del proyecto fue trabajar únicamente con la detección de vehículo en movimiento, seguirlo y aterrizar sobre él, pero luego de varias pruebas, se determinó que el *UAV* perdía la visión completa del vehículo si descendía bajo cierto umbral de

altura (entre 4.5 a 5 metros). A partir de ese momento, el sistema de detección y seguimiento tienen muchos problemas para establecer la posición del *UAV* respecto al vehículo. El área de visualización que se tiene por debajo de ese umbral de altura no permite la detección del objeto ubicado debajo del *UAV* (ver Figura 33).

Figura 32

Área de visión de la escena respecto a la altura del UAV

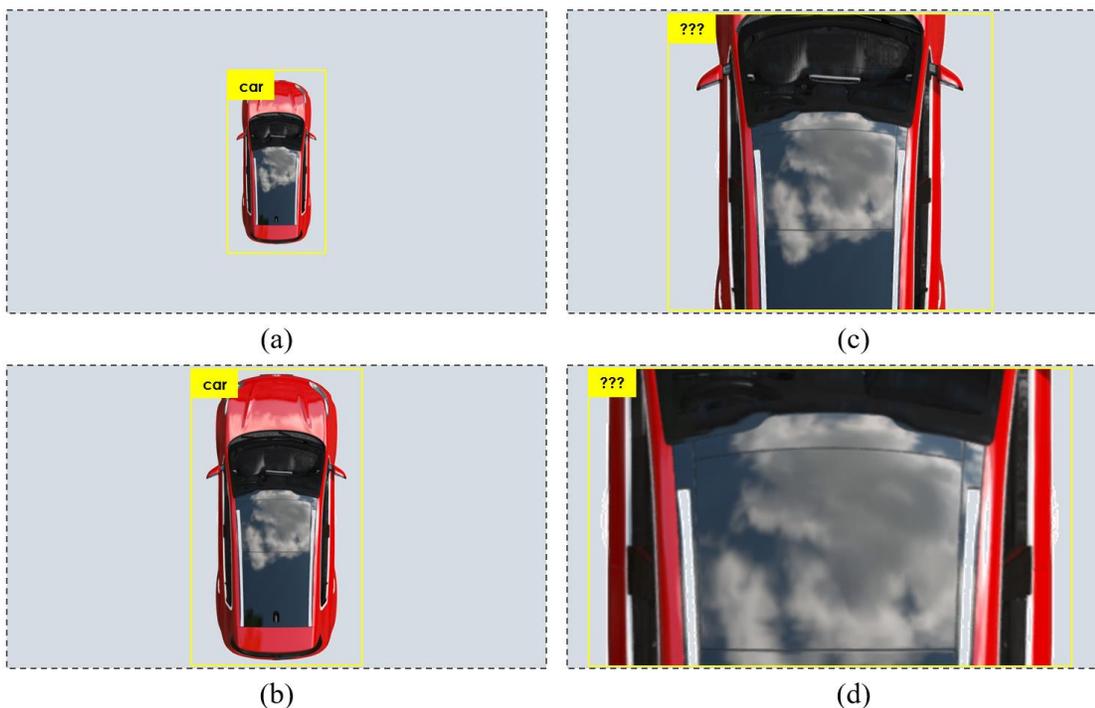


Nota. Esta gráfica representa el área de visión de la cámara del *UAV* a diferentes alturas con una vista cenital enfocado a un objeto en particular.

La Figura 32 (a) y (b), exhibe que en determinado rango de altitud del *UAV* es posible una detección adecuada del vehículo porque se puede observar completamente el contorno del objeto. Entonces, el primer proceso de detección depende de esta condición, es decir, la detección y seguimiento tendrán como target al vehículo, siempre y cuando la visualización de este sea completa, caso contrario, el algoritmo principal da paso a la siguiente búsqueda y detección del nuevo target.

Figura 33

Visión de la plataforma de aterrizaje respecto a su altura



Nota. La gráfica muestra visualmente el automóvil para el aterrizaje a diferentes alturas.

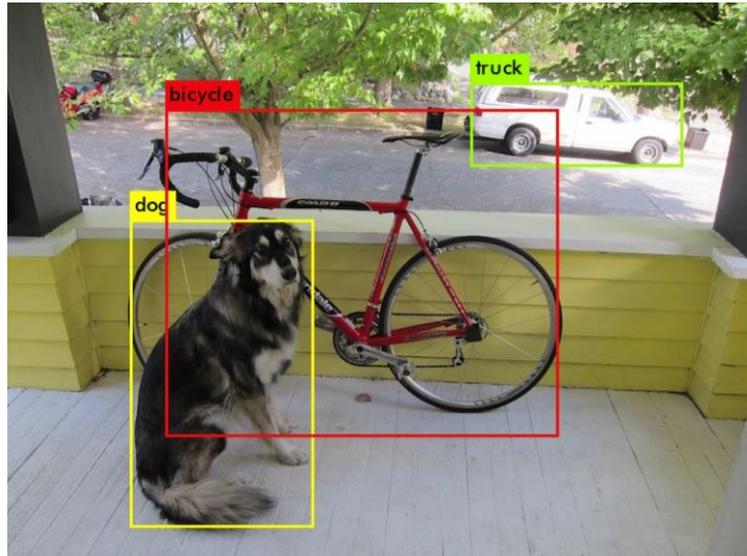
Detección del target con YOLO

Para la detección de objetos en este proyecto se utilizó YOLOv3, este sistema está implementado en un bucle repetitivo donde el streaming de video del drone es tratado como un conjunto de imágenes donde se realizan las detecciones.

El sistema puede detectar hasta 80 clases (objetos) diferentes en una misma imagen (ver Figura 34). Se necesita realizar un proceso de filtrado al total de las detecciones para eliminar los objetos que no son de interés.

Figura 34

Detección de objetos con YOLO



Nota. La gráfica muestra la detección de objetos dentro de una misma escena usando el sistema de detección YOLO. Tomado de “YOLOv3: An incremental improvement”, (Redmon & Farhadi, 2018).

¿Cómo se usa YOLO dentro del algoritmo?

Este sistema arroja un grupo información para cada detección dentro de una misma imagen. La información que entrega YOLO es:

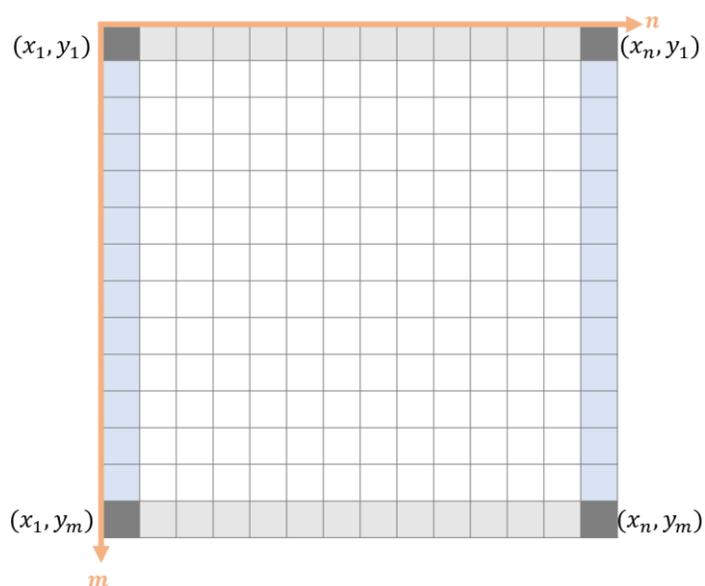
- Numero de objetos detectados.
- ID y nombre de la clase (ver Tabla 9)
- Matriz de cuadros delimitadores (información de la posición de los objetos en pixeles).
- Imagen de salida (con los cuadros delimitadores dibujados sobre ella).

Para el algoritmo de detección de este proyecto se utiliza los primeros tres datos. El número de objetos detectados, ID y nombre de la clase se usan básicamente para

algunas validaciones y filtrado para los objetos de interés a lo largo del algoritmo. La matriz de cuadros delimitadores contiene la posición de los vértices mínimo y máximo del cuadro que encierra al objeto detectado dentro de la imagen. La distribución de píxeles dentro de un cuadro parte desde la esquina superior izquierda y termina en la esquina inferior derecha (ver Figura 35).

Figura 35

Posición de píxeles dentro de la imagen



Nota. La Gráfica muestra como la computadora posiciona los píxeles de una imagen.

De donde:

- n : representa el número total de píxeles en el eje X de la imagen.
- m : representa el número total de píxeles en el eje Y de la imagen.

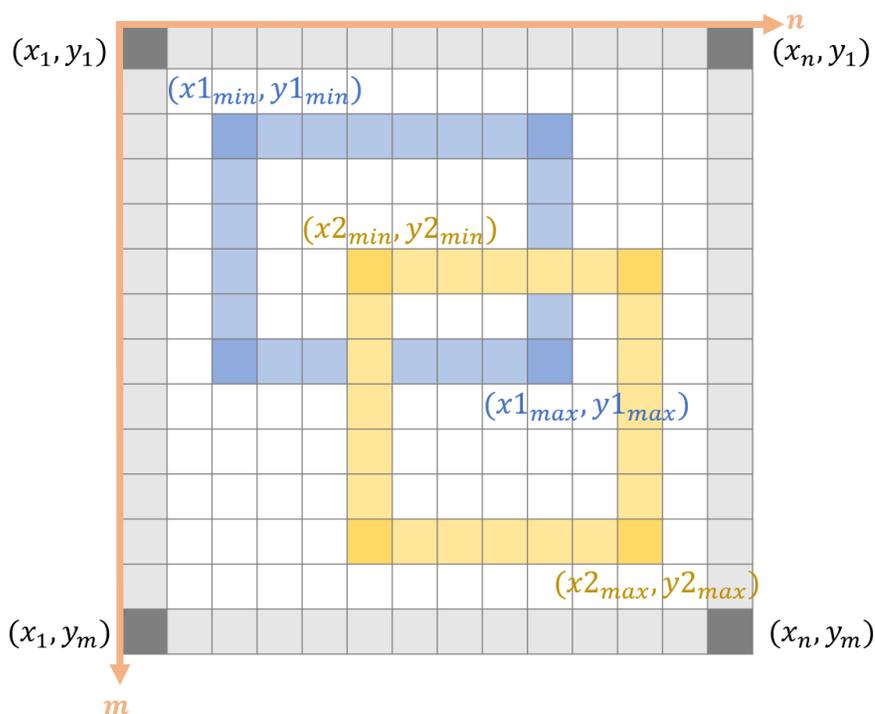
La matriz de cuadros delimitadores está distribuida por n filas y 4 columnas, las filas representan la cantidad de objetos reconocidos y las columnas contienen la posición de dos vértices (mínimo y máximo) del cuadro delimitador ubicados así:

$$\text{bounding boxes} = (x_{min} \ y_{min} \ x_{max} \ y_{max}) \quad (24)$$

La Figura 36 ejemplifica la creación de cuadros delimitadores en la detección de dos objetos utilizando YOLO, los datos que arroja el sistema son dos vértices para cada cuadro delimitador de objetos dentro de la imagen.

Figura 36

Cuadros delimitadores entregados por YOLO



Nota. La gráfica muestra un ejemplo de la creación de dos cuadros delimitadores y los datos que YOLO proporciona de su posición dentro de la imagen.

La detección de objetos con YOLO es representada en un conjunto de datos. Para el ejemplo de la Figura 36, la información que el sistema entregaría se muestra en la Tabla 10:

Tabla 10

Datos de salida de YOLO para dos detecciones de diferentes clases

Variables de salida	Datos
Número de objetos detectados	2
ID	(entre 0 a 79) (entre 0 a 79)
Nombre de la clase	(Clase ₁) (Clase ₂)
Matriz de cuadros delimitadores	$\begin{pmatrix} x1_{min} & y1_{min} & x1_{max} & y1_{max} \\ x2_{min} & y2_{min} & x2_{max} & y2_{max} \end{pmatrix}$

Nota. Esta tabla muestra un ejemplo de los datos que arrojaría el sistema de detección YOLO en caso de tener dos detecciones dentro de una misma imagen.

Figura 37

Detección del vehículo real con YOLO



Nota. Esta gráfica muestra un ejemplo de la detección del automóvil con el UAV volado a una altura sobre los 4 metros.

Primer proceso de detección

El UAV a una altura superior al umbral de 5 metros sobre el nivel del suelo puede capturar imágenes con una visión completa de autos pequeños y medianos. La Figura 37 muestra el sistema en marcha con la detección del vehículo (donde se realiza el aterrizaje automático) a una altura de aproximadamente 5 metros.

En la Tabla 11 se muestra la información de salida del sistema de detección para el vehículo blanco. Es importante mencionar que las dimensiones del fotograma mostrado en la Figura 37 son:

- $n = 856$ Número total de píxeles en el eje X de la imagen.
- $m = 480$ Número total de píxeles en el eje Y de la imagen.

Tabla 11

Información de detección del vehículo

Variables de salida	Datos
Número de objetos detectados	1
ID	(2)
Nombre de la clase	(Automovil)
Matriz de cuadros delimitadores	(499 20 728 372)

Nota. Esta tabla muestra los datos que arrojó el sistema de detección YOLO al detectar al automóvil en la imagen.

Para la variable de salida ID y Nombre de clase se debe revisar la Tabla 9, en este caso, la detección del objeto pertenece a la clase Automóvil, esta tiene el número 3 del

total de 80. El ID trabaja con esta numeración, pero comenzando desde el número 0, entonces el ID para Automóvil pasa a ser el 2. Esta variable de salida al igual que Nombre de clase y Matriz de cuadros delimitadores son tuplas que cambiarán de dimensiones para cada nuevo fotograma si hay más o menos detecciones.

Segundo proceso de detección

Para alturas menores al umbral de 5 metros, la visualización completa de automóviles grandes y medianos no abarca la totalidad de su figura, tal como se representa en la Figura 33 (c) y (d). El algoritmo de aterrizaje y seguimiento automático necesitan de algún punto de referencia para determinar la posición del UAV respecto al vehículo. Para alturas menores al umbral, el primer problema es la pérdida de la detección del automóvil. Al continuar con el descenso, poco a poco lo único que se puede observar en la imagen es el techo del vehículo, una superficie llana de un solo color, sin puntos referenciales ni objetos que se puedan utilizar para una nueva detección.

Figura 38

Vista cenital de la primera y segunda detección a diferentes alturas



Nota. Esta gráfica muestra la vista que tendría el UAV tanto para el automóvil como para el objeto utilizado en la etapa final del aterrizaje.

La solución a esto fue diseñar un segundo proceso de detección, con un nuevo target de menores dimensiones al techo de vehículo. Esto garantiza que su visualización y detección pueda mantenerse a menores alturas y lograr un seguimiento adecuado hasta conseguir finalmente el aterrizaje. La Figura 38 representa como el segundo proceso de detección ayuda a no perder el objeto de interés respecto a la primera detección, luego de haber descendido a alturas cercanas al vehículo. Es incorrecto concluir que se pudo haber omitido la primera detección y trabajar únicamente con la segunda. El primer diseño del algoritmo contemplaba únicamente la detección y seguimiento del vehículo, esto tiene buenos resultados para alturas mayores a 5 metros, por las grandes dimensiones del objeto, ocupando un área importante dentro de las imágenes captadas a esas alturas, ayudando a YOLO a reconocer el auto. En el caso de trabajar únicamente con el segundo proceso de detección, el algoritmo no es capaz de detectar ni seguir al objeto ubicado en el techo de automóvil, la distancia entre el nuevo target y el *UAV* es muy grande como para que se lo pueda detectar.

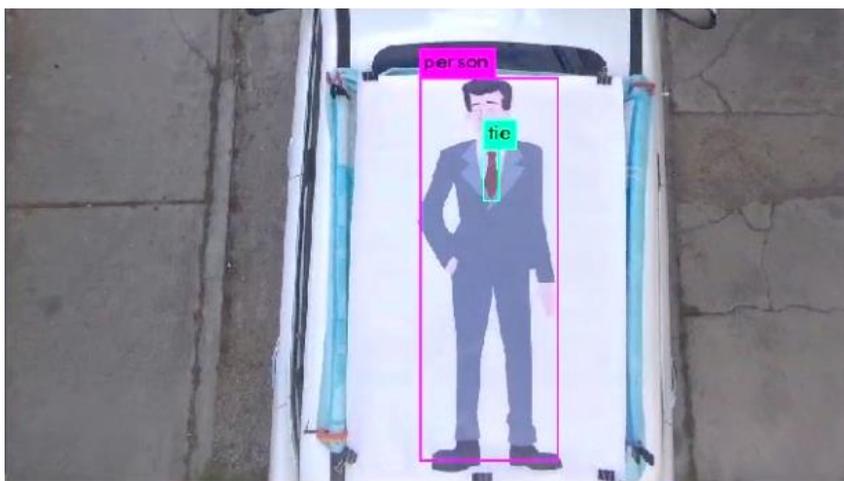
En la Figura 39 muestra el segundo proceso de detección; el nuevo target es una figura de un objeto específico dentro de las 80 clases que puede reconocer YOLO. Esta es una captura del streaming de video del algoritmo puesto en marcha. Y en la Tabla 12 se muestra la información de salida de YOLO para la detección del nuevo target. Se recuerda que las dimensiones en los fotogramas del streaming de video (Figura 39) también son:

- $n = 856$ Número total de píxeles en el eje X de la imagen.
- $m = 480$ Número total de píxeles en el eje Y de la imagen.

En este caso las variables de salida ID y Nombre de clase están en la Tabla 9, la detección del objeto pertenece a la clase Persona, con el ID de 0.

Figura 39

Detección real con YOLO del nuevo target



Nota. La figura muestra la detección del objeto de interés sobre el automóvil usando en la etapa final del aterrizaje automático.

Tabla 12

Información de salida para la detección del nuevo target

Variables de salida	Datos
Número de objetos detectados	1
ID	(0)
Nombre de la clase	(Persona)
Matriz de cuadros delimitadores	(418 68 559 466)

Nota. Esta tabla muestra los datos que arrojó el sistema de detección YOLO al detectar al objeto de interés montado sobre el automóvil.

Capítulo V

Seguimiento y aterrizaje automáticos del UAV

Se detalla la creación de algoritmos de seguimiento y aterrizaje automáticos del UAV sobre la plataforma móvil terrestre y el desarrollo de controladores para estas tareas. En el capítulo anterior se habló a detalle sobre el sistema de detección de objetos y de las herramientas utilizadas para desarrollar algoritmos que trabajen con la ubicación del target dentro del fotograma. Conocer la posición de la plataforma de aterrizaje respecto al UAV en tiempo real es fundamental para trabajar con los algoritmos de control y lograr un seguimiento adecuado con un descenso gradual hasta conseguir el aterrizaje del UAV.

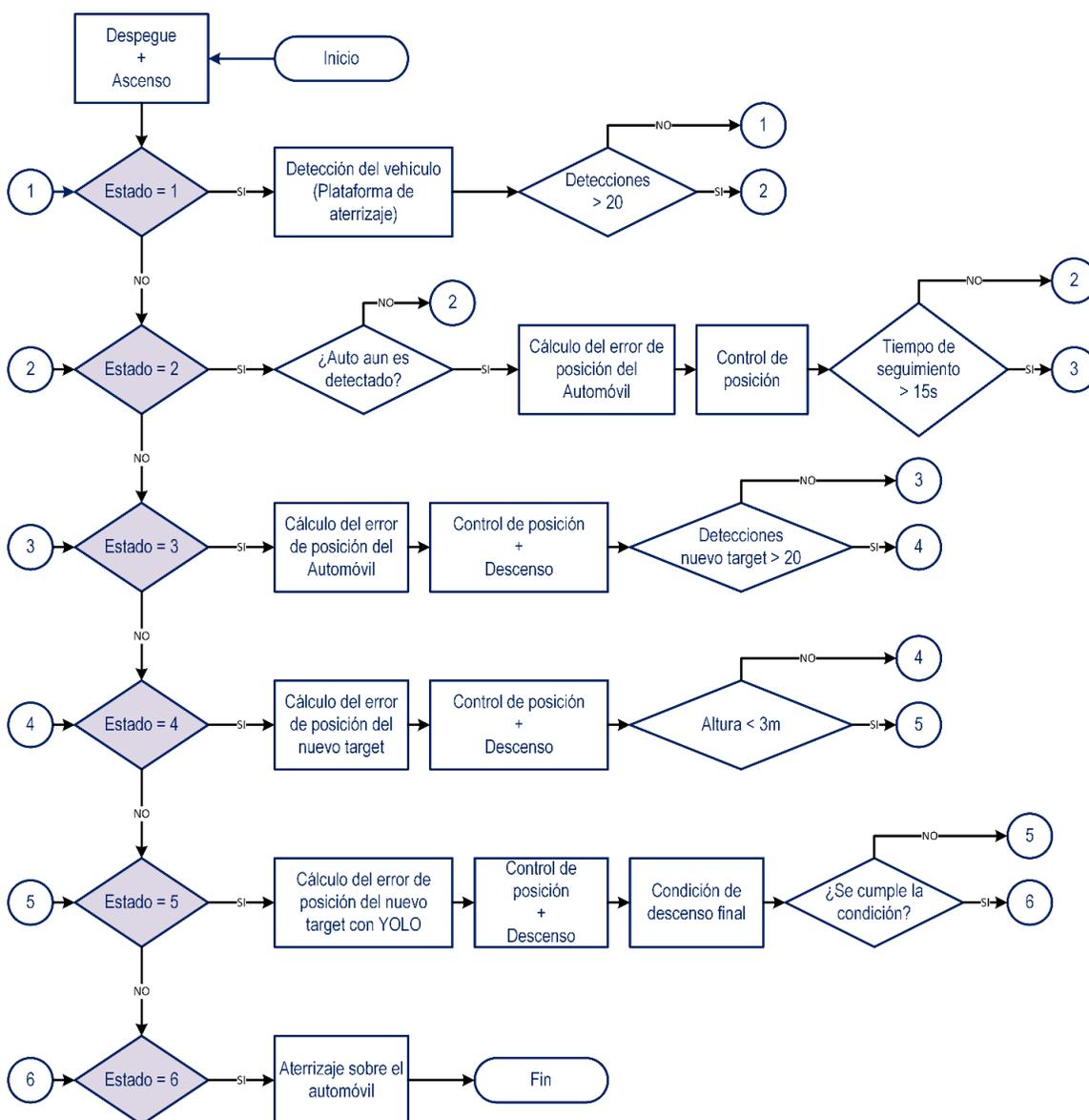
Descripción del algoritmo global de aterrizaje

El algoritmo del aterrizaje automático está desarrollado con varios procesos, tal como se muestra en la Figura 40. Está dividido en 6 estados diferentes para la toma de acciones ante posibles problemas como pérdida de la detección del vehículo, no detección del target colocado sobre el techo del vehículo, error en el seguimiento del target por problemas de escala ocasionados por la altura, entre otros.

Los seis estados mostrados en el diagrama de flujo del algoritmo general (ver Figura 40) están descritos por números, pero representan procesos dentro del algoritmo. Algunas se refieren al reconocimiento de objetos específicos en las escenas capturadas, para identificar a la plataforma móvil del resto de objetos. En cambio, otras acciones, como el control de seguimiento automático o las condiciones para el aterrizaje final se las detalla en este capítulo.

Figura 40

Diagrama de flujo del algoritmo global de aterrizaje automático



Nota. Esta gráfica muestra el diagrama de flujo programado para realizar la secuencia de aterrizaje automática considerando las detecciones del automóvil y del objeto de interés colocado sobre este.

Una explicación general de cada estado ayudará a la comprensión del rol que cumplen dentro del algoritmo. A continuación, se describe los estados que son parte del diagrama de flujo para el aterrizaje automático del *UAV*.

- **Estado 1: Búsqueda del automóvil**

Luego del despegue y un ascenso donde se debe indicar la altura de vuelo, el dron comienza a girar virtualmente su cámara hasta lograr una vista cenital (ver Figura 41), en ese momento se pone en marcha el sistema de detección de objetos con las modificaciones para posicionar únicamente el objeto de interés. El objetivo de este estado es la detección del objeto en cada fotograma que es enviado desde el dron a la estación en tierra.

Figura 41

Despegue y posicionamiento virtual de la cámara del UAV



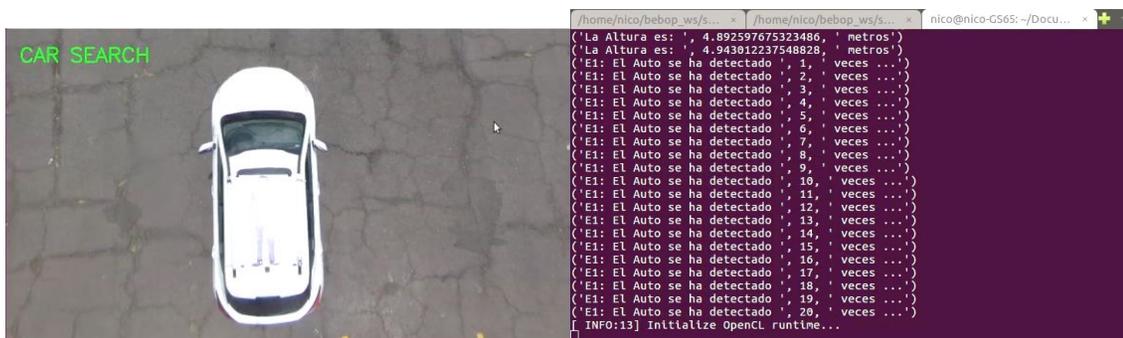
Nota. Esta gráfica muestra el despegue del *UAV*, la primera etapa del algoritmo global del aterrizaje automático.

Un promedio de 25 imágenes por segundos son procesadas con el fin de detectar al vehículo en las escenas capturadas (ver Figura 42). Al alcanzar las 20 detecciones seguidas, se calcula el promedio de los datos (posición de los vértices mínimo y máximo

del cuadro delimitador que encierra al objeto de interés dentro de la imagen). Este cálculo se almacena en una lista utilizada para inicializar el algoritmo de seguimiento en el estado posterior.

Figura 42

Detecciones seguidas del vehículo con YOLO



Nota. Esta gráfica muestra el estado 1 del algoritmo global del aterrizaje automático.

- **Estado 2: Seguimiento del automóvil**

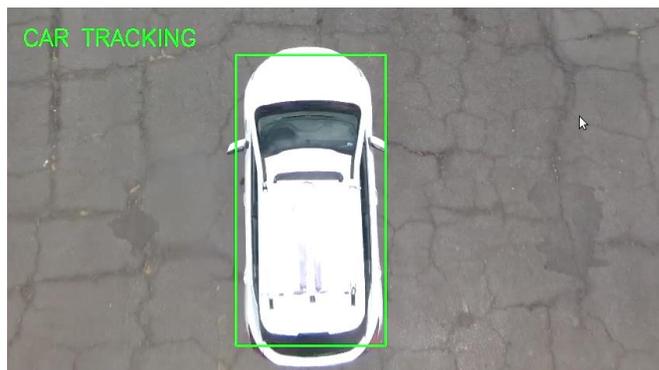
Ya conocida la posición de objeto dentro de la imagen, se ingresa los datos al algoritmo de seguimiento *KCF* para el seguimiento del objetivo señalado, la información de posición se actualiza en cada nuevo fotograma. Es decir, se conoce el vértice mínimo y máximo del cuadro delimitador del objeto a pesar de que este cambie de posición (automóvil comience una trayectoria rectilínea) (ver Figura 43).

Conocido ambos vértices de la detección que encierra al automóvil, se calcula el centroide del cuadro delimitador. Este nuevo dato representa la posición real de la plataforma respecto al *UAV*. Ahora ya es posible calcular el error de posición tanto para el eje *X* como en el *Y* para la implementación de los controladores que desplazan al *UAV* para realizar el seguimiento de la plataforma en movimiento.

Este procedimiento se realiza durante un tiempo determinado, sin que el que drone realice un descenso, solamente como demostración del seguimiento automático.

Figura 43

Cuadro de seguimiento del objeto de interés con algoritmo KCF



Nota. Esta gráfica muestra el estado 2 del algoritmo global del aterrizaje automático.

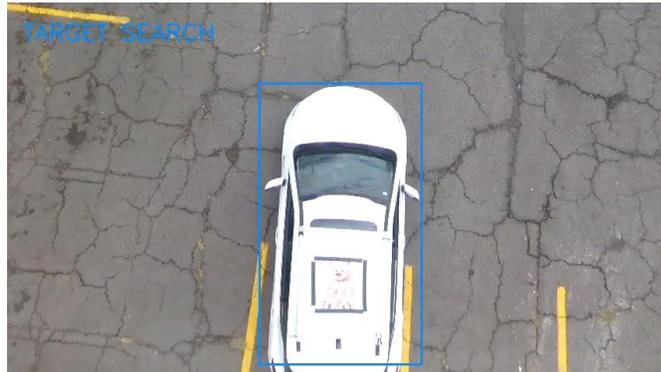
- **Estado 3: Búsqueda del nuevo target (imagen sobre el techo del auto)**

Para la búsqueda del nuevo target se realiza un control de posición más un descenso constante del *UAV* con los datos del posicionamiento del automóvil. Esto se realiza hasta que el drone disminuya su altura lo suficiente para que el sistema de detección reconozca al nuevo target en el techo del auto (ver Figura 44). En ese momento se repite el conteo sucesivo de 20 detecciones para calcular un promedio de la posición dentro de la imagen.

La duración de este estado muchas ocasiones es muy breve porque el *UAV* ha descendido lo suficiente como para detectar el nuevo target en las primeras iteraciones del estado, incluso la función que muestra la imagen dentro del algoritmo no se abre completamente antes de observarla.

Figura 44

Búsqueda del nuevo target



Nota. Esta gráfica muestra el estado 3 del algoritmo global del aterrizaje automático.

- **Estado 4: Seguimiento del nuevo target**

El estado 4 es muy similar al 2, difieren principalmente en el objeto de interés. Pasa del seguimiento del automóvil, al seguimiento de la imagen sobre el mismo, pero con la misma lógica de control.

Figura 45

Cuadro de seguimiento del nuevo target de interés con algoritmo KCF



Nota. Esta gráfica muestra el estado 4 del algoritmo global del aterrizaje automático.

La Figura 45 muestra el cuadro de seguimiento para el nuevo target. Otro cambio está es la condición de paso al siguiente estado, ya no depende de un tiempo determinado, ahora está sujeto a que drone baje de los 3 metros de altura, en ese instante el algoritmo da paso al siguiente estado.

- **Estado 5: Descenso hacia el target**

Para terminar la tarea de aterrizaje, se ha diseñado diferentes condiciones para la activación del tópic *land()* y finiquitar el aterrizaje. Estas condiciones contemplan parámetros como la altura, la escala del objeto de interés afectada por la variación de la altura y la perdida de la detección del target perjudicada por la corta distancia del drone y el auto.

Figura 46

Cuadro de detección y seguimiento del objeto antes del aterrizaje



Nota. Esta gráfica muestra el estado 5 del algoritmo global del aterrizaje automático.

Como se explicará más adelante, el cuadro delimitador que crea el algoritmo de seguimiento *KCF* se mantiene invariante en alto y ancho a pesar que el objeto cambie de escala debido a la variación de la altura de la cámara que captura la imagen. Este problema está presente en la etapa final del aterrizaje, porque el *UAV* durante la ejecución

final del algoritmo mantiene un descenso constante que produce que el objeto de interés ocupe un área mayor de la imagen, es decir que aumente su escala. En la Figura 46 se muestra la visualización del estado 5, el proceso antes de aterrizar completamente sobre el vehículo.

- **Estado 6: Aterrizaje final en el techo del automóvil**

A parte de finalizar el aterrizaje, se usa como medida de seguridad para un aterrizaje de emergencia ante cualquier problema que se pueda presentar.

Estimación del movimiento del *UAV*

Para realizar la estimación de movimiento del *UAV* se debe obtener su modelo del movimiento que describa su dinámica. Este modelo permite la creación de acciones de control para el seguimiento y aterrizaje automáticos. El proceso de obtención del modelo en *UAVs* se describe en varios artículos que realizan la estimación de movimiento utilizando una *IMU* (Unidad de Medición Inercial) como (Aguilar, Casaliglla, & Pólit, Obstacle Avoidance Based-Visual Navigation for Micro Aerial Vehicles. , 2017), (Khamseh Motlagh, Lotfi, & Taghirad, 2019) o (Yang, Lee, Jeon, & Lee, 2017), pero esta solución no es viable para este proyecto, tal como se menciona en (Salcedo Peña, 2018), (Grijalva Caisachana, 2019) y (Alvarez Samaniego, 2018) debido a que la baja tasa de refresco del tópico del drone Bebop 2 que envía los datos de odometría alcanzan como máximo 5 Hz, imposibilitando un aceptable cálculo del modelo matemático del movimiento del drone y la generación de acciones de control oportunas en tiempo real frente a perturbaciones para el seguimiento y aterrizaje de la plataforma en movimiento. Pero, considerando la velocidad de refresco del tópico encargado de enviar el streaming de video del *UAV* como mensajes tipo *ROS* hacia la estación en tierra alcanza los 30 Hz, una velocidad adecuada para lograr un bueno modelo matemático de la dinámica del drone. El modelado basado

en video o modelo servo visual trabaja con visión artificial y una transformada geométrica en imágenes sucesivas usando sus características locales para la detección, descripción y matching de *keypoints*. Esta idea ha sido aplicada en trabajos (Aguilar, Salcedo, Sandoval, & Cobeña, 2017),(Grijalva & Aguilar, 2019),(Orbea, et al., Math Model of UAV Multi Rotor Prototype with Fixed Wing Aerodynamic Structure for a Flight Simulator, 2017), (Aguilar W. G., Angulo, Costa, & Molina, 2014),(Aguilar, Angulo, & Pardo, Motion intention optimization for multicopter robust video stabilization, 2017). En (Salcedo Peña, 2018) se menciona que los movimientos del UAV en los ejes *X*, *Y* y *Z* se pueden vincular de forma particular a las diferentes acciones de movimiento descritas en la Tabla 7. El movimiento voluntario a lo largo de los diferentes ejes con respecto a las acciones de control se muestra en la Tabla 13 basada de (Aguilar & Angulo, Real-Time Model-Based Video Stabilization for Microaerial Vehicles, 2015).

Tabla 13

Movimiento en los ejes X, Y y Z vinculados a acciones de control

Parámetros de control	Rotación en Z	Eje X	Eje Y	Eje Z
PITCH	-	Voluntario	-	-
ROLL	-	-	Voluntario	-
YAW	Voluntario	-	-	-
ALTITUD	-	-	-	Voluntario

Nota. Esta tabla muestra las acciones de control pitch, roll, yaw y la altitud relacionadas con los movimientos en los ejes *X*, *Y* y *Z*. Tomado de “Aterrizaje automático de un vehículo aéreo no tripulado basado en seguimiento de puntos de interés para superficies móviles”, (Salcedo Peña, 2018)

Se llega a la conclusión que los desplazamientos voluntarios del *UAV* originados por las señales de control en los diferentes ejes de traslación (x, y, z) y rotación (z) tienen una relación individual entre sí, Tomando la decisión de trabajar con un modelo SISO (una entrada, una salida) para diseñar los controladores encargados del seguimiento y descenso automáticos. Esta conclusión es la misma a la que se llegó en (Salcedo Peña, 2018) y (Grijalva Caisachana, 2019).

Modelamiento servo visual

El modelamiento servo visual o modelo basado en video se ha presentado en (Aguilar & Angulo, Real-time video stabilization without phantom movements for micro aerial vehicles, 2014), (Aguilar, Salcedo, Sandoval, & Cobeña, 2017), también ha sido utilizado en varios proyectos de investigación para el modelamiento de drones como (Quisaguano Paredes, 2018), (Salcedo Peña, 2018), (Alvarez Samaniego, 2018), (Grijalva Caisachana, 2019) y (Camino Guzmán & Rojas Gómez, 2019) con resultados muy similares a los obtenidos con datos de sensores inerciales.

El modelamiento servo visual usa una transformación geométrica entre imágenes sucesivas utilizando matching entre puntos de interés coincidentes que permiten estimar los parámetros de movimiento entre la imagen actual y la anterior. Las imágenes utilizadas para este proceso son los fotogramas que componen el video capturado por el drone (un máximo de 30 fotogramas por segundo). Este procesamiento no se realiza durante la secuencia de vuelo para garantizar que el proceso sea lo más rápido y estable posible.

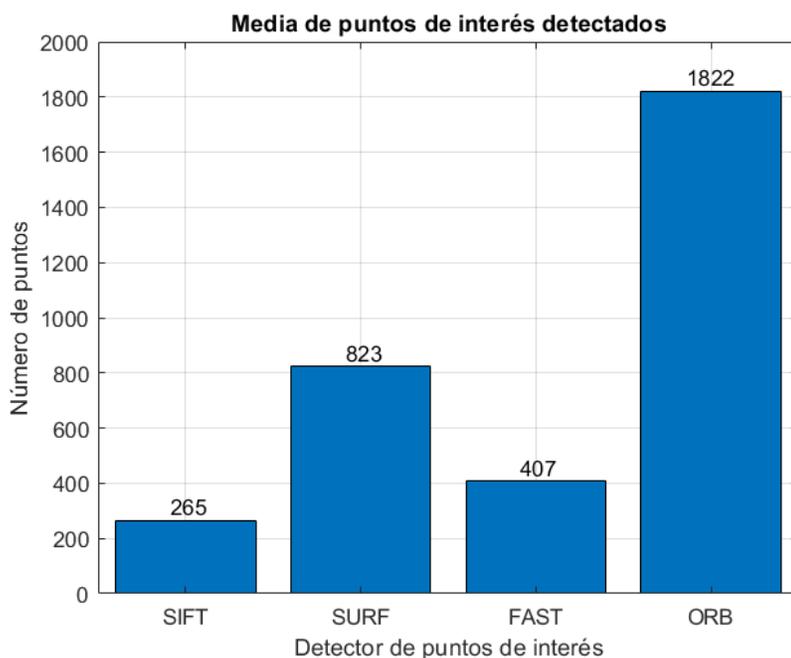
Detección y descripción de puntos de interés

El modelamiento servo visual trabaja con una transformada geométrica para determinar la variación de posición, ángulo o deformación entre imágenes sucesivas

dependiendo del tipo de transformación utilizada. Estos algoritmos a su vez trabajan con las características locales de cada fotograma para la detección, descripción y matching de *keypoints* para la estimación de parámetros de traslación, rotación o escala entre fotogramas sucesivos. Existen varios algoritmos para la detección y descripción de *keypoints* donde se destacan *SIFT*, *SURF*, *FAST* y *ORB*. Trabajos como (Salcedo Peña, 2018) o (Grijalva Caisachana, 2019) hacen una comparativa detallada del desempeño de estos métodos, concluyendo que *ORB* es la mejor opción en cuanto a una mayor cantidad de puntos de interés detectados y menor tiempo de detección tal como se muestran en las Figura 47 y Figura 48.

Figura 47

Cantidad de puntos de interés detectados

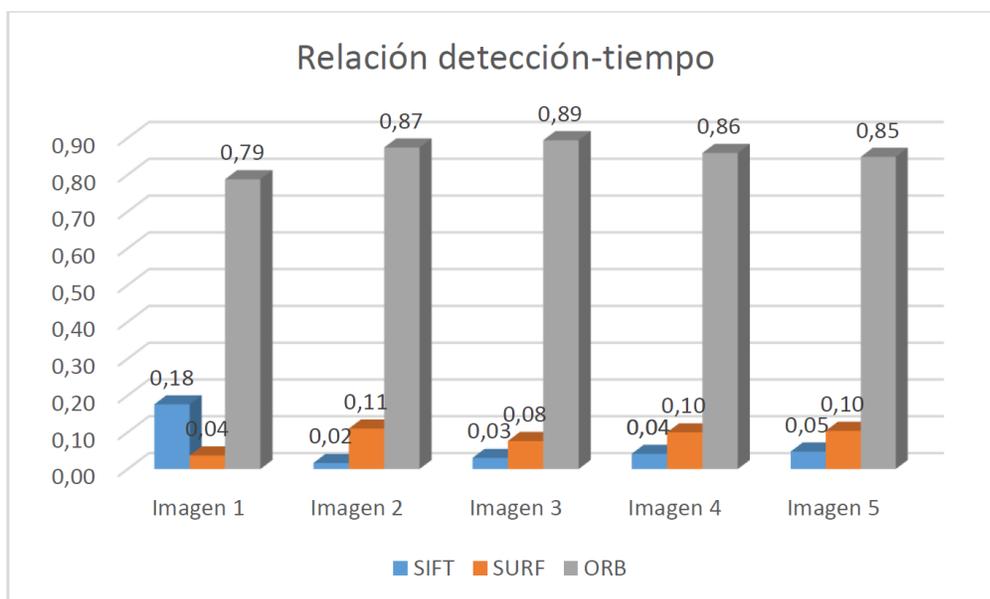


Nota. Esta gráfica representa el rendimiento del algoritmo *ORB* en cuanto a la detección de puntos de interés. Tomado de “*Sistema de navegación autónomo basado en visión para pistas de carreras de drones con marcas de superficie*”, (Grijalva Caisachana, 2019).

La Figura 47 representa la comparativa entre algoritmos de detección y descripción de *keypoints* realizada con un video capturado por el UAV Bebob 2 de una trayectoria trazada con marcas de color rojo sobre un piso uniforme gris en un ambiente con buenas condiciones de luz. La prueba arrojó 1822 puntos de interés utilizando el método *ORB*, esto significa que puede realizar la localización a una tasa de 150 FPS. Por otro lado, la menor cantidad de *keypoints* detectados con los métodos *SURF* y *SIFT* permite realizar detección a una velocidad de solamente 13 y 25 FPS respectivamente (Grijalva Caisachana, 2019).

Figura 48

Relación detección y tiempo de puntos de interés



Nota. Esta gráfica representa el desempeño del algoritmo *ORB* frente a *SIFT* y *SURF* evidenciando su superioridad respecto a la relación detección-tiempo de puntos de interés. Tomado de “*Aterrizaje automático de un vehículo aéreo no tripulado basado en seguimiento de puntos de interés para superficies móviles*”, (Salcedo Peña, 2018)

La Figura 48 representa la detección y descripción de puntos de interés de una misma imagen, pero en diferentes escenarios:

- **Imagen 1:** Vista del target capturada a 80 *cm*
- **Imagen 2:** Vista del target capturada a 50 *cm*
- **Imagen 3:** Vista inclinada del target capturada a 50 *cm*
- **Imagen 4:** Vista del target rotado y capturada a 120 *cm*
- **Imagen 5:** Vista diferente al target

Los resultados de esta comparación mostrada en la Figura 48 evidencian que el desempeño del algoritmo *ORB* es ampliamente superior. (Salcedo Peña, 2018) afirma que el tiempo de localización es menor que los otros métodos y que detecta una mayor cantidad de *keypoints* en relación al tiempo de procesamiento del algoritmo.

Considerando que, en ambos trabajos de investigación se llegó a la misma conclusión con pruebas muy similares a las necesidades de procesamiento entre imágenes para el modelamiento servo visual del *UAV* en este proyecto, se tomó la decisión de trabajar con el algoritmo *ORB* para la detección y descripción de *keypoints* e implementarlo en el algoritmo de la transformada geométrica.

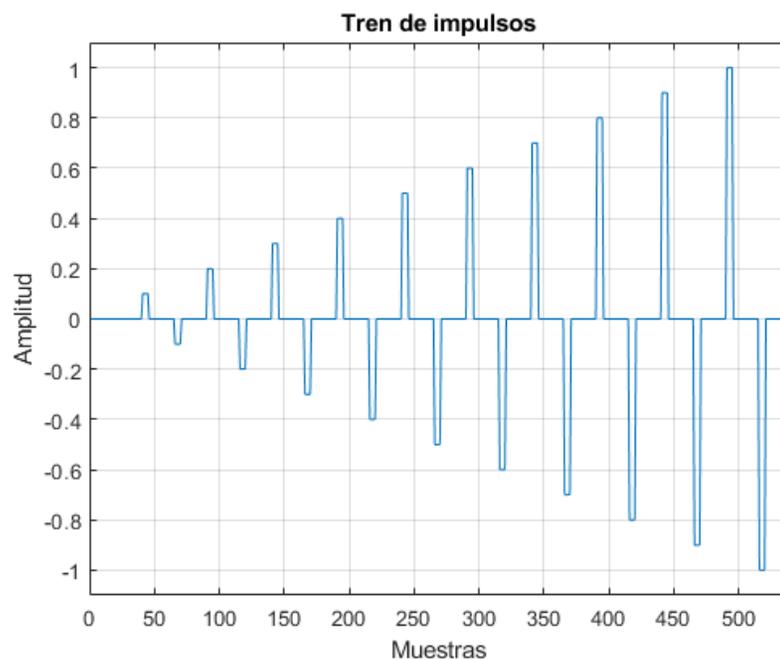
Obtención del modelo del movimiento acumulado

Para este proyecto de investigación, es necesario modelar el movimiento del *UAV* en los ejes *X* e *Y*, para diseñar los controladores encargados del seguimiento automático. Este rango de movimientos le permite al dron replicar la trayectoria rectilínea, considerando algunas desviaciones laterales de la plataforma móvil terrestre. Para generar la secuencia de vuelo restringida a un solo eje de manera voluntaria se utiliza el tópico de control de velocidad (*/bebop/cmd_vel*) descrito en la Tabla 7, de esta manera:

- Para la secuencia de vuelo alrededor del eje X (de adelante hacia atrás) se utiliza el tópico $linear.x(v_x)$, donde v_x debe ser un valor flotante cuyo rango va desde -1.0 a 1.0 .
- Para la secuencia de vuelo alrededor del eje Y (de derecha a izquierda) se utiliza el tópico $linear.y(v_y)$, donde v_y debe ser un valor flotante cuyo rango va desde -1.0 a 1.0 .

Figura 49

Tren de pulsos para las secuencias de vuelo del UAV



Nota. Esta gráfica muestra el tren de impulsos utilizados como señal de entrada para el modelamiento servo visual del *UAV*.

Las secuencias de vuelo se rigen a un tren de impulsos ascendentes en magnitud tanto de signo positivo como negativo, con un incremento gradual de 0.1 (ver Figura 49). Estos sucesivos mensajes de control producen que el *UAV* se desplace de adelante hacia

atrás o de izquierda a derecha dependiendo el tópico utilizado. Antes de realizar este proceso se consideraron los algunos aspectos descritos de la Tabla 14:

Tabla 14

Condiciones de vuelo y video del UAV para su modelamiento

CONDICIÓN	ESPECIFICACIÓN
Ángulo de visión de la cámara	Vista cenital
Altura de vuelo	Constante a 5 metros para garantizar una vista completa del automóvil (plataforma móvil de aterrizaje) dentro de la escena captada
Rango de movimientos	A lo largo del eje X e Y de manera independiente
Velocidad de desplazamiento	Definida por el tren de pulsos de la Figura 49
Tipo de terreno	Superficie plana

Nota. Esta tabla describe las condiciones del estado del *UAV* como también las del ambiente donde se realizó los vuelos.

A partir de videos grabados por el *UAV*, se realiza el proceso de estimación de movimiento analizando los cambios entre dos imágenes consecutivas expresados matemáticamente por la transformación geométrica que relaciona los puntos característicos en la primera imagen con sus correspondencias en la segunda imagen y así sucesivamente hasta llegar a los N fotogramas que componen el video. Este proyecto utiliza una transformada geométrica de similitud mediante la función `cv2.getAffineTransform` de *OpenCV* para obtener la expresión matemática que define los cambios de traslación entre fotogramas sucesivos en los ejes X e Y de manera independiente, la expresión resultante está dada por la siguiente matriz:

$$h = \begin{pmatrix} s \cdot \cos \theta & -s \cdot \sin \theta & t_x \\ s \cdot \sin \theta & s \cdot \cos \theta & t_y \end{pmatrix} \quad (25)$$

Los datos de la tercera columna (t_x, t_y) de la matriz de la ecuación (25) representan el cambio de posición en píxeles en los ejes X e Y respectivamente que existe de fotograma a fotograma. Para la estimación de movimiento se debe acumular estos datos de desplazamiento verticales u horizontales dependiendo del eje escogido para el modelamiento, sumando los datos de cada t_x, t_y generados con las N transformaciones realizadas por todos los N fotogramas de la grabación de vuelo del dron. De esta manera se generan N matrices, así:

$$\begin{aligned} h_1 &= \begin{pmatrix} s \cdot \cos \theta_1 & -s \cdot \sin \theta_1 & t_{x_1} \\ s \cdot \sin \theta_1 & s \cdot \cos \theta_1 & t_{y_1} \end{pmatrix} \\ h_2 &= \begin{pmatrix} s \cdot \cos \theta_2 & -s \cdot \sin \theta_2 & t_{x_2} \\ s \cdot \sin \theta_2 & s \cdot \cos \theta_2 & t_{y_2} \end{pmatrix} \\ &\vdots \\ h_N &= \begin{pmatrix} s \cdot \cos \theta_N & -s \cdot \sin \theta_N & t_{x_N} \\ s \cdot \sin \theta_N & s \cdot \cos \theta_N & t_{y_N} \end{pmatrix} \end{aligned} \quad (26)$$

Los elementos que representan los desplazamientos entre fotogramas de cada una de las matrices, están definidos por:

- $t_{x_1}, t_{x_2}, \dots, t_{x_N}$ para el eje X
- $t_{y_1}, t_{y_2}, \dots, t_{y_N}$ para el eje Y

Estos datos de desplazamiento se irán acumulando con sumas sucesivas dependiendo de las iteraciones del algoritmo. Por ejemplo, en la segunda iteración se sumarán dos valores t_{x_1} y t_{x_2} para el eje X , en cambio en la última iteración, la suma abarca todos los datos $t_{x_1}, t_{x_2}, \dots, t_{x_N}$ de los N fotogramas:

$$\begin{aligned}
 T_{X_1} &= t_{x_1} \\
 T_{X_2} &= t_{x_1} + t_{x_2} \\
 &\vdots \\
 T_{X_N} &= t_{x_1} + t_{x_2} + \dots + t_{x_N}
 \end{aligned}
 \tag{27}$$

Luego, cada suma acumulada de cada nueva iteración se almacena en una lista de Python para graficar la acumulación de movimiento en la secuencia de vuelo programada, de esta manera:

$$T_X = (T_{X_1} \quad T_{X_2} \quad \dots \quad T_{X_N}) \tag{28}$$

Este mismo procedimiento se replica para el eje Y obteniendo una nueva lista, descrita en la siguiente expresión:

$$T_Y = (T_{Y_1} \quad T_{Y_2} \quad \dots \quad T_{Y_N}) \tag{29}$$

Las listas generadas, es decir las expresiones mostradas en (28) y (29) se utilizan para graficar el movimiento acumulado de la secuencia de vuelo programa para el eje X y eje Y respectivamente con las señales de control de la Figura 49.

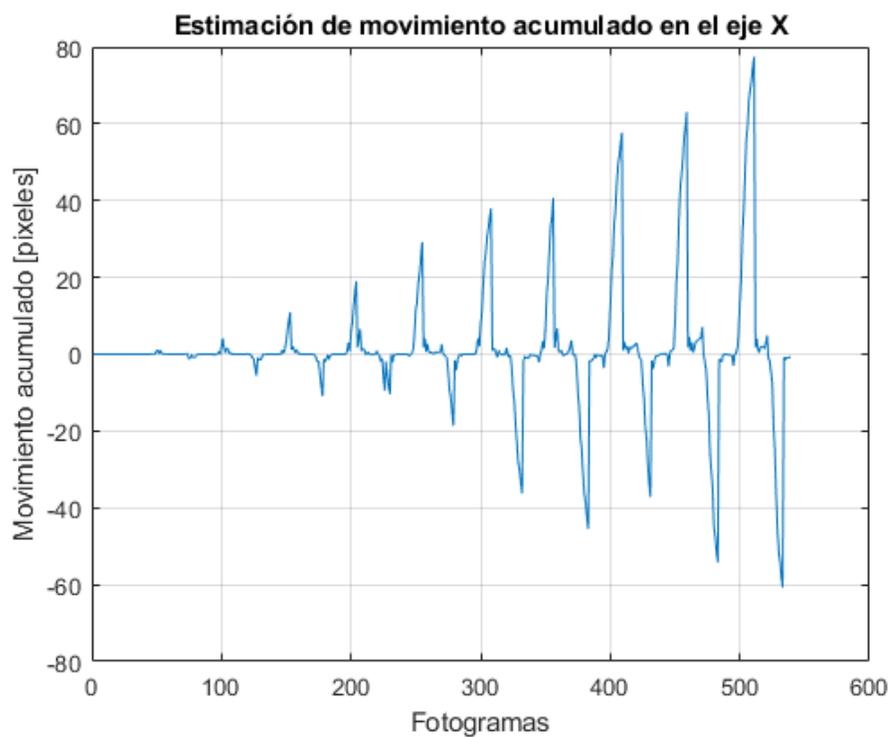
La Figura 50 y la Figura 51 representan la acumulación de movimiento en el eje X e Y respectivamente. Los datos de las gráficas resultantes más la señal de control (Figura 49) fueron utilizadas para la identificación del modelo de la planta. Para la obtención de los modelos, se trabajó con la herramienta de MATLAB denominada *systemIdentification* que arranca con la función *ident*.

Como se menciona en (Salcedo Peña, 2018) las gráficas de acumulación de movimiento pueden presentar datos estacionarios no representativos debido a perturbaciones ambientales o inherentes a las altas velocidades de vuelo del dron. Por esa razón, se realizaron las pruebas en un ambiente con pocas perturbaciones como

lapsos de bajas velocidades de viento y una iluminación adecuada (aproximadamente 2000 lux (Grijalva Caisachana, 2019)) y una superficie uniforme.

Figura 50

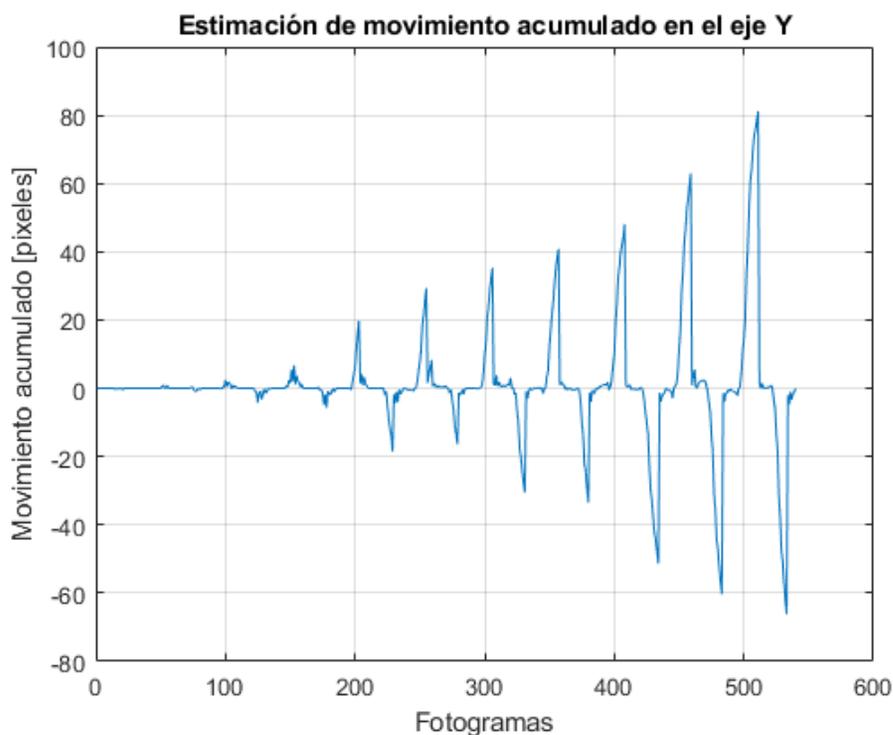
Estimación de movimiento acumulado en el eje X



Nota. Esta gráfica muestra la señal de salida del movimiento acumulado en el eje X.

Figura 51

Estimación de movimiento acumulado en el eje Y



Nota. Esta gráfica muestra la señal de salida del movimiento acumulado en el eje Y.

Obtención de la función de transferencias para el eje X

La herramienta de identificación del modelo trabaja con varios parámetros de entrada para el diseño de la función de transferencia $G(s)$, detallados en la Tabla 15.

Tabla 15

Parámetros para la función de estimación de movimiento en el eje X

Parámetro de entrada	Valor
Señal de entrada	Señal de control de la Figura 49
Modelo medido	Señal de la Figura 50

Parámetro de entrada	Valor
Tiempo de inicio	En 0.0 segundos
Tiempo de muestreo	Un promedio de 25 Hz (0.04 segundos)

Nota. Esta tabla detalla los parámetros utilizados en la herramienta de estimación del modelo para el control de movimiento en el eje X.

El tiempo de muestreo se obtuvo con la velocidad de refresco de los fotogramas enviados desde el dron a la estación en tierra. Teóricamente, alcanzaba una velocidad de 30 Hz, pero durante las diferentes pruebas la frecuencia de envío de los fotogramas se mantenía en un promedio de 25 Hz, que es el valor con el que se trabajó para la identificación del modelo. La función de transferencia para el eje X es de primer orden, mostrada en la expresión (30).

$$G(s) = \frac{K_p}{1 + T_p \cdot s} \quad (30)$$

Los valores resultantes del modelo son:

- $K_p = 87.0089$
- $T_p = 0.2285$

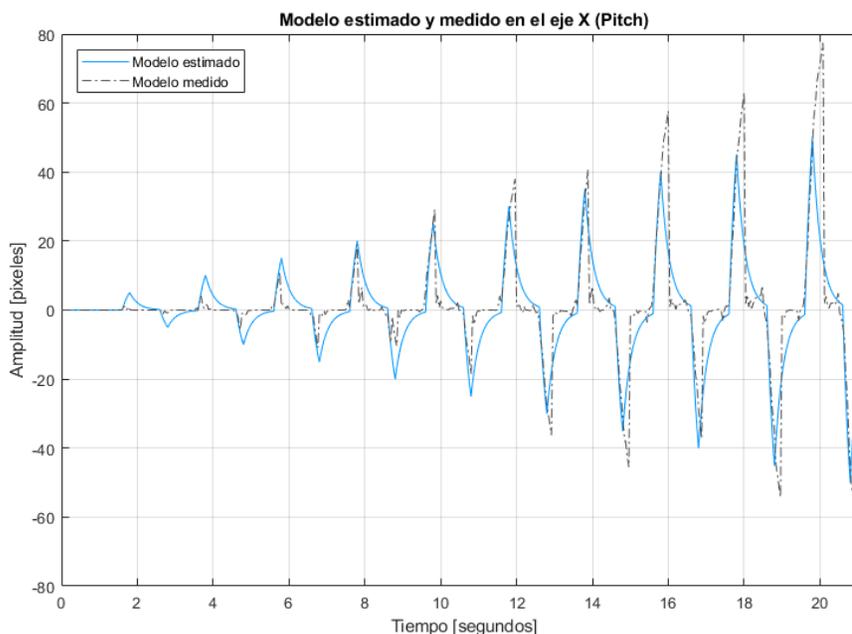
La expresión fina para esta función de transferencia quedaría definida por la expresión (31):

$$G(s) = \frac{87.0089}{1 + 0.2285s} \quad (31)$$

La Figura 52 muestra una comparativa entre el modelo estimado y el modelo medido con un dominio en el tiempo.

Figura 52

Modelo estimado con el medido para el eje X



Nota. Esta gráfica muestra la señal de movimiento acumulado y su estimación.

Obtención de la función de transferencias para el eje Y

Al igual que la identificación del modelo en para el eje X, la herramienta de MATLAB necesita conocer de los mismos parámetros para el eje Y. Es importante recordar que el tiempo de muestreo se rige por la velocidad de la tasa de fotogramas enviados a la estación en tierra, es decir, que se mantiene los 25 Hz o 0.04 segundos.

Tabla 16

Parámetros para la función de estimación de movimiento en el eje Y

Parámetro de entrada	Valor
Señal de entrada	Señal de la Figura 49

Parámetro de entrada	Valor
Modelo medido	Señal de la Figura 51
Tiempo de inicio	En 0.0 segundos
Tiempo de muestreo	Un promedio de 25 Hz (0.04 segundos)

Nota. Esta tabla detalla los parámetros utilizados en la herramienta de estimación del modelo para el control de movimiento en el eje Y.

La función de transferencia para el eje Y es de primer orden, mostrada en la expresión (32).

$$G(s) = \frac{K_p}{1 + T_p \cdot s} \quad (32)$$

Los valores resultantes del modelo son:

- $K_p = 95.4370$
- $T_p = 0.2882$

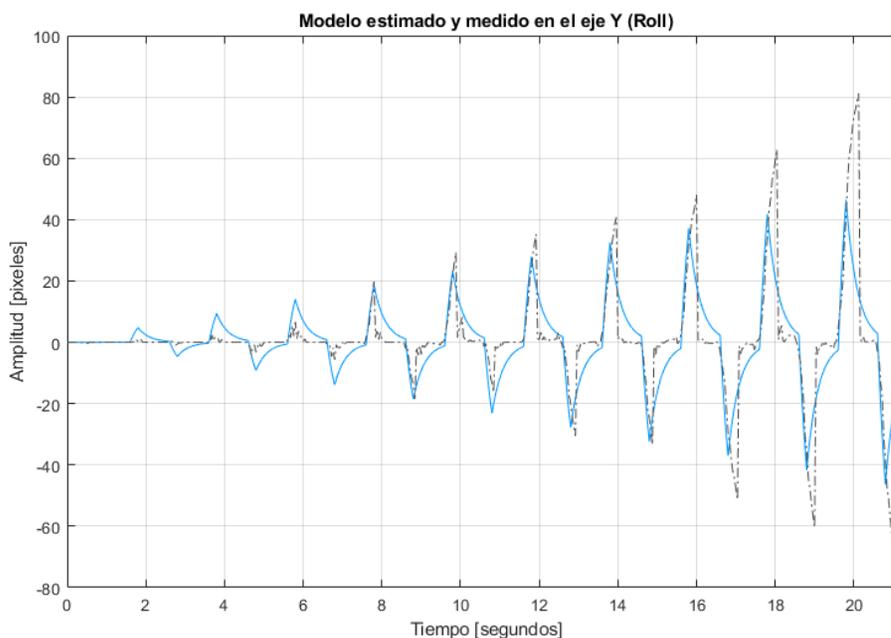
La expresión fina para esta función de transferencia quedaría definida por la expresión (33):

$$G(s) = \frac{95.437}{1 + 0.2882s} \quad (33)$$

La Figura 53 muestra una comparativa entre el modelo estimado y el modelo medido con un dominio en el tiempo.

Figura 53

Modelo estimado con el medido para el eje Y



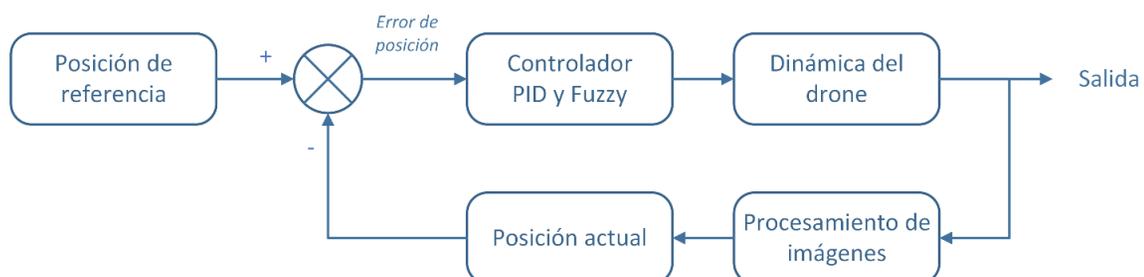
Nota. Esta gráfica muestra la señal de movimiento acumulado y su estimación.

Seguimiento de la plataforma móvil terrestre

El seguimiento del UAV a automóvil en movimiento está diseñado por dos métodos diferentes, el primero utiliza un controlador PID que necesita conocer la función de transferencia de la planta para realizar el diseño del controlador y posterior sintonización de sus ganancias para un desempeño adecuado. En el segundo método se diseña un controlador Difuso; este no necesita conocer el modelo de la planta, pero si un claro conocimiento del comportamiento de las diferentes acciones de control manual, de esta manera es posible transcribir el conocimiento del operador del UAV a una serie de términos lingüísticos para establecer las reglas necesarias que van a definir su comportamiento frente al seguimiento del automóvil y a posibles perturbaciones.

Figura 54

Lazo de control para los desplazamientos en Roll y Pitch del UAV



Nota. Esta gráfica muestra un diagrama para la implementación del sistema de seguimiento de la plataforma utilizando un controlador PID o un difuso.

El objetivo de ambos controladores es mantener al *UAV* cerca del automóvil en movimiento, siguiendo su trayectoria y contrarrestando los posibles y diferentes desplazamientos laterales. El procedimiento de los sistemas de seguimiento es:

- Detección y reconocimiento de la plataforma móvil del resto de objetos de la escena.
- Lectura de la posición de los vértices del cuadro delimitador que encierra a la plataforma.
- Calcular el centroide del cuadro delimitador (posición actual del dron respecto a la plataforma).
- Calcular el error de posición actual del dron con la posición de referencia (centro de la imagen capturada por el dron).
- Enviar el error de posición al control en lazo cerrado (ver Figura 54) para dirigir el movimiento del *UAV* en el eje *X* e *Y*.

- Disminuir gradualmente la altura del UAV hasta lograr el aterrizaje sobre la plataforma móvil.

Es importante recordar que el diseño de los controladores se los realizó para el movimiento en el eje X (Pitch) y para el eje Y (Roll) por separado, es decir, se trabajó con un sistema SISO en ambos casos.

Diseño del Controlador PID

Para diseñar un controlador PID es necesario conocer el modelo de la planta, en este caso el drone. Este modelamiento no se realizó con los datos inerciales de la *IMU* del dispositivo por su baja velocidad de comunicación, por esa razón se trabajó con un modelado servo visual utilizando visión por computadora aplicada a la grabación de una secuencia de vuelo. Se obtuvo las funciones de transferencia para el movimiento acumulado en el eje X e Y descritas en las ecuaciones (31) y (33).

Se trabaja con dos controladores por separado, uno para el eje X (Pitch) y otro para el eje Y (Roll). En el diseño y sintonización de los parámetros de ambos controladores se usa la herramienta de MATLAB llamada *pidTunner*.

Controlador PID para el eje X (Pitch)

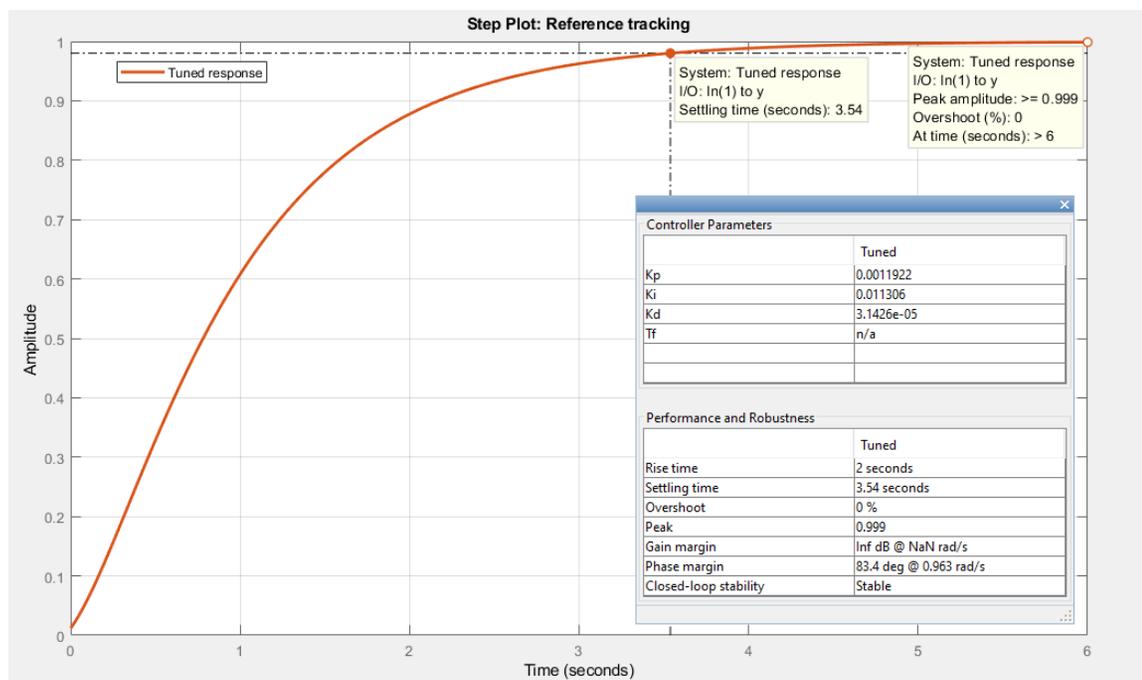
La herramienta de MATLAB necesita como parámetros de entrada la función de transferencia del movimiento acumulado en pitch (Expresión (31)). También se debe especificar algunos parámetros de rendimiento como:

- *Overshoot* $M_p = 0\%$
- *Tiempo de asentamiento* $t_s = 3.5$ segundos

Las dimensiones de la imagen donde se realiza la detección de los objetos tienen un área de 856×480 píxeles. Se considera que el tiempo de establecimiento para el movimiento en pitch debe ser más bajo que para roll porque la relación de aspecto de la imagen es más pequeña en el eje X (sección vertical) necesitando una respuesta más rápida ante perturbaciones. Es por eso que se contempla un tiempo de establecimiento debe bordear los 3.5 segundos.

Figura 55

Respuesta del controlador al escalón unitario del movimiento en el eje X



Nota. Esta gráfica muestra los resultados del diseño del controlador PID para el eje X .

Los parámetros resultantes tras la sintonización del controlador son las ganancias para el control en el eje X (ver Figura 55) representados en su forma paralela y tienen los siguientes valores:

- $K_p = 0.0011922$

- $K_i = 0.011306$
- $K_d = 3.1426 \times 10^{-5}$

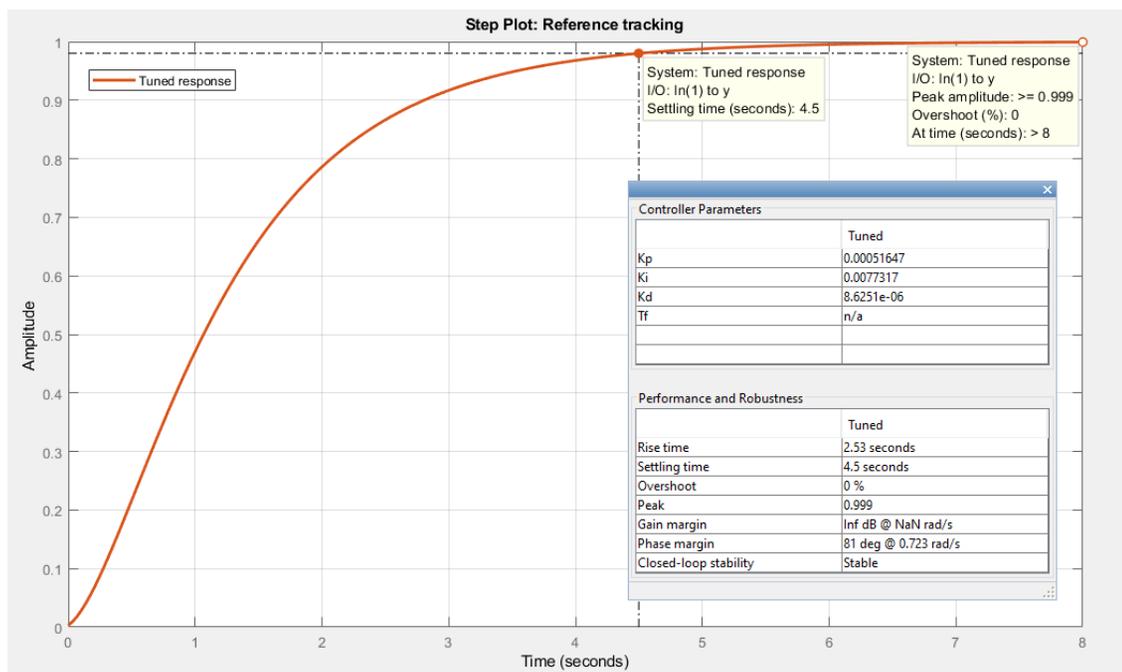
Controlador PID para el eje Y (Roll)

La función de transferencia del movimiento acumulado para roll, mostrada en la expresión (33) es necesaria para el diseño del controlador usando la herramienta *pidTuner*. También se debe especificar algunos parámetros de rendimiento como:

- *Overshoot* $M_p = 0\%$
- *Tiempo de asentamiento* $t_s = 4.5$ segundos

Figura 56

Respuesta del controlador al escalón unitario del movimiento en el eje Y



Nota. Esta gráfica muestra los resultados del diseño del controlador PID para el eje Y.

Se considera que el tiempo de establecimiento para el movimiento en roll debe ser más alto que para pitch porque el área de la imagen es mayor en el eje Y (sección horizontal) permitiendo un mayor tiempo de respuesta ante perturbaciones. Se contempla un tiempo de establecimiento no mayor a los 4.5 segundos.

Los parámetros resultantes tras la sintonización del controlador son las ganancias para el control en el eje Y (ver Figura 56) representados en su forma paralela y tienen los siguientes valores:

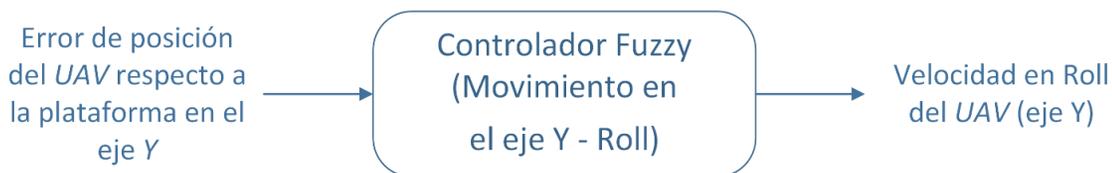
- $K_p = 0.00051647$
- $K_i = 0.0077317$
- $K_d = 8.6251 \times 10^{-6}$

Diseño del Controlador Fuzzy

El primer paso para el diseño del controlador difuso es definir y especificar las variables que entran y salen del controlador. La Figura 57 y Figura 58 describe las variables utilizadas para el seguimiento de la plataforma. Como se ha mencionado, se ha diseñado un controlador por separado para los movimientos en pitch y roll.

Figura 57

Variables de entradas y salida para el control de posición en roll



Nota. Esta gráfica representa las variables de entrada y salida usadas para el diseño del controlador difuso en el eje Y .

Figura 58

Variables de entradas y salida para el control de posición en pitch



Nota. Esta gráfica representa las variables de entrada y salida usadas para el diseño del controlador difuso en el eje X .

El segundo paso es definir las variables de entrada y salida descritas en la Figura 57 y Figura 58 como variables lingüísticas. Para el control de seguimiento se establecen dos variables lingüísticas para cada tipo de movimiento (pitch y roll). Estas variables tienen de 5 propiedades:

- x : Su nombre
- X : Universo de discurso
- $T(x)$: Términos lingüísticos que acepta la variable
- G : Regla sintáctica que genera los valores lingüísticos
- M : Regla semántica que asocia cada término lingüístico con su significado

Variable lingüística del error de posición en el eje Y (Roll)

Para describir a una variable lingüística se debe especificar sus propiedades. Para el movimiento en roll o a lo ancho de la imagen, el controlador tiene una variable de entrada (ver Figura 57).

La variable de entrada tiene las siguientes características:

- **Nombre de la variable (y):** Error de posición del dron en el eje Y (roll) o en el ancho de la imagen.

- **Universo de discurso (Y):** [1; 856] pixeles (Ancho de la imagen usada en la detección de objetos)

En la Tabla 17 se describen los términos lingüísticos y los conjuntos difusos asociados a estos dentro del universo de discurso antes especificado.

Tabla 17

Definición de la variable lingüística del error de posición en el eje Y

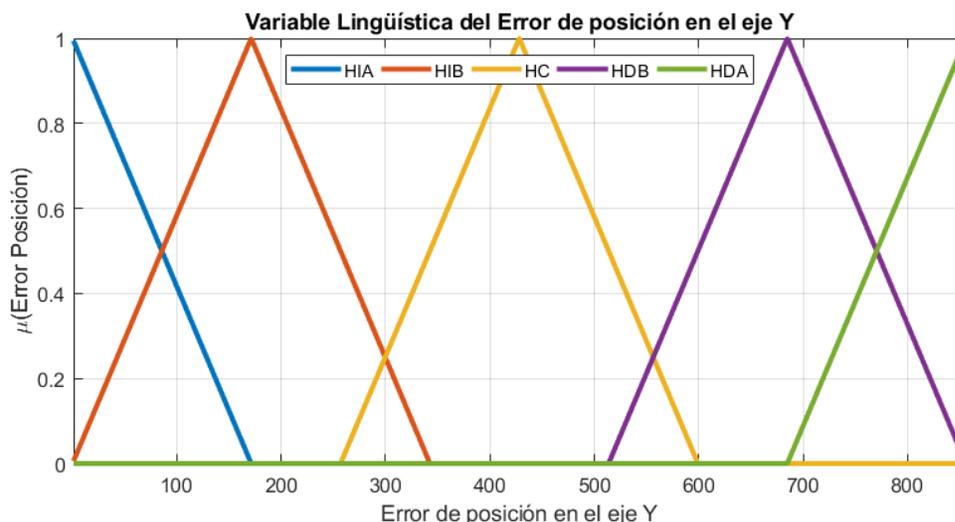
Valor lingüístico	Función de membresía Conjunto difuso	Descripción
HIA (Horizontal Izquierda Alto)	Triangular abierta por la izquierda $M(HIA) = [0; 0; 171.2]$	El error de posición del dron a lo ancho de la imagen es Alto y a la Izquierda
HIB (Horizontal Izquierda Bajo)	Triangular $M(HIB) = [0; 171.2; 342.4]$	El error de posición del dron a lo ancho de la imagen es Bajo y a la Izquierda
HC (Horizontal Central)	Triangular $M(HC) = [256.8; 428; 599.2]$	El error de posición del dron a lo ancho de la imagen es Cero
HDB (Horizontal Derecho Bajo)	Triangular $M(HDB) = [513.6; 684.8; 856]$	El error de posición del dron a lo ancho de la imagen es Bajo y a la Derecha
HDA (Horizontal Derecha Alto)	Triangular abierta por la derecha $M(HDA) = [684.8; 684.8; 856]$	El error de posición del dron a lo ancho de la imagen es Alto y a la Derecha

Nota. En esta tabla se describe las variables lingüísticas utilizadas para definir el error de posición alrededor del eje Y.

La definición de esta variable lingüística se puede resumir de manera gráfica, tal como se muestra en la Figura 59.

Figura 59

Definición de la variable lingüística para el error de posición en el eje Y



Nota. Esta gráfica muestra las funciones de membresía que definen el error de posición a lo largo del eje Y.

Variable lingüística de la velocidad del UAV en eje Y (Roll)

Para la velocidad del dron a lo largo del eje Y (roll) ante posibles desplazamientos laterales de la plataforma móvil mientras sigue su trayectoria rectilínea, el controlador tiene una variable de salida (ver Figura 57).

La variable de salida tiene las siguientes características:

- **Nombre de la variable (roll):** Velocidad del UAV a lo largo del eje Y (roll) o a lo ancho de la imagen.

- **Universo de discurso (*ROLL*):** $[-0.5; 0.5]$ magnitud de velocidad en números decimales.

Tabla 18Definición de la variable lingüística de la velocidad del UAV en el eje *Y*

Valor lingüístico	Función de membresía Conjunto difuso	Descripción
RNA (Roll Negativo Alto)	Triangular abierta por la izquierda $M(RNA) = [-0.5; -0.5; -0.1875]$	La velocidad en eje <i>Y</i> (roll) del drone debe ser Negativa y Alta
RNB (Roll Negativo Bajo)	Triangular $M(RNB) = [-0.25; -0.125; 0.0]$	La velocidad en eje <i>Y</i> (roll) del drone debe ser Negativa y Baja
RC (Roll Cero)	Triangular $M(RC) = [-0.125; 0.0; 0.125]$	La velocidad en eje <i>Y</i> (roll) del drone debe ser Cero
RPB (Roll Positivo Bajo)	Triangular $M(RPB) = [0.0; 0.125; 0.25]$	La velocidad en eje <i>Y</i> (roll) del drone debe ser Positiva y Baja
RPA (Roll Positivo Alto)	Triangular abierta por la derecha $M(RPA) = [0.1875; 0.5; 0.5]$	La velocidad en eje <i>Y</i> (roll) del drone debe ser Positiva y Alta

Nota. Esta gráfica muestra las funciones de membresía que definen el error de posición a lo largo del eje *Y*.

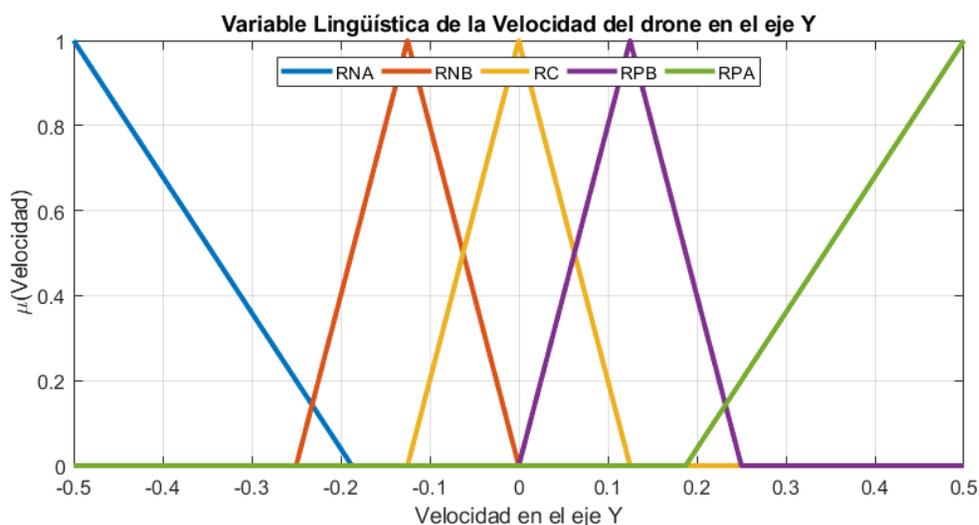
El tópico de velocidad a lo largo del eje *Y* “*bebop.twist.linear.y*” descrito en la Tabla 7 recibe valores flotantes desde -1.0 hasta 1.0 . La razón de trabajar únicamente

con magnitudes desde -0.5 a 0.5 para el universo de discurso se debe a que el seguimiento de la plataforma móvil no requiere velocidades tan altas, se llegó a esta conclusión tras realizar varias pruebas con diferentes universos de discurso de mayores rangos. Se determinó que para valores superiores a $|0.5|$, la respuesta de velocidad del controlador provoca una sobrecompensación del error de posición, generando a su vez un nuevo error, pero en dirección contraria que origina un movimiento oscilatorio del drone nada deseable.

En la Tabla 18 se describen los términos lingüísticos y los conjuntos difusos asociados a estos dentro del universo de discurso antes especificado. La definición de esta variable lingüística se puede resumir de manera gráfica, tal como se muestra en la Figura 60.

Figura 60

Definición de la variable lingüística para la velocidad del dron en el eje Y



Nota. Esta gráfica muestra las funciones de membresía que definen la velocidad del UAV a lo largo del eje Y.

Reglas de control para el seguimiento en el eje Y (Roll)

El control de movimiento en el eje Y necesita cinco términos lingüísticos (para la variable de entrada y salida) para de asociar estos términos uno a uno (ver Tabla 19).

Tabla 19

Reglas de control para el seguimiento del target en el eje Y (Roll)

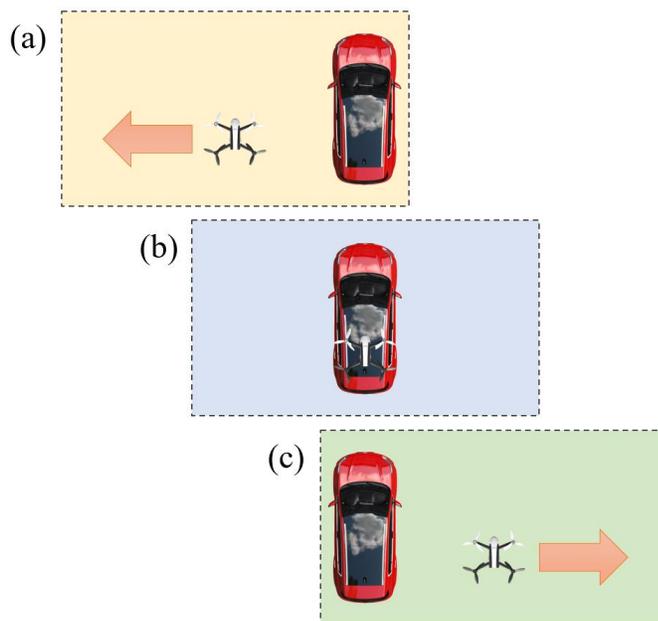
Error de posición	Velocidad	Descripción de la regla
HIA	RPA	Si el error de posición es Alto y a la Izquierda, entonces la velocidad del drone en Roll es Alta y Positiva (movimiento rápido a la izquierda)
HIB	RPB	Si el error de posición es Bajo y a la Izquierda, entonces la velocidad del drone en Roll es Baja y Positiva (movimiento lento a la izquierda)
HC	RC	Si el error de posición es Cero, entonces la velocidad del drone en Roll es Cero (no hay movimiento)
HDB	RNB	Si el error de posición es Bajo y a la Derecha, entonces la velocidad del drone en Roll es Baja y Negativa (movimiento lento a la derecha)
HDA	RNA	Si el error de posición es Alto y a la Derecha, entonces la velocidad del drone en Roll es Alta y Negativa (movimiento rápido a la derecha)

Nota. Esta tabla describe las reglas de control que relacionan el error de posición con la velocidad del UAV a lo largo del eje Y.

Puede parecer un poco contradictorio que para corregir el error de posición de la plataforma respecto al drone, este deba desplazarse al mismo extremo (izquierda o derecha) del error, pero esto se explica claramente en la Figura 61.

Figura 61

Posición del target respecto al movimiento lateral del UAV



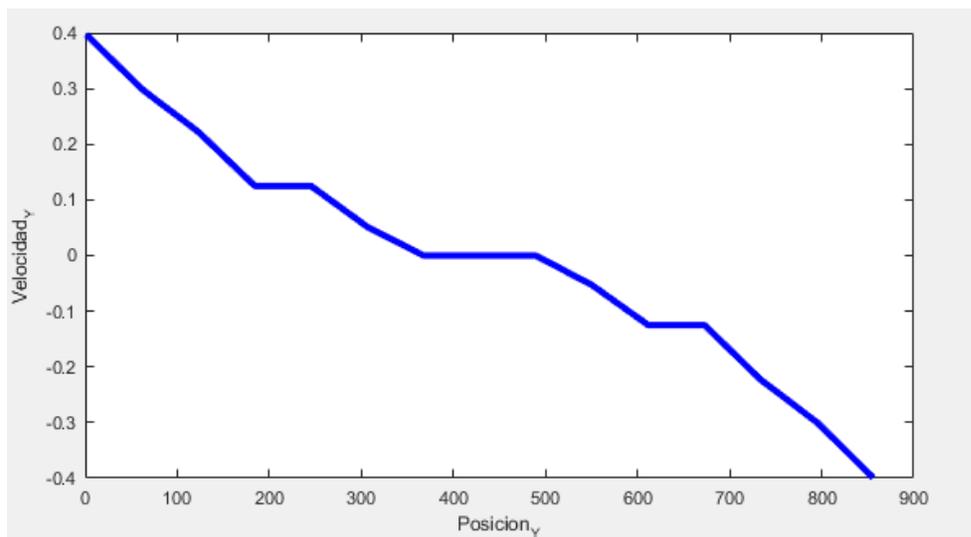
Nota. Esta gráfica muestra la posición del target al lado opuesto que se desplaza el UAV.

Cuando el UAV se desplaza a la izquierda, la plataforma se ubica en la parte derecha de la imagen capturada (Figura 61 (a)) y viceversa, si el UAV se desplaza a la derecha la plataforma se ubica en la parte izquierda de la imagen (Figura 61 (c)). Por esta razón, cuando el error de posición se describe a la izquierda, las reglas de control definen valores positivos de velocidad en Roll, es decir, provoca un desplazamiento a la izquierda centrando nuevamente al UAV sobre la plataforma (Figura 61 (b)). La misma lógica se usa para el error de posición a la derecha.

La curva de control (ver Figura 62) relaciona las variables del controlador, es decir, la entrada (error de posición en el eje Y) vs la salida (velocidad del UAV en el eje Y). Representando el estado del error de posición en todo su universo de discurso y los valores que generaría el controlador para cada valor del error.

Figura 62

Curva de control para el seguimiento en el eje Y



Nota. Esta gráfica representa el comportamiento que tendrá la velocidad del UAV a lo largo del eje Y respecto al error de posición en el mismo eje.

Variable lingüística del error de posición en el eje X (Pitch)

El movimiento en el eje X (pitch) o alto de la imagen, el controlador tiene una variable de entrada (ver Figura 58).

La variable de entrada tiene las siguientes características:

- **Nombre de la variable (x):** Error de posición del dron en el eje X (pitch) o en el alto de la imagen.
- **Universo de discurso (X):** [1; 480] pixeles (Alto de la imagen usada en la detección de objetos)

En la Tabla 20 se describen los términos lingüísticos y los conjuntos difusos asociados a estos dentro del universo de discurso antes especificado.

Tabla 20

Definición de la variable lingüística del error de posición en el eje X

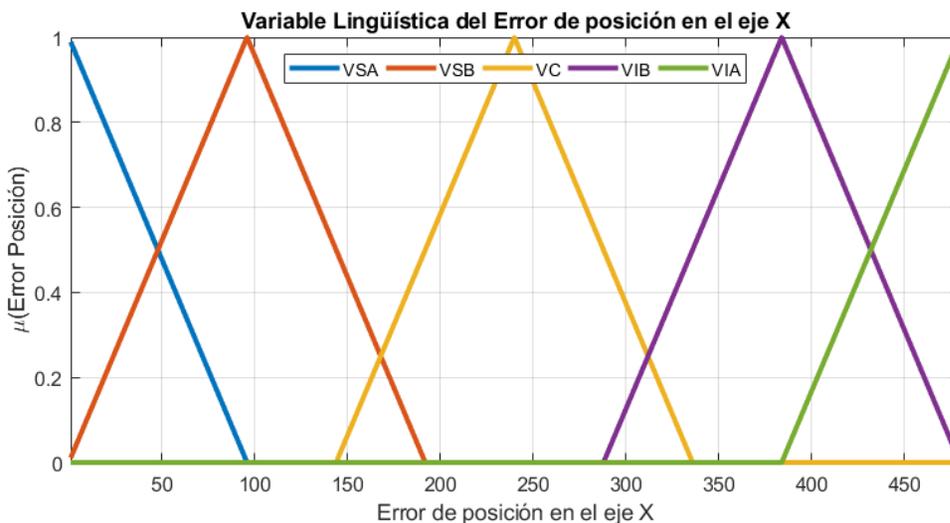
Valor lingüístico	Función de membresía Conjunto difuso	Descripción
VSA (Vertical Superior Alto)	Triangular abierta por la izquierda $M(VSA) = [0; 0; 96]$	El error de posición del drone a lo alto de la imagen es Alto en la parte Superior
VSB (Vertical Superior Bajo)	Triangular $M(VIA) = [0; 96; 192]$	El error de posición del drone a lo alto de la imagen es Bajo en la parte Superior
VC (Vertical Central)	Triangular $M(VC) = [144; 240; 336]$	El error de posición del drone a lo alto de la imagen es Cero
VIB (Vertical Inferior Bajo)	Triangular $M(VIB) = [288; 384; 480]$	El error de posición del drone a lo alto de la imagen es Bajo en la parte Inferior
VIA (Vertical Inferior Alto)	Triangular abierta por la derecha $M(VIA) = [384; 480; 480]$	El error de posición del drone a lo alto de la imagen es Alto en la parte Inferior

Nota. En esta tabla se describe las variables lingüísticas utilizadas para definir el error de posición alrededor del eje Y.

La definición de esta variable lingüística se puede resumir de manera gráfica, tal como se muestra en la Figura 63.

Figura 63

Definición de la variable lingüística para el error de posición en el eje X



Nota. Esta gráfica muestra las funciones de membresía que definen el error de posición a lo largo del eje X.

Variable lingüística de la velocidad del UAV en eje X (Pitch)

Para la velocidad del dron a lo largo del eje X (pitch) siguiendo la trayectoria hacia adelante y atrás de la plataforma móvil el controlador tiene una variable de salida (ver Figura 58).

La variable de salida tiene las siguientes características:

- **Nombre de la variable (*pitch*):** Velocidad del UAV a lo largo del eje X (pitch) o a lo alto de la imagen.
- **Universo de discurso (*PITCH*):** $[-0.5; 0.5]$ magnitud de velocidad en número decimales.

El t3pico de velocidad a los largo del eje X “*bebop.twist.linear.x*” descrito en la Tabla 7 recibe valores flotantes desde -1.0 hasta 1.0 pero 3nicamente se considera valores desde -0.5 a 0.5 para el universo de discurso. La explicaci3n es la misma de la variable de salida para la velocidad en el eje Y descrita en la secci3n 0.

En la Tabla 21 se describen los t3rminos ling33sticos y los conjuntos difusos asociados a estos dentro del universo de discurso antes especificado.

Tabla 21

Definici3n de la variable ling33stica de la velocidad del UAV en el eje X

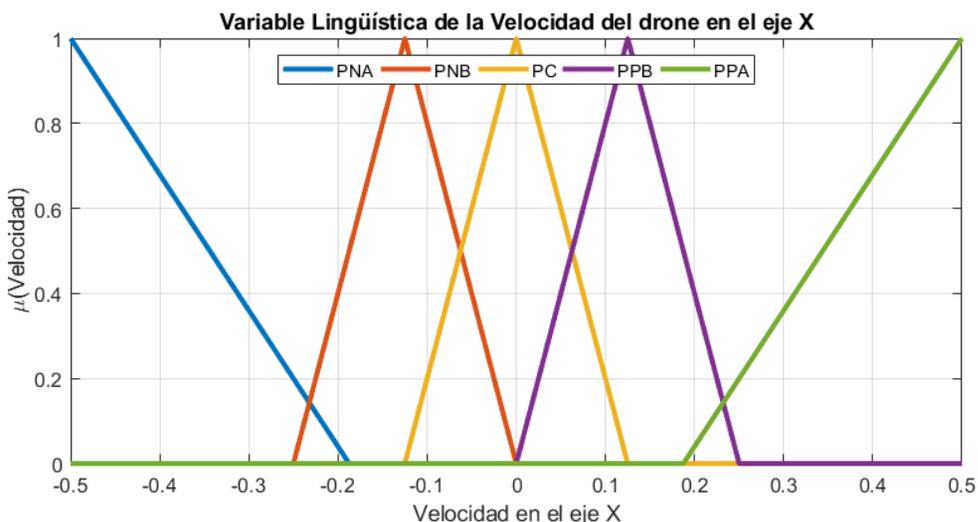
Valor ling33stico	Funci3n de membres3a Conjunto difuso	Descripci3n
PNA (Pitch Negativo Alto)	Triangular abierta por la izquierda $M(PNA) = [-0.5; -0.5; -0.1875]$	La velocidad en eje X (pitch) del dron e debe ser Negativa y Alta
PNB (Pitch Negativo Bajo)	Triangular $M(PNB) = [-0.25; -0.125; 0.0]$	La velocidad en eje X (pitch) del dron e debe ser Negativa y Baja
PC (Pitch Cero)	Triangular $M(PR) = [-0.125; 0.0; 0.125]$	La velocidad en eje X (pitch) del dron e debe ser Cero
PPB (Pitch Positivo Bajo)	Triangular $M(PPB) = [0.0; 0.125; 0.25]$	La velocidad en eje X (pitch) del dron e debe ser Positiva y Baja
PPA (Pitch Positivo Alto)	Triangular abierta por la derecha $M(PPA) = [0.1875; 0.5; 0.5]$	La velocidad en eje X (pitch) del dron e debe ser Positiva y Alta

Nota. Esta gr3fica muestra las funciones de membres3a que definen el error de posici3n a lo largo del eje X .

La definición de esta variable lingüística se puede resumir de manera gráfica, tal como se muestra en la Figura 64.

Figura 64

Definición de la variable lingüística para el error de posición en el eje X



Nota. Esta gráfica muestra las funciones de membresía que definen la velocidad del UAV a lo largo del eje X.

Reglas de control para el seguimiento en el eje X (Pitch)

El control de movimiento en el eje X necesita cinco términos lingüísticos (para la variable de entrada y salida) para de asociar estos términos uno a uno (ver Tabla 22).

Tabla 22

Reglas de control para el seguimiento del target en el eje X (Pitch)

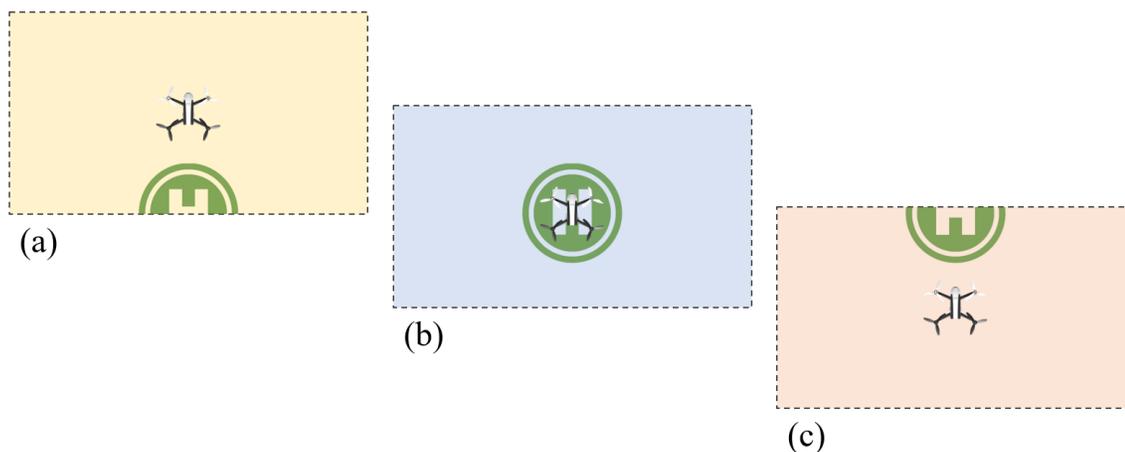
Error de posición	Velocidad	Descripción de la regla
VSA	PPA	Si el error de posición es Alto y de la región Superior, entonces la velocidad del dron en Pitch es Alta y Positiva (movimiento rápido hacia adelante)

VSB	PPB	Si el error de posición es Bajo y de la región Superior, entonces la velocidad del drone en Pitch es Baja y Positiva (movimiento lento hacia adelante)
VC	PC	Si el error de posición es Cero, entonces la velocidad del drone en Pitch es Cero (no hay movimiento)
VIB	PNB	Si el error de posición es Bajo y de la región Inferior, entonces la velocidad del drone en Pitch es Baja y Positiva (movimiento lento hacia atrás)
VIA	PNA	Si el error de posición es Alto y de la región Inferior, entonces la velocidad del drone en Pitch es Alta y Positiva (movimiento rápido hacia atrás)

Nota. Esta tabla describe las reglas de control que relacionan el error de posición con la velocidad del UAV a lo largo del eje X .

Figura 65

Posición del target respecto al movimiento frontal y trasero del UAV



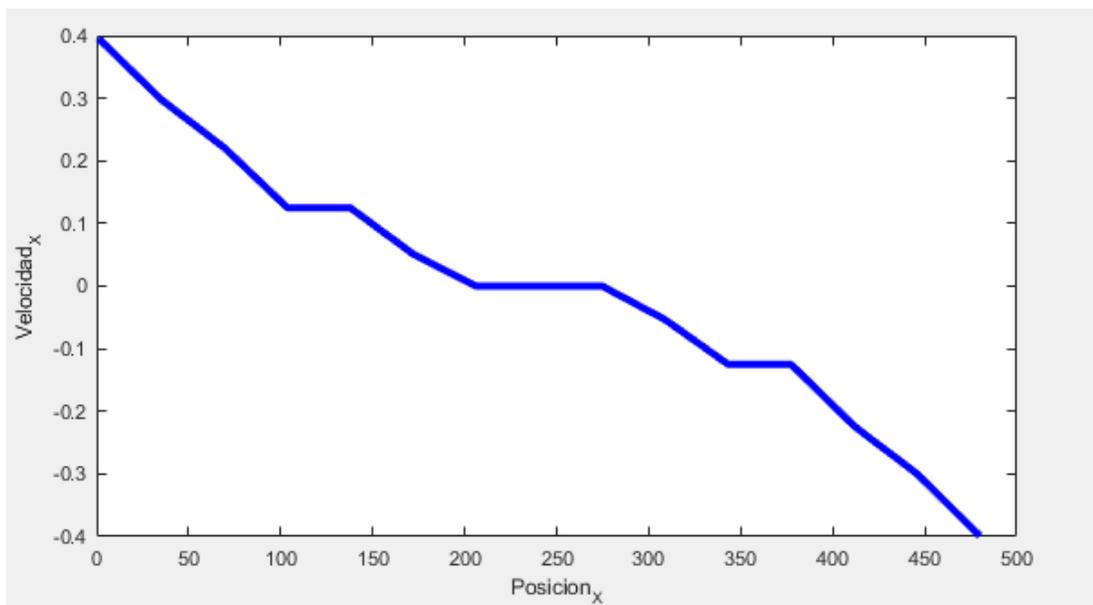
Nota. Esta gráfica muestra la posición del target al lado opuesto que se desplaza el UAV.

Estas reglas de control también pueden parecer un poco contradictorias porque corrigen el error de posición de la plataforma respecto al dron, con un desplazamiento

hacia el mismo lugar (adelante o atrás) del error, pero esto se explica claramente en la Figura 65.

Figura 66

Curva de control para el seguimiento en el eje X



Nota. Esta gráfica representa el comportamiento que tendrá la velocidad del *UAV* a lo largo del eje Y respecto al error de posición en el mismo eje.

Cuando el *UAV* se desplaza a adelante, la plataforma se ubica en la parte de abajo de la imagen capturada (Figura 65 (a)) y viceversa, si el *UAV* se desplaza hacia atrás la plataforma se ubica en la parte superior de la imagen (Figura 65 (c)). Por esta razón, cuando el error de posición se describa hacia abajo, las reglas de control definen valores positivos de velocidad en Pitch, es decir, provoca un desplazamiento para adelante centrando nuevamente al *UAV* sobre la plataforma (Figura 65 (b)). La misma lógica se usa para el error de posición hacia abajo de la imagen.

La curva de control (ver Figura 66) relaciona las variables del controlador, es decir, la entrada (error de posición en el eje X) vs la salida (velocidad del *UAV* a lo largo del eje X). Representando el estado del error de posición en todo su universo de discurso y los valores que generaría el controlador para cada valor del error.

Aterrizaje automático del *UAV* sobre la plataforma móvil

El desarrollo de este sistema de aterrizaje considera una situación donde el *UAV* se encuentre inicialmente en vuelo a una altura suficiente para la detección de automóviles, es decir que este proyecto al igual que (Salcedo Peña, 2018) comienza con un posicionamiento semi manual del *UAV* sobre la plataforma terrestre que marcará una trayectoria rectilínea. El aterrizaje automático del *UAV* no es una acción aislada de todo lo descrito hasta ahora, al contrario. El control del aterrizaje automático parte desde la detección de la plataforma de aterrizaje, continúa con la implementación de los controladores para el seguimiento del automóvil y finalmente termina con un descenso gradual hasta hacer contacto con el vehículo.

Condiciones para el descenso final

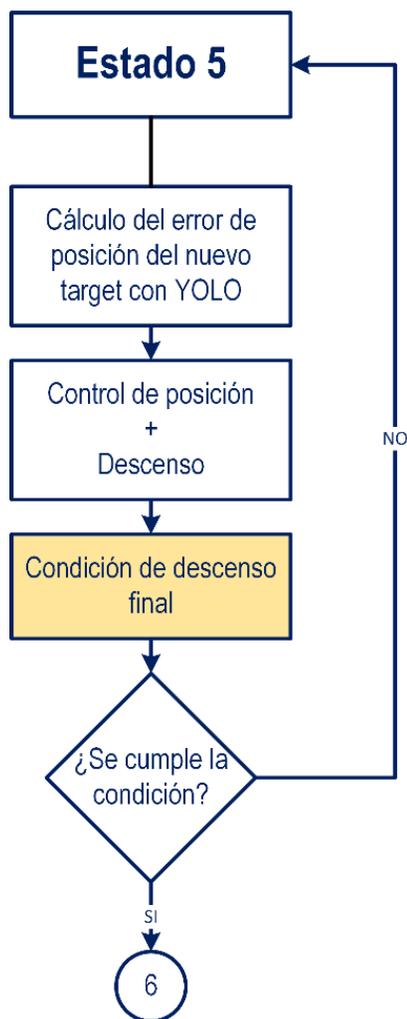
Al iniciar este capítulo se muestra el diagrama de flujo (ver Figura 40) que describe los procesos utilizados para lograr el aterrizaje automático sobre la plataforma móvil. Recordando, el estado 5 incluía un proceso llamado: Condición de descenso final (ver Figura 67). Este proceso incluye diferentes consideraciones antes de llamar a la acción final de aterrizaje.

Estas consideraciones contemplan algunos escenarios donde sería propicio ejecutar la acción final para terminar el aterrizaje. Cada panorama analiza una o varias variables que pueden generar un indicador para terminar el proceso. Fueron diseñados y puestos a prueba para determinar cuál de todos muestra un resultado final. Es importante

aclarar que todas estas consideraciones fueron pensadas para el momento donde el dron se encuentra lo suficientemente cerca para terminar el vuelo.

Figura 67

Estado 5 del diagrama de flujo general del aterrizaje automático



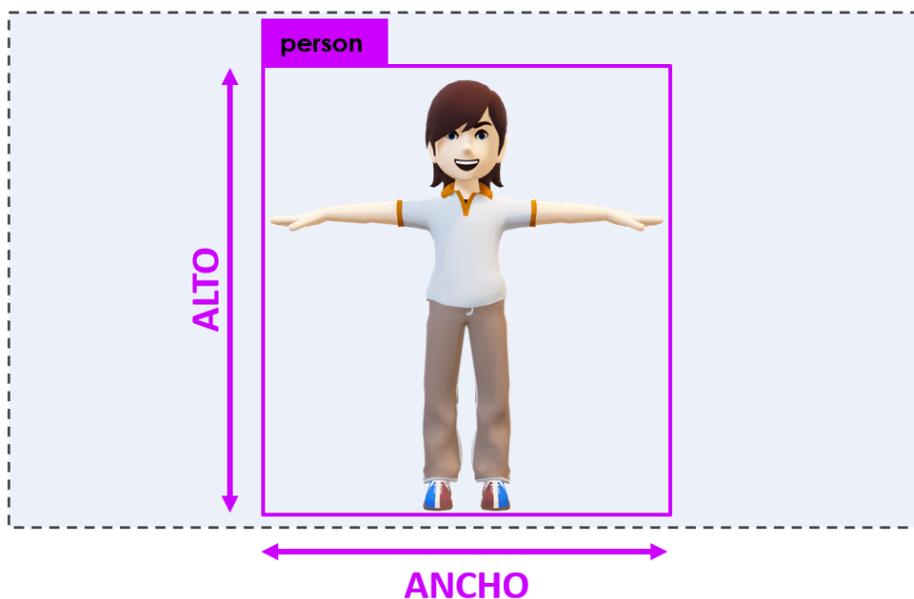
Nota. Esta gráfica muestra el proceso del estado 5 donde se describe de la etapa final del aterrizaje automático.

Condición 1: Por el ancho y alto del target

Los indicadores de este escenario son el ancho y alto del cuadro delimitador de la imagen reconocida por el sistema de detección *YOLO* (ver Figura 68). Se busca generar la señal de culminación de la tarea cuando el *UAV* sobrevuele cerca de la imagen, al encontrarse a una distancia lo suficientemente corta para que el cuadro delimitador de la detección abarque una gran cantidad del área total de visión de la escena, cumplirá la condición para dar paso al estado 6, terminando con el aterrizaje.

Figura 68

Alto y ancho del cuadro delimitador de la detección de objetos



Nota. Esta figura representa el área de detección del cuadro delimitador generado por *YOLO* como condición para el descenso final del *UAV*.

Existen tres formas para generar la señal de activación, estas son:

- Cuando el alto del cuadro delimitador sea mayor a un porcentaje del alto total de imagen (480 pixeles), por ejemplo, si el alto del cuadro delimitador alcance el 90% o 432 pixeles.
- Cuando el ancho del cuadro delimitador sea mayor a un porcentaje del ancho total de imagen (856 pixeles), por ejemplo, si el ancho del cuadro delimitador alcance el 90% o 774 pixeles.
- Cuando el alto y ancho del cuadro delimitador sean mayores a un porcentaje del total del alto y ancho total de imagen (480×856 pixeles).

Condición 2: Por la no detección del target

Cuando el *UAV* se posiciona sobre la imagen colocada en el techo del auto y la distancia que los separa es tan corta lo que en las imágenes capturadas por la cámara del dron no se puede detectar ningún objeto en particular (ver Figura 69). Si el target no es detectado n veces sucesivas, la señal para finalizar el aterrizaje automático se activará.

Figura 69

Condición para finalizar el aterrizaje con la no identificación del target



Nota. Esta figura representa la pérdida de detección del objeto con el sistema YOLO debido a la corta distancia vertical entre el *UAV* y el objeto de interés.

Condición 3: Por la altura del drone

Esta condición es una de las más fáciles de implementar, pero presenta algunos problemas por la baja tasa de refresco de la información de la altura de vuelo (máximo 5 Hz). Si se condiciona la señal de finalización del aterrizaje a cierta altura, la respuesta del algoritmo no sería inmediata y el drone podría mantener el descenso a pesar de estar muy cerca del vehículo provocando colisiones que posiblemente lo arrojen fuera del área de aterrizaje.

Luego de realizar numerosas pruebas, se determinó que la mejor opción para terminar el descenso del *UAV* sobre la plataforma de aterrizaje, es la condición 1, porque es adaptable al tamaño de la plataforma de aterrizaje y a las dimensiones del target. Esta condición también presentó los resultados con mayor repetibilidad, la cercanía del *UAV* a la plataforma al momento del aterrizaje final fue muy similar a lo largo de todas las pruebas, por el contrario, la condición 2 y 3 arrojaron resultados muy variantes.

Capítulo VI

Pruebas y resultados

En este capítulo se busca evaluar el desempeño de los principales procesos que son parte del algoritmo global del aterrizaje automático, algunos fueron diseñados con métodos diferentes o bajo una configuración distinta para lograr el mejor resultado conjunto. Para determinar qué método o configuración funciona mejor bajo las mismas condiciones se prepararon algunas pruebas especializadas. Los procedimientos críticos puestos a prueba son:

- **Detección de objetos.** Sistema que permite la identificación de la plataforma específica de aterrizaje y su posición dentro de las imágenes capturadas por el *UAV*.
- **Control de seguimiento de la plataforma.** Sistema de control en lazo cerrado que trabaja con los actuadores del *UAV* para el seguimiento automático de la plataforma en movimiento.
- **Tiempo de aterrizaje.** El tiempo que necesita exclusivamente proceso de aterrizaje final con la implementación de dos controladores diferentes.
- **Posición final sobre la plataforma.** La posición relativa del *UAV* a la plataforma tras terminar el aterrizaje automático.

El diseño de cada prueba realizada y la mención de los parámetros a evaluar para los procedimientos señalados se detallan a lo largo de este capítulo.

Posicionamiento y seguimiento de objetos detectados

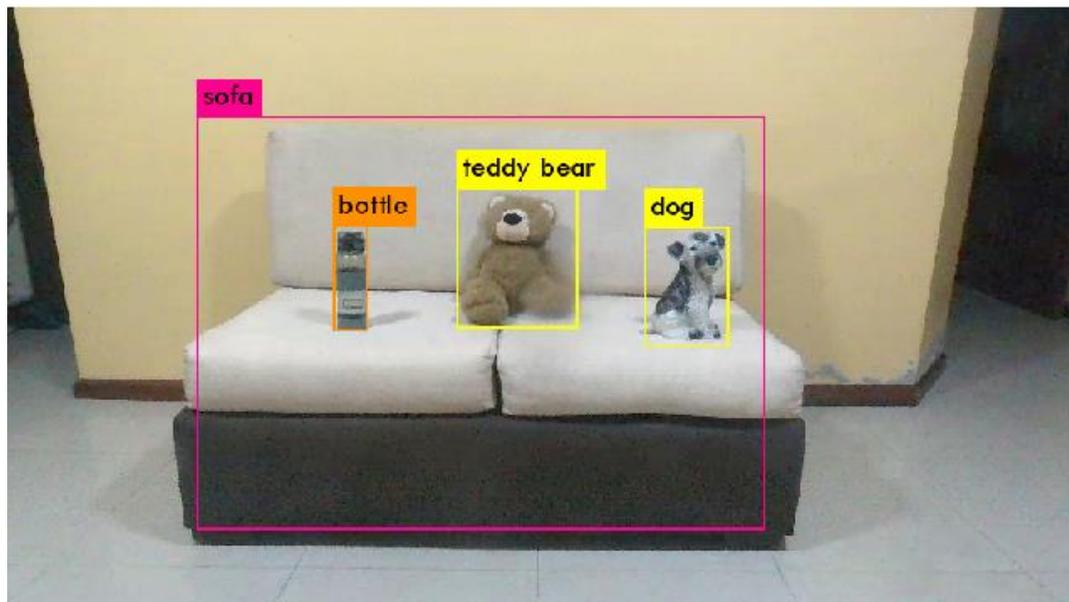
Como se ha mencionado en secciones anteriores, se utiliza un sistema de detección de objetos en tiempo real llamado *YOLO* con la capacidad de reconocer 80

clases diferentes (ver Tabla 9). La evaluación de este sistema busca determinar el desempeño de las herramientas trabajando en conjunto con las imágenes captadas por el UAV. Se considero poner a prueba dos parámetros, el primero es la veracidad del posicionamiento de objetos detectados dentro de la imagen y el segundo es la capacidad de seguimiento a diferentes alturas.

Para esta prueba se colocó un grupo de objetos dentro de las 80 clases entrenadas para verificar la veracidad del posicionamiento que entrega el cuadro delimitador del sistema dentro de la imagen y verificarlo de manera visual. La Figura 70 muestra el señalamiento de la detección de los objetos.

Figura 70

Detección de objetos en imágenes del UAV



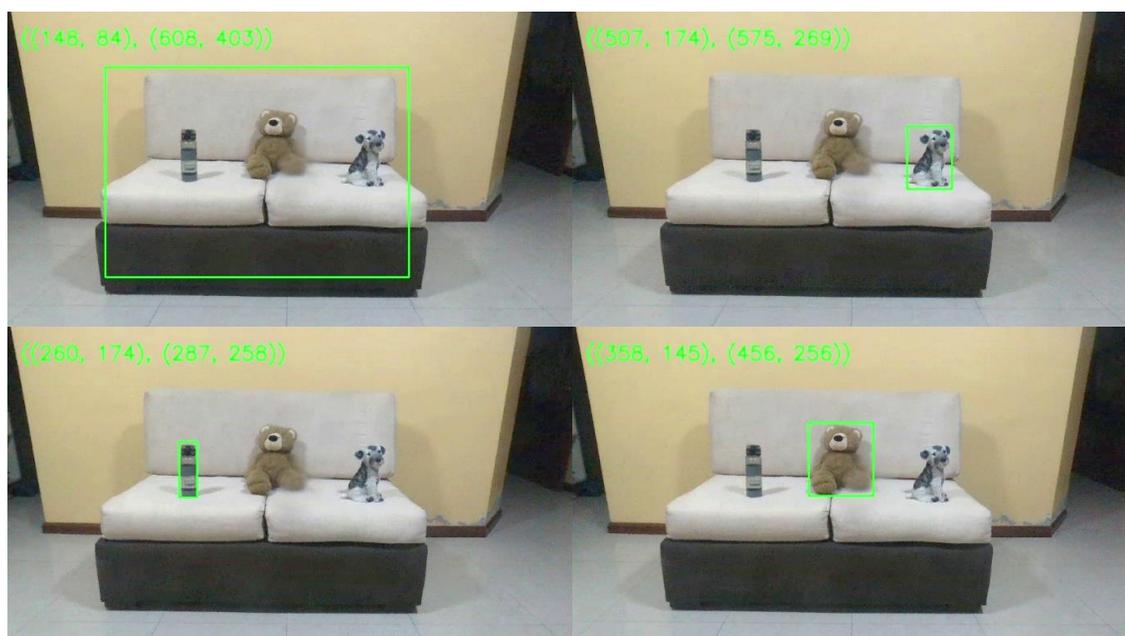
Nota. Esta imagen muestra el sistema de detección de objetos YOLO implementado en varios objetos dentro de una escena capturada por el UAV.

Con una rápida inspección visual del señalamiento de los cuadros delimitadores se puede determinar que:

- El montaje de *YOLO* en la estación en tierra como parte del algoritmo de comunicación y control del *UAV* arroja resultados muy satisfactorios en el posicionamiento de los objetos detectados.
- La velocidad de procesamiento del sistema de detección de objetos varía entre los 24 a 25 *Hz*, siendo muy adecuado porque la tasa de refresco de las imágenes capturadas por el *UAV* mantiene velocidades similares, permitiendo la detección de la plataforma en todas las imágenes recibidas. Es importante destacar que la velocidad de *YOLO* depende de algunas configuraciones de su sistema, pero sobre todo de los recursos del equipo donde esté implementado.

Figura 71

Cuadro delimitador del algoritmo de seguimiento KCF con imagen inicial



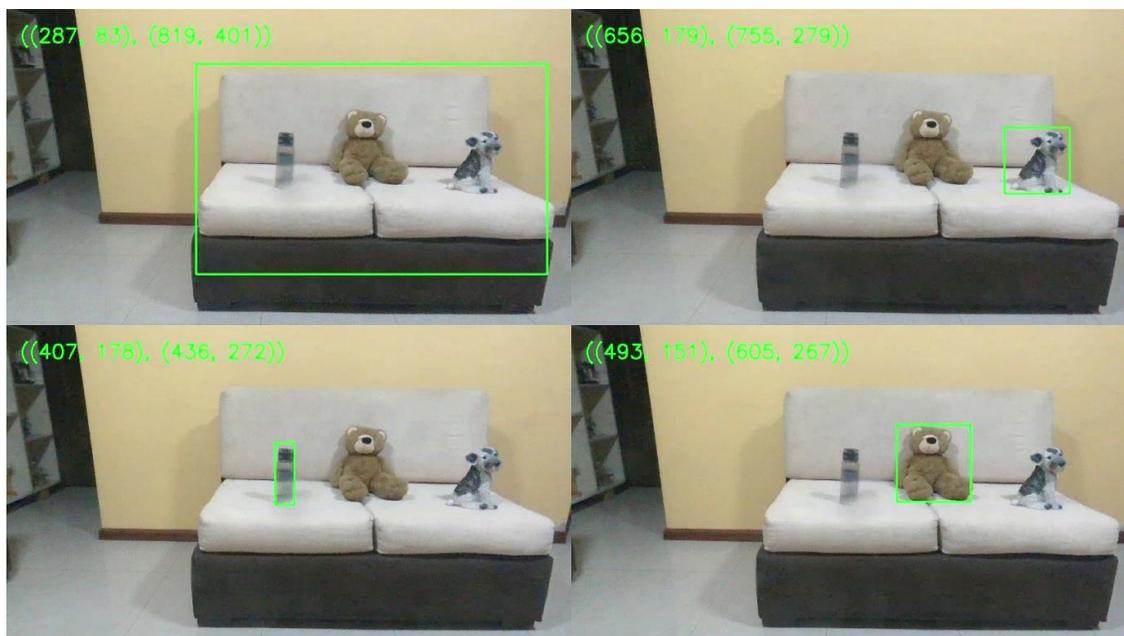
Nota. La imagen muestra la implementación del fotograma inicial del algoritmo KCF.

Como ya se determinó que *YOLO* es adecuado para trabajar junto al algoritmo de comunicación y control del dron, la siguiente prueba busca evaluar el desempeño del algoritmo de seguimiento *KFC* a partir de la información de los cuadros delimitadores generados por *YOLO*. Se trabaja con cada uno de los objetos detectados como lo muestra la Figura 71 para evaluar el desempeño del seguimiento.

En la Figura 71 se muestra un cuadro de color verde que encierra al objeto previamente detectado, este señalamiento ya no denota la detección de *YOLO*, ahora representa el objetivo de seguimiento del algoritmo *KCF*. En la Figura 72 se muestra el desempeño del seguimiento, el *UAV* fue rotado en el eje *Z* en sentido anti horario y a pesar de eso el cuadro delimitador mantiene el seguimiento.

Figura 72

Algoritmo de seguimiento KCF con imagen desplazada



Nota. Esta imagen muestra como el algoritmo de seguimiento *KCF* mantiene el cuadro delimitador sobre el objeto a pesar de su desplazamiento dentro de la imagen.

Para establecer los resultados de la prueba de seguimiento con el algoritmo *KCF* se realiza una inspección visual entre la Figura 71 y Figura 72, determinando que:

- El seguimiento de los objetos de interés tiene un buen rendimiento y no se ve afectado por la posición ni área que ocupe dentro de la imagen.
- Para esta prueba, las características de seguimiento *KCF*, que mantienen el alto y ancho del cuadro de detección de YOLO produjeron un gran resultado, pero pueden ser un problema cuando el objeto cambia de escala, dicho de otra manera, que el *UAV* varíe su altura. La siguiente prueba demuestra el problema mencionado.

Seguimiento del target respecto a la altura del *UAV*

Este proyecto contempla el seguimiento y aterrizaje de *UAV* desde una altura adecuada para la detección de objetos con el sistema *YOLO*, el primer paso es la detección de la plataforma móvil luego enviar la información del cuadro delimitador *KCF* y trabajar con el sistema de seguimiento.

Figura 73

Detección y señalamiento del objeto para el seguimiento

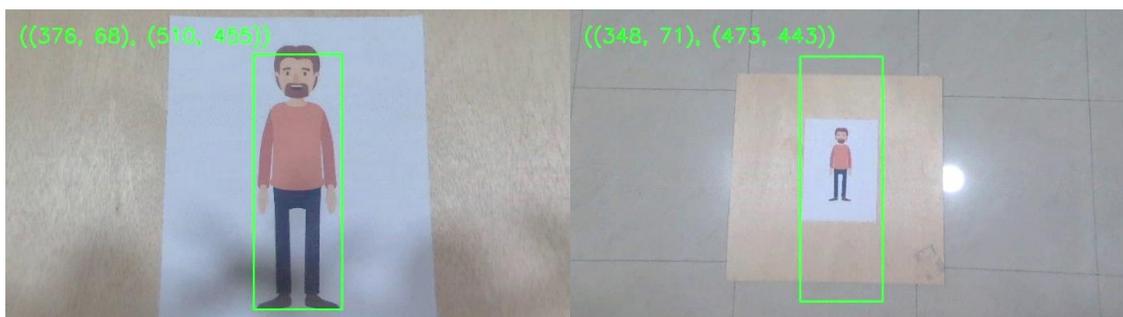


Nota. Esta imagen muestra como el sistema *YOLO* envía la posición del objeto al algoritmo *KCF* marcando el mismo cuadro delimitador para el seguimiento.

Se realiza una prueba para determinar si la variación de la altura afecta al cuadro que señala el objetivo del seguimiento con *KCF*, en la Figura 73 se muestra la detección con YOLO e inmediato señalamiento del objetivo de seguimiento a una altura inicial de 100cm . La prueba consiste en verificar si el señalamiento del objetivo de seguimiento mantiene la veracidad de su posición real dentro de la imagen a pesar de variar la altura del dron. Se crean escenarios a dos alturas diferentes. La Figura 74 muestra el resultado del seguimiento a una altura de 100cm y 25cm .

Figura 74

Error de escala con KCF



Nota. Esta imagen representa el error del algoritmo de seguimiento respecto a la escala, al variar la altura de la cámara que captura al target.

Con los resultados se determina que:

- El algoritmo de seguimiento *KCF* presenta problemas respecto a la escala de la imagen, es decir, que mientras el dron cambie su altura las dimensiones del objeto de interés también variarán, pero el cuadro de señalamiento se mantendrá del mismo alto y ancho perjudicando la veracidad del seguimiento del target.

- Las pruebas también señalan que YOLO no presenta el problema de escala de KCF (ver Figura 75), por lo que la etapa final del aterrizaje automático trabaja directamente con este sistema.

Figura 75

Comportamiento de YOLO a diferentes escalas



Nota. Esta imagen muestra el sistema de detección YOLO no presenta problemas de escala al variar la altura de la cámara que captura al target.

Sistema de control para el seguimiento de la plataforma

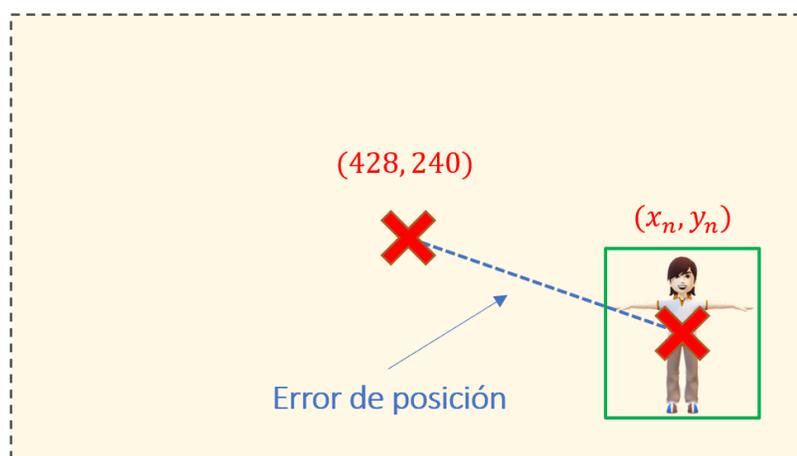
Las pruebas hechas para la evaluación del sistema de seguimiento automático de un objeto específico buscan determinar qué controlador tiene mejor desempeño frente al error de posición del objeto al centro de la imagen capturada por el *UAV* (ver Figura 76).

Para determinar este error de posición, se limitó al algoritmo a la etapa de detección y seguimiento, además se estableció un tiempo límite de 25 segundos antes de terminar la prueba, en ese lapso el *UAV* vuela sobre la plataforma de aterrizaje (una superficie de 60×60 cm con una imagen estampada dentro de las 80 clases que YOLO puede detectar) controlando que el centro de la imagen capturada coincida con el centro de la plataforma. La prueba consiste en desplazar manualmente al *UAV* una vez que se haya posicionado el centro de su imagen con el centro del objeto de seguimiento, en ese

momento se provocará una perturbación empujando levemente al dron en el eje X (adelante y atrás) e Y (izquierda y derecha), se busca comparar las respuestas de ambos controladores ante estos desplazamientos.

Figura 76

Distancia entre los centros de la imagen y del cuadro del target



Nota. Esta gráfica representa el concepto error de posición cálculo desde los centros de la imagen y del cuadro delimitador usando píxeles como unidad de medida.

Prueba del control de seguimiento en el eje X positivo

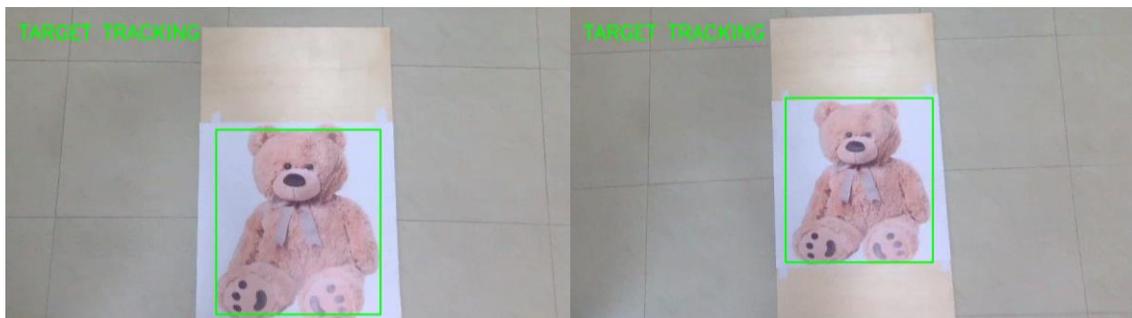
El objetivo de estas pruebas es comparar el rendimiento del controlador PID vs el controlador Difuso bajo las mismas condiciones, aplicando una perturbación con un desplazamiento del dron hacia delante (pitch positivo), el objeto de interés pareciera moverse hacia abajo en la imagen, este efecto visual se explica en la Figura 65. El desplazamiento a lo largo del eje X se representa en la Figura 77.

Al desplazar el dron hacia delante, el algoritmo de seguimiento contrarresta esta acción provocando que los actuadores traten de mover al dron en dirección contraria (pitch negativo) para nuevamente ubicar el centro de la imagen en el centro del cuadro

que encierra al objetivo del seguimiento. La Figura 78 muestra la evolución del error de posición en los 25 segundos programados para conocer el comportamiento de los controladores.

Figura 77

Desplazamiento hacia delante del UAV



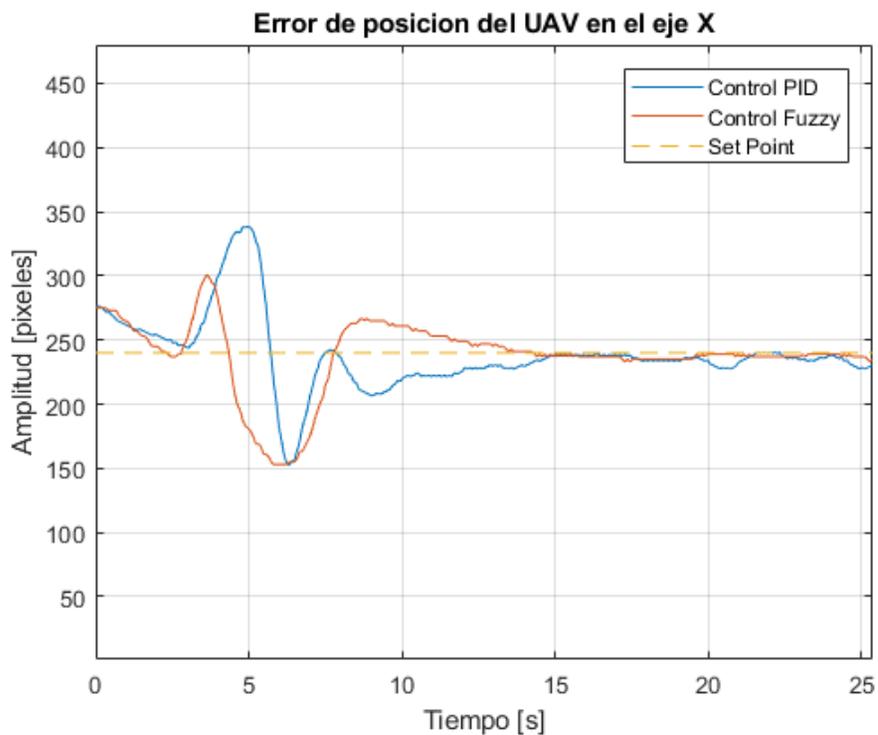
Nota. Esta imagen representa el desplazamiento hacia adelante del UAV, dejando a la plataforma de aterrizaje inmóvil.

Clos resultados mostrados en la Figura 78 se determina que:

- Al comparar las respuestas de los controladores se puede determinar que el control PID presenta un mayor overshoot que el control difuso, pero este último se comporta mejor en estado estable, no presenta las pequeñas oscilaciones como si lo hace el control PID.
- El tiempo de establecimiento de ambos controladores de muy similar, alrededor de los 15s.

Figura 78

Error de posición en eje X con perturbación del UAV hacia delante



Nota. Esta imagen representa el comportamiento del seguimiento usando dos controladores diferentes frente a una perturbación desplazando el UAV hacia delante.

Prueba del control de seguimiento en el eje X negativo

De igual manera se aplicará una perturbación con un desplazamiento del dron hacia atrás (pitch negativo), el objeto de interés pareciera moverse hacia arriba en la imagen. El desplazamiento a lo largo del eje X se representa en la Figura 79.

El algoritmo de seguimiento contrarresta esta acción provocando que los actuadores traten de mover dron en dirección contraria (pitch positivo) para nuevamente ubicar el centro de los fotogramas en el centro del cuadro que encierra al objetivo del seguimiento. La Figura 80 representa la evolución del error de posición.

Figura 79

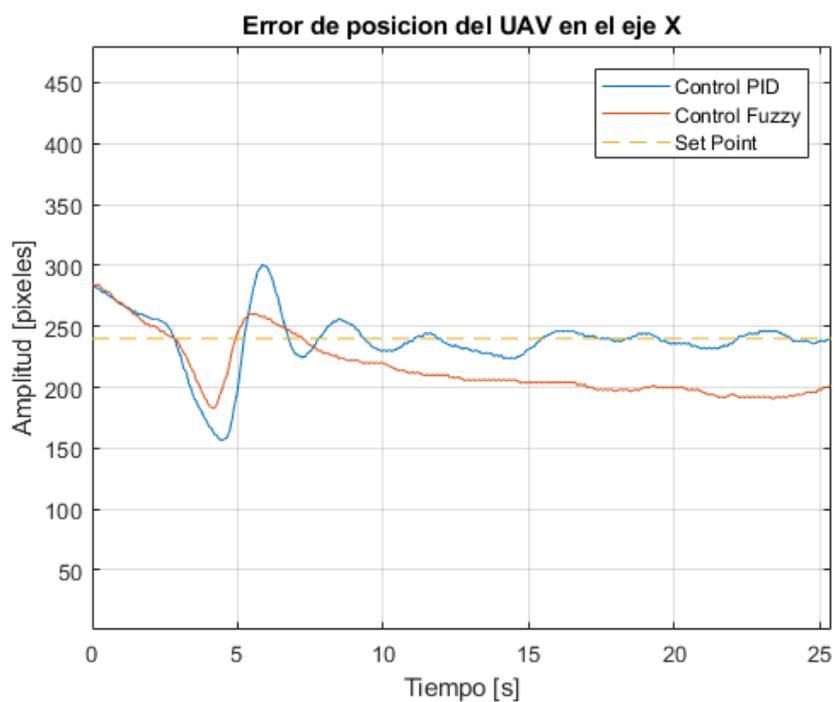
Desplazamiento hacia atrás del UAV



Nota. Esta imagen representa el desplazamiento hacia atrás del UAV, dejando a la plataforma de aterrizaje inmóvil.

Figura 80

Error de posición en eje X con perturbación del UAV hacia atrás



Nota. Esta imagen representa el comportamiento del seguimiento usando dos controladores diferentes frente a una perturbación desplazando el UAV hacia atrás.

Con los resultados mostrados en la Figura 80.

- La respuesta del controlador Fuzzy presenta un error en estado estacionario evidente, el dron no se ubicaría completamente en el centro de la plataforma. En cambio, el controlador PID, a pesar de presentar un mayor overshoot tiende a mantener al dron centrado respecto al objeto de interés.
- EL controlador PID mantiene los 15s en su tiempo de establecimiento.

Prueba del control de seguimiento en el eje Y positivo

Se aplicará una perturbación con un desplazamiento del dron hacia su lado izquierdo (roll positivo), el objeto de interés pareciera moverse hacia la derecha de la imagen. El desplazamiento a lo largo del eje Y se representa en la Figura 81.

Figura 81

Desplazamiento hacia la izquierda del UAV

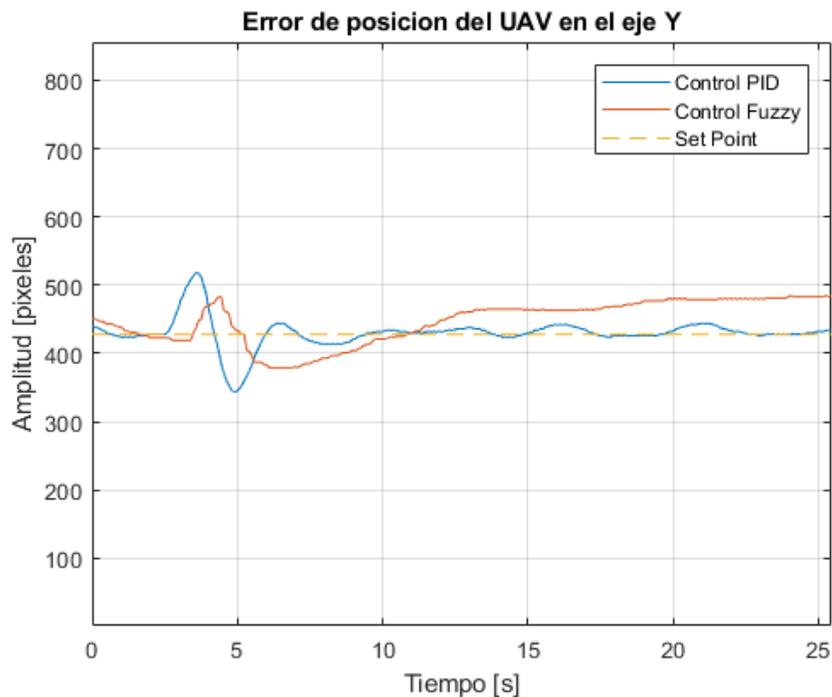


Nota. Esta imagen representa el desplazamiento hacia la izquierda del UAV, dejando a la plataforma de aterrizaje inmóvil.

El algoritmo de seguimiento contrarresta esta acción provocando que los actuadores traten de mover dron en dirección contraria (roll negativo) para nuevamente ubicar el centro de la imagen en el centro del cuadro que encierra al objetivo del seguimiento. La Figura 82 representa la evolución del error de posición.

Figura 82

Error de posición en eje Y con perturbación del UAV hacia la izquierda



Nota. Esta imagen representa el comportamiento del seguimiento usando dos controladores diferentes frente a una perturbación desplazando el UAV hacia la izquierda.

La Figura 82 exhibe los resultados para concluir que:

- La respuesta del controlador PID tiene un overshoot mayor que el controlador Difuso, sus resultados son muy similares a la prueba realiza en el eje X con un desplazamiento hacia atrás. Con el controlador PID el drone tiene a estabilizar su imagen en valores muy cercanos al set point pero con pequeñas oscilaciones. Al contrario, el controlador difuso, mantiene un error en estado estacionario que físicamente se refleja en un desplazamiento lateral del centro de la plataforma no deseado.

- EL tiempo de establecimiento del controlador PID se redujo a aproximadamente 10s.

Prueba del control de seguimiento en el eje Y negativo

Se aplicará una perturbación con un desplazamiento del dron hacia su lado derecho (roll negativo), el objeto de interés pareciera moverse hacia la derecha de la imagen. El desplazamiento a lo largo del eje Y se representa en la Figura 83.

Figura 83

Desplazamiento hacia la derecha del UAV



Nota. Esta imagen representa el desplazamiento hacia la derecha del UAV, dejando a la plataforma de aterrizaje inmóvil.

El algoritmo de seguimiento contrarresta esta acción provocando que los actuadores traten de mover dron en dirección contraria (roll positivo) para nuevamente ubicar el centro de la imagen en el centro del cuadro que encierra al objetivo del seguimiento. La Figura 84 representa la evolución del error de posición.

Con los resultados mostrados en la Figura 84 se determina que:

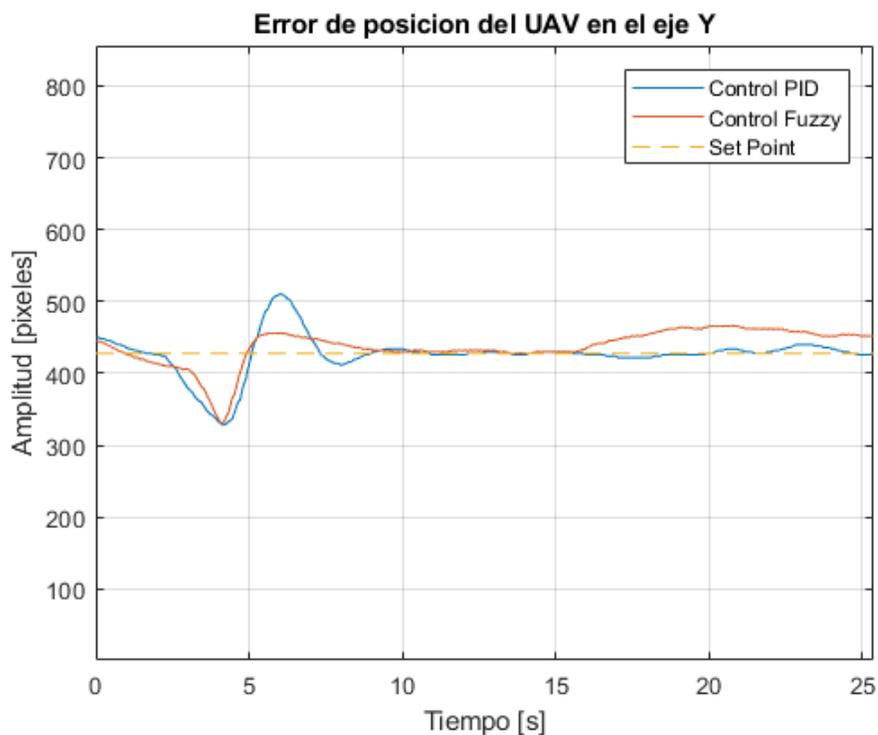
- Esta prueba en particular muestra una respuesta del controlador difuso con un overshoot menor que el controlador PID y hasta cierto punto, una respuesta en estado estacionario muy adecuada hasta el segundo 15, a partir de ese instante

comienza a notarse un error de posición, en cambio el control PID al igual que el resto de las pruebas mantiene un comportamiento adecuado en estado estacionario.

- El tiempo de establecimiento de ambos controladores es menor similar y menor a los 10s.

Figura 84

Error de posición en eje Y con perturbación del UAV hacia la derecha



Nota. Esta imagen representa el comportamiento del seguimiento usando dos controladores diferentes frente a una perturbación desplazando el UAV hacia la derecha.

Tiempo y posición final del UAV sobre la plataforma de aterrizaje

El algoritmo de aterrizaje automático, en su estado 5 (ver Figura 40) ejecuta un descenso gradual del UAV hasta cumplir una condición para terminar completamente con

el seguimiento y aterrizaje sobre la plataforma móvil. En secciones anteriores procedimientos considerados como detonantes de la señal de culminación del proceso. Tras numerosas pruebas se determinó que la mejor alternativa es la condición 1: Por el alto o ancho del cuadro de detección del objeto. Bajo este requisito, se realizaron algunas pruebas para determinar el tiempo promedio de aterrizaje con la implementación del controlador PID y Fuzzy.

La Tabla 23 registra el tiempo del reconocimiento más el descenso final del dron usando el controlador PID, el tiempo total es la suma de estos valores más aproximadamente 5 segundos programados del seguimiento.

Tabla 23

Tiempos de detección y descenso con el controlador PID

Número de prueba con controlador PID	Tiempo de detección [s]	Tiempo de descenso [s]	Tiempo total [s]
Prueba 1	0.8567	3.9142	9.8008
Prueba 2	0.3329	4.0796	9.4500
Prueba 3	0.3991	4.5733	9.9994
Prueba 4	0.3123	3.7347	9.0947

Nota. Esta tabla muestra los tiempos de descenso y detección utilizados en 4 pruebas similares utilizando el controlador PID con los mismos parámetros de altura y distancia recorrida.

En la Tabla 24 se registra los mismos parámetros resultantes de la utilización de controlador Difuso.

Tabla 24

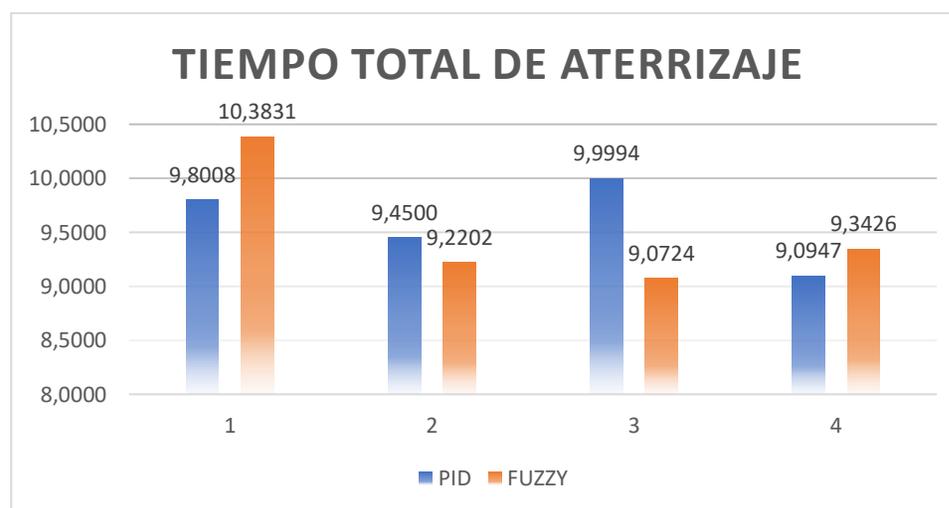
Tiempos de detección y descenso con el controlador Difuso

Número de prueba con controlador Fuzzy	Tiempo de detección [s]	Tiempo de descenso [s]	Tiempo total [s]
Prueba 1	0.4934	4.889	10.3831
Prueba 2	0.3540	3.8368	9.2202
Prueba 3	0.3125	3.7476	9.0724
Prueba 4	0.3974	3.9268	9.3426

Nota. Esta tabla muestra los tiempos de descenso y detección utilizados en 4 pruebas similares utilizando el controlador PID con los mismos parámetros de altura y distancia recorrida.

Figura 85

Tiempo total de aterrizaje entre el controlador PID y Fuzzy



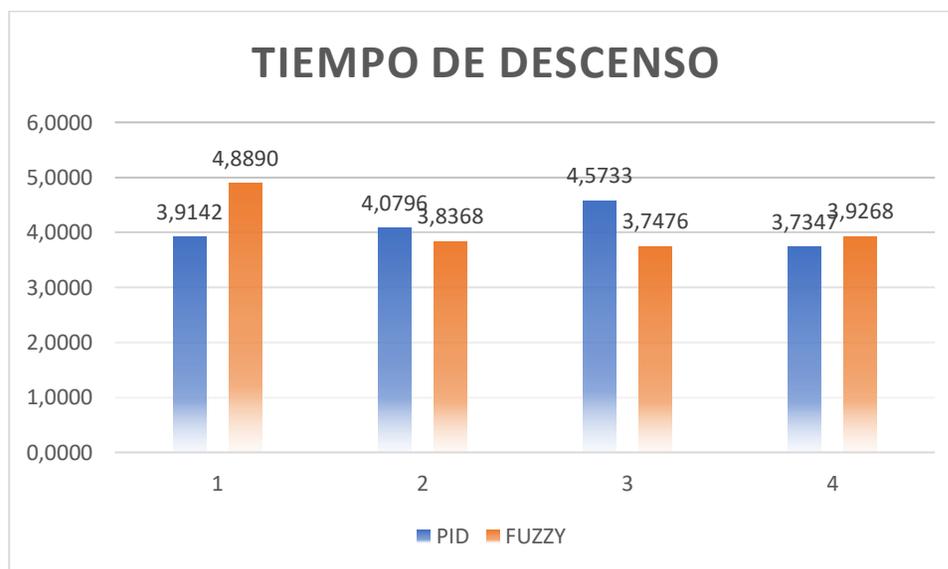
Nota. Esta gráfica representa una comparativa del tiempo total de aterrizaje del UAV usando de dos controladores diferentes.

El tiempo total de aterrizaje de para ambos controladores suma el tiempo de detección, seguimiento y descenso final, la Figura 85 evidencia valores muy variantes, no es posible afirmar que un controlador sea más rápido que otro.

Una alternativa para determinar que controlador tarda menos es comprar solamente el tiempo de descenso del UAV para cada controlador, tal como se aprecia en la Figura 86.

Figura 86

Tiempo de descenso entre el controlador PID y Fuzzy



Nota. Esta gráfica representa una comparativa del tiempo del descenso final del UAV sobre la plataforma móvil usando de dos controladores diferentes.

Con el tiempo de descenso registrado en ambos controladores no es posible definir que uno sea más rápido que otro, no existe un claro indicador para hacerlo.

Un parámetro importante es la posición del UAV dentro de la plataforma al terminar el aterrizaje, se recuerda que en estas pruebas se usó una superficie de madera

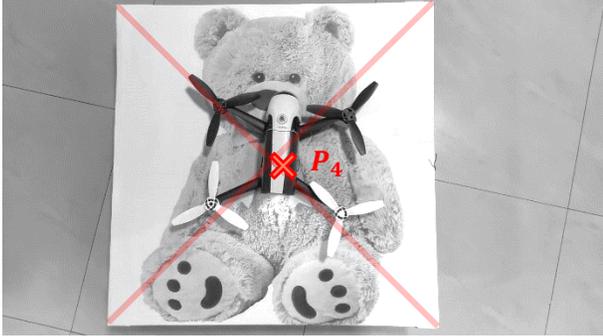
de $60 \times 60\text{cm}$ con una imagen apta para la detección de YOLO, es decir que sea parte de las 80 clases de entrenamiento del sistema. La distancia recorrida por la plataforma móvil en cada prueba es aproximadamente 220cm .

En la Tabla 25 se muestra la posición final del UAV sobre la plataforma móvil para el algoritmo que trabaja con el controlador PID, se calcula al error de posición del centro de la plataforma.

Tabla 25

Error de posición del UAV sobre la plataforma con controlador PID

IMAGEN DE LA PRUEBA	POSICIÓN FINAL DEL UAV
<p data-bbox="537 961 662 993" style="text-align: center;">Prueba 1</p> 	<p data-bbox="1052 961 1292 993" style="text-align: center;">$P_1(25\text{cm} \ 25\text{cm})$</p> $Error_{P_1}$ $= \sqrt{(30 - 25)^2 + (30 - 25)^2} \text{ cm}$ $Error_{P_1} = \sqrt{25 + 25} \text{ cm}$ $Error_{P_1} = 7.07 \text{ cm}$
<p data-bbox="537 1419 662 1451" style="text-align: center;">Prueba 2</p> 	<p data-bbox="1052 1419 1292 1451" style="text-align: center;">$P_2(32\text{cm} \ 30\text{cm})$</p> $Error_{P_1}$ $= \sqrt{(30 - 32)^2 + (30 - 30)^2} \text{ cm}$ $Error_{P_1} = \sqrt{4 + 0} \text{ cm}$ $Error_{P_1} = 2.00 \text{ cm}$

IMAGEN DE LA PRUEBA	POSICIÓN FINAL DEL UAV
<p data-bbox="537 352 662 384" style="text-align: center;">Prueba 3</p> 	<p data-bbox="1052 352 1295 384" style="text-align: center;">$P_3(41cm \ 24cm)$</p> $Error_{P_1}$ $= \sqrt{(30 - 41)^2 + (30 - 24)^2} \text{ cm}$ $Error_{P_1} = \sqrt{121 + 36} \text{ cm}$ $Error_{P_1} = 12.53 \text{ cm}$
<p data-bbox="537 806 662 837" style="text-align: center;">Prueba 4</p> 	<p data-bbox="1052 806 1295 837" style="text-align: center;">$P_4(31cm \ 32cm)$</p> $Error_{P_1}$ $= \sqrt{(30 - 31)^2 + (30 - 32)^2} \text{ cm}$ $Error_{P_1} = \sqrt{1 + 4} \text{ cm}$ $Error_{P_1} = 2.24 \text{ cm}$

Nota. Esta tabla muestra la posición final del UAV tras el descenso final usando el controlador PID.

En la Tabla 26 se muestra el posicionamiento final del UAV sobre la plataforma móvil para el algoritmo que trabaja con el controlador Difuso, también se calcula al error de posición del centro de la plataforma.

Tabla 26

Error de posición del UAV sobre la plataforma con controlador difuso

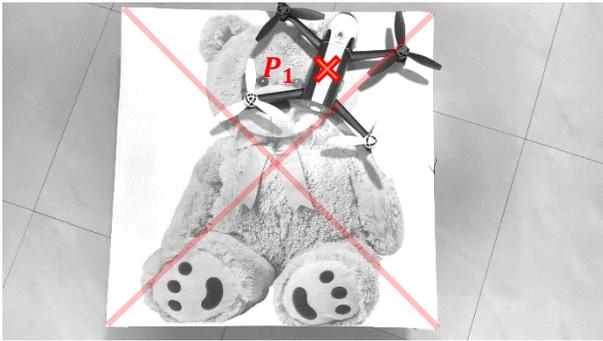
IMAGEN DE LA PRUEBA	POSICIÓN FINAL DEL UAV
<p data-bbox="537 516 659 543">Prueba 1</p> 	<p data-bbox="1052 516 1292 543">$P_1(39cm \ 14cm)$</p> $Error_{P_1}$ $= \sqrt{(30 - 39)^2 + (30 - 14)^2} \text{ cm}$ $Error_{P_1} = \sqrt{81 + 256} \text{ cm}$ $Error_{P_1} = 18.36 \text{ cm}$
<p data-bbox="537 963 659 991">Prueba 2</p> 	<p data-bbox="1052 963 1292 991">$P_2(29cm \ 53cm)$</p> $Error_{P_1}$ $= \sqrt{(30 - 29)^2 + (30 - 53)^2} \text{ cm}$ $Error_{P_1} = \sqrt{1 + 529} \text{ cm}$ $Error_{P_1} = 23.02 \text{ cm}$
<p data-bbox="537 1411 659 1438">Prueba 3</p> 	<p data-bbox="1052 1411 1292 1438">$P_3(43cm \ 38cm)$</p> $Error_{P_1}$ $= \sqrt{(30 - 43)^2 + (30 - 38)^2} \text{ cm}$ $Error_{P_1} = \sqrt{169 + 64} \text{ cm}$ $Error_{P_1} = 15.26 \text{ cm}$

IMAGEN DE LA PRUEBA	POSICIÓN FINAL DEL UAV
<p data-bbox="537 352 665 384" style="text-align: center;">Prueba 4</p> 	<p data-bbox="1049 346 1295 380" style="text-align: center;">$P_4(25cm \ 13cm)$</p> $Error_{P_1}$ $= \sqrt{(30 - 25)^2 + (30 - 13)^2} \text{ cm}$ $Error_{P_1} = \sqrt{25 + 289} \text{ cm}$ $Error_{P_1} = 17.72 \text{ cm}$

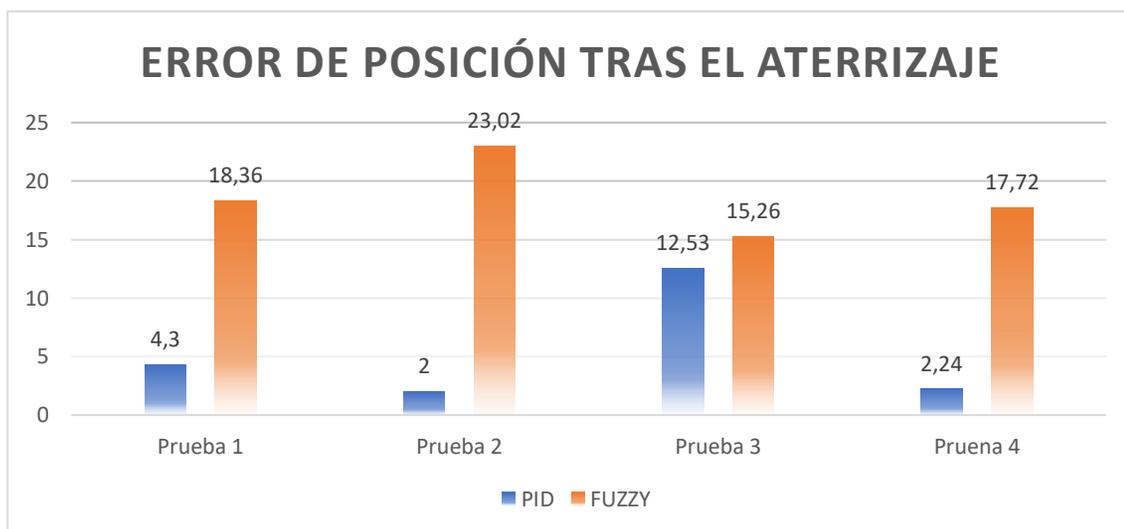
Nota. Esta tabla muestra la posición final del UAV tras el descenso final usando el controlador difuso. En la Figura 87 se muestra una comparativa de los errores de posición del UAV respecto al centro de la plataforma al usar el controlador Difuso y el PID.

Se determina que:

- El error de posición del UAV al centro de la plataforma de aterrizaje es claramente menor al utilizar el controlador PID, la distancia entre el centro del dron y de la plataforma de aterrizaje no supera los 13cm.
- A pesar que el controlador Difuso tiene un desempeño menor que el controlador PID, la mayor distancia al centro de la plataforma de aterrizaje es 23,02cm que es relativamente baja si se considera un área de aterrizaje mayor como el techo de un vehículo.

Figura 87

Error de posición del UAV sobre la plataforma móvil al aterrizar



Nota. Esta gráfica muestra una comparativa entre el error de posición del *UAV* respecto al centro de la superficie de aterrizaje al terminas el descenso.

Prueba del algoritmo completo

En la Figura 88 se muestra el algoritmo en funcionamiento bajo la implementación del controlador PID, la altura con la que parte el *UAV* es de 7 metros a nivel del piso. El tiempo total que tomó el aterrizaje automático es 43 segundos, donde el automóvil recorrió una distancia aproximada de 60 metros, a una velocidad promedio de 5 *km/h*.

Figura 88

Funcionamiento del algoritmo de aterrizaje con el controlador PID





Nota. Estas imágenes muestran el seguimiento y posterior aterrizaje con un descenso gradual del UAV hasta alcanzar el techo del automóvil en movimiento.

Capítulo VII

Conclusiones y recomendaciones

Conclusiones

Se diseñó un sistema de detección y seguimiento visual para plataformas móviles aptas para el aterrizaje automático de un *UAV* utilizando un sistema de detección de objetos en tiempo real llamado YOLO como método de reconocimiento y discriminación de la plataforma de aterrizaje del resto de objetos dentro de la imagen capturada. La detección del objeto de interés proporciona su posición en píxeles dentro de la imagen, información utilizada para la implementación del algoritmo de seguimiento KCF.

El algoritmo de seguimiento KCF genera un cuadro sobre el objeto de interés que representa su posición dentro de la imagen a partir de la información del sistema de detección YOLO, pero el alto y ancho del cuadro permanecen constantes a pesar de la variación de escala del objeto marcado producida por el cambio de la altura del *UAV*. Se concluyó que el algoritmo KFC necesita una actualización constante de la escala del objeto de interés si la captura de la imagen es sometida a variaciones de altura.

Se evaluó la implementación del sistema de detección YOLO en el streaming de video enviado por el *UAV* a una tasa de 25 *Hz* utilizando objetos de distintos tamaños en diferentes ubicaciones dentro de la imagen. La velocidad de detección del sistema YOLO mantenía un promedio de 25 FPS gracias a la intervención de una Unidad de Procesamiento Gráfico (GPU) permitiendo la detección del objeto de interés en todos los fotogramas del video. Se concluye que para un adecuado rendimiento del sistema de detección YOLO (superior a los 20 FPS) en el reconocimiento de la plataforma se necesita de altos recursos computacionales como una GPU de buenas características.

Se diseñó dos sistemas de control de movimientos en los ejes X e Y para el seguimiento de la plataforma móvil previamente detectada. El control PID fue diseñado a partir del modelamiento servo visual y el control difuso utilizó la experiencia del operador para la elaboración de las reglas de control. En la comparación de ambos controladores se concluyó que las respuestas del controlador difuso presentan un menor overshoot, en cambio las respuestas del control PID muestran un menor tiempo de establecimiento y un error en estado estacionario de 3%, siendo fuertes indicadores a considerar en la implementación en el sistema presentando un mejor desempeño.

En las pruebas realizadas para la evaluación de la etapa final del algoritmo de aterrizaje del *UAV* sobre una plataforma móvil de $60 \times 60\text{cm}$ con una imagen estampada se pudo concluir que el tiempo necesario para el seguimiento y descenso final del *UAV* bajo la implementación del controlador PID es muy similar a la del control Difuso. Por otro lado, la prueba de error de posición final que evalúa la distancia del centro del *UAV* respecto al centro de la plataforma, determinó que el controlador PID es muy superior, el error promedio en 4 pruebas similares es 5.27cm mientras que el controlador Difuso presentó un error promedio de 18.59cm .

Se determinó que para la detección y seguimiento del automóvil en la etapa inicial del algoritmo completo para de aterrizaje automático se la debe hacer a una altura superior a los 5 metros desde el piso para que las imágenes capturadas con una vista cenital del *UAV* contengan por completo la figura del automóvil y el sistema de detección YOLO logre su correcto reconocimiento.

Recomendaciones

Se recomienda hacer una inspección del *UAV* previa al vuelo para asegurarse de la correcta instalación y ajuste de sus hélices para evitar bruscas interrupciones en el

vuelo. La acertada y firme colocación de su batería impedirá un repentina y aparatosa caída del UAV a raíz de movimientos rápidos que se puedan presentar.

Para vuelos en interiores, se recomienda tener mucho cuidado con la superficie del lugar, si esta presenta una textura muy uniforme y reflejante el UAV tiende a desplazarse sin ninguna acción de control a lo largo de los ejes X o Y . Se recomienda colocar algunos objetos de referencia sobre la superficie de la posible trayectoria a cursar. Y para vuelos en exteriores se recomienda respetar las restricciones de la guía de vuelo del fabricante para el vehículo como: no volar en condiciones meteorológicas adversas (lluvia, nieve, niebla, vientos fuertes ni durante la noche).

El sistema de aterrizaje automático usa visión computacional y algoritmos de inteligencia artificial para determinar la posición del final de aterrizaje. En este tipo de tareas, la velocidad de respuesta del algoritmo es importante para generar acciones de control dependiendo del resultado del procesamiento de imágenes y evitar colisiones o accidentes durante el vuelo automático. Se recomienda trabajar con equipos que incorporen hardware especializado para el procesamiento de imágenes como GPUs para lograr una respuesta en tiempo real.

Se recomienda el uso del controlador de ROS llamado "bebop_autonomy" para entablar la comunicación inalámbrica *Wi-Fi* entre el UAV Bebop 2 de la marca Parrot y la estación en tierra. Este controlador permite acceder a las imágenes del drone o parámetros de odometría como también acciones el envío de acciones de control para definir movimientos de traslación o rotación en los tres ejes X , Y y Z .

Referencias bibliográficas

- Aguilar, W. G., Morales, S., Ruiz, H., & Abad, V. (2017). RRT* GL Based Path Planning for Virtual Aerial Navigation. *Augmented Reality, Virtual Reality, and Computer Graphics*, 176–184.
- Aguilar, W. G., & Angulo, C. (2012). Compensación de los efectos generados en la imagen por el control de navegación del robot Aibo ERS 7. *VII Congreso de Ciencia y Tecnología ESPE*. Sangolquí, Ecuador.
- Aguilar, W. G., & Angulo, C. (2012). Compensación y aprendizaje de efectos generados en la imagen durante el desplazamiento de un robot. *X Simposio CEA de Ingeniería de Control*. Barcelona, Spain.
- Aguilar, W. G., & Angulo, C. (2013). Estabilización robusta de vídeo basada en diferencia de nivel de gris. *VIII Congreso de Ciencia y Tecnología ESPE*. Sangolquí, Ecuador.
- Aguilar, W. G., & Angulo, C. (2014). Compensación de los Efectos Generados en la Imagen por el Control de Navegación del Robot Aibo ERS 7. *In Revista Digital Congreso de Ciencia y Tecnología: Memorias. Sesiones Técnicas*, 155-160.
- Aguilar, W. G., & Angulo, C. (2014). Estabilización de vídeo en micro vehículos aéreos y su aplicación en la detección de caras. *IX Congreso de Ciencia y Tecnología ESPE*. Sangolquí, Ecuador.
- Aguilar, W. G., & Angulo, C. (2014). Real-time video stabilization without phantom movements for micro aerial vehicles. *EURASIP Journal on Image and Video Processing*, 2014(1).

- Aguilar, W. G., & Angulo, C. (2014). Real-time video stabilization without phantom movements for micro aerial vehicles. *EURASIP Journal on Image and Video Processing*, 1, 1-13.
- Aguilar, W. G., & Angulo, C. (2014). Robust video stabilization based on motion intention for low-cost micro aerial vehicles. *2014 IEEE 11th International Multi-Conference on Systems, Signals & Devices (SSD14)*.
- Aguilar, W. G., & Angulo, C. (2015). Real-Time Model-Based Video Stabilization for Microaerial Vehicles. *Neural Processing Letters*, 43(2), 459–477.
- Aguilar, W. G., & Angulo, C. (2016). Real-Time Model-Based Video Stabilization for Microaerial Vehicles. *Neural Processing Letters*, 43(2), 459-477.
- Aguilar, W. G., & Morales, S. (2016). 3D Environment Mapping Using the Kinect V2 and Path Planning Based on RRT Algorithms. *Electronics*, 5(4), 70.
- Aguilar, W. G., & Morales, S. G. (2016). 3D Environment Mapping Using the Kinect V2 and Path Planning Based on RRT Algorithms. *Electronics*, 5(4), 70.
- Aguilar, W. G., Abad, V., Ruiz, H., Aguilar, J., & Aguilar-Castillo, F. (2017). RRT-Based Path Planning for Virtual Bronchoscopy Simulator. En *Lecture Notes in Computer Science* (págs. 155-165).
- Aguilar, W. G., Álvarez, L., Grijalva, S., & Rojas, I. (2019). Monocular Vision-Based Dynamic Moving Obstacles Detection and Avoidance. *Lecture Notes in Computer Science*, 386–398.

- Aguilar, W. G., Álvarez, L., Grijalva, S., & Rojas, I. (2019). Monocular Vision-Based Dynamic Moving Obstacles Detection and Avoidance. En *Lecture Notes in Computer Science* (págs. 386-398). Germany: Springer.
- Aguilar, W. G., Angulo, C., & Costa-Castello, R. (2017). Autonomous Navigation Control for Quadrotors in Trajectories Tracking. En *Lecture Notes in Computer Science* (págs. 287-297).
- Aguilar, W. G., Angulo, C., & Pardo, J. A. (2017). Motion intention optimization for multirotor robust video stabilization. *2017 CHILEAN Conference on Electrical, Electronics Engineering, Information and Communication Technologies*. (CHILECON).
- Aguilar, W. G., Angulo, C., & Pardo, J. A. (2017). Motion intention optimization for multirotor robust video stabilization. *Chilean Conference on Electrical, Electronics Engineering, Information and Communication Technologies (CHILECON)*. Pucón, Chile.
- Aguilar, W. G., Angulo, C., Costa, R., & Molina, L. (2014). Control autónomo de cuadricópteros para seguimiento de trayectorias. *IX Congreso de Ciencia y Tecnología ESPE*. Sangolquí, Ecuador.
- Aguilar, W. G., Casaliglla, V. P., & Pólit, J. L. (2017). Obstacle Avoidance Based-Visual Navigation for Micro Aerial Vehicles. *Electronics*, 6(1), 10.
- Aguilar, W. G., Casaliglla, V. P., & Pólit, J. L. (2017). Obstacle Avoidance Based-Visual Navigation for Micro Aerial Vehicles. . *Electronics*, 6(1).

- Aguilar, W. G., Costa Castello, R., & Angulo, C. (2017). Autonomous Navigation Control for Quadrotors in Trajectories Tracking. *Lecture Notes in Computer Science*, 287-297.
- Aguilar, W. G., Luna, M. A., Moya, J. F., Abad, V., Parra, H., & Ruiz, H. (2017). Pedestrian Detection for UAVs Using Cascade Classifiers with Meanshift. *2017 IEEE 11th International Conference on Semantic Computing (ICSC)*.
- Aguilar, W. G., Luna, M. A., Moya, J. F., Abad, V., Ruiz, H., Parra, H., & Angulo, C. (2017). Pedestrian Detection for UAVs Using Cascade Classifiers and Saliency Maps. *Lecture Notes in Computer Science*, 563–574.
- Aguilar, W. G., Luna, M. A., Moya, J. F., Abad, V., Ruiz, H., Parra, H., & Angulo, C. (2017). Pedestrian Detection for UAVs Using Cascade Classifiers and Saliency Maps. En *Lecture Notes in Computer Science* (págs. 563-574).
- Aguilar, W. G., Luna, M. A., Moya, J. F., Abad, V., Ruiz, H., Parra, H., & Lopez, W. (2017). Cascade Classifiers and Saliency Maps Based People Detection. *Augmented Reality, Virtual Reality, and Computer Graphics*, 501–510.
- Aguilar, W. G., Luna, M. A., Moya, J. F., Luna, M. P., Abad, V., Ruiz, H., & Parra, H. (2017). Real-Time Detection and Simulation of Abnormal Crowd Behavior. En *Lecture Notes in Computer Science* (págs. 420-428).
- Aguilar, W. G., Luna, M. A., Ruiz, H., Moya, J. F., Luna, M. P., Abad, V., & Parra, H. (2017). Statistical Abnormal Crowd Behavior Detection and Simulation for Real-Time Applications. En *Lecture Notes in Computer Science* (págs. 671-682).

- Aguilar, W. G., Luna, M., Moya, J. F., Luna, M. P., Abad, V., Ruiz, H., & Parra, H. (2017). Real-Time Detection and Simulation of Abnormal Crowd Behavior. *Augmented Reality, Virtual Reality, and Computer Graphics*, 420–428.
- Aguilar, W. G., Manosalvas, J. F., Guillén, J. A., & Collaguazo, B. (2018). Robust Motion Estimation Based on Multiple Monocular Camera for Indoor Autonomous Navigation of Micro Aerial Vehicle. *Augmented Reality, Virtual Reality, and Computer Graphics*.
- Aguilar, W. G., Manosalvas, J. F., Guillén, J. A., & Collaguazo, B. (2018). Robust Motion Estimation Based on Multiple Monocular Camera for Indoor Autonomous Navigation of Micro Aerial Vehicle. *International Conference on Augmented Reality, Virtual Reality and Computer Graphics* (págs. 547-561). Springer.
- Aguilar, W. G., Morales, S., Ruiz, H., & Abad, V. (2017). RRT* GL Based Optimal Path Planning for Real-Time Navigation of UAVs. *Lecture Notes in Computer Science*, 585-595.
- Aguilar, W. G., Morales, S., Ruiz, H., & Abad, V. (2017). RRT* GL Based Path Planning for Virtual Aerial Navigation. En *Lecture Notes in Computer Science* (págs. 176-184).
- Aguilar, W. G., Quisaguano, F., Rodríguez, G., Alvarez, L., Limaico, A., & Sandoval, D. (2018). Convolutional Neuronal Networks Based Monocular Object Detection and Depth Perception for Micro UAVs. *Lecture Notes in Computer Science*, 401-410.
- Aguilar, W. G., Salcedo, V. S., Sandoval, D. S., & Cobeña, B. (2017). Developing of a Video-Based Model for UAV Autonomous Navigation. *Computational Neuroscience*, 94–105.

- Aguilar, W. G., Salcedo, V., Sandoval, D., & Cobeña, B. (2017). Developing of a Video-Based Model for UAV Autonomous Navigation. *Communications in Computer and Information Science*, 94-105.
- Aguilar, W. G., Sandoval, S., Limaico, A., Villegas-Pico, M., & Asimbaya, I. (2019). Path Planning Based Navigation Using LIDAR for an Ackerman Unmanned Ground Vehicle. En *Lecture Notes in Computer Science* (págs. 399-410). Germany: Springer.
- Alvarez Samaniego, L. G. (2018). *Desarrollo de un sistema vinculado a un micro-UAV para la detección y evasión de objetos dinámicos a partir de imágenes monoculares*. Sangolquí: Universidad de las Fuerzas Armadas ESPE.
- Amaguaña, F., Collaguazo, B., Tituaña, J., & Aguilar, W. G. (2018). Simulation System Based on Augmented Reality for Optimization of Training Tactics on Military Operations. *International Conference on Augmented Reality, Virtual Reality and Computer Graphics* (págs. 394-403). Springer.
- Andrea, C. C., Byron, J. Q., Jorge, P. I., Inti, T. C., & Aguilar, W. G. (2018). Geolocation and Counting of People with Aerial Thermal Imaging for Rescue Purposes. *International Conference on Augmented Reality, Virtual Reality and Computer Graphics* (págs. 171-182). Springer.
- Bay, H., Tuytelaars, T., & Van Gool, L. (2006). SURF: Speeded Up Robust Features. *Lecture Notes in Computer Science*, 404–417.
- Belkacem, K., & Ghazzawi, Y. (2011). Robust PID Controller Design for an UAV Flight Control System. *Proceedings of the World Congress on Engineering and Computer Science Vol II*. San Francisco.

- Bianco, S., Mazzini, D., Pau, D., & Schettini, R. (2015). Local detectors and compact descriptors for visual search: a quantitative comparison. *Digital Signal Process*, 44, 1-13.
- Cabras, P., Rosell, J., Pérez, A., Aguilar, W. G., & Rosell, A. (2011). Haptic-based navigation for the virtual bronchoscopy. *18th IFAC World Congress*. Milano, Italy.
- Camino Guzmán, S. M., & Rojas Gómez, M. I. (2019). *Sistema aéreo terrestre basado en robótica cooperativa para detección de minas antipersonales*. Sangolquí: Universidad de las Fuerzas Armadas ESPE.
- Campoy, P., Correa, J. F., Mondragón, I., Martínez, C., Olivares, M., Mejías, L., & Artieda, J. (2008). Computer Vision Onboard UAVs for Civilian Tasks. *Journal of Intelligent and Robotics Systems*, 105-135.
- Cervantes, L., & Castillo, O. (2015). Type-2 Fuzzy Logic Aggregation of Multiple Fuzzy Controllers for Airplane Flight Control. *Information Sciences*, 247-256.
- Clough, B. T. (2002). Metrics, Schmetrics! How The Heck Do You Determine A UAV's Autonomy Anyway? *Proceedings of the performance metrics for intelligent systems workshop (PerMIS)*, (págs. 1-7). Maryland.
- Galindo, R., Aguilar, W. G., & Reyes Ch, R. P. (2019). Landmark based eye ratio estimation for driver fatigue detection. En *Lecture Notes in Computer Science* (págs. 565-576). Germany: Springer.
- Gomes Carreira, T. (2013). Quadcopter Automatic Landing on a Docking Station. *Master Thesis, Instituto Superior Tecnico (IST)*.

- Gomez-Ojeda, R., & Gonzalez-Jimenez, J. (2016). Robust Stereo Visual Odometry through a Probabilistic Combination of Points and Line Segments. *2016 IEEE International Conference on Robotics and Automation (ICRA)*.
- Grijalva Caisachana, S. (2019). *Sistema de navegación autónomo basado en visión para pistas de carreras de drones con marcas de superficie*. Sangolquí: Universidad de las Fuerzas Armadas ESPE.
- Grijalva, S., & Aguilar, W. G. (2019). Landmark-Based Virtual Path Estimation for Assisted UAV FPV Tele-Operation with Augmented Reality. En *Lecture Notes in Computer Science* (págs. 688-700). Germany: Springer.
- Hartley, R., & Zisserman, A. (2003). *Multiple View Geometry in computer vision*. United States of America by Cambridge University Press, New York: Cambridge University Press.
- Hassaballah, M., Abdelmgeid, A. A., & Alshazly, H. A. (2016). Image Features Detection, Description and Matching. *Studies in Computational Intelligence*, 11-45.
- He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas: IEEE.
- Henriques, J. F., Caseiro, R., Martins, P., & Batista, J. (2015). High-Speed Tracking with Kernelized Correlation Filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence* , 37(3), 583–596.

- Hui, J. (6 de Marzo de 2018). *Medium*. Obtenido de mAP (mean Average Precision) for Object Detection: https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173
- Ibarra, L. M., Freire, A. F., Minango, S. N., RocioVelasco, N. d., Chang, E. O., & Chipantasi, D. J. (2017). *Applying artificial vision techniques and artificial neural networks to autonomous quadcopter landing*. Recuperado el 13 de 8 de 2020, de <http://xplore.staging.ieee.org/ielx7/8232983/8247444/08247524.pdf?arnumber=8247524>
- Jantzen, J. (2007). *Foundations of Fuzzy Control*. England: Library of Congress Cataloging-in-Publication Data.
- Jara-Olmedo, A., Medina-Pazmiño, W., Mesías, R., Araujo-Villaroel, B., Aguilar, W. G., & Pardo, J. A. (2018). Interface of Optimal Electro-Optical/Infrared for Unmanned Aerial Vehicles. En *Smart Innovation, Systems and Technologies* (págs. 372-380).
- Jara-Olmedo, A., Medina-Pazmiño, W., Tozer, T., Aguilar, W. G., & Pardo, J. A. (2018). E-services from Emergency Communication Network: Aerial Platform Evaluation. *International Conference on eDemocracy & eGovernment (ICEDEG)* (págs. 251-256). IEEE.
- Jiménez Camacho , E. (2009). *Medición de distancias por medio de procesamiento de imágenes y triangulación, haciendo uso de cámaras de video*. Puebla: Escuela de Ingeniería, Universidad de las Américas Puebla.
- Joshi, P. (2015). *OpenCV with Python By Example. Build real-world computer vision applications and develop cool demos using OpenCV for Python*. Birmingham: Packt Publishing Ltd.

- Jung, Y., Lee, D., & Bang, H. (2015). Close-Range Vision Navigation and Guidance for rotary UAV Autonomous Landing. *IEEE International Conference on Automation Science and Engineering (CASE)*, 342-347.
- Kada, B., & Ghazzawi, Y. (2011). Robust PID controller design for an UAV flight control system. *Proceedings of the World Congress on Engineering and Computer Science*, (págs. 1-6).
- Kalal, Z., Mikolajczyk, K., & Matas, J. (2010). Forward-Backward Error: Automatic Detection of Tracking Failures. *2010 20th International Conference on Pattern Recognition*.
- Kayacan, E., & Maslim, R. (2017). Type-2 Fuzzy Logic Trajectory Tracking Control of Quadrotor VTOL Aircraft With Elliptic Membership Functions. *IEEE/ASME Transactions on Mechatronics Vol 22*, 339-348.
- Kendoul, F. (2012). Survey of advances in guidance, navigation, and control of unmanned rotorcraft systems. *Journal of Field Robotica*, 315-378.
- Khamseh Motlagh, H. D., Lotfi, F., & Taghirad, H. H. (2019). Position Estimation for Drones based on Visual SLAM and IMU in GPS-denied Environment. *International Conference on Robotics and Mechatronics (ICRoM)*. Tehran.
- Kim, J. W., Jung, Y., Lee, D., & Shim, D. H. (2014). Outdoor Autonomous Landing on a Moving Platform for Quadrotors using an Omnidirectional Camera . *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*, 1243-1256.
- King, J. R. (2017). *Quadrotor Visual Servoing for Automatic Landing*. University of Toronto Institute for Aerospace Studies.

- Lee, H., Jung, S., & Shim, D. H. (2016). Vision-based UAV landing on the moving vehicle. *2016 International conference on unmanned aircraft systems* (págs. 1-7). Arlington: IEEE.
- Lehtola, V., Huttunen, H., Christophe, F., & Mikkonen, T. (2017). Evaluation of Visual Tracking Algorithms for Embedded Devices. *Lecture Notes in Computer Science*, 88–97.
- Ling, K. (2014). Precision Landing of a Quadrotor UAV on a Moving Target Using Low-Cost Sensors. *Master's Thesis, University of Waterloo*.
- Liu, J. (2018). *Intelligent Control Design and MatLab Simulation*. Beijing: Tsinghua University Press, Beijing and Springer Nature Singapore Pte Ltd.
- Luukkonen, T. (2011). *Modelling and control of quadcopter*. Espoo.
- Madison, R., Andrews, G., DeBitetto, P., Rasmussen, S., & Bottkol, M. (2007). Vision-Aided Navigation for Small UAVs in GPS-Challenged Environments. *AIAA Infotech@Aerospace 2007 Conference and Exhibit*.
- Maimone, M., Cheng, Y., & Matthies, L. (2007). Two Years of Visual Odometry on the Mars Exploration Rovers. *Journal of Field Robotics*, 24(3), 169-186.
- Merz, T., Duranti, S., & Conte, G. (2004). Autonomous Landing of an Unmanned Helicopter based on Vision and Inertial Sensing. *Proceedings of the 9th International Symposium on Experimental Robotics*. Singapore.
- Mian, A. S., Bennamoun, M., & Owens, R. (2007). Keypoint Detection and Local Feature Matching for Textured 3D Face Recognition. *International Journal of Computer Vision*, 79(1), 1–12.

- Monajjemi, M. (2015). *bebop_autonomy*. Obtenido de bebop_autonomy - ROS Driver for Parrot Bebop Drone (quadrocopter) 1.0 & 2.0: <https://bebop-autonomy.readthedocs.io/en/latest/index.html>
- Murphy, R. R., Steimle, E., Griffin, C., Cullins, C., Hall, M., & Pratt, K. (2008). Cooperative Use of Unmanned Sea Surface and Micro Aerial Vehicles at Hurricane Wilma. *Journal of Field Robotics*, 164-180.
- Nourani-Vatani, N., & Koerich Borges, P. V. (2011). Correlation-Based Visual Odometry for Ground Vehicles. *Journal of Field Robotics*, 28(5), 742-768.
- Orbea, D., Moposita, J., Aguilar, W. G., Paredes, M., León, G., & Jara-Olmedo, A. (2017). Math Model of UAV Multi Rotor Prototype with Fixed Wing Aerodynamic Structure for a Flight Simulator. En *Lecture Notes in Computer Science* (págs. 199-211).
- Orbea, D., Moposita, J., Aguilar, W. G., Paredes, M., Reyes, R. P., & Montoya, L. (2017). Vertical take off and landing with fixed rotor. *Chilean Conference on Electrical, Electronics Engineering, Information and Communication Technologies (CHILECON)*. Pucón, Chile.
- Otto, A., Agatz, N., Campbell, J., Golden, B., & Pesch, E. (2018). Optimization approaches for civil applications of unmanned aerial vehicles (UAVs) or aerial drones: A survey. *Networks*, 411-458.
- Pardo, C. (Marzo de 2020). *picuino*. Obtenido de Controlador PID: <https://www.picuino.com/es/arduprog/control-pid.html>
- Pardo, J. A., Aguilar, W. G., & Toulkeridis, T. (2017). Wireless communication system for the transmission of thermal images from a UAV. *Chilean Conference on Electrical,*

Electronics Engineering, Information and Communication Technologies (CHILECON). Pucón, Chile.

Parrot *Bebop 2 Drone*. (2020). Obtenido de User Guide: <https://www.parrot.com/es/drones/parrot-bebop-2>

Prates, P. A., Mendoca, R., Lourenco, A., Marques, F., Matos-Carvalho, J. P., & Barata, J. (2018). Vision-based UAV detection and tracking using motion signatures. *IEEE Industrial Cyber-Physical Systems (ICPS)*, 482-487.

Quan, Q. (2017). *Introduction to Multicopter Design and Control*. Singapore: Springer Singapore.

Quisaguano Paredes, F. J. (2018). *Desarrollo de un sistema simultáneo de seguimiento de persona y evasión de obstáculos no definidos usando estimación de profundidad para imágenes monoculares en un micro-UAV*. Sangolquí: Universidad de las Fuerzas Armadas ESPE.

Redmon, J., & Farhadi, A. (2018). YOLOv3: An incremental improvement. *arXiv preprint arXiv*.

Redmon, J., Divvala Santosh, Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 779-788.

ROS wiki AaronMR. (21 de 06 de 2014). *ROS.org*. Obtenido de ROS Concepts: <http://wiki.ros.org/ROS/Concepts>

Rosten, E., & Drummond, T. (2006). Machine Learning for High-Speed Corner Detection. *9th European Conference on Computer Vision*, (págs. 440-443).

- Rublee, E., Rabaud, V., Konolige, K., & Bradski, G. (2011). ORB: an efficient alternative to SIFT or SURF. *Computer Vision (ICCV)* (págs. 2564-2571). IEEE international conference.
- Salcedo Peña, V. S. (2018). *Aterrizaje automático de un vehículo aéreo no tripulado basado en seguimiento de puntos de interés para superficies móviles*. Sangolquí: Carrera de Ingeniería en Electrónica, Automatización y Control. Universidad de las Fuerzas Armadas ESPE.
- Sarabakha, A., Fu, C., Kayacan, E., & Kumbasar, T. (2017). Type-2 Fuzzy Logic Controllers Made EvenSimpler: From Design to Deployment for UAVs. *IEEE Transactions on Industrial Electronics*, 5069-5077.
- Saripalli, S., & Sukhatme, G. V. (2003). Landing on a Moving Target Using an Autonomous Helicopter. *Proceedings of the International Conference on Field and Service Robotics*.
- Scaramuzza, D., & Fraundorfer, F. (2011). Visual Odometry [Tutorial]. *IEEE Robotics & Automation Magazine*, 18(4), 80–92.
- Simeone, O. (2018). A Very Brief Introduction to Machine Learning With Applications to Communication Systems. *IEEE Transactions on Cognitive Communications and Networking*.
- Strydom, R., Thurrowgood, S., & Srinivasan, M. V. (2014). Visual Odometry: Autonomous UAV Navigation using Optic Flow and Stereo. *Proceedings of Australasian Conference on Robotics and Automation*. Melbourne.
- Szeliski, R. (2010). *Computer Vision: Algorithms and Applications*, 1st ed. Springer.

- Turkoglu, K., & Jafarov, E. M. (2006). H inf. Loop Shaping Robust Control vs. Classical PI(D) Control: A case study on the Longitudinal Dynamics of Hezarfen UAV. *WSEAS International Conference on Dynamical Systems and Control*, (págs. 105-110). Bucharest.
- Turkoglu, K., Ozdemir, U., Nikbay, M., & Jafarov, E. M. (2008). PID Parameter Optimization of an UAV Longitudinal Flight Control System. *World Academy of Science, Engineering and Technology Vol II*, 35-40.
- Valero Lavid, C. (2017). *Desarrollo de aplicaciones basadas en visión con entornos ROS para el Drone Bebop 2*. Universidad de Alcalá Escuela Politécnica Superior.
- Vázquez Ruano, A. (2017). *Control system for autonomous landing on moving*. Universidad Carlos III de Madrid.
- Viala, C. R. (2006). *Caracterización y optimización del proceso de calibrado de cámaras basado en plantilla bidimensional (Tesis Doctoral)*. Universidad Politécnica de Valencia.
- Yang, H., Lee, Y., Jeon, S. Y., & Lee, D. (2017). Multi-rotor drone tutorial: systems, mechanics, control and state estimation. *Intelligent Service Robotics*, 10(2), 79-93.
- Zadeh, L. A. (1996). *Fuzzy Sets, Fuzzy Logic, Fuzzy Systems*. World Scientific Press.
- Zhu, Y., Shen, X., & Chen, H. (2015). Copy-move forgery detection based on scaled ORB. *Multimedia Tools and Applications*, 75(6), 3221–3233.