



UNIVERSIDAD DE GRANADA

TRABAJO FIN DE GRADO
INGENIERÍA EN TECNOLOGIAS DE TELECOMUNICACIÓN

Sistema de control de drones para vigilancia

Desarrollo de algoritmos de control y aplicaciones de
técnicas de visión artificial para la navegación autónoma de
UAV en interiores

Autor

Juan Manuel Tejada Triviño

Directores

Pablo Rodríguez Martín
Juan José Ramos Muñoz



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE
TELECOMUNICACIÓN

Granada, 22 de agosto de 2024



Sistema de control de drones para vigilancia

Desarrollo de algoritmos de control y aplicaciones de
técnicas de visión artificial para la navegación autónoma de
UAV en interiores

Autor

Juan Manuel Tejada Triviño

Directores

Pablo Rodríguez Martín
Juan José Ramos Muñoz

Sistema de control de drones para vigilancia: Desarrollo de algoritmos de control y aplicaciones de técnicas de visión artificial para la navegación autónoma de UAV en interiores

Juan Manuel Tejada Triviño

Palabras clave: UAV, Dron, Simulación, Control autónomo, YOLOv, Navegación, Videovigilancia, Visión por computador, Mapeo, ARUCO, Reconocimiento de objetos, Inteligencia artificial, Tello, Low-Level Protocol, Planificación de rutas, Navegación autónoma, Implementación, Simulador, Optimización, Detección de objetos

Resumen

Este proyecto se centra en el desarrollo y evaluación de tecnologías para la navegación autónoma de UAVs (Vehículos Aéreos no Tripulados o Unmanned Aerial Vehicle) en entornos interiores, con un enfoque particular en la videovigilancia activa y la integración de sistemas de visión por computador. Se analizan los fundamentos teóricos de la videovigilancia, comparando los enfoques pasivos y activos, y se exploran tecnologías clave como la óptica, los marcadores ARUCO y el algoritmo de detección YOLO (You Only Look Once). Además, se abordan los parámetros esenciales para el control autónomo en interiores, así como los diferentes tipos de UAVs, junto con las tecnologías de control autónomo y los algoritmos de navegación y evasión de obstáculos.

El estudio incluye una revisión de las herramientas de simulación de vuelo como Webots, Gazebo y Microsoft Flight Simulator, evaluando sus capacidades y limitaciones en el contexto de la simulación de operaciones en interiores. Asimismo, se examina el marco normativo actual y se proponen mejoras para la seguridad y el cumplimiento regulatorio en el uso de UAVs en espacios confinados.

Finalmente, se propone un sistema de navegación autónoma basado en la integración de ARUCO y YOLOv5, implementado y probado tanto en simuladores como en aplicaciones reales. Se analiza cómo esta integración mejora la precisión en la orientación y posicionamiento del dron, y se evalúa la efectividad de las simulaciones en la preparación para vuelos reales. El trabajo concluye con una discusión sobre la integración de técnicas avanzadas de SLAM y edge computing para optimizar la operación de UAVs en entornos complejos para un desarrollo futuro.

Drone control system for surveillance: Development of control algorithms and applications of artificial vision techniques for autonomous navigation of UAVs indoors

Juan Manuel Tejada Triviño

Keywords: UAV, Drone, Simulation, Autonomous Control, YOLOv, Navigation, Computer Vision, Mapping, ARUCO, Object Recognition, Artificial Intelligence, Tello, Low-Level Protocol, Autonomous Navigation, Implementation, Simulators, Optimization, Object Detection

Abstract

This project focuses on the development and evaluation of advanced technologies for autonomous navigation of UAVs (Unmanned Aerial Vehicles) in indoor environments, with a particular emphasis on active video surveillance and the integration of computer vision systems. The theoretical foundations of video surveillance are analyzed, comparing passive and active approaches, and key technologies such as optics, ARUCO markers, and the YOLO (You Only Look Once) detection algorithm are explored. Additionally, essential parameters for autonomous indoor control are addressed, along with different types of UAVs, autonomous control technologies, and obstacle navigation and avoidance algorithms.

The study includes a comprehensive review of flight simulation tools such as Webots, Gazebo, and Microsoft Flight Simulator, evaluating their capabilities and limitations in the context of simulating indoor operations. Furthermore, the current regulatory framework is examined, and improvements are proposed for safety and regulatory compliance in the use of UAVs in confined spaces.

Finally, an autonomous navigation system based on the integration of ARUCO and YOLOv5 is proposed, implemented, and tested both in simulators and real-world applications. The analysis includes how this integration enhances the drone's accuracy in orientation and positioning, and the effectiveness of simulations in preparing for real flights is evaluated. The work concludes with a discussion on the integration of advanced SLAM techniques and edge computing to optimize UAV operations in complex environments.

Yo, **Juan Manuel Tejada Triviño**, alumno de la titulación Grado en Tecnologías de la Telecomunicación de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 77770478A, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Juan Manuel Tejada Triviño

Granada a 22 de agosto de 2024 .

D. Juan José Ramos Muñoz, Profesor del Departamento Teoría de la Señal, Telemática y Comunicaciones de la Universidad de Granada.

D. Pablo Rodríguez Martín , Personal Investigador del Departamento Teoría de la Señal, Telemática y Comunicaciones de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado *Sistema de control de drones para vigilancia: Desarrollo de algoritmos de control y aplicaciones de técnicas de visión artificial para la navegación autónoma de UAV en interiores*, ha sido realizado bajo su supervisión por **Juan Manuel Tejada Triviño**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 22 de agosto de 2024 .

Los directores:

Juan José Ramos Muñoz

Pablo Rodríguez Martín

Agradecimientos

Expresar mi más sincero agradecimiento a mi familia y amigos. A ustedes, que han sido mi refugio y mi fortaleza en los momentos más difíciles, gracias por su amor incondicional, por su apoyo constante y por creer en mí incluso cuando yo dudaba. Su compañía y palabras de aliento han sido un pilar fundamental en este proceso, y sin ustedes, este logro no habría sido posible.

Quiero también extender mi más profundo agradecimiento, a los directores (Juan José Ramos Muñoz y Pablo Rodríguez Martín) por su valiosa orientación y apoyo durante la realización de mi Trabajo de Fin de Grado. Su dedicación y conocimiento han sido fundamentales para mi crecimiento académico y profesional.

Agradezco su paciencia y comprensión durante los momentos de dificultad, así como su entusiasmo y motivación constante. Su experiencia y sabiduría han sido una fuente de inspiración y me han permitido alcanzar metas que, al inicio de este camino, parecían inalcanzables.

Gracias por su paciencia, sabiduría y constante motivación. Este logro no habría sido posible sin su inestimable ayuda.

Índice general

1. Introducción	1
1.1. Motivación y objetivos del proyecto	2
1.2. Estructura de la memoria	3
2. Planificación	5
2.1. Metodología del proyecto	5
2.2. Presupuesto del Proyecto	8
3. Estado del arte	9
3.1. Videovigilancia pasiva vs activa	9
3.2. Fundamentos teóricos de vuelo y navegación con UAVs	11
3.3. Tipos de UAVs	12
3.4. Videovigilancia, Robots, UAVs	15
3.4.1. Modelos de UAVs para Interiores	20
3.4.2. Tello EDU	20
3.4.3. Spark	21
3.4.4. Crazyflie	22
3.4.5. Parrot Bebop 2	22
3.4.6. Comparativa de UAVs para interiores	23
3.4.7. Bibliotecas de programación para el dron Tello	23
3.5. Reconocimiento y Detección de Patrones	24
3.5.1. Sistemas de Marcadores	24
3.5.2. Comparativa de sistemas de marcadores	26
3.5.3. Detección y localización de Objetos en Tiempo Real	33
3.6. Seguridad y regulaciones para el vuelo de UAVs en interiores	42
3.6.1. Marco Normativo Actual y Desafíos de Seguridad	42
3.6.2. Propuestas para Mejorar la Seguridad y Cumplimiento Normativo	42
3.7. Navegación autónoma y mapeo en interiores	43
3.7.1. Desafíos actuales en la navegación autónoma de UAVs en interiores	43
3.7.2. Tecnologías de comunicación para operaciones en interiores	44

3.8.	Simulación de Vuelo	44
3.8.1.	Webots	45
3.8.2.	Gazebo	45
3.8.3.	Microsoft Flight Simulator	45
3.8.4.	Comparativa de los simuladores	46
4.	Herramientas Hardware y Software	47
4.1.	Hardware	47
4.1.1.	Drone Tello	47
4.2.	Software	48
4.2.1.	Protocolo de comunicación con el dron	48
4.2.2.	Descripción y funcionalidades de la biblioteca Telopy	52
4.2.3.	ARUCO	55
4.2.4.	YOLOv5	55
4.2.5.	Webots	56
5.	Desarrollo del sistema	59
5.1.	Descripción general del sistema	59
5.1.1.	Soluciones propuestas para la navegación y mapeo avanzado	59
5.1.2.	Retos a resolver en base a las herramientas	61
5.2.	Preparación y Configuración previa a la implementación	62
5.2.1.	Configuraciones Previas	62
5.2.2.	Mediciones de Error y Análisis	62
5.3.	Implementación de ARUCO para navegación autónoma	64
5.3.1.	Configuración de ARUCO	65
5.4.	Implementación en el simulador	66
5.4.1.	Creación de entornos y robots simulados que replican condiciones reales	67
5.4.2.	Implementación de API de Tello en webots	69
5.5.	Configuración de Yolov5	72
5.5.1.	Integración de Aruco con Yolov5	72
5.6.	Configuración del sistema para navegación autónoma indoor	74
5.6.1.	Parametrización de las Etiquetas ARUCO	74
5.6.2.	Interfaz Gráfica	75
5.6.3.	Control de Navegación	78
5.6.4.	Flujo del Control Principal	81
5.6.5.	Sistemas de alertas basado en Telegram	86
6.	Evaluación	91
6.1.	Experimentos de calibración y mediciones de error en ARUCO	91
6.1.1.	Ánálsis de cómo ARUCO mejora la orientación y posicionamiento del dron	96
6.2.	Experimentos en el Simulador	96

6.2.1. Implementación en el Simulador	97
6.2.2. Implementación en el dron real	101
6.3. Evaluación del sistema de alertas	103
Conclusiones y Líneas Futuras	105
6.3.1. Diferencias y Consideraciones entre la Implementación en Vida Real vs. Simulador	106
6.3.2. Evaluación de la Efectividad de la Simulación en la Preparación para Vuelos Reales	107
6.4. Lineas Futuras	107
6.4.1. Módulo ESP32	107
6.5. Integración de SLAM con Tello	108
6.6. Integración de SLAM con ARUCO en la Navegación Autónoma	109
6.6.1. Análisis de la mejora en la navegación y mapeo a través de esta integración	110
6.7. Edge Computing + ESP32	111
6.8. Gestión de flotas de UAVs en entornos confinados	111
6.8.1. Solución propuesta para la optimización del flujo de trabajo de la flota	112
Anexo	114

Índice de figuras

2.1. Diagrama de Gantt	6
3.1. Esquema de Orientación en UAVs [1].	12
3.2. UAV Tipo C0.	13
3.3. UAV Tipo C1.	13
3.4. UAV Tipo C2.	14
3.5. UAV Tipo C3 - C7.	14
3.6. Plataforma de integración de sensores en UAV Lattice [2].	16
3.7. UAV de Intercepción - Anvil [2].	17
3.8. Radar WISP [2].	17
3.9. UAV de Seguimiento RoadRunner [2].	18
3.10. Perro Robot para Vigilancia [3].	18
3.11. TelloEDU.	21
3.12. Spark.	21
3.13. Crazyflie	22
3.14. Parrot Bebop 2.	23
3.15. Esquema de funcionamiento de Sistemas de Marcadores.	25
3.16. Parámetros de extracción de bits Aruco.	32
3.17. Comparativa versiones de Yolo [4].	37
3.18. Comparativa diferentes versiones YOLOv5.	39
3.19. Comparativa de rendimiento YOLOv5 contra sus competidores.	41
3.20. Comparativa de velocidad YOLOv5 contra sus competidores.	41
4.1. TelloEDU User Manual.	48
4.2. Yolov5 Checkout	56
4.3. Webots Transfer	57
5.1. Descripción del Sistema	60
5.2. Proceso de Calibración de Cámara	63
5.3. Error de Reproyección	64
5.4. Intrucciones para seleccionar el entorno	67
5.5. Mapa Simulado	68
5.6. Intrucciones para importar el robot	68
5.7. Robot en Webots	69

5.8. Instrucciones para importar el script de control	69
5.9. integración de detección de personas con YOLO y detección de etiquetas con ARUCO	74
5.10. Control de flujo GUI	77
5.11. Control de flujo para el seguimiento de etiquetas	79
5.12. Control del estado de Mapeo	83
5.13. Control de flujo para verificar estado de Mapeo	84
5.14. Crear bot de telegram	87
5.15. Control de flujo para el envio de alertas	89
6.1. Herramienta para de medición de errores para los ángulos . .	92
6.2. Errores en Ángulos Detectados por ARUCO	93
6.3. Distribución de Errores Absolutos por Ángulo Real	93
6.4. Errores Promedios por Ángulo Real con Tamaño de Círculo según Error	94
6.5. Ángulos Reales Absolutos frente al Error	94
6.6. Errores en Distancias Detectadas por ARUCO	95
6.7. Distribución de Errores Absolutos por Distancia Real	95
6.8. Mapa y recorrido en el simulador	96
6.9. Estado Mapping	98
6.10. Finalizado Mapping	98
6.11. Estado de Seguimiento de Puerta (Etiqueta 5)	99
6.12. Estado de Seguimiento de Puerta (Stop Puerta Cerrada) . .	99
6.13. Estado de Pasado puerta (pass)	100
6.14. Vuelta a Estado de Mapping	100
6.15. Final del recorrido	100
6.16. Tiempo en completar el recorrido.	101
6.17. Implemetación de Estado Mapeo en condiciones reales . .	102
6.18. Estado Mapeo en condiciones reales con Yolo y ARUCO . .	102
6.19. Implemetación de Estado Mapeo en condiciones reales . .	103
6.20. Evaluación del sistema de detección	104
6.21. ESP32.	107
6.22. ESP32 Pinout.	108
6.23. SLAM implementado en Tello EDU	109
6.24. SLAM fusionado con ARUCO	110
6.25. SLAM fusionado con ArUco en UAV	110
6.26. ESP32 onboard TELLO	111
6.27. PINOUT Batería de TELLO	111
6.28. Diagrama de EdgeComputing con Drones	112
6.29. Diagrama para el control de colmenas mediante AP	113

Índice de tablas

2.1. Presupuesto general del proyecto de drones	8
3.1. Bibliotecas de Programación para Tello	24
3.2. Propiedades de los diccionarios ARUCO	27
3.3. Parámetros de la clase Dictionary	30
3.4. Parámetros de umbralización	30
3.5. Parámetros de filtrado de contornos	31
3.6. Parámetros de extracción de bits	31
3.7. Parámetros de identificación de marcadores	32
3.8. Parámetros de refinamiento de esquinas	32
3.9. Comparativa de versiones YOLO[5]	38
3.10. Comparativa de especificaciones técnicas de las versiones de YOLOv5	39
3.11. Training parameters for YOLOv5 and Faster R-CNN	40
4.1. Tello General UDP Packet Structure	49
4.2. Variables de la clase LogNewMvoFeedback	54
4.3. Variables de la clase LogImuAtti	54
5.1. Parametrización de las Etiquetas ARUCO	75
6.1. Resumen Estadístico de los Errores Angulares	93
6.2. Resumen Estadístico de los Errores de Distancia	94
6.3. Datasheet Drone DJI Tello EDU.	114
6.4. Tello Message IDs and Meanings (Part 1)	115
6.5. Tello Message IDs and Meanings (Part 2)	116
6.6. Funciones y Descripciones de Tellopy	117
6.7. Funciones y Descripciones de Tellopy	118
6.8. Eventos y Estados de Tellopy	119
6.9. Variables de la clase FlightData	120

Capítulo 1

Introducción

Este proyecto se centra en explorar las aplicaciones innovadoras de los drones en el campo de la videovigilancia y la navegación autónoma en interiores, subrayando la sinergia entre tecnologías emergentes y las crecientes demandas de seguridad y monitoreo en entornos. A medida que avanzan los sistemas robóticos y las tecnologías de visión por computadora, los drones están empezando a desempeñar un papel cada vez más importante en varias industrias, desde la seguridad hasta la logística, proporcionando nuevas perspectivas y capacidades que antes eran inimaginables. Sin embargo, la adaptación de los drones para su uso en entornos interiores lleva a desafíos particulares, como la necesidad de una navegación controlada, a través de un mapeo —el proceso de crear un modelo o representación simple del entorno interior para permitir una navegación autónoma y segura—.

En este contexto, el presente proyecto se propone superar las limitaciones de las técnicas tradicionales de vigilancia como las cámaras de seguridad, a través de la implementación de drones equipados con tecnologías avanzadas de visión por computadora y algoritmos de navegación autónoma. Estos sistemas permitirán realizar tareas de videovigilancia de manera más eficiente y efectiva, ofreciendo soluciones adaptativas que pueden interactuar y reaccionar en tiempo real a las condiciones del entorno.

El sistema desarrollado en este proyecto se compone de dos partes fundamentales. La primera es la navegación y mapeo, donde el dron, a través de un mapeo simple y la implementación de algoritmos de navegación diseñados específicamente para este entorno, podrá moverse de manera autónoma dentro de espacios interiores. La segunda parte se enfoca en la investigación y implementación de algoritmos de visión por computadora, no solo para mejorar la navegación y el mapeo, sino también para el reconocimiento de objetos. Se explorarán diferentes algoritmos, evaluando sus capacidades y adaptabilidad, para determinar cuál se ajusta mejor a las necesidades específicas de nuestro proyecto.

La investigación y el desarrollo en este proyecto se fundamentan en una comprensión de los últimos avances en robótica, navegación y procesamiento de imágenes, junto con un estudio de las necesidades operativas específicas para drones en interiores. El proyecto aspira a desarrollar un marco de trabajo que aborde desde el diseño técnico hasta la implementación práctica y la evaluación de rendimiento en escenarios de vida real.

Este documento detalla la planificación estratégica y la metodología propuesta para el desarrollo del proyecto, incluyendo fases de diseño, simulación, implementación y pruebas. También abarca la estructura presupuestaria necesaria para garantizar la adquisición de recursos tecnológicos y humanos adecuados, y una revisión de la normativa vigente para asegurar que todas las operaciones se realicen dentro de un marco legal y seguro. La memoria del proyecto está diseñada para ofrecer una narrativa coherente y detallada de cada etapa del proyecto, proporcionando análisis técnicos, resultados de pruebas y recomendaciones para futuras investigaciones y aplicaciones.

1.1. Motivación y objetivos del proyecto

La principal motivación detrás de este proyecto surge de la necesidad de abordar las limitaciones inherentes a los sistemas de videovigilancia tradicionales como las cámaras de seguridad, las cuales a menudo se ven restringidas por su incapacidad para adaptarse y reaccionar de manera dinámica a los cambios ambientales y a las situaciones en tiempo real. Frente a este desafío, la implementación de drones representa una oportunidad transformadora para revolucionar estos sistemas. Equipados con tecnologías avanzadas de visión por computador y algoritmos de navegación, estos drones están diseñados para operar eficazmente dentro de espacios cerrados, realizando tareas de vigilancia con una mayor operatividad. Esta capacidad para operar de forma autónoma no solo mejora la calidad y la cobertura de la vigilancia, sino que también contribuye significativamente a la seguridad y gestión operativa de las instalaciones.

La videovigilancia, utilizando drones equipados con tecnología avanzada, ofrece la capacidad de detectar intrusiones y responder rápidamente antes de que se materialicen amenazas. Esta tecnología también se ha implementado en diversos sectores, como en empresas como IKEA y Amazon, que han comenzado a utilizar drones en sus almacenes para gestionar inventarios, demostrando cómo puede ser aplicada tanto en entornos de seguridad como en la optimización de procesos logísticos[6]. La capacidad de reaccionar en tiempo real es una de las mayores ventajas de la videovigilancia activa con drones, reduciendo significativamente el riesgo de infiltraciones y mejorando la seguridad en tiempo real.

Para lograr una integración efectiva de los drones en los sistemas de seguridad y vigilancia interiores, este proyecto se ha fijado varios objetivos claros y medibles, estructurados para abordar tanto los desafíos técnicos como operativos:

1. **Desarrollo de un sistema de control y navegación en interiores para drones:** Este objetivo busca crear un marco de navegación que permita a los drones orientarse y maniobrar de manera independiente, utilizando tecnologías de mapeo en tiempo real que son críticas para la operación autónoma.
2. **Implementación de una solución eficiente de videovigilancia mediante drones:** El proyecto tiene como objetivo desarrollar un sistema de drones que no solo capture imágenes, sino que también sea capaz de interpretar y responder a eventos en tiempo real, adaptándose a las dinámicas del entorno para mejorar la vigilancia y la respuesta a incidentes.

1.2. Estructura de la memoria

Esta memoria está estructurada para proporcionar una visión completa y detallada del proyecto de desarrollo e implementación de un sistema de drones para navegación y videovigilancia autónoma en interiores. La estructura está diseñada para guiar al lector a través de las diferentes etapas del proyecto, desde la concepción inicial hasta los resultados finales y su análisis. A continuación se detallan los capítulos y las principales secciones que componen esta memoria:

- **Capítulo 1: Introducción** - Este capítulo establece el contexto del proyecto, incluyendo la motivación detrás del desarrollo del sistema de drones, los objetivos específicos que el proyecto pretende alcanzar, y una descripción de los tipos de UAVs utilizados. Además, se aborda la planificación y metodología del proyecto, ofreciendo un esbozo del enfoque adoptado y los recursos estimados necesarios para su realización.
- **Capítulo 2: Planificación y Metodología** - En este capítulo se presenta la planificación del proyecto, incluyendo el cronograma y la distribución de tareas. Además, se describe la metodología adoptada para el desarrollo del sistema y los recursos estimados necesarios para su implementación.
- **Capítulo 3: Estado del arte** - Se revisan los fundamentos teóricos relevantes para el proyecto, incluyendo tecnologías actuales de video-

vigilancia, robots y UAVs. Este capítulo profundiza en los sistemas de visión por computador, destacando tecnologías clave, además de explorar los modelos de UAVs diseñados específicamente para operar en interiores.

- **Capítulo 4: Herramientas Hardware y Software** - Este capítulo cataloga y describe en detalle el hardware y software utilizado en el proyecto, desde los componentes físicos, hasta los sistemas de software que facilitan el control de rutas, mapeo y reconocimiento por inteligencia artificial.
- **Capítulo 5: Desarrollo del sistema** - Expone el diseño operativo del sistema de drones, incluyendo la implementación de técnicas de mapeo y navegación, así como las pruebas realizadas para validar la eficacia del sistema. Este capítulo también cubre la integración y evaluación de los componentes del sistema en simulaciones y aplicaciones del mundo real.
- **Capítulo 6: Evaluación del Sistema** - En este capítulo se llevará a cabo una evaluación exhaustiva del sistema de drones desarrollado. Se analizarán los resultados de las pruebas realizadas en el capítulo anterior, evaluando la precisión, eficiencia y robustez del sistema en diferentes escenarios. Se compararán estos resultados con los objetivos planteados al inicio del proyecto y se identificarán posibles áreas de mejora.
- **Capítulo 7: Conclusiones y trabajo futuro** - Recapitula los logros del proyecto, discute la efectividad del sistema desarrollado y esboza las áreas para futuras investigaciones y mejoras del sistema de drones.

Cada capítulo está diseñado para proporcionar una comprensión integral de los aspectos del proyecto, asegurando que todos los detalles técnicos y teóricos sean accesibles y claros para los lectores interesados en la tecnología de drones y sus aplicaciones prácticas.

Capítulo 2

Planificación

2.1. Metodología del proyecto

El presente documento se divide en seis capítulos, los cuales se detallan a continuación:

1. **Planificación y Metodología** La organización del proyecto, que abarca tanto el calendario como la asignación de responsabilidades. También se detalla la metodología seleccionada para llevar a cabo el desarrollo del sistema y los recursos previstos para su ejecución.
2. **Estado del Arte:** Esta fase implica una profunda investigación de las últimas tecnologías en drones, con un enfoque específico en sistemas de navegación autónoma, algoritmos de vuelo inteligente y tecnologías de videovigilancia. Se estudiarán sistemas para comunicaciones en tiempo real, redes móviles 5G para transmisión de datos de alta velocidad, y el uso de inteligencia artificial para optimizar la operación autónoma de drones.
3. **Selección de herramientas Hardware y Software:** Se seleccionarán las herramientas Hardware y Software que mejor se adapten al proyecto, de tal manera que se puedan completar las tareas de navegación y mapeo
4. **Desarrollo del Sistema:** En esta fase el sistema será desarrollado y probado para asegurar su funcionamiento estable y seguro en diferentes escenarios de uso, como la navegación en espacios cerrados y la detección de objetos en movimiento. Se realizarán pruebas experimentales cuyos resultados serán analizados para evaluar la precisión y la eficiencia del sistema. A partir de estos análisis, se diseñarán los algoritmos

que ajusten dinámicamente los parámetros de vuelo y videovigilancia según las condiciones operativas en tiempo real.

5. **Evaluación:** En esta sección se evaluarán todos los sistemas desarrollados en las fases anteriores. La evaluación incluirá una revisión exhaustiva de la funcionalidad, eficiencia y precisión de los sistemas de navegación autónoma, comunicación en tiempo real, y videovigilancia.
6. **Líneas Futuras:** En esta fase se analizarán las posibilidades futuras del proyecto, identificando áreas de mejora y expansión. Se documentarán las recomendaciones para investigaciones y desarrollos adicionales que puedan optimizar o ampliar el sistema, considerando nuevas tecnologías emergentes y potenciales aplicaciones.

Si se distribuyen estas fases de una forma más concisa a lo largo del tiempo se podría modelar la elaboración del proyecto mediante el siguiente diagrama de Gantt (Fig 2.1):

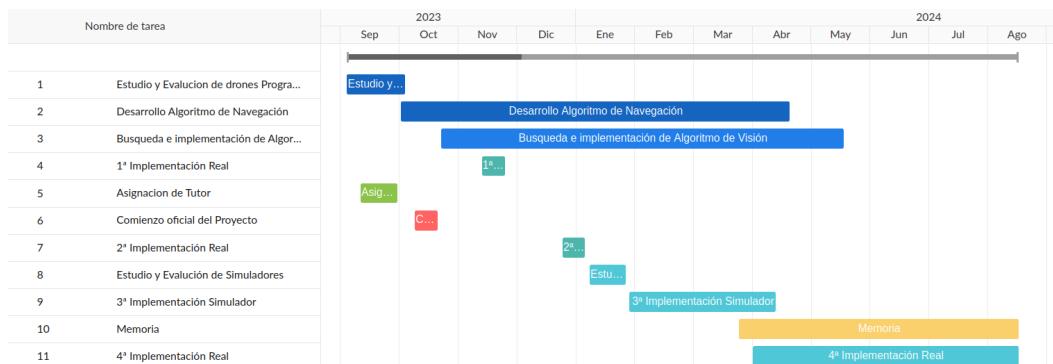


Figura 2.1: Diagrama de Gantt

1. **Estudio y Evaluación de drones Programables (1 SEP - 2 OCT):** Durante este periodo, se realizará una búsqueda de diferentes tipos de drones con el objetivo de encontrar el que mejor se ajustará a las necesidades del proyecto. Se considerarán varios modelos y características específicas para asegurarse de que el dron seleccionado cumpla con los requisitos técnicos y funcionales necesarios.
2. **Desarrollo de Algoritmo de Navegación (26 SEP - 16 ABR):** En esta fase, se llevará a cabo una fase de desarrollo para crear el algoritmo de mapeo más adecuado para el proyecto. Se desarrollarán diversos algoritmos, evaluando su rendimiento y compatibilidad con el dron seleccionado y los objetivos del proyecto.

3. **Búsqueda e implementación de Algoritmo de Visión (17 OCT - 22 MAY)**: Este periodo estará dedicado a la búsqueda del algoritmo de visión óptimo para el proyecto. Se investigarán varias opciones, considerando aspectos como la precisión, velocidad y capacidad de integración con el sistema del dron.
4. **1^a Implementación (9 NOV - 15 NOV)**: Durante esta etapa, se realizará la primera prueba de implementación de los algoritmos en el dron real. Esta fase incluirá la integración y prueba del algoritmo seleccionado, evaluando su desempeño en condiciones reales de operación y realizando ajustes necesarios para mejorar su funcionamiento.
5. **Asignación de Tutor y Comienzo oficial del proyecto (4 SEP - 20 OCT)**: Durante este periodo, se dedicará tiempo a realizar los trámites necesarios para inscribir el proyecto, además de la búsqueda de un tutor para Trabajo de Fin de Grado (TFG).
6. **2^a Implementación (30 DIC - 5 ENE)**: Se llevará a cabo una segunda implementación de los Algoritmos en el dron real. Esta fase incluirá la integración del nuevo algoritmo y la evaluación de su rendimiento en el dron en condiciones reales de operación.
7. **Estudio y Evaluación de Simuladores (6 ENE - 23 ENE)**: Debido al tiempo que tomarán las pruebas en el dron real, se procederá a investigar y probar diferentes simuladores, encontrando el que mejor se ajustará a las necesidades del proyecto.
8. **3^a Implementación en Simulador (23 ENE - 5 ABR)**: Se realizará la implementación y ajustes del algoritmo y las funcionalidades del proyecto en el simulador seleccionado. Esta etapa incluirá pruebas exhaustivas para asegurar que todo funcionará correctamente en el entorno simulado.
9. **4^a Implementación Real (18 ABR - 21 JUL)**: Despues de que el proyecto funcionara adecuadamente en el simulador, se procedió a la implementación final en el dron real. Esta fase involucró la integración y prueba del sistema completo en el dron, asegurando que todo funcionara correctamente en condiciones reales.
10. **Memoria (26 MAR - 16 AGO)**: Durante el tiempo que se realizarán las implementaciones, se trabajará simultáneamente en la redacción de la memoria del TFG. Esta documentación incluirá todos los aspectos del proyecto, desde la investigación inicial hasta las pruebas finales.

2.2. Presupuesto del Proyecto

Para el desarrollo del proyecto de drones, se ha elaborado el siguiente presupuesto detallado que incluye los costos de los materiales, el tiempo empleado en cada fase y los costos operacionales asociados:

Detalles	Cantidad	Precio por unidad
Materiales		
Drone y componentes	1x	200,00€
Software de control y simulación	Licencias	0,00€
Vida útil Ordenador	1 año	120,00 €
Sub-total		320,00€
Tiempo empleado		
Investigación y desarrollo	50h	12€/h
Preparación y pruebas	100h	12€/h
Desarrollo de software y hardware	100h	12€/h
Redacción de la memoria	50h	12€/h
Tiempo total empleado	300 horas	3.600,00€
Operación		
Configuración y despliegue inicial	1x	300,00€
Mantenimiento y actualizaciones	1 año	200,00€
Sub-total		500,00€
Coste total del proyecto (sin I.V.A.)		4.420,00€
Coste total del proyecto (21 % I.V.A.)		5.407,00€

Tabla 2.1: Presupuesto general del proyecto de drones

Capítulo 3

Estado del arte

En esta sección se investiga el estado actual de las tecnologías relacionadas con la navegación autónoma y la videovigilancia mediante drones. Se explora lo que está disponible actualmente y cómo se están resolviendo problemas similares. Inicialmente, se discute el problema de la localización, examinando los diferentes métodos que se utilizan para el seguimiento mediante sistemas de cámaras monocular. A continuación, se revisan las distintas metodologías para obtener información sobre la escala y la profundidad en estos sistemas. Posteriormente, se presentan las metodologías empleadas para la construcción y almacenamiento de mapas de obstáculos. Finalmente, se revisan trabajos que utilizan diferentes metodologías para combinar la generación de mapas y de obstáculos, buscando optimizar tanto la precisión en la localización como la eficacia en la detección de obstáculos para mejorar la navegación de los drones en entornos complejos.

3.1. Videovigilancia pasiva vs activa

La videovigilancia, como componente crucial de los sistemas modernos de seguridad, puede clasificarse en dos categorías principales: pasiva y activa. Cada tipo tiene su enfoque y metodología específicos, ofreciendo distintas ventajas y enfrentándose a diversos desafíos en el ámbito de la seguridad[7][8][9].

3.1.0.1. Videovigilancia Pasiva

La videovigilancia pasiva se refiere a los sistemas que registran y monitorean actividades sin intervención activa en los eventos en tiempo real. Estos sistemas están compuestos principalmente por cámaras estáticas que graban continuamente o por detección de movimiento, almacenando la información

para revisiones futuras. Son ideales para áreas que requieren supervisión constante pero donde la probabilidad de incidentes inmediatos es baja. La principal limitación de la videovigilancia pasiva es su naturaleza reactiva; el sistema no puede intervenir durante un incidente, sino que solo proporciona evidencia visual posterior, como es el caso de:

- **Tiendas Minoristas:** En muchas tiendas, se utilizan cámaras de seguridad que graban continuamente o cuando se detecta movimiento. Estas cámaras no están supervisadas en tiempo real, pero las grabaciones se revisan después en caso de robos o incidentes. Es un enfoque económico que permite documentar eventos para su análisis posterior.
- **Estacionamientos Públicos:** Los estacionamientos suelen utilizar sistemas de videovigilancia pasiva para monitorear la entrada y salida de vehículos. Las cámaras graban las placas de los autos y el entorno, pero no intervienen directamente en tiempo real. En caso de un robo de vehículo, las grabaciones se revisan para identificar a los responsables.

3.1.0.2. Videovigilancia Activa

En contraste, la videovigilancia activa incluye la participación dinámica en la monitorización y la respuesta inmediata a los incidentes a medida que ocurren. Este enfoque utiliza drones, robots, y en algunos casos, intervenciones humanas para interactuar directamente con el entorno vigilado. Por ejemplo, un dron equipado con cámaras puede no solo grabar, sino también seguir a un intruso en tiempo real, proporcionando información constante a las fuerzas de seguridad para una respuesta rápida y efectiva.

La inclusión de tecnologías avanzadas como la inteligencia artificial y el aprendizaje automático ha ampliado aún más las capacidades de la videovigilancia activa, permitiendo sistemas que no solo detectan anomalías sino que también predicen patrones de comportamiento potencialmente peligrosos antes de que se manifiesten en amenazas, como es el caso de:

- **Monitoreo Urbano en Ciudades Inteligentes:** Algunas ciudades, como Singapur, han implementado sistemas de videovigilancia activa donde las cámaras están conectadas a centros de control que supervisan el tráfico y las actividades en tiempo real. Estas cámaras pueden alertar automáticamente a las autoridades sobre comportamientos sospechosos o accidentes, permitiendo una respuesta inmediata.
- **Seguridad en Infraestructuras Críticas:** En lugares como plantas nucleares o instalaciones militares, se utilizan drones y robots equipados con cámaras para monitorear el área. Estos sistemas pueden

seguir a intrusos y activar protocolos de seguridad, enviando alertas al personal para una intervención rápida.

3.1.0.3. Comparación y Aplicaciones

Mientras que la videovigilancia pasiva sigue siendo valiosa por su simplicidad y coste relativamente bajo, la videovigilancia activa está ganando terreno en aplicaciones críticas, como el control de áreas urbanas concurridas, protección de infraestructuras críticas y operaciones de respuesta rápida en emergencias. La elección entre videovigilancia pasiva y activa depende de varios factores, incluyendo el nivel de riesgo, la naturaleza del área a ser vigilada, y los recursos disponibles para la gestión de la seguridad.

En resumen, mientras la videovigilancia pasiva se centra en la observación y registro, la activa se enfoca en la interacción y la intervención, lo que refleja un cambio hacia sistemas de seguridad más proactivos y adaptativos en el mundo contemporáneo.

3.2. Fundamentos teóricos de vuelo y navegación con UAVs

Se abordaran los principios teóricos que fundamentan las tecnologías y metodologías aplicadas en el desarrollo de sistemas de drones para navegación y videovigilancia autónoma. Los fundamentos teóricos son cruciales para entender cómo se integran y aplican diversas tecnologías emergentes en el diseño y funcionamiento de los UAVs (Unmanned Aerial Vehicles), particularmente en ambientes interiores y para tareas de videovigilancia.

3.2.0.1. Principios de Orientación en la Navegación de UAV

La navegación autónoma en los drones se fundamenta en el control preciso de su orientación y posición dentro de un entorno. Los drones utilizan controladores de vuelo avanzados que regulan tres ejes de movimiento críticos para su operación: yaw, pitch y roll [1].

- **Yaw:** Este controla la rotación del drone alrededor de su eje vertical, esencial para ajustar la dirección en la que el drone está apuntando. Es utilizado para maniobras direccionales, permitiendo que el drone gire a la izquierda o derecha.
- **Pitch:** Regula la inclinación del drone hacia adelante o hacia atrás. Modificar el pitch afecta directamente la velocidad de avance o retro-

ceso del drone y es crucial para el control de la velocidad durante el vuelo.

- **Roll:** Ajusta la inclinación lateral del drone, lo que impacta su habilidad para moverse lateralmente o mantener la estabilidad en condiciones de viento.

La habilidad de un dron para controlar estos ejes de forma precisa es facilitada por giroscopios y acelerómetros que proporcionan datos en tiempo real, permitiendo ajustes dinámicos durante el vuelo para mantener la estabilidad y la trayectoria deseadas.

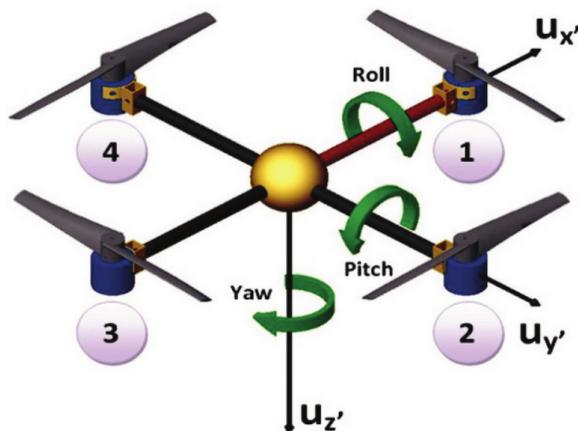


Figura 3.1: Esquema de Orientación en UAVs [1].

Una vez establecido el control sobre la orientación del drone, es esencial integrar sistemas de localización como el GPS (Global Positioning System). Estos sistemas son fundamentales para permitir que el drone no solo se mantenga estable en el aire, sino también para que pueda orientarse y moverse de manera autónoma dentro de su entorno. El GPS proporciona datos de posicionamiento global que son útiles en entornos abiertos.

La combinación de control avanzado de orientación con sistemas de localización precisa permite a los drones realizar tareas de navegación y videovigilancia autónoma, abriendo camino a aplicaciones innovadoras en numerosos campos industriales y comerciales.

3.3. Tipos de UAVs

La clasificación de los drones según la normativa vigente de EASA (Agencia de la Unión Europea para la Seguridad Aérea) incluye seis clases, definidas según su masa máxima al despegue (MTOW) y otras características

específicas. Estas clases están reguladas por los reglamentos (UE) 2019/947 y (UE) 2019/945, con las clases C5 y C6 añadidas en el Reglamento Delegado (UE) 2020/1058 [10].

Requisitos para Cada Clase de Drones

1. Clase C0



Figura 3.2: UAV Tipo C0.

- MTOW no superior a 250g.
- Velocidad de vuelo máxima de 19m/s.
- Diseñados para minimizar daños a personas.
- Deben incluir instrucciones de uso del fabricante.

2. Clase C1



Figura 3.3: UAV Tipo C1.

- MTOW entre 250g y 900g.
- Energía de impacto máxima de 80 julios.
- Velocidad máxima de 19m/s.

- Sistema de geoconsciencia y recuperación automática.
- Identificación a distancia y marcado con un número de serie único.
- Equipados con luces para visibilidad nocturna.

3. Clase C2



Figura 3.4: UAV Tipo C2.

- MTOW hasta 4 kg.
- Función de baja velocidad con límite de 3 m/s.
- Sistema de geoconsciencia.
- Identificación a distancia y número de serie único.
- Sistema de seguridad para terminación de vuelo.
- Iluminación para visibilidad nocturna.

4. Clase C3 - C6



Figura 3.5: UAV Tipo C3 - C7.

- MTOW hasta 25 kg.
- Longitud máxima de 3 metros.
- Sistema de geoconsciencia y protección de datos.
- Sistema de terminación de vuelo.
- Identificación a distancia con iluminación nocturna.

3.4. Videovigilancia, Robots, UAVs

La integración de la robótica y la tecnología de UAVs en los sistemas de videovigilancia representa un avance significativo en cómo se realizan la seguridad y la supervisión en muchos sectores. Estos sistemas no solo extienden las capacidades de los métodos tradicionales de vigilancia fija, sino que también introducen nuevas formas de recopilar y analizar datos en entornos dinámicos y a menudo impredecibles.

3.4.0.1. Aplicaciones en Seguridad y Vigilancia

Los robots y UAVs equipados con cámaras pueden patrullar áreas extensas y realizar tareas de monitoreo continuo, lo que los hace ideales para la vigilancia de infraestructuras críticas, como plantas de energía, aeropuertos y fronteras. Su capacidad para navegar de manera autónoma permite una respuesta rápida ante eventos inusuales o emergencias, mejorando significativamente los tiempos de reacción en comparación con los sistemas de seguridad convencionales.

3.4.0.2. Desafíos Ambientales y Soluciones Integradas

Los drones que operan en interiores deben enfrentarse a un conjunto variable de desafíos ambientales, como la interferencia de señales, fluctuaciones de luz, y obstáculos impredecibles como el movimiento de personas y la apertura y cierre de puertas. La solución para estos desafíos reside en un sistema integrado que utiliza YOLO para la detección de objetos, marcas Aruco para construir un mapa y algoritmos de SLAM para localización del entorno y navegar a través de él.

3.4.0.3. Solución de Problemas Específicos

- **Acceso a Áreas Peligrosas** Los UAVs, en particular, solucionan el problema del acceso a áreas difíciles o peligrosas para los humanos. Por ejemplo, pueden proporcionar imágenes en tiempo real de zonas afectadas por desastres naturales sin poner en riesgo vidas humanas. Esto es crucial para la planificación de rescates y para evaluar el daño de manera eficiente y segura.
- **Mejora de la Cobertura de Vigilancia** La videovigilancia mediante UAVs permite una cobertura más amplia y detallada. En el contexto urbano, pueden ayudar en la gestión del tráfico y en la supervisión de eventos públicos, proporcionando vistas aéreas que serían imposibles de lograr mediante cámaras estáticas.

3.4.0.4. Implementaciones Industriales de sistemas de vigilancia

Anduril Industries Es una empresa tecnológica estadounidense que se especializa en el desarrollo de sistemas autónomos avanzados para la defensa. Utiliza su plataforma de software Lattice para conectar capacidades autónomas de sentido y control de mando con hardware modular y escalable. Aquí te presento un resumen de los productos mencionados[2]:

- **Lattice:** Es un sistema operativo impulsado por inteligencia artificial que coordina sistemas de detección y mando y control. Lattice agrega datos de sensores distribuidos para ofrecer una solución integrada que permite a las entidades de defensa y seguridad manejar misiones de forma más eficiente (Fig 3.6).



Figura 3.6: Plataforma de integración de sensores en UAV Lattice [2].

- **Anvil:** Es un sistema utilizado para la intercepción autónoma de amenazas de drones. Navega de manera autónoma para interceptar drones potencialmente peligrosos, utilizando velocidad y guía terminal a bordo para entregar energía cinética y neutralizar la amenaza con daño colateral mínimo (Fig 3.7).



Figura 3.7: UAV de Intercepción - Anvil [2].

- **Wisp:** Es un Radar que proporciona imágenes de alta calidad en un área amplia para detección automática de amenazas y conciencia situacional persistente. Está diseñado con un imager IR transformador que combina AI en tiempo real con un diseño compacto de hardware (Fig 3.8).



Figura 3.8: Radar WISP [2].

- **Roadrunner:** Es un vehículo aéreo autónomo con capacidades de despegue y aterrizaje vertical (VTOL), lo que le otorga flexibilidad para lanzarse y retornar rápidamente. Está diseñado para ofrecer un rendimiento extraordinario a bajo costo y es adecuado para varias misiones de defensa y seguridad (Fig 3.9).



Figura 3.9: UAV de Seguimiento RoadRunner [2].

Prosegur Security Prosegur ha desarrollado "Yellow", un perro robot que, además de realizar tareas de vigilancia convencional, está equipado para interactuar con otros sistemas de seguridad, lo que mejora significativamente las capacidades de respuesta y análisis en tiempo real de situaciones de seguridad[11].

Boston Dynamics Otra empresa destacada en este ámbito es Boston Dynamics, cuyo robot "Spot" se ha adaptado para patrullas de seguridad, inspecciones y operaciones en entornos peligrosos [3]. Spot es capaz de navegar por terrenos difíciles y realizar tareas de vigilancia sin necesidad de guía humana directa, lo que lo hace ideal para operaciones en zonas de alto riesgo (Fig 3.10).



Figura 3.10: Perro Robot para Vigilancia [3].

Estas tecnologías no solo proporcionan soluciones de vigilancia más eficientes sino que también reducen los riesgos para los seres humanos al poder operar en entornos peligrosos o inaccesibles, asegurando una respuesta más rápida y segura ante incidentes.

3.4.0.5. Ventajas y Desventajas de los sistemas existentes

Aunque las implementaciones industriales como las desarrolladas por Anduril Industries, Prosegur, y Boston Dynamics ofrecen soluciones robustas y avanzadas en el ámbito de la autonomía y la vigilancia, presentan ciertas limitaciones en comparación con nuestro sistema de drones autónomos para interiores. A continuación, se detallan los principales puntos a considerar:

Anduril Industries Los sistemas de Anduril, como Lattice y Anvil, están altamente optimizados para aplicaciones en defensa y entornos exteriores amplios, ofreciendo integración de sensores avanzados y capacidades de intercepción autónoma. Sin embargo, presentan las siguientes desventajas en comparación con nuestro sistema:

- **Enfoque en aplicaciones militares:** Los productos de Anduril están diseñados principalmente para escenarios militares y de defensa, lo que los hace menos adecuados para aplicaciones civiles y comerciales en interiores.
- **Costo elevado:** La tecnología avanzada y el enfoque en la defensa hacen que sus sistemas sean costosos, lo que puede no ser justificable para empresas que buscan soluciones para vigilancia en interiores.
- **Complejidad de integración:** La integración de sus sistemas requiere infraestructura compleja y especializada, lo que puede ser una barrera para implementaciones más simples o en espacios cerrados.

Nuestro sistema, por otro lado, ofrece una solución específica para interiores, optimizada para operar en espacios reducidos y con un costo significativamente menor, facilitando la adopción en sectores comerciales y empresariales.

Prosegur Security El robot "Yellow" de Prosegur es innovador en la integración con otros sistemas de seguridad y en la capacidad de operar en vigilancia convencional. No obstante, tiene ciertas limitaciones:

- **Limitación en movilidad:** Como un robot terrestre, "Yellow" no puede acceder a zonas elevadas o de difícil acceso, limitando su efectividad en ciertos entornos interiores.
- **Menor versatilidad:** Al ser un dispositivo especializado, su adaptabilidad para diferentes tipos de misiones o entornos es limitada comparada con un sistema de drones.

En contraste, nuestro sistema es capaz de maniobrar en espacios tridimensionales, superando obstáculos y accediendo a áreas que un robot terrestre no podría alcanzar.

Boston Dynamics "Spot" de Boston Dynamics es un robot con alta capacidad para patrullas y operaciones en entornos peligrosos. Sin embargo, al igual que "Yellow", presenta desventajas en entornos interiores:

- **Limitaciones en entornos cerrados:** Aunque Spot es ágil y capaz de moverse por terrenos difíciles, su diseño para exteriores puede limitar su eficiencia en espacios confinados o interiores con restricciones arquitectónicas.
- **Alto costo y mantenimiento:** La avanzada tecnología de Spot implica un costo elevado y una mayor necesidad de mantenimiento especializado, lo cual puede no ser sostenible para aplicaciones de bajo presupuesto.

Nuestro sistema está específicamente diseñado para maximizar la eficiencia en interiores, ofreciendo una solución más económica y con menor requerimiento de mantenimiento, ideal para vigilancia y monitoreo en entornos cerrados.

Conclusión En resumen, aunque las soluciones mencionadas representan el pináculo de la tecnología en sus respectivos campos, nuestro sistema de dron autónomo para interiores se destaca por su adaptabilidad a espacios reducidos, su facilidad de implementación y un costo significativamente menor, lo que lo convierte en una alternativa más accesible y eficaz para aplicaciones en interiores.

3.4.1. Modelos de UAVs para Interiores

Los UAVs para interiores difieren significativamente de sus equivalentes de exteriores, destacando por su pequeño tamaño, mayor maniobrabilidad y capacidad de vuelo en espacios reducidos. Estas cualidades son esenciales para tareas que van desde la inspección detallada de infraestructuras hasta la interacción segura en ambientes poblados.

3.4.2. Tello EDU

El Tello EDU (Fig 3.11) es un dron de interior de gama económica y gran versatilidad. Destacado por su bajo costo y por contar con un SDK (Software

Development Kit) avanzado, se ha ganado la preferencia de la comunidad para una variedad de proyectos, desde educativos hasta de investigación. Se explica mas en detalle en el capítulo 4.1.1.



Figura 3.11: TelloEDU.

3.4.3. Spark

Por otro lado, el Spark (Fig 3.12), si bien es un UAV más avanzado y con características de vuelo inteligentes como el reconocimiento facial y de gestos, tiene un costo mayor [12]. A pesar de su mayor precio, no cuenta con el mismo nivel de apoyo comunitario que Tello ni con un acceso abierto a su código fuente. La falta de una amplia disponibilidad de proyectos de código abierto y una menor flexibilidad para la personalización lo hacen menos atractivo para entornos donde la experimentación y la adaptación continua son clave.



Figura 3.12: Spark.

3.4.4. Crazyflie

El Crazyflie (Fig 3.13) es un dron modular y altamente personalizable que ha capturado la atención de la comunidad investigadora y educativa [13]. Su tamaño extremadamente reducido y su arquitectura de código abierto lo convierten en una plataforma ideal para experimentación y desarrollo en entornos cerrados. Además de su versatilidad, el Crazyflie cuenta con una amplia gama de módulos y sensores adicionales que permiten a los desarrolladores expandir sus capacidades de vuelo y navegación. Sin embargo, su complejidad y la necesidad de un mayor conocimiento técnico para su configuración y operación pueden representar un desafío para los usuarios menos experimentados.



Figura 3.13: Crazyflie

3.4.5. Parrot Bebop 2

El Parrot Bebop 2 (Fig 3.14) es un dron que, aunque principalmente diseñado para exteriores, se ha utilizado en entornos interiores debido a su estabilidad de vuelo y características avanzadas de grabación [14]. Equipado con una cámara de alta resolución y capacidades de vuelo autónomo, el Bebop 2 ofrece una experiencia de usuario avanzada. No obstante, su tamaño y peso lo hacen menos adecuado para espacios reducidos, y su mayor precio, junto con una menor flexibilidad para la personalización y modificación, lo colocan en una posición menos favorable en comparación con otras opciones enfocadas en entornos interiores.



Figura 3.14: Parrot Bebop 2.

3.4.6. Comparativa de UAVs para interiores

Al evaluar las opciones de drones para aplicaciones en interiores, el Tello EDU emerge como la elección más equilibrada, combinando potencia, flexibilidad y un precio accesible. A diferencia del Spark, que aunque ofrece tecnologías avanzadas como el reconocimiento facial, su elevado costo y la menor flexibilidad en cuanto a personalización lo colocan en desventaja. El Crazyflie, aunque increíblemente versátil y adaptable, requiere un mayor conocimiento técnico, lo que podría limitar su uso en proyectos más sencillos. El Bebop 2, por otro lado, está diseñado principalmente para exteriores, y su tamaño y costo lo hacen menos ideal para espacios reducidos.

El Tello EDU, sin embargo, se destaca no solo por su accesibilidad y un SDK robusto que permite una amplia gama de configuraciones, sino también por el apoyo de una comunidad activa que ha contribuido significativamente haciendo ingeniería inversa de su protocolo de comunicación a bajo nivel[15]. Esto ha permitido a los usuarios explorar profundamente sus capacidades y adaptar el dron para diversas aplicaciones, convirtiéndolo en la opción preferida para quienes buscan un equilibrio entre costo, potencia y posibilidades de personalización en proyectos de interiores.

3.4.7. Bibliotecas de programación para el dron Tello

Existen varias API no oficiales que facilitan la creación de aplicaciones para interactuar con el Tello. Gracias a estas API, es posible desarrollar aplicaciones de escritorio y móviles para el Tello en muchas plataformas utilizando diversos lenguajes de programación.

Las API listadas a continuación 3.1, en su mayoría, superan con creces

las capacidades limitadas de la API oficial proporcionada por Ryze.

Lenguaje	API	Windows	MacOS	Linux	Android	iOS
C#	TelloLib	Sí	No	No	Sí	No
Dart	Ryze SDK 1.3	Sí	Sí	Sí	Sí	Sí
Go	Gobot	Sí	Sí	Sí	?	?
Go	Tello	Sí	Sí	Sí	?	?
Python	pytello	?	?	Sí	?	?
Python	TelloPy	?	?	Sí	?	?
Python	tello-asyncio	Sí	Sí	Sí	?	?
Ruby	tello gem	Sí	Sí	Sí	?	?
Rust	rust-tello	Sí	Sí	Sí	?	?
Processing	Ryze SDK 1.3	Sí	Sí	Sí	No	?
Scratch2 (Raspberry 3)	Ryze SDK 1.3	No	?	Sí	No	?

Tabla 3.1: Bibliotecas de Programación para Tello

La biblioteca Tellopy [16] ha sido elegida para este proyecto porque está basado en el protocolo de bajo nivel del dron[15], lo que le permite ofrecer una funcionalidad completa similar a la de la aplicación original de Tello. El uso del SDK oficial introduce un retraso de aproximadamente un segundo, lo cual es inaceptable para la captura de datos en tiempo real. Además, Tellopy está escrito en Python, lo que facilita su integración con otras herramientas y librerías como OpenCV, YOLOv, etc. Se explica con mas detalle en 4.2.2

3.5. Reconocimiento y Detección de Patrones

El reconocimiento de patrones es un campo de la inteligencia artificial que permite a los sistemas informáticos obtener información significativa de imágenes digitales, vídeos y otros inputs visuales, y actuar o hacer recomendaciones basadas en esa información. En el contexto de los UAVs y sistemas de seguridad, la visión por computador juega un papel crítico en la navegación, la detección de objetos y la interacción con el entorno.

3.5.1. Sistemas de Marcadores

Existen varios sistemas de marcadores utilizados en aplicaciones de visión por computadora y realidad aumentada. Cada uno de estos sistemas tiene características que lo hacen adecuado para diferentes tipos de aplicaciones, incluyendo la localización y navegación de drones en interiores.

3.5.1.1. ARUCO

ARUCO es un sistema de marcadores basado en patrones cuadrados simples que son fácilmente reconocibles en imágenes digitales. Este sistema se ha convertido en una herramienta esencial en diversas aplicaciones de realidad aumentada y robótica, particularmente valiosa en contextos donde la precisión y la rapidez en la localización son cruciales [17].

Los marcadores ARUCO, cuando se configuran correctamente, no solo permiten la detección rápida de puntos de interés, sino que también facilitan la medición precisa del ángulo y la distancia hasta estos puntos, con un margen de error mínimo que puede reducirse a solo unos pocos milímetros. Esta precisión es posible gracias a la robustez del algoritmo de detección de Aruco, que interpreta los marcadores dentro del campo visual de la cámara y calcula la posición y orientación relativas con alta fiabilidad.

En el ámbito de los drones, el uso de ARUCO es particularmente relevante para operaciones que requieren alta precisión en la calibración espacial. Los marcadores proporcionan puntos de referencia clave que permiten a los drones realizar tareas críticas como el aterrizaje autónomo y la navegación precisa en entornos interiores, donde el uso de GPS no es viable (Fig 3.15). Esta capacidad es fundamental, por ejemplo, en escenarios de rescate o inspección en estructuras cerradas, donde la precisión en el posicionamiento del dron puede impactar directamente en el éxito de la misión[18].

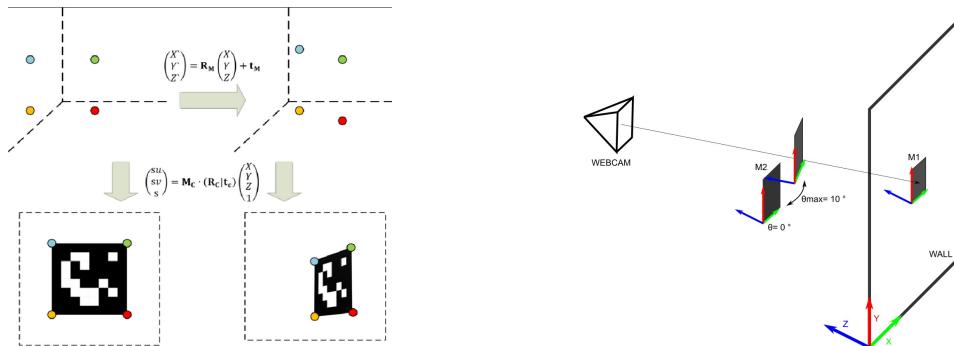


Figura 3.15: Esquema de funcionamiento de Sistemas de Marcadores.

Además, el sistema Aruco es altamente configurable, lo que permite a los ingenieros y desarrolladores ajustar la escala y los parámetros de detección según las necesidades específicas del proyecto, optimizando así el rendimiento del sistema en diversas condiciones operativas.

3.5.1.2. AprilTag

AprilTag es otro sistema de marcadores visuales similar a ARUCO, que también utiliza patrones cuadrados con códigos binarios para la detección y localización en entornos tridimensionales. La principal ventaja de AprilTag es su robustez frente a variaciones en las condiciones de iluminación y su capacidad para operar en entornos más dinámicos, donde los marcadores pueden ser parcialmente ocluidos. Además, AprilTag es conocido por su bajo requerimiento computacional, lo que lo hace ideal para dispositivos con recursos limitados, como drones de bajo costo o robots móviles[19].

3.5.1.3. ARToolKit

ARToolKit es uno de los sistemas de marcadores más antiguos y ampliamente utilizados en la realidad aumentada. A diferencia de ARUCO y AprilTag, ARToolKit no solo se enfoca en la detección de marcadores cuadrados, sino que también permite la integración de modelos 3D en tiempo real, basados en la posición y orientación de los marcadores. Aunque es menos preciso que ARUCO en términos de localización, ARToolKit sigue siendo popular en aplicaciones donde la interacción en tiempo real con objetos virtuales es más importante que la precisión espacial absoluta[20].

3.5.1.4. Chilitags

Chilitags es otro sistema de marcadores diseñado para aplicaciones de visión por computadora, particularmente en entornos de bajo costo y con recursos limitados. Los marcadores Chilitags son altamente configurables y pueden ser utilizados en diferentes tamaños y formas, lo que permite su adaptación a una amplia gama de aplicaciones. Aunque no es tan preciso como ARUCO, Chilitags es valorado por su flexibilidad y facilidad de uso en proyectos de prototipado rápido[21].

3.5.2. Comparativa de sistemas de marcadores

La elección de un sistema de marcadores depende en gran medida de los requisitos específicos de la aplicación. ARUCO se destaca sobre otros sistemas por varias razones clave:

- **Precisión:** ARUCO ofrece una alta precisión en la detección y localización de marcadores, lo cual es fundamental en aplicaciones donde la exactitud en la medición del ángulo y la distancia es crítica, como en la navegación de drones y robótica.

- **Robustez:** El algoritmo de ARUCO es robusto frente a diferentes condiciones de iluminación y puede manejar con eficacia situaciones donde los marcadores están parcialmente fuera del campo visual. Esto lo hace ideal para entornos complejos y dinámicos.
- **Eficiencia computacional:** Aunque hay sistemas como AprilTag que también son eficientes, ARUCO ofrece un buen equilibrio entre precisión y eficiencia computacional, permitiendo su uso en dispositivos con capacidades de procesamiento moderadas sin comprometer el rendimiento.
- **Integración y soporte:** ARUCO tiene un amplio soporte en la comunidad de desarrollo y está bien integrado en muchas bibliotecas de visión por computadora, como OpenCV, lo que facilita su implementación en diversos proyectos.

La elección de ARUCO sobre otros sistemas como AprilTag o ARToolKit suele estar motivada por la necesidad de una alta precisión en la detección de marcadores y la capacidad para operar en entornos con iluminación variable y posibles occlusiones. Además, su balance entre precisión y eficiencia lo convierte en una opción preferida para aplicaciones críticas como la navegación de drones en entornos interiores o la robótica de alta precisión.

3.5.2.1. Marcadores y Diccionarios de ARUCO

Un marcador ARUCO es un cuadrado sintético con un borde negro ancho y una matriz binaria interna que determina su identificador (id). El tamaño del marcador determina el tamaño de la matriz interna. Por ejemplo, un tamaño de marcador de 4x4 está compuesto por 16 bits [22][23].

Un diccionario de marcadores es el conjunto de marcadores considerados en una aplicación específica. Los principales atributos de un diccionario son el tamaño del diccionario y el tamaño del marcador 3.2.

Propiedad	Descripción
Tamaño del diccionario	Número de marcadores que componen el diccionario (4x4 5x5 6x6 7x7).
Tamaño del marcador	Tamaño de los marcadores (número de bits/módulos).

Tabla 3.2: Propiedades de los diccionarios ARUCO

El módulo ARUCO incluye algunos diccionarios predefinidos con diferentes tamaños de diccionario y tamaños de marcador.

3.5.2.2. Creación de Marcadores

Antes de su detección, los marcadores deben imprimirse para ser colocados en el entorno. Las imágenes de los marcadores pueden generarse usando la función `generateImageMarker()` 3.1. Cabe mencionar que OpenCV también proporciona un recubrimiento en Python para llamar a estas funciones, lo que facilita su uso en este lenguaje. A continuación se muestra un ejemplo de cómo generar y guardar un marcador ArUco utilizando OpenCV en C++.

Listing 3.1: Generación y guardado de un marcador ArUco usando OpenCV

```
1 cv::Mat markerImage;
2 cv::aruco::Dictionary dictionary =
3 cv::aruco::getPredefinedDictionary(cv::aruco::DICT_6X6_250);
4 cv::aruco::generateImageMarker(dictionary, 23, 200, markerImage, 1);
5 cv::imwrite("marker23.png", markerImage);
```

3.5.2.3. Parámetros de `cv:generateImageMarker()`

- **Primer parámetro:** El objeto `cv::aruco::Dictionary` creado previamente.
- **Segundo parámetro:** El id del marcador, en este caso el marcador 23 del diccionario `cv::aruco::DICT6X6250`.
- **Tercer parámetro:** El tamaño de la imagen del marcador de salida. En este caso, la imagen de salida tendrá un tamaño de 200x200 píxeles.
- **Cuarto parámetro:** La imagen de salida.
- **Último parámetro:** Parámetro opcional para especificar el ancho del borde negro del marcador.

3.5.2.4. Detección de marcadores

Dada una imagen que contiene marcadores ArUco, el proceso de detección debe devolver una lista de marcadores detectados.

1. **Detección de candidatos a marcadores:** En este paso se analizan las imágenes para encontrar formas cuadradas que sean candidatas a marcadores.
2. **Verificación de candidatos:** Se analiza la codificación interna de los candidatos para determinar si realmente son marcadores.

La detección de marcadores se realiza en la función, `cv::ArucoDetector::detectMarkers()`. Esta función es la más importante del módulo 3.2, ya que el resto de la funcionalidad se basa en los marcadores detectados por `detectMarkers()`.

Listing 3.2: Detección de marcadores ArUco usando OpenCV

```

1 cv::Mat inputImage;
2 // ... leer inputImage ...
3 std::vector<int> markerIds;
4 std::vector<std::vector<cv::Point2f>> markerCorners,
5     rejectedCandidates;
6 cv::aruco::DetectorParameters detectorParams = cv::aruco::
7     DetectorParameters();
8 cv::aruco::Dictionary dictionary =
9 cv::aruco::getPredefinedDictionary(cv::aruco::DICT_6X6_250);
10 cv::aruco::ArucoDetector detector(dictionary, detectorParams);
11 detector.detectMarkers(inputImage, markerCorners, markerIds,
12     rejectedCandidates);

```

3.5.2.5. Estimación de Pose

La estimación de pose de la cámara puede realizarse después de detectar los marcadores.

- **corners:** Vector de esquinas de los marcadores devuelto por la función `cv::aruco::ArucoDetector::detectMarkers()`.
- **marker side:** Tamaño del lado del marcador en metros o en otra unidad.
- **camMatrix y distCoeffs:** Parámetros de calibración de la cámara.
- **rvecs y tvecs:** Vectores de rotación y de traslación respectivamente para cada uno de los marcadores detectados en `corners`.

OpenCV proporciona una función para dibujar los ejes, por lo que se puede verificar la estimación de la pose:

3.5.2.6. Selección de Diccionarios

El módulo ARUCO proporciona la clase `Dictionary` para representar un diccionario de marcadores 3.3.

Propiedad	Descripción
<code>bytesList</code>	Información del código del marcador.
<code>markerSize</code>	Número de bits por dimensión.
<code>maxCorrectionBits</code>	Número máximo de bits que pueden ser corregidos.

Tabla 3.3: Parámetros de la clase `Dictionary`

3.5.2.7. Parámetros del detector

Uno de los parámetros de `cv::ArucoDetector` es un objeto `cv::DetectorParameters`. Este objeto incluye todas las opciones que pueden personalizarse durante el proceso de detección de marcadores.

Uno de los primeros pasos en el proceso de detección de marcadores es la umbralización adaptativa de la imagen de entrada 3.4.

Parámetro	Descripción
<code>adaptiveThreshWinSizeMin</code>	Tamaño mínimo de la ventana de umbralización.
<code>adaptiveThreshWinSizeMax</code>	Tamaño máximo de la ventana de umbralización.
<code>adaptiveThreshWinSizeStep</code>	Incremento del tamaño de la ventana de umbralización.
<code>adaptiveThreshConstant</code>	Valor constante añadido en la operación de umbralización.

Tabla 3.4: Parámetros de umbralización

3.5.2.8. Filtrado de contornos

Después de la umbralización, se detectan contornos. Sin embargo, no todos los contornos son considerados candidatos a marcadores. Se filtran en diferentes etapas 3.5.

Parámetro	Descripción
<code>minMarkerPerimeterRate</code>	Tamaño mínimo de un marcador.
<code>maxMarkerPerimeterRate</code>	Tamaño máximo de un marcador.
<code>polygonalApproxAccuracyRate</code>	Error máximo que puede producir la aproximación poligonal.
<code>minCornerDistanceRate</code>	Distancia mínima entre cualquier par de esquinas en el mismo marcador.
<code>minMarkerDistanceRate</code>	Distancia mínima entre cualquier par de esquinas de dos marcadores diferentes.
<code>minDistanceToBorder</code>	Distancia mínima de cualquier esquina del marcador al borde de la imagen.

Tabla 3.5: Parámetros de filtrado de contornos

3.5.2.9. Extracción de bits

Después de la detección de candidatos, se analizan los bits de cada candidato para determinar si son marcadores (Tab 3.6 y Fig 3.16).

Parámetro	Descripción
<code>markerBorderBits</code>	Ancho del borde del marcador.
<code>minOtsuStdDev</code>	Desviación estándar mínima de los valores de píxeles para realizar la umbralización de Otsu.
<code>perspectiveRemovePixelPerCell</code>	Número de píxeles por celda después de corregir la distorsión de perspectiva.
<code>perspectiveRemoveIgnoredMarginPerCell</code>	Número de píxeles ignorados en los márgenes de las celdas.

Tabla 3.6: Parámetros de extracción de bits

3.5.2.10. Identificación de marcadores

Después de extraer los bits, el siguiente paso es verificar si el código extraído pertenece al diccionario de marcadores y, si es necesario, se puede

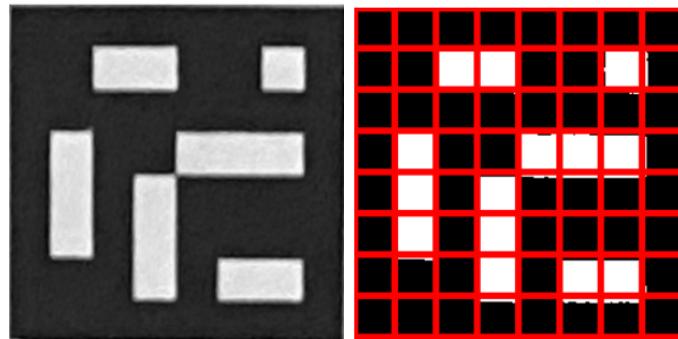


Figura 3.16: Parámetros de extracción de bits Aruco.

realizar la corrección de errores 3.7.

Parámetro	Descripción
<code>maxErroneousBitsInBorderRate</code>	Número permitido de bits erróneos en el borde.
<code>errorCorrectionRate</code>	Tasa de corrección de errores permitida.

Tabla 3.7: Parámetros de identificación de marcadores

3.5.2.11. Refinamiento de esquinas

Después de que los marcadores han sido detectados e identificados, el último paso es realizar un refinamiento subpíxel de las posiciones de las esquinas 3.8.

Parámetro	Descripción
<code>cornerRefinementMethod</code>	Método de refinamiento subpíxel de las esquinas.
<code>cornerRefinementWinSize</code>	Tamaño máximo de la ventana para el proceso de refinamiento de esquinas.
<code>relativeCornerRefinementWinSize</code>	Tamaño de ventana dinámico para el refinamiento de esquinas relativo al tamaño del módulo Aruco.
<code>cornerRefinementMinAccuracy</code>	Valor de error mínimo antes de detener el proceso de refinamiento subpíxel.

Tabla 3.8: Parámetros de refinamiento de esquinas

3.5.3. Detección y localización de Objetos en Tiempo Real

La detección y localización de objetos en tiempo real es fundamental en una variedad de aplicaciones tecnológicas, desde la conducción autónoma y la robótica hasta la realidad aumentada y la seguridad. Diversos métodos y algoritmos se han desarrollado para optimizar esta tarea, permitiendo identificar y ubicar objetos de manera rápida y precisa dentro de una escena.

3.5.3.1. YOLO (You Only Look Once)

YOLO es un sistema de detección de objetos que se destaca por su capacidad para realizar detecciones en tiempo real, procesando imágenes en una sola evaluación de la red. Esto lo diferencia notablemente de otros métodos que requieren múltiples pasos por una red para detectar objetos[24].

Ventajas:

- **Alta velocidad:** Procesa imágenes en tiempo real, permitiendo múltiples cuadros por segundo con una sola pasada a través de la red.
- **Aportes de la comunidad:** Amplio soporte y mejoras continuas por parte de la comunidad, lo que ha llevado a versiones optimizadas para diferentes usos.
- **Flexibilidad y escalabilidad:** Adaptable a una variedad de dispositivos, desde sistemas embebidos hasta servidores de alto rendimiento.
- **Eficiencia en el consumo de recursos:** Optimizado para funcionar en dispositivos con recursos limitados, como móviles y drones.

Desventajas:

- **Menor precisión en objetos pequeños:** Puede ser menos preciso en la detección de objetos muy pequeños en comparación con métodos más avanzados como Faster R-CNN.
- **Compromiso entre precisión y velocidad:** Aunque balanceado, puede no ser la mejor opción cuando la precisión es más importante que la velocidad.

3.5.3.2. MediaPipe

MediaPipe, desarrollado por Google, es un marco multiplataforma para la percepción de objetos y el análisis de video en tiempo real. A diferencia de YOLO, que se centra principalmente en la detección de objetos genéricos, MediaPipe se utiliza ampliamente para el seguimiento y la detección de

características específicas del cuerpo humano, como manos, rostro y cuerpo completo, con gran precisión y eficiencia. MediaPipe emplea modelos avanzados de aprendizaje profundo optimizados para su uso en dispositivos móviles y embebidos, lo que lo hace ideal para aplicaciones en tiempo real que requieren un bajo consumo de recursos[25].

Ventajas:

- **Precisión en características humanas:** Excepcional en el seguimiento y detección de manos, rostro y cuerpo, ideal para aplicaciones de realidad aumentada y interacción hombre-máquina.
- **Optimización para dispositivos móviles:** Diseñado para funcionar eficientemente en dispositivos móviles y sistemas embebidos con un bajo consumo de recursos.
- **Fácil integración en aplicaciones AR/VR:** Ideal para aplicaciones de realidad aumentada y virtual que requieren detección precisa de características humanas.

Desventajas:

- **Limitado a casos específicos:** Menos versátil para la detección de objetos genéricos fuera del ámbito de características humanas.
- **Menor adaptabilidad a tareas generales:** No es tan efectivo para aplicaciones que requieren detección de una amplia gama de objetos.

3.5.3.3. SSD (Single Shot MultiBox Detector)

El SSD es otro método de detección de objetos en tiempo real que, al igual que YOLO, realiza predicciones en una sola pasada a través de la red. SSD genera múltiples cajas delimitadoras de diferentes tamaños y relaciones de aspecto a partir de varias ubicaciones de la imagen, prediciendo simultáneamente las clases de objetos y ajustando las cajas para que se ajusten mejor a los objetos detectados. Aunque SSD es ligeramente menos preciso que YOLO en algunas tareas, se destaca por su balance entre velocidad y precisión, lo que lo convierte en una opción popular en aplicaciones de tiempo real[26].

Ventajas:

- **Buena velocidad y precisión:** Ofrece un equilibrio entre velocidad y precisión, adecuado para aplicaciones en tiempo real.
- **Versatilidad:** Capaz de manejar diferentes tamaños de objetos con alta eficiencia, lo que lo hace útil en diversas aplicaciones.

- **Eficiencia en tiempo real:** Similar a YOLO, puede funcionar en tiempo real con un rendimiento sólido.

Desventajas:

- **Menor precisión en objetos pequeños:** Puede tener dificultades para detectar objetos muy pequeños con la misma precisión que otros métodos.
- **Requerimientos de recursos:** Puede necesitar más recursos que YOLO en algunas configuraciones, lo que lo hace menos adecuado para dispositivos con hardware limitado.

3.5.3.4. Faster R-CNN

Faster R-CNN es un método que combina la precisión del R-CNN original con la velocidad mejorada gracias a la introducción de la Red de Propuestas de Regiones (RPN), que permite generar propuestas de regiones y realizar la clasificación en una sola etapa. Aunque no es tan rápido como YOLO o SSD, Faster R-CNN ofrece una mayor precisión, especialmente en tareas donde la detección exacta es crítica. Este método es frecuentemente utilizado en aplicaciones donde la precisión tiene prioridad sobre la velocidad, como en sistemas de vigilancia y análisis de imágenes médicas[27].

Ventajas:

- **Alta precisión:** Uno de los métodos más precisos, ideal para aplicaciones donde la exactitud es crítica, como en el análisis de imágenes médicas.
- **Detección robusta:** Su enfoque basado en la generación de propuestas de regiones permite una detección más robusta en escenarios complejos.

Desventajas:

- **Menor velocidad:** Es significativamente más lento que YOLO y SSD, lo que lo hace menos adecuado para aplicaciones que requieren detección en tiempo real.
- **Alto consumo de recursos:** Requiere más capacidad computacional, limitando su uso en dispositivos de bajo rendimiento.

3.5.3.5. Comparación sistemas de detección de objetos

La elección de YOLO como el método preferido para este proyecto de detección y localización de objetos en tiempo real se basa en sus múltiples ventajas:

- **Velocidad:** YOLO es capaz de procesar imágenes en tiempo real con alta velocidad, lo que es crucial en aplicaciones donde la rapidez es esencial.
- **Aportes de la comunidad:** La fuerte comunidad de desarrolladores que respalda YOLO garantiza que esté en constante evolución, adaptándose a nuevas necesidades y mejorando su rendimiento.
- **Flexibilidad y escalabilidad:** La capacidad de YOLO para adaptarse a diversas plataformas, desde dispositivos móviles hasta entornos de alto rendimiento, lo hace extremadamente versátil.
- **Eficiencia en el consumo de recursos:** YOLO ha sido optimizado para operar con recursos limitados, lo que lo convierte en una opción ideal para aplicaciones móviles y embebidas.

Aunque MediaPipe, SSD y Faster R-CNN ofrecen ventajas en áreas específicas, como la precisión en características humanas o la detección en escenarios complejos, YOLO es la opción más equilibrada para aplicaciones que requieren una detección y localización rápidas, precisas y eficientes en tiempo real.

3.5.3.6. Evolución de YOLO: Comparativa de YOLOv4 a YOLOv8

YOLO es un sistema de detección de objetos destacado en el campo de la visión por computadora. Desde su cuarta versión hasta la más reciente, YOLO ha continuado evolucionando, ofreciendo mejoras significativas en precisión y eficiencia. A continuación se presenta una comparativa de las versiones de YOLOv4 a YOLOv8, con un enfoque en su consumo de recursos y rendimiento (Fig 3.9).

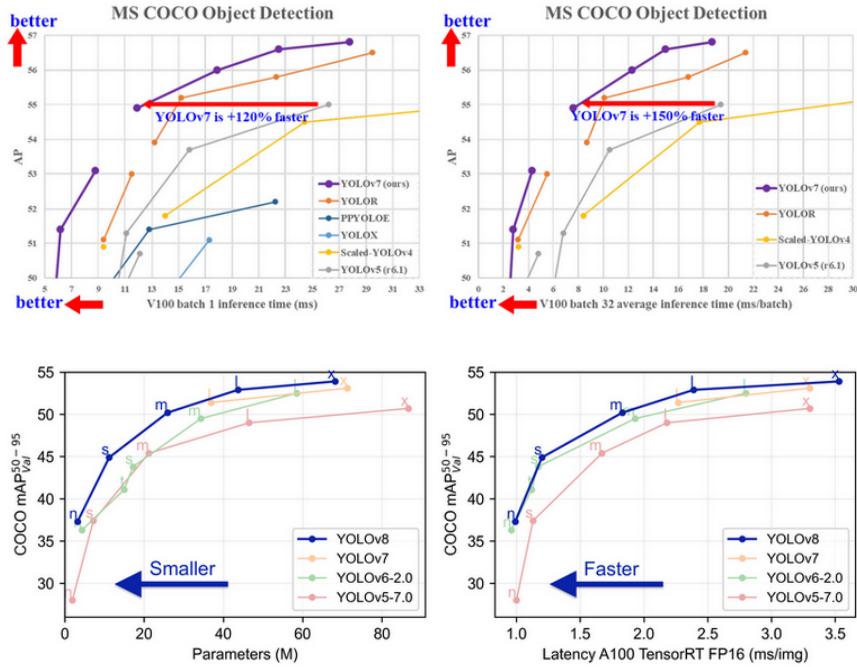


Figura 3.17: Comparativa versiones de Yolo [4].

Comparación de Consumo y Rendimiento Para aplicaciones donde el consumo de recursos es crítico, YOLOv5, especialmente en sus versiones más pequeñas como YOLOv5s, es conocido por ser ligero y rápido. Por otro lado, para máxima precisión donde los recursos no son una limitación, YOLOv7 y YOLOv8 son más adecuados.

Ventajas de YOLOv5 YOLOv5 es una implementación no oficial del sistema YOLO que ha ganado popularidad debido a su flexibilidad y facilidad de uso. Las principales ventajas de YOLOv5 incluyen su excepcional velocidad y precisión en la detección de objetos, crucial para aplicaciones que demandan respuestas inmediatas, como la navegación autónoma de drones en interiores. Esta versión de YOLO es particularmente notable por su capacidad para ser entrenada rápidamente en conjuntos de datos personalizados, lo que permite una adaptación eficaz a entornos dinámicos y cambiantes.

Diferencias entre las Versiones de YOLOv5 YOLOv5 viene en varias versiones que se adaptan a diferentes necesidades y capacidades computacionales, desde YOLOv5s (small) hasta YOLOv5x (extra large) como se puede ver en la figura 3.18 y tabla 3.10. A continuación se describen las diferencias clave entre estas versiones:

Versión	Lanzamiento	Características clave	Consumo	Rendimiento
YOLOv4	Abril 2020	CSPDarknet53 como backbone, Mish activation, Cross-stage partial connections (CSP), Self-adversarial training	Moderado	Alta eficacia en comparación con versiones anteriores
YOLOv5	Junio 2020	Altamente personalizable, mejoras en la arquitectura, no es una versión oficial	Bajo a moderado	Excelente equilibrio entre velocidad y precisión
YOLOv6	2022	Diseñado para ser más rápido y eficiente en hardware limitado	Moderado	Mejora significativa en la velocidad, ideal para aplicaciones en tiempo real
YOLOv7	2022	E-ELAN y E-RepLK para mejorar la precisión sin aumentar la carga computacional	Alto	Muy alta precisión, adecuado para aplicaciones que requieren gran detalle en la detección
YOLOv8	2023	Mejoras en precisión y eficiencia con innovaciones en arquitectura de redes neuronales	Alto	Aún en evaluación, promete ser el más avanzado en precisión y capacidades

Tabla 3.9: Comparativa de versiones YOLO[5]

- **YOLOv5s (Small):** Es la versión más ligera, diseñada para entornos donde los recursos computacionales son limitados o para aplicaciones que requieren una velocidad extremadamente alta. A pesar de su tamaño reducido, mantiene una precisión razonable, lo que la hace ideal para dispositivos móviles o sistemas embebidos.
- **YOLOv5m (Medium):** Ofrece un equilibrio entre velocidad y precisión. Es adecuado para aplicaciones que necesitan un buen rendimiento de detección sin el costo computacional de las versiones más grandes.
- **YOLOv5l (Large):** Mejora la precisión a costa de un mayor consumo de recursos. Esta versión es preferida cuando la precisión es más crítica y se dispone de capacidad computacional adecuada.
- **YOLOv5x (Extra Large):** Proporciona la mejor precisión entre todas las variantes de YOLOv5. Aunque requiere significativos recursos computacionales, es ideal para aplicaciones en las que la precisión en la detección de objetos es la prioridad máxima, como en sistemas de vigilancia avanzada o en investigaciones donde se requieren detalles finos.

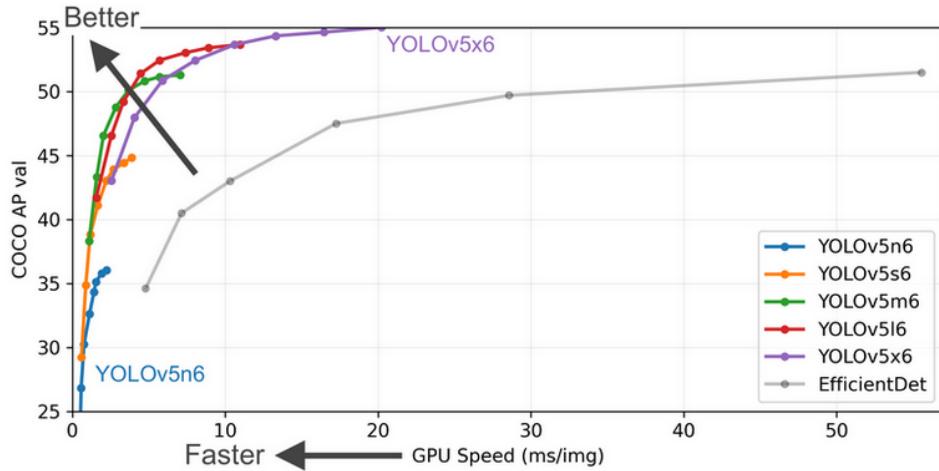


Figura 3.18: Comparativa diferentes versiones YOLOv5.

Estas variantes de YOLOv5 permiten una personalización y escalabilidad según las necesidades específicas del proyecto, asegurando que los desarrolladores puedan elegir la versión adecuada que equilibre la velocidad, precisión y consumo de recursos para su aplicación específica[28].

Modelo	Size (pixels)	mAP	Tiempo	Params	FLOPs
YOLOv5n	640x640	28.0	45.7 ms	1.9M	4.5
YOLOv5s	640x640	37.4	56.8 ms	7.2M	16.5
YOLOv5m	640x640	45.4	64.1 ms	21.2M	49.0
YOLOv5l	640x640	49.0	67.3 ms	46.5M	109.1
YOLOv5x	640x640	50.7	68.9 ms	86.7M	205.7
YOLOv5n6	640x640	36.0	54.4 ms	3.2M	4.6
YOLOv5s6	1280x1280	44.8	63.7 ms	12.6M	16.8
YOLOv5m6	1280x1280	51.3	69.3 ms	35.7M	50.0
YOLOv5l6	1280x1280	53.7	71.3 ms	76.8M	111.4
YOLOv5x6	1280x1536	26.2/-	19.4/- ms	140.7/-	209.8/-

Tabla 3.10: Comparativa de especificaciones técnicas de las versiones de YOLOv5

- **Modelo:** Nombre del modelo de YOLOv5.
- **Size (pixels):** Tamaño de entrada de la imagen en píxeles.
- **mAP:** Precisión media (mean Average Precision) en el conjunto de validación, calculada con un umbral IoU del 50
- **Tiempo:** Tiempo de inferencia en milisegundos en una CPU V100.
- **Params:** Número de parámetros del modelo en millones.
- **FLOPs:** Operaciones de coma flotante por segundo (Floating Point Operations per Second) a una resolución 'Size', en miles de millones.

Comparativa con Competidores A diferencia de otros modelos como SSD (Single Shot MultiBox Detector) o sistemas basados en R-CNN (Region-based Convolutional Neural Networks), YOLOv5 ofrece un balance óptimo entre velocidad y precisión. Mientras que R-CNN y sus variantes, como Faster R-CNN, son altamente precisos, su proceso de detección es más lento debido al método de propuesta de regiones que utilizan. Por otro lado, SSD es más rápido que R-CNN pero tiende a ser menos preciso en objetos pequeños, un área donde YOLOv5 también muestra un rendimiento robusto

3.11.

Parámetros	YOLOv5	ResNet50 (FPN)	VGG16, MVGG16	MobNet2, InceV3
Epochs	1200	100	100	100
Tasa de Aprendizaje	0.0032	0.005	0.0001	0.0001
Optimizador	SGD	SGD	Adam	Adam
Anchor Sizes	Dinamic	32 to 512	8 to 512	8 to 512
Tiempo de Inferencia	0.009s	0.065s	0.112s	0.052s
Tiempo de Entrenamiento	12s	134s	80s	54s
Tamaño del Modelo (MB)	14.8	165.7	134.5	339.6

Tabla 3.11: Training parameters for YOLOv5 and Faster R-CNN

- **Epochs:** Número de iteraciones completas sobre el conjunto de datos durante el entrenamiento.
- **Tasa de Aprendizaje:** La velocidad a la que el modelo ajusta sus pesos durante el entrenamiento.
- **Optimizador:** Algoritmo utilizado para ajustar los pesos del modelo durante el entrenamiento.

- **Tamaños de Anclas:** Las dimensiones de las cajas delimitadoras (anclas) utilizadas en el modelo.
- **Tiempo de Inferencia:** El tiempo que tarda el modelo en hacer una predicción, medido en segundos.
- **Tiempo de Entrenamiento:** El tiempo necesario para entrenar el modelo por una época, medido en segundos.
- **Tamaño del Modelo (MB):** El tamaño del archivo del modelo entrenado, medido en megabytes.

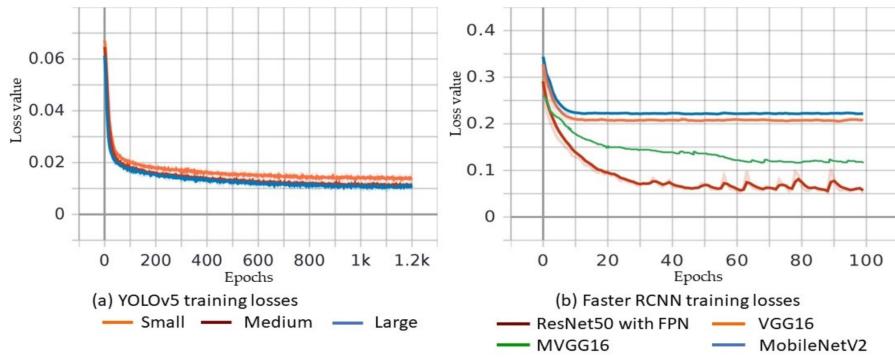


Figura 3.19: Comparativa de rendimiento YOLOv5 contra sus competidores.

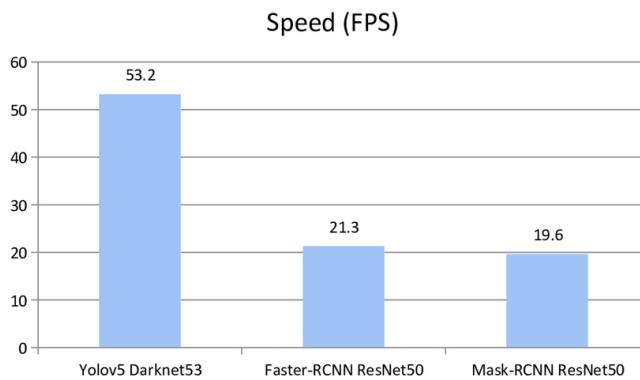


Figura 3.20: Comparativa de velocidad YOLOv5 contra sus competidores.

Esta decisión se basa en la necesidad de un sistema de visión por computador que no solo sea rápido y preciso, sino también adaptable a las condiciones variables y a menudo impredecibles que caracterizan los entornos interiores donde operará el dron (Fig 3.19 y 3.20) [29].

3.6. Seguridad y regulaciones para el vuelo de UAVs en interiores

La seguridad y el cumplimiento normativo son aspectos críticos en la operación de UAVs, especialmente en entornos interiores donde el riesgo de colisión y daño a personas y propiedades es mayor. Comprender el marco normativo actual y desarrollar propuestas para mejorar la seguridad es esencial para la implementación exitosa de estos sistemas.

3.6.1. Marco Normativo Actual y Desafíos de Seguridad

El vuelo de UAVs en interiores enfrenta varios desafíos de seguridad, como la proximidad a personas y objetos, el espacio confinado y la posibilidad de interferencias electromagnéticas. Aunque muchas regulaciones se enfocan en el vuelo al aire libre, algunas leyes y normativas también se aplican a operaciones interiores, especialmente en espacios públicos o comerciales.

Por ejemplo, en algunos países, el uso de drones en interiores de edificios comerciales requiere permisos especiales y debe cumplir con las normativas de seguridad laboral y protección civil. Además, en la Unión Europea, el Reglamento de Ejecución (UE) 2019/947 establece que cualquier operación de UAV, incluidas las interiores, debe cumplir con requisitos específicos de seguridad, como la identificación electrónica del dron y la autorización previa en ciertas categorías de operación. En los Estados Unidos, la FAA (Federal Aviation Administration) regula el uso de drones bajo la Parte 107, que incluye disposiciones aplicables para vuelos en interiores, como el requerimiento de que el piloto mantenga la línea de vista directa con el UAV en todo momento.

La falta de una regulación específica y uniforme para el vuelo de drones en interiores puede generar incertidumbre y riesgos adicionales, destacando la necesidad de un marco normativo claro y coherente.

3.6.2. Propuestas para Mejorar la Seguridad y Cumplimiento Normativo

Para mejorar la seguridad y asegurar el cumplimiento normativo en el vuelo de UAVs en interiores, proponemos la implementación de pruebas rigurosas y redundancias en los sistemas de control del dron. Estas medidas incluyen:

- **Pruebas Extensivas:** Antes de la operación en entornos reales, se deben realizar pruebas exhaustivas en entornos simulados y controlados.

dos. Esto permite identificar y corregir posibles fallos en el sistema sin poner en riesgo a personas o propiedades.

- **Redundancia en Sistemas Críticos:** Implementar redundancias en los sistemas de navegación y control para asegurar que el dron pueda mantener un vuelo seguro incluso en caso de fallos parciales. Esto incluye sistemas de detección y evasión de obstáculos, así como mecanismos de recuperación ante pérdida de señal.
- **Protocolos de Seguridad:** Desarrollar y seguir protocolos estrictos de seguridad que incluyan zonas restringidas, límites de velocidad y altitud, y procedimientos de emergencia. Estos protocolos deben ser comunicados claramente a todos los operadores y personal involucrado.
- **Capacitación de Operadores:** Asegurar que los operadores de UAVs estén adecuadamente capacitados en el uso seguro y eficiente de los drones, incluyendo la respuesta a situaciones de emergencia y la comprensión de las normativas aplicables.

La implementación de estas medidas no solo protege la seguridad de los individuos, sino que también asegura que las operaciones de UAVs en interiores cumplan con las regulaciones existentes, como el Reglamento de Ejecución (UE) 2019/947 en Europa o la Parte 107 de la FAA en los Estados Unidos, minimizando el riesgo de incidentes y mejorando la aceptación y viabilidad de estas tecnologías en aplicaciones comerciales e industriales.

3.7. Navegación autónoma y mapeo en interiores

La navegación autónoma de drones ha experimentado un avance significativo en los últimos años, impulsada por la miniaturización de sensores, el aumento de la potencia de computación y el desarrollo de algoritmos más sofisticados. En entornos interiores, donde los sistemas GPS no son confiables, la navegación autónoma y el mapeo se vuelven cruciales para que los drones realicen tareas como inspección y vigilancia.

3.7.1. Desafíos actuales en la navegación autónoma de UAVs en interiores

A pesar de los avances, la navegación autónoma de drones en interiores aún enfrenta diversos desafíos:

- **Sensores limitados:** Los drones pequeños generalmente tienen sensores de menor calidad, lo que dificulta la percepción precisa del entorno.

- **Baterías de corta duración:** Los drones pequeños suelen tener baterías de menor capacidad, lo que limita el tiempo de vuelo autónomo.
- **Algoritmos computacionalmente costosos:** Los algoritmos de mapeo y navegación avanzados pueden requerir mucha potencia de computación, lo que es un problema para los drones con recursos limitados.
- **Entornos dinámicos:** Los entornos interiores pueden cambiar con frecuencia, lo que dificulta que los drones creen mapas precisos y navegar de manera segura.

3.7.2. Tecnologías de comunicación para operaciones en interiores

Existen varias tecnologías de comunicación esenciales para la navegación de drones en interiores, cada una con sus propias ventajas y limitaciones. Estos sistemas de comunicación son fundamentales para garantizar un control preciso y seguro en entornos cerrados, donde la falta de señales GPS y otros desafíos pueden complicar la operación de los UAVs [30].

- **WiFi:** Comúnmente utilizado debido a su disponibilidad y facilidad de implementación. Es adecuado para entornos donde la cobertura WiFi es robusta, pero puede ser limitado por interferencias, alcance y mayor BW (Bandwidth o ancho de banda).
- **4G:** Ofrece mayor cobertura comparado con el WiFi, pero introduce un delay que puede afectar el control en tiempo real de los drones.
- **5G:** Promete resolver los problemas de latencia del 4G, ofreciendo una conexión ultrarrápida y de baja latencia, ideal para aplicaciones de control autónomo en tiempo real.
- **LoRa:** Ideal para comunicaciones a larga distancia con bajo consumo de energía, aunque con una menor velocidad de datos. Es útil para aplicaciones que requieren transmisiones ocasionales de datos de estado o comandos de control básico.
- **Bluetooth:** Utilizado para comunicaciones a corta distancia, es útil para configuraciones iniciales o para comunicaciones de backup en caso de fallos en otros sistemas.

3.8. Simulación de Vuelo

La simulación de vuelo de drones es una herramienta esencial para el desarrollo y prueba de algoritmos, permitiendo a los investigadores y desa-

rrolladores experimentar en un entorno controlado antes de realizar pruebas en el mundo real. Mientras que las plataformas de simulación ofrecen una variedad de características y niveles de flexibilidad que pueden influir en la elección de la herramienta adecuada para un proyecto específico, el uso de un dron físico conlleva una serie de desafíos adicionales. Al probar directamente con el dron real, se enfrentan problemas como la limitada duración de la batería, el desgaste de los componentes y la posibilidad de daños en el hardware debido a errores en el código o en la operación. Estos factores no solo aumentan los costos, sino que también limitan el tiempo disponible para realizar pruebas, ya que los drones requieren mantenimiento y recarga frecuente. En contraste, un simulador permite realizar pruebas extensivas sin estas limitaciones físicas, facilitando la iteración rápida y segura en el desarrollo de algoritmos.

3.8.1. Webots

Webots es un simulador de robótica que permite un control total sobre la simulación, ofreciendo soporte para múltiples lenguajes de programación como C, Java, Python, entre otros. Es gratuito y de código abierto, lo que facilita la modificación y personalización según las necesidades del usuario. Además, Webots cuenta con una gran comunidad que respalda su uso, proporcionando numerosos proyectos y ejemplos que pueden servir como base para nuevos desarrollos. La interoperabilidad de lenguajes de programación y su naturaleza completamente libre lo convierten en una herramienta versátil y poderosa para la simulación de drones[31].

3.8.2. Gazebo

Gazebo es otro simulador popular en el ámbito de la robótica, pero presenta algunas limitaciones en comparación con Webots. Aunque ofrece un entorno robusto para la simulación, su comunidad es menos extensa y proporciona menos posibilidades y libertad dentro del simulador. Gazebo no es completamente de código abierto, lo que puede restringir la capacidad de personalización. Además, no permite la misma interoperabilidad de lenguajes de programación que Webots, limitando así su flexibilidad en ciertos proyectos[32].

3.8.3. Microsoft Flight Simulator

Microsoft Flight Simulator no es de código abierto y está menos orientado a la simulación de drones en comparación con las otras plataformas mencionadas. Está diseñado principalmente para simular vuelos de avio-

nes comerciales, ofreciendo menos libertad para crear y modificar elementos dentro del simulador. La comunidad de desarrolladores enfocados en drones es menor y los proyectos disponibles son limitados. Esto hace que sea menos adecuado para proyectos de simulación de drones que requieren un alto grado de personalización y experimentación[33].

3.8.4. Comparativa de los simuladores

Al comparar Webots, Gazebo y Microsoft Flight Simulator, es evidente que Webots ofrece la mayor flexibilidad y soporte para la simulación de drones. Webots proporciona un control total sobre la simulación, soporte para múltiples lenguajes de programación, y una comunidad activa que facilita el desarrollo y la colaboración. Gazebo, aunque útil, carece de la misma libertad y flexibilidad, y Microsoft Flight Simulator está más orientado a la simulación de vuelos comerciales con menos opciones para la personalización de drones.

Por estas razones, hemos escogido Webots como la plataforma de simulación ideal para nuestro proyecto. Su capacidad para adaptarse a diversas necesidades y su fuerte respaldo comunitario lo hacen la opción más adecuada para el desarrollo y prueba de algoritmos de control autónomo de UAVs.

Capítulo 4

Herramientas Hardware y Software

En este capítulo, se detallarán las herramientas de hardware y software utilizadas en el desarrollo y ejecución del proyecto de navegación autónoma y mapeo de UAVs en interiores. La correcta elección y configuración de estos componentes es crucial para asegurar el éxito del sistema propuesto. Se abordarán tanto los dispositivos físicos (hardware) como las plataformas y algoritmos (software) que permiten la implementación y operación eficiente de los drones en entornos complejos.

4.1. Hardware

A continuación, se describirán las especificaciones de todos los componentes hardware utilizados en el proyecto.

4.1.1. Drone Tello

Como se ha mencionado previamente en el capítulo 2, en este proyecto se utilizará el drone DJI Tello EDU 4.1 de la marca Ryze Tech. Esta aeronave es un pequeño cuadricóptero (4 rotores) de fácil accesibilidad con grandes posibilidades educativas debido a la facilidad de ser programado. Puede ser programado en diferentes lenguajes como Scratch, Python, Swift, Go, C, Javascript y Matlab [34]. A pesar de no ser un drone con la última tecnología en sensores, es una alternativa muy recomendada por su bajo precio, su versatilidad y su tamaño, en comparación con otros modelos.

El DJI Tello EDU cuenta con un sistema de posicionamiento visual, un controlador de vuelo, un sistema de transmisión de video, un sistema de

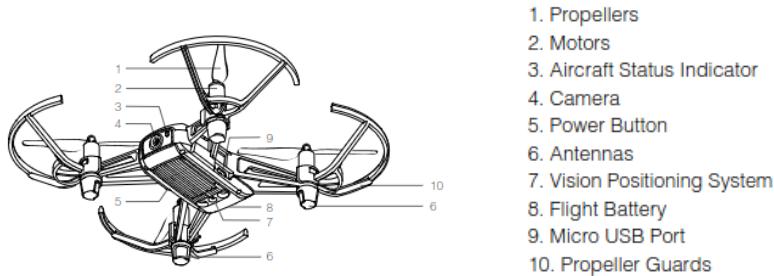


Figura 4.1: TelloEDU User Manual.

propulsión, una IMU y una batería de vuelo 6.3, con un SDK avanzado, el cual permite programar varios drones en forma de enjambre para que vuelen en conjunto y sintonía.

Destacado por su bajo costo y por contar con un SDK avanzado[35], se ha ganado la preferencia de la comunidad para una variedad de proyectos, desde educativos hasta de investigación. La ingeniería inversa que ha hecho la propia comunidad, del protocolo a bajo nivel del Tello ha permitido a los desarrolladores obtener un control más profundo del dron, abriendo la puerta a una personalización y adaptación sin precedentes para una gama de aplicaciones en interiores. Estas características han sido decisivas para elegir el Tello como plataforma principal en nuestro proyecto, buscando aprovechar su accesibilidad y el extenso soporte comunitario que respalda su uso.

4.2. Software

A continuación, se describirán las herramientas y bibliotecas utilizadas en la parte software del proyecto.

4.2.1. Protocolo de comunicación con el dron

El SDK proporcionado por los fabricantes incluye todo lo necesario para interactuar con el dron y realizar una amplia gama de funciones[35]. Sin embargo, este SDK introduce un retraso tanto en la transmisión de comandos al dron como en la recepción de los mismos, además de un delay en la transmisión del video que el dron envía, lo cual es inaceptable para situaciones que requieren una respuesta en tiempo real. Debido a estas limitaciones, la comunidad de desarrolladores ha colaborado de manera significativa para descifrar el protocolo de bajo nivel que utiliza el Tello para comunicarse con su aplicación nativa. Gracias a la ingeniería inversa, se han logrado identificar todos los parámetros y comandos en código hexadecimal, lo que permite

un control absoluto del dron y soluciona el problema del delay. Por ejemplo, es posible eliminar el límite de altura de 30 metros que tiene el dron, optimizando así su rendimiento y capacidad de respuesta en tiempo real.[15].

4.2.1.1. Descripción del protocolo de bajo nivel

Mucho esfuerzo colaborativo ha tenido lugar en sitios como el foro oficial para decodificar el protocolo de bajo nivel que el Tello usa para comunicarse con su aplicación nativa [36].

El Tello se comunica con su controlador vía Wi-Fi en el puerto predefinido 8889 utilizando mensajes UDP. La mayoría de estos mensajes consisten en un paquete estructurado de datos con el siguiente formato general:

Posición (B)	Contenido	Comentarios
0	Header	Siempre 0xCC
1-2	Packet Size	Tamaño total del paquete de 13 bits
3	CRC-8	CRC desde el encabezado hasta el tamaño del paquete
4	Packet Type Info	Bits: F—T—TYP—SUB - Ver abajo
5-6	Message ID	Little-endian - Ver abajo
7-8	Sequence No.	Little-endian - O 0 para algunos tipos, o ascendente para otros
9...	Payload	Opcional, varía según el tipo de paquete
End-1, End	CRC16	CRC desde el encabezado hasta el final del payload

Tabla 4.1: Tello General UDP Packet Structure

Estructura general del paquete UDP de Tello

Cada paquete de datos sigue un formato estructurado que consta de varios campos. A continuación, se describe cada uno de estos campos en detalle:

Posición 0 - Encabezado (Header)

- **Contenido:** Siempre tiene el valor 0xCC.

- Este es un valor fijo que indica el inicio de un paquete de datos.

Posiciones 1-2 - Tamaño del paquete (Packet Size)

- **Contenido:** Un valor de 13 bits que indica el tamaño total del paquete.
 - El campo de "Packet Size" se codifica en un formato *little-endian*:
 - Los primeros 8 bits (el primer byte) se almacenan de manera normal.
 - El segundo byte contiene 5 bits que forman parte del tamaño del paquete y 3 bits adicionales que se combinan con el siguiente campo (Información del Tipo de Paquete).
 - La fórmula de decodificación:

$$\text{size} = \text{buffer}[1] + ((\text{buffer}[2] \ll 8) \gg 3)$$

- Los 3 bits restantes del segundo byte en el campo "Packet Size" se utilizan como parte del siguiente campo (Packet Type Info), que especifica el tipo y subtipo del paquete. Estos bits no contribuyen al tamaño del paquete, sino que ayudan a clasificar y determinar cómo se debe procesar el contenido del paquete..

Posición 3 - CRC-8

- **Contenido:** Un valor CRC de 8 bits.
- Este valor se calcula desde el Encabezado hasta el Tamaño del Paquete, y se utiliza para verificar la integridad de estos campos.

Posición 4 - Información del tipo de paquete (Packet Type Info)

- **Formato de Bits:**
 - **Bit 0:** 1 si el paquete es *desde* el dron.
 - **Bit 1:** 1 si el paquete es *hacia* el dron.
 - **Bits 2-4:** 3 bits que indican el tipo de paquete.
 - **Bits 5-7:** 3 bits que indican el subtipo de paquete, generalmente 0.

Posiciones 5-6 - ID del Mensaje (Message ID)

- **Contenido:** Un identificador de mensaje codificado en formato *little-endian*.

Posiciones 7-8 - Número de Secuencia (Sequence No.)

- **Contenido:** Un número de secuencia también codificado en *little-endian*.

Posición 9 en adelante - Carga Útil (Payload)

- **Contenido:** Datos adicionales que varían según el tipo de paquete.

Posición Final y penúltima - CRC-16

- **Contenido:** Un valor CRC de 16 bits.

IDs de los mensajes de comunicación del dron

En el protocolo de comunicación del dron Tello, cada mensaje que se intercambia entre el dron y su controlador está identificado por un código único denominado **ID de mensaje**. Estos IDs están codificados en formato hexadecimal y se utilizan para indicar la naturaleza de cada mensaje, permitiendo que el dron y el controlador comprendan y procesen la información de manera adecuada.

Las tablas que se presentan a continuación enumeran los IDs de mensajes conocidos hasta la fecha, junto con la función específica que desempeñan en la comunicación con el dron. Además, se proporciona la dirección de la comunicación, que puede ser unidireccional hacia el dron (\rightarrow), desde el dron (\leftarrow), o bidireccional (\leftrightarrow). En algunos casos, también se incluyen comentarios que ofrecen detalles adicionales sobre el comportamiento o el uso del mensaje.

Estas tablas son esenciales para comprender cómo se configura y opera el dron Tello, facilitando el desarrollo de aplicaciones y scripts personalizados que interactúan directamente con el dron mediante comandos específicos. A medida que se descubren y documentan más IDs de mensajes, estas tablas se actualizan para reflejar el conocimiento más reciente sobre el protocolo de Tello.

Las tablas a continuación presentan una referencia rápida y organizada para los desarrolladores y entusiastas que trabajan con el protocolo del dron Tello 6.4 6.5.

4.2.2. Descripción y funcionalidades de la biblioteca Tellopy

Tellopy proporciona todas las funcionalidades básicas necesarias para controlar un dron Tello. A continuación, se presentan las principales funcionalidades disponibles, organizadas en las tablas 6.6 6.7.

Algunas de las funciones más importantes incluyen `connect()`, `quit()`, `log()`, `set_throttle()`, `set_roll()`, `set_pitch()` y `set_yaw()`. Estas funciones permiten establecer la conexión inicial con el dron, gestionar la salida de la aplicación, y controlar los movimientos fundamentales del dron en vuelo.

- `connect()`: Esta función inicia la conexión entre el controlador y el dron, permitiendo la comunicación y control del dispositivo.
- `quit()`: Utilizada para detener los hilos internos y finalizar la conexión de manera segura, garantizando que todos los procesos se cierren correctamente.
- `log(level)`: Permite ajustar el nivel de detalle de los mensajes de salida, lo que es útil para depuración y monitoreo de las operaciones del dron.
- `set_throttle(throttle)`: Controla el movimiento vertical del dron, permitiendo ascender o descender ajustando el valor de la aceleración.
- `set_roll(roll)`: Modifica la inclinación lateral del dron, es decir, su movimiento hacia la izquierda o derecha.
- `set_pitch(pitch)`: Gestiona la inclinación hacia adelante o hacia atrás del dron, controlando su desplazamiento horizontal.
- `set_yaw(yaw)`: Controla la rotación del dron sobre su eje vertical, permitiendo que gire a la izquierda o derecha.

Estas funciones son esenciales para la operación básica del dron, permitiendo tanto la conexión y gestión del sistema, como el control preciso del vuelo en todas las direcciones y orientaciones.

4.2.2.1. Eventos y Estados

La tabla 6.8 describe los eventos y estados que se pueden utilizar con Tellopy para monitorear y controlar el dron, es fundamental comprenderlos para una operación eficiente. Entre los eventos más importantes se encuentran:

- **EVENT_CONNECTED:** Indica que la conexión con el dron ha sido establecida exitosamente, permitiendo el inicio de las operaciones.
- **EVENT_WIFI:** Proporciona información sobre la señal WiFi, lo cual es crucial para asegurar una comunicación estable y sin interrupciones entre el controlador y el dron.
- **EVENT_FLIGHT_DATA:** Este evento actualiza datos vitales del vuelo como la velocidad, nivel de batería y altitud del dron, esenciales para monitorizar y controlar en tiempo real el estado del vuelo.
- **EVENT_LOG_DATA:** Recoge y procesa los datos de registro de los sensores del dron, como el IMU y el giroscopio, ofreciendo una visión detallada de todos los eventos internos y condiciones operativas del dron.

Estos eventos son críticos para mantener el control y asegurar la seguridad durante el vuelo, proporcionando la información necesaria para tomar decisiones en tiempo real y reaccionar ante cualquier cambio en las condiciones del dron.

4.2.2.2. FlightData

La clase FlightData captura todos los registros relacionados con el estado del dron durante el vuelo. En la tabla 6.9 se presentan las principales variables de esta clase, los registros juegan un papel crucial para garantizar la seguridad y eficiencia del vuelo. Entre los registros más importantes se encuentran:

- **battery_percentage:** Proporciona el porcentaje de batería restante, lo cual es vital para planificar la duración del vuelo y evitar emergencias debido a falta de energía.
- **em_open y em_sky:** Estos indicadores de emergencia son fundamentales para detectar situaciones críticas.
- **fly_speed y ground_speed:** Miden la velocidad de vuelo y la velocidad sobre el suelo, respectivamente, proporcionando datos esenciales para el control de la trayectoria y la estabilidad del dron durante el vuelo.
- **imu_state:** Monitorea el estado del IMU (Unidad de Medición Inercial), que incluye sensores como acelerómetros y giroscopios. Este sensor es clave para mantener la orientación y estabilidad del dron en vuelo.

4.2.2.3. Clase LogNewMvoFeedback

La clase LogNewMvoFeedback captura todos los registros relacionados con la velocidad y posicionamiento del dron durante el vuelo. En la tabla 4.2 se presentan las principales variables de esta clase:

Variable	Descripción
log	Registro de eventos.
count	Contador de eventos.
vel_x	Velocidad en el eje X.
vel_y	Velocidad en el eje Y.
vel_z	Velocidad en el eje Z.
pos_x	Posición en el eje X.
pos_y	Posición en el eje Y.
pos_z	Posición en el eje Z.

Tabla 4.2: Variables de la clase LogNewMvoFeedback

4.2.2.4. Clase LogImuAtti

La clase LogImuAtti captura todos los registros relacionados con la IMU del dron durante el vuelo. En la tabla 4.3 presentan las principales variables de esta clase:

Variable	Descripción
log	Registro de eventos.
count	Contador de eventos.
acc_x	Aceleración en el eje X.
acc_y	Aceleración en el eje Y.
acc_z	Aceleración en el eje Z.
gyro_x	Velocidad angular en el eje X.
gyro_y	Velocidad angular en el eje Y.
gyro_z	Velocidad angular en el eje Z.
q0	Componente q0 del cuaternión.
q1	Componente q1 del cuaternión.
q2	Componente q2 del cuaternión.
q3	Componente q3 del cuaternión.
vg_x	Velocidad del vector g en el eje X.
vg_y	Velocidad del vector g en el eje Y.
vg_z	Velocidad del vector g en el eje Z.

Tabla 4.3: Variables de la clase LogImuAtti

4.2.3. ARUCO

Como se ha explicado en el capítulo 3.5.2.1, la biblioteca ARUCO será la elegida en nuestro proyecto para la detección de marcadores fiduciales en nuestras aplicaciones de visión artificial. ARUCO es una herramienta que permite identificar estos marcadores, los cuales son cuadrados sintéticos con un borde negro ancho y una matriz interna binaria que determina su identificador (ID). Esta elección se basa en su eficacia y precisión en este tipo de tareas.

4.2.3.1. Uso de marcas Aruco para la navegación y mapeo

Para el sistema de posicionamiento se van a utilizar los vectores `rvecs` y `tvecs`, que son los vectores de rotación y de traslación respectivamente para cada uno de los marcadores detectados en `corners` 3.5.2.5.

El vector `rvec` proporciona el ángulo de cada etiqueta, es decir, `pitch`, `roll` y `yaw` de cada etiqueta, mientras que el vector `tvec` proporciona la posición en los ejes `x`, `y` y `z` de la etiqueta, utilizando las técnicas de traslación 3D a 2D anteriormente descritas.

Con estos parámetros se podrá situar el dron, solamente con una cámara en la posición donde se desee.

4.2.4. YOLOv5

YOLOv5 (You Only Look Once, versión 5) es un modelo avanzado de detección de objetos que ha sido ampliamente adoptado por su precisión y eficiencia. Los motivos por los que ha sido escogido se encuentran detallados en el capítulo 3.5.3.5.

4.2.4.1. Descripción y Funcionalidades

YOLOv5 es un modelo que se puede entrenar para la detección de objetos en imágenes y videos (Fig 4.2). Algunas de sus principales funcionalidades incluyen:

- **Detección en tiempo real:** Capaz de procesar imágenes y videos en tiempo real, ofreciendo detecciones rápidas y precisas.
- **Entrenamiento personalizado:** Permite entrenar el modelo con conjuntos de datos personalizados para detectar objetos específicos según las necesidades del usuario.

- **Eficiencia computacional:** Optimizado para funcionar en una amplia gama de dispositivos, desde potentes servidores hasta dispositivos móviles con recursos limitados.
- **Soporte para múltiples clases:** Puede detectar y clasificar múltiples objetos diferentes en una sola imagen.
- **Implementación sencilla:** Dispone de herramientas y bibliotecas que facilitan su integración en diferentes aplicaciones y sistemas.
- **Flexibilidad en el ajuste de parámetros:** Los usuarios pueden ajustar diversos hiperparámetros para optimizar el rendimiento del modelo según sus necesidades específicas.



Figura 4.2: Yolov5 Checkout

4.2.5. Webots

Webots es un simulador avanzado para la modelación y simulación de robots y UAVs (Unmanned Aerial Vehicles). Es open-source, lo que proporciona un control total sobre la simulación y ofrece un gran soporte comunitario. Los motivos por los que ha sido elegido se encuentran detallados en el capítulo 3.8 [31].

4.2.5.1. Características y ventajas para simulación de UAVs

Una de las grandes ventajas de Webots es su gran capacidad para establecer todo tipo de parámetros en una simulación, haciendo posible crear escenarios lo más realistas posibles. Además, cuenta con una amplia variedad de proyectos aportados por la comunidad, como drones y robots, que se asemejan mucho a la realidad. Gracias a esto, se puede acelerar el proceso Simulación - Realidad.

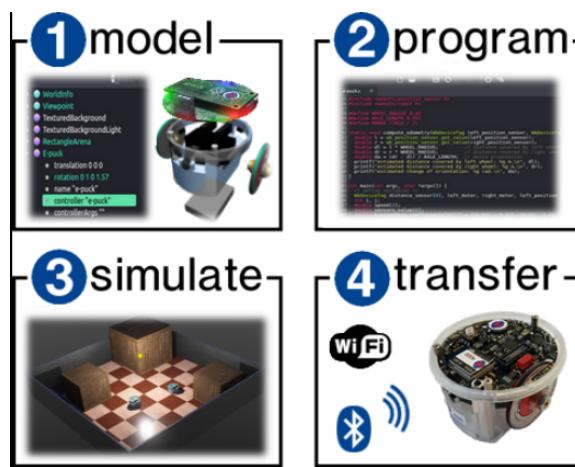


Figura 4.3: Webots Transfer

4.2.5.2. Proceso de simulación de entornos y pruebas

Webots ofrece una gran cantidad de entornos precreados tanto por la comunidad como por los desarrolladores del software. Esto permite que personas sin conocimientos profundos en la configuración de parámetros de simulación puedan simular proyectos que ya están desarrollando en la realidad, como es el caso de este proyecto.

4.2.5.3. Lenguajes de Programación

Los controladores de Webots pueden escribirse en C/C++, Java, Python o MATLAB. A pesar de sus diferencias sintácticas, todos estos lenguajes comparten la misma implementación de bajo nivel. Siempre que la secuencia de llamadas a funciones/métodos no varíe, cada lenguaje de programación producirá exactamente los mismos resultados de simulación. Por lo tanto, los conceptos explicados aquí con ejemplos en C también se aplican a C++, Java, Python y MATLAB.

4.2.5.4. Entornos de desarrollo

Webots trae un IDE (Integrated Development Environment) entorno de desarrollo integrado o entornos de terceros para desarrollar controladores de Webots, lo que da juego a la imaginación, entre los cuales se pueden destacar.

- **Webots Built-in Editor:** Uso del editor integrado de Webots.
- **The standard File Hierarchy of a Project:** Jerarquía estándar de archivos de un proyecto.
- **Compiling Controllers in a Terminal:** Compilación de controladores en una terminal.
- **Using Webots Makefiles:** Uso de Makefiles de Webots.
- **Debugging C/C++ Controllers:** Depuración de controladores en C/C++.
- **Starting Webots Remotely:** Iniciando Webots remotamente a través (ssh).
- **Running Extern Robot Controllers:** Ejecutando controladores externos de robots.
- **Transfer to Your Own Robot:** Transferencia a su propio robot.

Capítulo 5

Desarrollo del sistema

5.1. Descripción general del sistema

Este sistema de vigilancia está diseñado para monitorizar edificios utilizando un dron autónomo. El dron se comunica a través de una conexión Wi-Fi con un ordenador central, el cual recibe y controla en tiempo real las imágenes capturadas por el dron.

Para la localización y navegación, el dron emplea el sistema de marcadores ARUCO, que le permite orientarse y seguir rutas predefinidas dentro del área de vigilancia. Además, incorpora el modelo YOLOv5 para el reconocimiento de objetos, lo que le permite identificar y clasificar diferentes tipos de objetos en su campo de visión, como personas, vehículos y animales.

Cuando el dron detecta alguno de estos objetos, automáticamente captura una imagen y la envía a un grupo de Telegram predeterminado, alertando al equipo de seguridad o al responsable del sistema. Esto garantiza una vigilancia eficaz y permite la rápida intervención en caso de detectar actividades inusuales (Fig 5.1).

5.1.1. Soluciones propuestas para la navegación y mapeo avanzado

Para abordar estos desafíos, este proyecto propone una solución para la navegación autónoma y mapeo en interiores que utiliza marcas fiduciales (ARUCO), YOLOv5[37][28]. Esta combinación ofrece las siguientes ventajas:

- **Marcas fiduciales:** La utilización de marcas fiduciales, como ARUCO, permite a los drones determinar su posición y orientación con precisión utilizando solo una cámara. Esto reduce la necesidad de sensores costosos y computacionalmente intensivos, además, Aruco per-

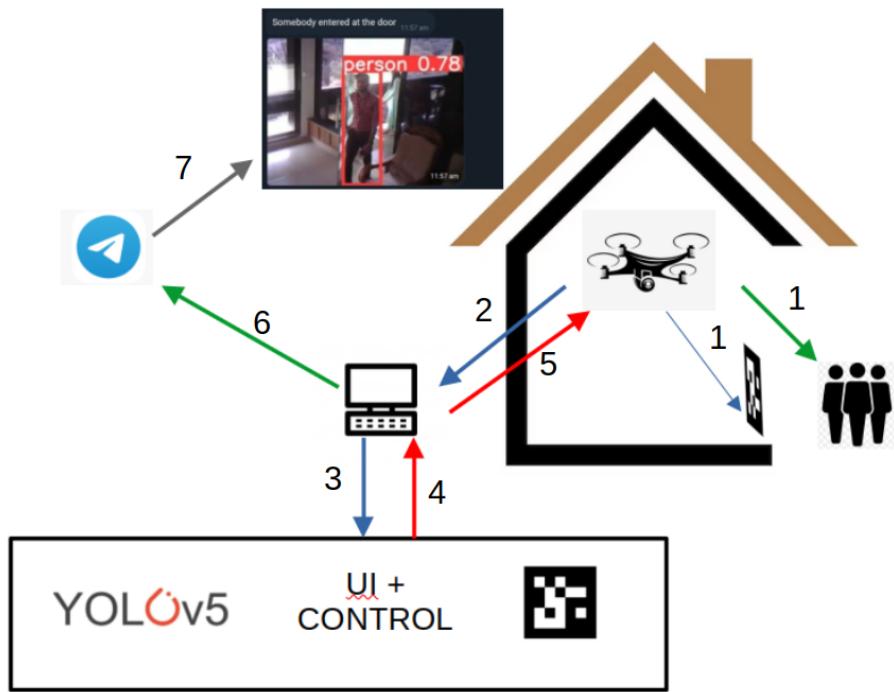


Figura 5.1: Descripción del Sistema

mite una localización precisa y eficiente del dron, incluso con una sola cámara.

- **Algoritmos ligeros:** El desarrollo de algoritmos de mapeo y navegación más ligeros y eficientes computacionalmente permite su implementación en drones con recursos limitados.
- **Reconocimiento de objetos:** YOLOv5, un algoritmo de reconocimiento de objetos liviano y eficiente, se puede utilizar para detectar y rastrear objetos de interés en el entorno.

5.1.1.1. Ventajas de la propuesta

La propuesta de este proyecto tiene las siguientes ventajas:

- **Bajo costo:** La utilización de marcas fiduciales y algoritmos ligeros reduce la necesidad de hardware y software costosos.
- **Robustez:** La combinación de diferentes tecnologías aumenta la robustez de la navegación y el mapeo en entornos dinámicos.

- **Escalabilidad:** La solución se puede adaptar a diferentes drones y entornos.

La navegación autónoma y el mapeo son cruciales para las aplicaciones de drones en interiores. Este proyecto propone una solución viable y escalable que utiliza marcas fiduciales y YOLOv5 para hacer la navegación en interiores posible.

5.1.2. Retos a resolver en base a las herramientas

En esta sección se describen los principales retos que se deben enfrentar y cómo cada herramienta clave contribuye a la resolución del problema:

- **Cámara:** Capturar imágenes del entorno interior para detectar y localizar etiquetas Aruco, que son fundamentales para la navegación precisa del dron en ausencia de GPS.
- **Datos de vuelo del dron:** Obtener información en tiempo real sobre la posición, velocidad y orientación del dron. Estos datos se integran en el sistema de navegación para ajustar continuamente la ruta del dron y asegurar su estabilidad y precisión.
- **Uso de marcas para localización:** Utilizar estas etiquetas o marcas para el mapeo preciso del entorno. Las etiquetas sirven como puntos de referencia que ayudan al dron a determinar su ubicación exacta dentro del espacio interior.
- **Integración en tiempo real para la detección de objetos:** Este modelo de detección de objetos permite identificar y clasificar diversos objetos en el entorno. Esto mejora la capacidad del dron para evitar obstáculos y reconocer elementos específicos, lo cual es crucial para la navegación autónoma.
- **Uso de simuladores:** Aprovechar las capacidades de este simulador open-source para crear y probar entornos virtuales detallados. Esto permite ajustar y optimizar el sistema en un entorno controlado antes de implementarlo en el mundo real, reduciendo riesgos y costos.
- **Algoritmos de posicionamiento y mapeo:** Implementar algoritmos avanzados para crear y actualizar el mapa del entorno. Estos algoritmos aseguran que el dron pueda navegar de manera autónoma y precisa, incluso en entornos dinámicos y cambiantes.
- **Interfaz de usuario:** Desarrollar una interfaz que permita a los operadores monitorizar el estado del dron, visualizar el mapa generado y enviar comandos de navegación de manera intuitiva. Esto facilita el control y supervisión del sistema, mejorando la experiencia del usuario.

5.2. Preparación y Configuración previa a la implementación

A continuación, se procederá a la configuración e instalación de las bibliotecas necesarias para el correcto funcionamiento del dron, así como para todas las herramientas previamente mencionadas que se utilizarán en el desarrollo y operación del proyecto. Este proceso asegurará que todos los componentes estén preparados y optimizados para la implementación, garantizando una integración fluida y eficiente del sistema.

5.2.1. Configuraciones Previas

Para utilizar la biblioteca `tellopy`, es necesario instalar la versión 4.2.0.* de OpenCV, que es la versión compatible. Dependiendo del sistema operativo y de la versión de Python que tengas instalado, deberás seleccionar una versión específica dentro de ese rango. Por ejemplo, en el caso de Linux Mint Ubuntu 22.04, la versión adecuada es `opencv-python 4.2.0.34`, aunque esto puede variar según los detalles mencionados anteriormente.

Además, para usar Aruco es necesario calibrar la cámara. La calibración se realiza utilizando un tablero de ajedrez, lo cual permite obtener los parámetros intrínsecos y extrínsecos de la cámara. La calibración es crucial para corregir las distorsiones de la lente y asegurar que las medidas obtenidas sean precisas[38].

- **Matriz de Calibración:** Durante la calibración de la cámara, se genera una matriz de calibración que incluye parámetros intrínsecos como la distancia focal (f_x, f_y) y el punto principal (c_x, c_y). Esta matriz se calcula mediante la captura de varias imágenes del tablero de ajedrez desde diferentes ángulos y posiciones 5.2.

- **Cálculo de la Matriz:** Utilizando las imágenes capturadas del tablero, se detectan los puntos de las esquinas del tablero. Estos puntos se utilizan para resolver un sistema de ecuaciones que permite calcular los parámetros intrínsecos y extrínsecos de la cámara, resultando en una matriz de calibración precisa. La calibración también ayuda a determinar los coeficientes de distorsión que corrigen las deformaciones causadas por la lente.

Para que la calibración tenga un error aceptable se deben hacer alrededor de 30 fotos en diferentes angulos y distancias.

5.2.2. Mediciones de Error y Análisis

La calibración de la cámara genera errores que deben ser evaluados para asegurar la precisión del sistema. A continuación, se explican los errores que

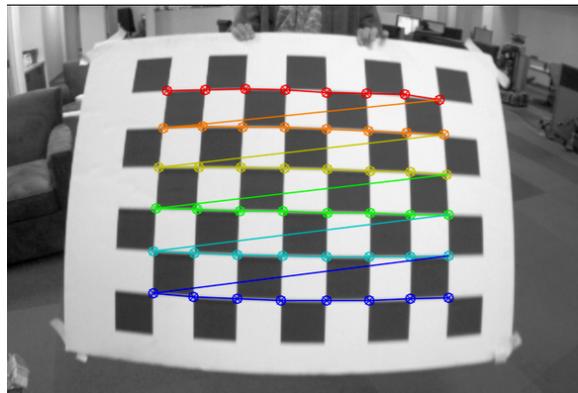


Figura 5.2: Proceso de Calibración de Cámara

mas importan:

- **Error de reproyección:** La reproyección es el proceso de proyectar puntos en el espacio tridimensional (3D) a un plano bidimensional (2D), como una imagen o una fotografía. En el contexto de visión por computadora y calibración de cámaras, la reproyección se refiere a cómo se mapearían las coordenadas de un punto 3D, dado un modelo de cámara, a las coordenadas 2D en la imagen capturada por esa cámara. Este error mide la discrepancia entre las coordenadas proyectadas de los puntos del tablero en la imagen y las coordenadas reales detectadas. Se calcula como la media de las distancias entre estos puntos en todas las imágenes utilizadas para la calibración. La fórmula para el error de reproyección es 5.1:

$$\text{Error de reproyección} = \frac{1}{N} \sum_{i=1}^N \sqrt{(x_i - x'_i)^2 + (y_i - y'_i)^2} \quad (5.1)$$

N representa el número total de puntos de datos considerados en el cálculo del error de reproyección. Es decir, es la cantidad de puntos de referencia para los cuales se está calculando la diferencia entre las coordenadas proyectadas (x'_i, y'_i) y las coordenadas originales (x_i, y_i) . Este promedio da una idea del error global de reproyección para el conjunto de datos considerado.

- **Error de distorsión:** La distorsión de la lente puede causar que las líneas rectas aparezcan curvas en la imagen. Los coeficientes de distorsión se calculan para corregir estos efectos. El error de distorsión evalúa la eficacia de estos coeficientes para corregir la imagen.

- **Error intrínseco y extrínseco:** Los parámetros intrínsecos (como la distancia focal y el punto principal) y los parámetros extrínsecos (como la rotación y la translación de la cámara) también pueden tener errores que afectan la precisión de la calibración. Estos errores se evalúan comparando

los parámetros calculados con los valores teóricos o esperados.

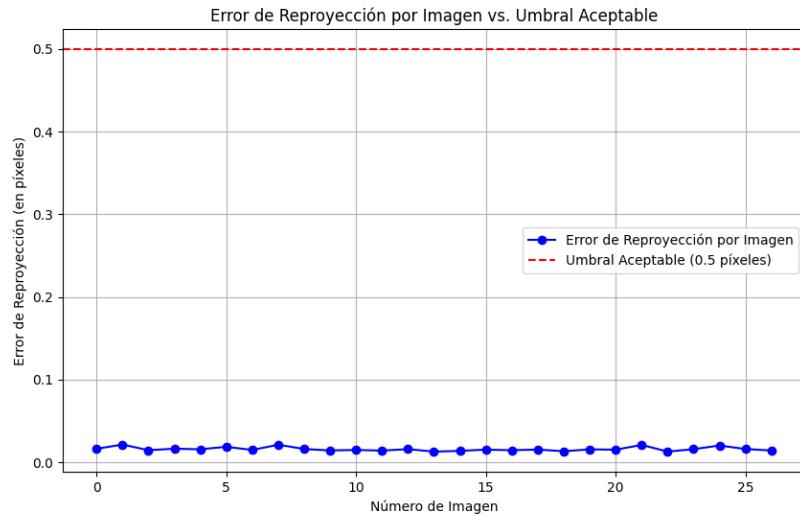


Figura 5.3: Error de Reproyección

Un error de reproyección de hasta 0.5 píxeles se considera generalmente bueno porque implica que la discrepancia promedio entre las coordenadas proyectadas y las coordenadas reales es menor que la mitad de un píxel. Dado que la resolución de las cámaras modernas es alta y los píxeles son muy pequeños, un error de menos de 0.5 píxeles es prácticamente insignificante y asegura una alta precisión en la calibración. Esto es crucial para aplicaciones que requieren medidas exactas, como la detección de marcadores Aruco en visión por computador 5.3.

Para todas las demás herramientas o bibliotecas no se necesita configuración inicial.

5.3. Implementación de ARUCO para navegación autónoma

Los marcadores ARUCO son una herramienta invaluable para el posicionamiento y mapeo en entornos donde el GPS no es accesible o donde sistemas como LIDAR (Light Detection and Ranging) no pueden ser utilizados. ARUCO proporciona una solución más económica y efectiva para estas aplicaciones, facilitando la navegación autónoma en interiores y otros espacios complejos.

5.3.1. Configuración de ARUCO

Para detectar los marcadores ARUCO, es fundamental configurar adecuadamente los parámetros de detección 5.1. A continuación, se explican las funciones utilizadas y su propósito:

Listing 5.1: Detección de Marcadores ARUCO

```

1 def detect_ArUco(img):
2     Detected_ArUco_markers = {}
3     gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
4     aruco_dict = aruco.Dictionary_get(aruco.DICT_4X4_250)
5     parameters = aruco.DetectorParameters_create()
6
7     corners, ids, _ = aruco.detectMarkers(gray, aruco_dict, parameters
8                                         =parameters)
9
10    if ids is not None:
11        for i in range(len(ids)):
12            Detected_ArUco_markers[str(ids[i][0])] = corners[i]
13
14    return Detected_ArUco_markers

```

Esta función convierte la imagen a escala de grises y utiliza un diccionario de marcadores ARUCO (`aruco.DICT_4X4_250`) y parámetros de detección predeterminados. La función `detectMarkers` 5.2 detecta los marcadores en la imagen y devuelve sus esquinas y IDs. Los marcadores detectados se almacenan en un diccionario.

Listing 5.2: Cálculo de la Orientación en Grados

```

1 def Calculate_orientation_in_degree(Detected_ArUco_markers,
2                                     camera_matrix, dist_coeffs):
3     ArUco_marker_orientations = {}
4     for aruco_id, corners in Detected_ArUco_markers.items():
5         rvec, tvec, _ = aruco.estimatePoseSingleMarkers(corners, SIZE,
6                                                       camera_matrix, dist_coeffs)
7         rmat = cv.Rodrigues(rvec)[0]
8         sy = np.sqrt(rmat[0,0] * rmat[0,0] + rmat[1,0] * rmat[1,0])
9         singular = sy < 1e-6
10        if not singular:
11            pitch = np.arctan2(rmat[2,1], rmat[2,2])
12            yaw = np.arctan2(-rmat[2,0], sy)
13            roll = np.arctan2(rmat[1,0], rmat[0,0])
14        else:
15            pitch = np.arctan2(-rmat[1,2], rmat[1,1])
16            yaw = np.arctan2(-rmat[2,0], sy)
17            roll = 0
18        pitch_deg = np.degrees(pitch)
19        yaw_deg = np.degrees(yaw)
20        roll_deg = np.degrees(roll)
21        pitch_deg = (pitch_deg + 180) % 360 - 180
22        yaw_deg = (yaw_deg + 180) % 360 - 180
23        roll_deg = (roll_deg + 180) % 360 - 180
24
25        ArUco_marker_orientations[aruco_id] = {
26            'yaw': yaw_deg,
27            'pitch': pitch_deg,
28            'roll': roll_deg
29        }
30
31    return ArUco_marker_orientations

```

```

25     'pitch': pitch_deg,
26     'roll': roll_deg,
27     'rvec': rvec[0],
28     'tvec': tvec[0]
29 }
30 return ArUco_marker_orientations

```

- **Entrada:** Recibe los marcadores detectados, la matriz de la cámara y los coeficientes de distorsión.
- **Pose del marcador:** Utiliza `aruco.estimatePoseSingleMarkers` para obtener los vectores de rotación (`rvec`) y traslación (`tvec`) del marcador.
- **Conversión:** Convierte `rvec` en una matriz de rotación (`rmat`) utilizando `cv.Rodrigues`.
- **Cálculo de orientación:**
 - Calcula el factor `sy` para determinar si la matriz de rotación es singular.
 - Si no es singular, calcula los ángulos de yaw, pitch y roll usando funciones trigonométricas (`np.arctan2`).
 - Si es singular, ajusta los cálculos para manejar esta condición.
 - Convierte los ángulos de radianes a grados y los ajusta al rango $[-180, 180]$ grados.
- **Almacenamiento de resultados:** Guarda los ángulos (yaw, pitch, roll) y los vectores (`rvec` y `tvec`) en un diccionario para cada marcador.
- **Retorno:** Devuelve un diccionario que contiene las orientaciones y los vectores de todos los marcadores detectados.

La sección dedicada a los experimentos para el estudio de los errores asociados a ARUCO se encuentra en el capítulo 6.1

5.4. Implementación en el simulador

Se procederá a explicar cómo configurar el entorno y el robot en el simulador Webots para replicar condiciones reales de operación. Se utilizará este simulador debido a las limitaciones inherentes al uso del dron real, tales como la duración limitada de las baterías y otros problemas que ya se han mencionado anteriormente.

Es necesario implementar un dron en el simulador que cuente con un sistema de movimiento similar al del dron Tello, cuyas características ya se han descrito previamente. Además, se recreará un mapa detallado de un edificio en el simulador. Este entorno virtual permitirá probar y evaluar los algoritmos de navegación y control de manera eficiente antes de llevarlos al dron físico.

5.4.1. Creación de entornos y robots simulados que replican condiciones reales

Para realizar una simulación efectiva, se escogió un entorno realista disponible en Open Sample World, concretamente el archivo `complete_apartment.wbt` (Fig 5.10). Este entorno proporciona un espacio cerrado adecuado para la simulación de vuelo en interiores (Fig 5.11).

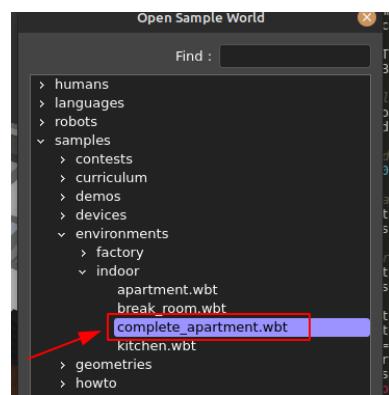


Figura 5.4: Instrucciones para seleccionar el entorno



Figura 5.5: Mapa Simulado

Para el robot, inicialmente se seleccionó la versión del DJI Mavic, disponible en los ejemplos de robots de Webots. Este modelo permite configurar el dron para que se mueva de manera similar a como lo hace la biblioteca `tellopy`. Sin embargo, se encontró un problema en la programación del robot que solo permite volar en espacios abiertos, lo cual no era adecuado para la simulación en interiores.

Por lo tanto, se buscó un dron alternativo y se encontró el modelo Crazyflie, las instrucciones para importar el robot en nuestro entorno se pueden ver en la figuras 5.12 y 5.13, para añadirle los scripts de control de vuelo (Fig 5.14), además este permite volar en entornos cerrados y se puede configurar para que se mueva como lo hace la biblioteca `tellopy`, como se describirá en el siguiente apartado.

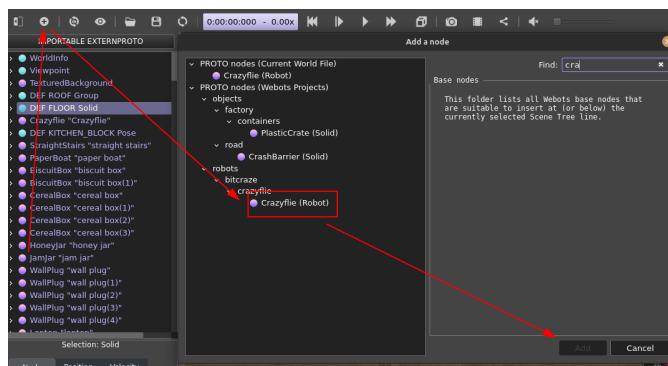


Figura 5.6: Intrucciones para importar el robot

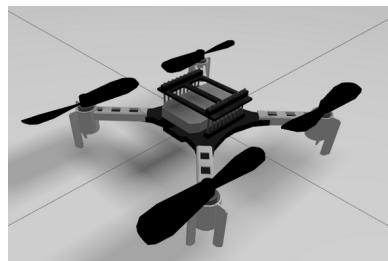


Figura 5.7: Robot en Webots

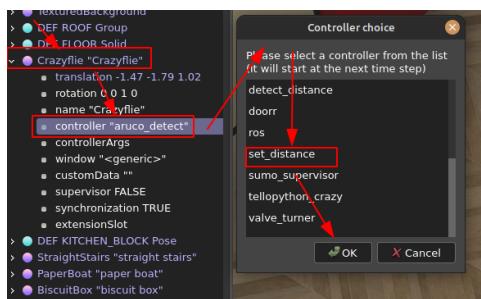


Figura 5.8: Instrucciones para importar el script de control

5.4.2. Implementación de API de Tello en webots

Para asegurar que el dron simulado en Webots se comporte de manera similar al dron real controlado por la API de `tellopy`, se ha desarrollado una API específica que garantiza la similitud en el control del dron en el simulador. Cabe destacar que no se ha comprobado que las velocidades del dron real y del dron simulado sean exactamente iguales; únicamente se ha asegurado que el control de las mismas se realice de manera similar. Esta API permite replicar con precisión las funciones de movimiento en todas las direcciones, tal como se haría con el dron físico.

A continuación, se detallan las funciones de `tellopy` 5.3 utilizadas para el control del dron real y su correspondiente implementación en Webots:

Listing 5.3: Funciones de Movimiento en Tellopy

```

1 def set_throttle(self, throttle):
2     """
3         Set_throttle controls the vertical up and down motion of the drone
4
5             Pass in an int from -1.0 ~ 1.0. (positive value means upward)
6
7             if self.left_y != self._fix_range(throttle):
8                 log.info('set_throttle(val=%4.2f)' % throttle)
9                 self.left_y = self._fix_range(throttle)

```

```

10 def set_yaw(self, yaw):
11     """
12         Set_yaw controls the left and right rotation of the drone.
13         Pass in an int from -1.0 ~ 1.0. (positive value will make the
14             drone turn to the right)
15     """
16     if self.left_x != self.__fix_range(yaw):
17         log.info('set_yaw(val=%4.2f)' % yaw)
18         self.left_x = self.__fix_range(yaw)
19
20 def set_pitch(self, pitch):
21     """
22         Set_pitch controls the forward and backward tilt of the drone.
23         Pass in an int from -1.0 ~ 1.0. (positive value will make the
24             drone move forward)
25     """
26     if self.right_y != self.__fix_range(pitch):
27         log.info('set_pitch(val=%4.2f)' % pitch)
28         self.right_y = self.__fix_range(pitch)
29
30 def set_roll(self, roll):
31     """
32         Set_roll controls the side to side tilt of the drone.
33         Pass in an int from -1.0 ~ 1.0. (positive value will make the
34             drone move to the right)
35     """
36     if self.right_x != self.__fix_range(roll):
37         log.info('set_roll(val=%4.2f)' % roll)
38         self.right_x = self.__fix_range(roll)

```

Estas funciones de `tellopy` controlan el movimiento vertical (`set_throttle`), la rotación (`set_yaw`), el movimiento hacia adelante y hacia atrás (`set_pitch`), y el movimiento lateral (`set_roll`). La implementación de estas funciones en Webots 5.4 se realiza de la siguiente manera:

Listing 5.4: Implementación de Funciones de Movimiento en Webots

```

1 # Ajustar valor dentro del rango -1.0 a 1.0
2 def fix_range(value):
3     return max(min(value, 1.0), -1.0)
4
5 def set_throttle(value):
6     global height_diff_desired, height_desired
7     height_diff_desired = fix_range(value)
8     height_desired += height_diff_desired * dt
9
10 def set_yaw(value):
11     global yaw_desired
12     yaw_desired = fix_range(value)
13
14 def set_pitch(value):
15     global forward_desired
16     forward_desired = fix_range(value)
17
18 def set_roll(value):
19     global sideways_desired
20     sideways_desired = fix_range(value)

```

Comentarios sobre las funciones:

- **fix_range(value)**: Esta función se asegura de que el valor de entrada esté dentro del rango permitido de -1.0 a 1.0. Este ajuste es fundamental para evitar que se pasen valores que puedan ser interpretados incorrectamente por el simulador o el dron real.
- **set_throttle(value)**: Controla el movimiento vertical del dron (subida y bajada). En Webots, además de ajustar el valor dentro del rango permitido, se ha modificado la función para que afecte directamente a **height_diff_desired**, que luego se utiliza para actualizar **height_desired** en función del tiempo (**dt**). Este cambio es necesario para simular de manera continua el cambio de altura en el dron, lo cual es una diferencia clave respecto al control discreto en el dron real.
- **set_yaw(value)**: Controla la rotación horizontal del dron (giro hacia la izquierda o derecha). En el simulador, el valor ajustado se asigna a **yaw_desired**, que probablemente es utilizado en otro lugar del código para aplicar la rotación. Este cambio refleja la necesidad de mantener el control de la orientación del dron en un contexto simulado.
- **set_pitch(value)**: Controla la inclinación hacia adelante o hacia atrás del dron, determinando su movimiento en el eje longitudinal. En Webots, se asigna a **forward_desired**, que luego se traduce en movimiento hacia adelante o hacia atrás. Este cambio asegura que el comportamiento simulado de movimiento hacia adelante o atrás sea continuo y controlado.
- **set_roll(value)**: Controla la inclinación lateral del dron (movimiento hacia la izquierda o derecha). El valor ajustado se asigna a **sideways_desired** en el simulador, lo que permite que el dron simulado se desplace lateralmente de manera controlada y fluida, al igual que en el dron real.

Al configurar estas funciones en Webots, se logra que el dron simulado tenga las mismas órdenes de movimiento y se mueva de la misma manera que el dron real controlado con **tellopy**. Sin embargo, es importante recalcar que esta implementación no garantiza que las velocidades sean idénticas entre el dron simulado y el dron real, sino que se centra en replicar el control de movimiento de manera similar, permitiendo realizar pruebas realistas en el entorno simulado. El código se puede ver en el GitHub.

5.5. Configuración de Yolov5

En esta sección, se explicará cómo configurar Yolov5 para la detección de objetos y cómo integrar el sistema de detección con la biblioteca Aruco utilizando OpenCV[28].

Yolov5 es un modelo de detección de objetos desarrollado por Ultralytics que se puede utilizar de manera eficiente con PyTorch. Para configurarlo, necesitamos cargar el modelo preentrenado [39], realizar inferencias en las imágenes, y procesar los resultados de las detecciones.

En caso de entrenar un modelo manualmente, es acosejable hacerlo con las imágenes de la propia cámara del dron.

El código de que ilustra la configuración y el uso de Yolov5 para detectar personas y objetos se puede ver en 6.5

Explicación de las Funciones

- `get_center(bbox)`: Esta función toma las coordenadas de una caja delimitadora (`bbox`) y calcula su centro. Las coordenadas de entrada son `xmin`, `ymin`, `xmax`, `ymax`, y la función retorna las coordenadas (`xc`, `yc`) del centro de la caja.
- `load_model()`: Esta función carga el modelo preentrenado de Yolov5 desde el repositorio de Ultralytics en PyTorch Hub. Muestra mensajes en consola durante el proceso de carga del modelo y retorna el modelo cargado.
- `get_bboxes(preds)`: Esta función toma las predicciones (`preds`) obtenidas del modelo y filtra las cajas delimitadoras con una confianza mayor o igual al 30 % (`confidence >= 0.3`). Retorna un `DataFrame` con las coordenadas de las cajas (`xmin`, `ymin`, `xmax`, `ymax`), la confianza y el nombre del objeto detectado.
- `detector(frame, model)`: Esta función realiza la detección de objetos en un fotograma (`frame`) utilizando el modelo de Yolov5. Ejecuta la inferencia, procesa las predicciones para obtener las cajas delimitadoras y dibuja círculos y rectángulos alrededor de los objetos detectados en el fotograma. Además, muestra el nombre del objeto y la confianza en la imagen. Retorna el fotograma procesado.

5.5.1. Integración de Aruco con Yolov5

Dado que Yolov5 es compatible con OpenCV y Aruco también es una biblioteca basada en OpenCV, no hay problemas de compatibilidad entre

ellos. La integración de Aruco con Yolov5 puede ser muy útil para aplicaciones que requieren la detección y el seguimiento de marcadores junto con otros objetos en la misma escena 5.5.

La forma de conectar ambas tecnologías sería la siguiente:

Listing 5.5: Conexión YOLOv5 con ARUCO

```
1 from detector import detector, load_model
2
3 # Cargar el modelo de Yolov5
4 model = load_model()
5
6 # Capturar una imagen o video frame (img)
7 # ... (codigo para capturar la imagen)
8
9 # Detectar objetos en la imagen usando Yolov5
10 img = detector(img, model)
11
12 # Mostrar la imagen procesada
13 cv2.imshow('Original', img)
```

En este ejemplo, primero cargamos el modelo de Yolov5 utilizando la función `load_model`. Luego, capturamos una imagen o un fotograma de video y pasamos esta imagen a la función `detector` para realizar la detección de objetos. Finalmente, mostramos la imagen procesada utilizando OpenCV.

Esta integración permite combinar la detección de objetos de Yolov5 con la detección de marcadores Aruco en un flujo de trabajo unificado, aprovechando la capacidad de ambas bibliotecas para trabajar con imágenes y videos como se muestra en las figura 5.9.



Figura 5.9: integración de detección de personas con YOLO y detección de etiquetas con ARUCO

5.6. Configuración del sistema para navegación autónoma indoor

En esta sección, se describirá la configuración del sistema para la navegación autónoma en interiores, incluyendo la parametrización de las etiquetas ARUCO, la explicación detallada de los diagramas de estados utilizados para el control del sistema, y la interfaz gráfica de usuario desarrollada para visualizar los datos obtenidos.

5.6.1. Parametrización de las Etiquetas ARUCO

Para el sistema de navegación autónoma, se utilizarán etiquetas ARUCO que nos permiten identificar y localizar diferentes puntos de interés en el entorno. Las etiquetas están clasificadas según el siguiente esquema:

- **Etiquetas para Paredes:** IDs que terminan en 0 o 1.
- **Etiquetas para Marcos de Puertas:** IDs que terminan en 5.

- **Etiquetas para Puertas:** IDs que terminan en 4, que indican si las puertas están abiertas.
- **Etiquetas para Pasillos:** IDs que terminan en 6.

A continuación, se presenta una tabla con la parametrización de las etiquetas ARUCO:

Tipo de Etiqueta	Patrón de ID	Descripción
Etiquetas para Paredes	Terminan en 0 o 1	Ubicación de las paredes
Etiquetas para Marcos	Terminan en 5	Ubicación de los marcos de puertas
Etiquetas para Puertas	Terminan en 4	Indican si las puertas están abiertas
Etiquetas para Pasillos	Terminan en 6	Indican si hay un pasillo
Etiquetas libres	Terminan en 2, 3, 7, 8 y 9	Para la escalabilidad del proyecto

Tabla 5.1: Parametrización de las Etiquetas ARUCO

5.6.2. Interfaz Gráfica

Basado en los datos obtenidos de las etiquetas ARUCO, se ha desarrollado una interfaz gráfica de usuario GUI (Graphic User Interface o Interfaz Gráfica de Usuario) utilizando OpenCV (`cv2`). La interfaz muestra un plano en base a los datos recogidos y un minimapa que indica la dirección hacia la cual apunta la cámara del dron. Adicionalmente, se presenta en tiempo real lo que la cámara del dron está capturando 5.10.

A continuación, se describe cada paso del diagrama:

1. **Procesar Orientaciones (distancia, yaw, pitch, roll):** Este bloque representa el procesamiento de datos de orientación, que incluye distancia, *yaw* (giro), *pitch* (inclinación) y *roll* (alabeo).
2. **¿Hay Orientaciones?:** Aquí se realiza una verificación para determinar si se han recibido orientaciones válidas.
 - **Sí:** Si hay orientaciones, el flujo continúa hacia “Dibujar Marcador en GUI”.
 - **No:** Si no hay orientaciones, el flujo se dirige a “Guardar registros de distancias y ángulos”.
3. **Dibujar Marcador en GUI:** Si hay orientaciones, se dibuja un marcador en la GUI para representar la orientación procesada.
4. **¿Está modo mapeo activado?:** Se verifica si el modo de mapeo está activado en el sistema.

- **Sí:** Si el modo de mapeo está activado, se muestra el modo de mapeo en la GUI y el flujo vuelve al procesamiento de orientaciones.
 - **No:** Si el modo de mapeo no está activado, el flujo sigue a “Guardar registros de distancias y ángulos”.
5. **Guardar registros de distancias y ángulos:** Si no hay orientaciones o si el modo de mapeo no está activado, se guarda la información de las distancias y ángulos registrados.
 6. **¿Hay registros de distancias y ángulos?:** Se verifica si existen registros de distancias y ángulos almacenados.
 - **Sí:** Si existen registros, se calculan promedios para cada registro.
 - **No:** Si no hay registros, el flujo sigue a “Dibujar Mapa en base a registros”.
 7. **Calcular Promedios para cada registro:** Aquí se realiza el cálculo de los promedios basados en los registros disponibles de distancias y ángulos.
 8. **Mostrar Mapa y Minimapá con datos finales:** Finalmente, se muestra en la GUI el mapa y un minimapa con los datos finales procesados.

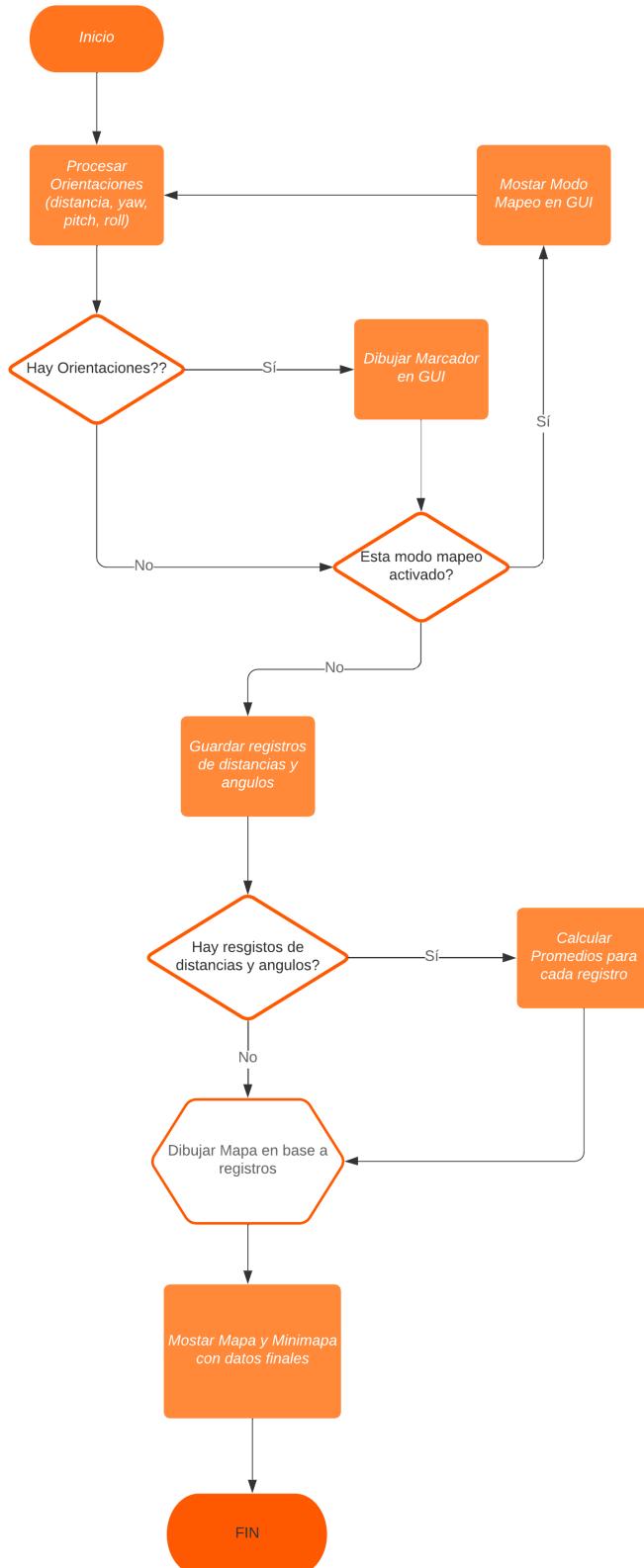


Figura 5.10: Control de flujo GUI

La GUI se compone de los siguientes elementos principales:

■ **Plano General:**

- Utiliza los datos de las distancias y orientaciones de las etiquetas ARUCO para dibujar un mapa en 2D.
- Se representa la posición de las paredes, puertas y marcos de puertas según los IDs de las etiquetas.

■ **Minimap:**

- Indica la dirección hacia la cual apunta la cámara del dron.
- Muestra un círculo representando el campo de visión del dron.
- Incluye un indicador de norte para facilitar la orientación.

■ **Vista de Cámara:**

- Muestra en tiempo real lo que la cámara del dron está capturando.
- Superpone las detecciones de las etiquetas ARUCO en la imagen de la cámara para una mejor visualización.

El fragmento de código 6.6 ilustra cómo se dibujan estos elementos en la GUI utilizando OpenCV.

5.6.3. Control de Navegación

El diagrama 5.11 representa un flujo de control para un dron, donde cada estado representa una función o comportamiento específico que el dron debe ejecutar. A continuación, se detalla cada estado y sus transiciones:

1. **Estado de Mapeo (Control de Navegación):** El primer paso después del inicio es entrar en el estado de mapeo, donde se controla la navegación del dron.
2. **¿Etiqueta ID = 5?:** Se verifica si la etiqueta con ID 5 está presente.
 - **Sí:** Si la etiqueta es 5, el flujo continúa hacia el “Estado de control de roll”.
 - **No:** Si la etiqueta no es 5, el flujo vuelve al “Estado de Mapeo (Control de Navegación)” .

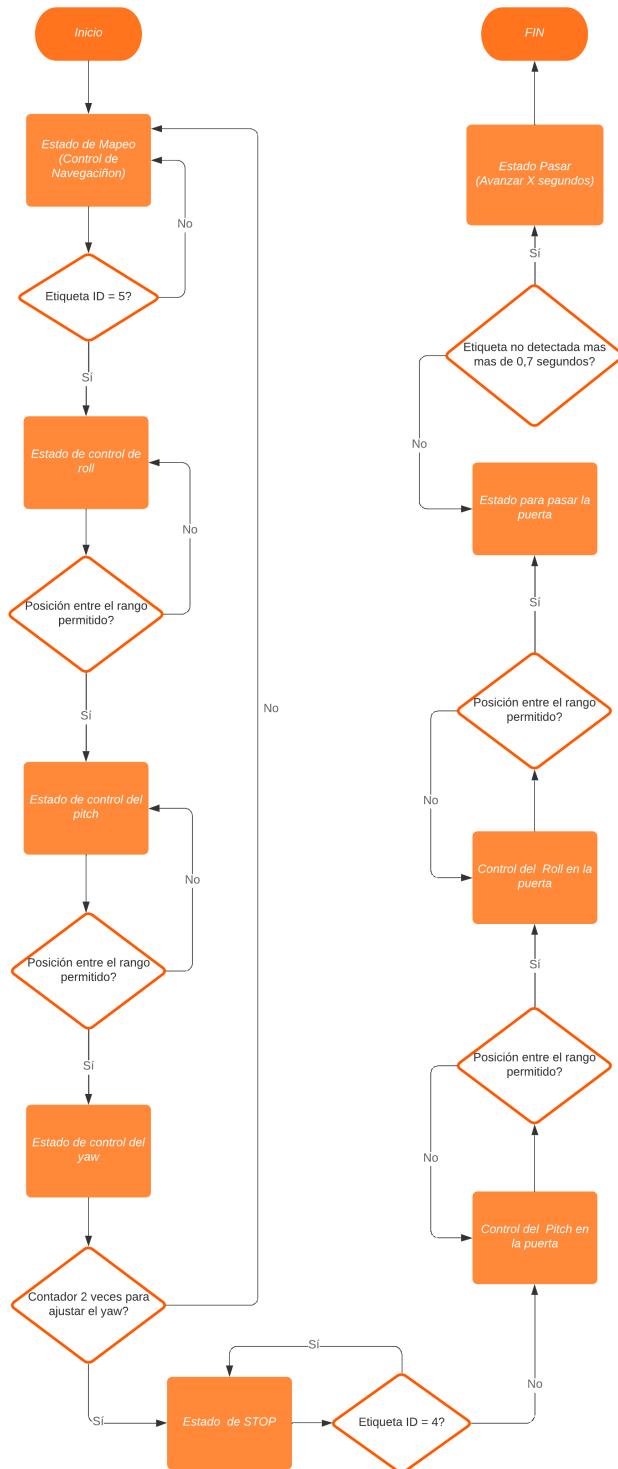


Figura 5.11: Control de flujo para el seguimiento de etiquetas

3. **Estado de control de roll:** En este estado, se controla el roll del dron.
4. **¿Posición entre el rango permitido?:** Se verifica si la posición del dron está dentro del rango permitido.
 - **Sí:** Si la posición está dentro del rango, el flujo avanza al “Estado de control del pitch”.
 - **No:** Si no está dentro del rango, el flujo regresa al “Estado de control de roll”.
5. **Estado de control del pitch:** En este estado, se controla el pitch del dron.
6. **¿Posición entre el rango permitido?:** Se verifica nuevamente si la posición está dentro del rango permitido.
 - **Sí:** Si la posición es adecuada, el flujo continúa hacia el “Estado de control del yaw”.
 - **No:** Si la posición no está dentro del rango, el flujo vuelve al “Estado de control del pitch”.
7. **Estado de control del yaw:** Aquí se controla el yaw del dron.
8. **¿Contador 2 veces para ajustar el yaw?:** Se verifica si se ha contado dos veces para ajustar el yaw.
 - **Sí:** Si se ha contado dos veces, se pasa al “Estado de STOP”.
 - **No:** Si no se ha contado dos veces, el flujo regresa al “Estado de control del yaw”.
9. **Estado de STOP:** El dron entra en un estado de parada.
10. **¿Etiqueta ID = 4?:** Se verifica si la etiqueta es ID 4.
 - **Sí:** Si es 4, el flujo va hacia “Control del Pitch en la puerta”.
 - **No:** Si no es 4, el flujo regresa al “Estado de Mapeo (Control de Navegación)”.
11. **Control del Pitch en la puerta:** Se controla el pitch del dron al pasar por una puerta.
12. **¿Posición entre el rango permitido?:** Se verifica si la posición es adecuada.
 - **Sí:** Si la posición es correcta, el flujo avanza al “Control del Roll en la puerta”.

- **No:** Si la posición no está en el rango permitido, el flujo regresa a “Control del Pitch en la puerta”.
13. **Control del Roll en la puerta:** Se controla el roll del dron al pasar por la puerta.
14. **¿Posición entre el rango permitido?:** Se verifica nuevamente si la posición está dentro del rango permitido.
- **Sí:** Si la posición es adecuada, el flujo avanza al siguiente paso.
 - **No:** Si no está dentro del rango, el flujo regresa al “Control del Roll en la puerta”.
15. **¿Etiqueta no detectada más de 0,7 segundos?:** Se verifica si la etiqueta no ha sido detectada por más de 0,7 segundos.
- **Sí:** Si la etiqueta no ha sido detectada en ese tiempo, se avanza al “Estado Pasar (Avanzar X segundos)”.
 - **No:** Si la etiqueta sigue siendo detectada, se regresa al “Estado para pasar la puerta”.
16. **Estado para pasar la puerta:** El dron entra en un estado para pasar la puerta.
17. **Estado Pasar (Avanzar X segundos):** Finalmente, el dron avanza durante un tiempo determinado.

Este diagrama representa un flujo de control específico para un dron que necesita detectar marcadores ARUCO y ajustar sus parámetros de vuelo (*roll, pitch, yaw, throttle*) para navegar a través de su entorno, particularmente en zonas específicas como puertas. Cada transición está condicionada por eventos específicos como la detección de marcadores ARUCO y el ajuste de ciertos parámetros dentro de rangos predefinidos.

El fragmento de código 6.7 muestra como se manejan los estados dentro de la función:

5.6.4. Flujo del Control Principal

El flujo principal del programa consta de dos estados de verificación. El primero es un estado de mapeo, en el cual el dron realizará dos giros completos para recopilar todos los datos posibles utilizando ARUCO y Yolov5. El segundo estado verificará si el mapeo ha sido completado exitosamente; de ser así, se activará la GUI y el estado de navegación.

A continuación se van a explicar detalladamente los diagramas que controla el flujo principal

5.6.4.1. Estado de Mapeo

En el primer estado del programa 5.12, el dron entra en modo de mapeo, donde realiza dos giros completos para escanear y capturar datos del entorno utilizando las tecnologías ARUCO y Yolov5. Este proceso es fundamental para construir un mapa detallado del área, lo que permitirá la navegación precisa y la interacción con el entorno.

1. **Procesar valor inicial del yaw:** El dron comienza procesando el valor inicial del yaw, que es la orientación actual.
2. **¿Yaw inicial es None?:** Se verifica si el valor inicial del yaw es *None* (es decir, si no se ha establecido).
 - **Sí:** Si el valor inicial es *None*, se calcula el valor actual del yaw y se acumula.
 - **No:** Si el valor inicial no es *None*, se omite esta parte y se continúa.
3. **Calcular valor yaw actual y acumularlo:** Si el yaw inicial era *None*, ahora se calcula y se acumula el valor del yaw actual.
4. **Girar Dron hasta valor yaw = 720 grados:** El dron gira hasta que el valor acumulado del yaw alcance los 720 grados, lo que representa dos vueltas completas.
5. **¿Valor del yaw es superior a 720 grados?:** Se verifica si el valor acumulado del yaw ha superado los 720 grados.
 - **Sí:** Si el valor es superior a 720 grados, el proceso continúa hacia “Mapeo Completado”.
 - **No:** Si el valor no ha alcanzado los 720 grados, el dron continúa girando y acumulando el yaw.
6. **Mapeo Completado, Sobrescribir Variables, Parar el giro del Dron:** Una vez que el dron ha completado las dos vueltas (720 grados), se considera que el mapeo está completado. Se sobrescriben las variables necesarias y se detiene el giro del dron.

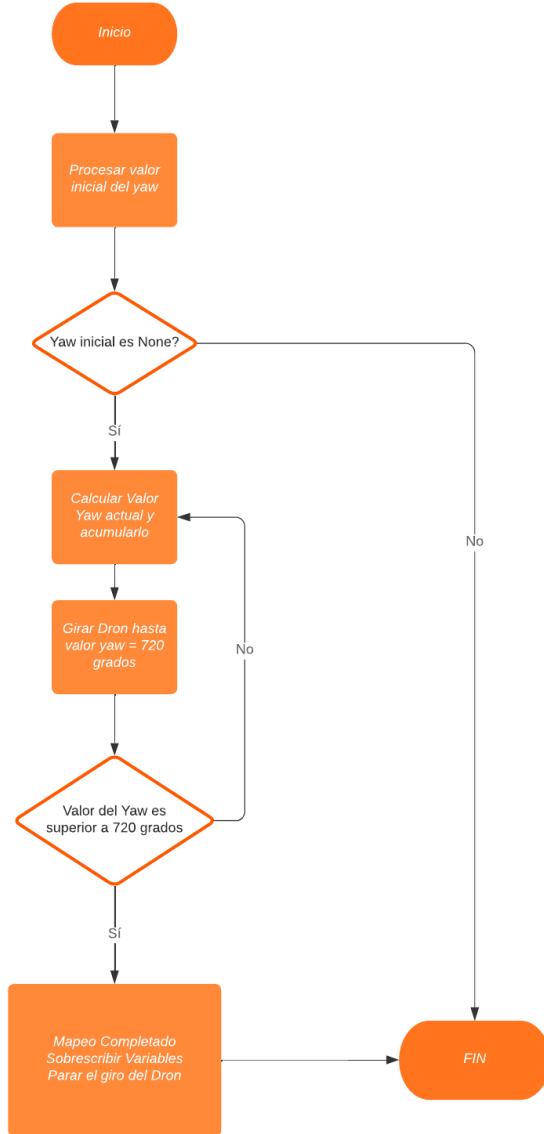


Figura 5.12: Control del estado de Mapeo

El fragmento de código 6.8 muestra como se maneja el control del Yaw.

5.6.4.2. Verificación del estado Mapping

Una vez completado el mapeo, el programa entra en el segundo estado, donde verifica si el proceso de mapeo se ha realizado correctamente. Si el

mapeo es exitoso, se activará la interfaz gráfica de usuario (GUI) y se habilitará el estado de navegación, permitiendo al dron moverse de manera segura y eficiente dentro del entorno previamente mapeado.

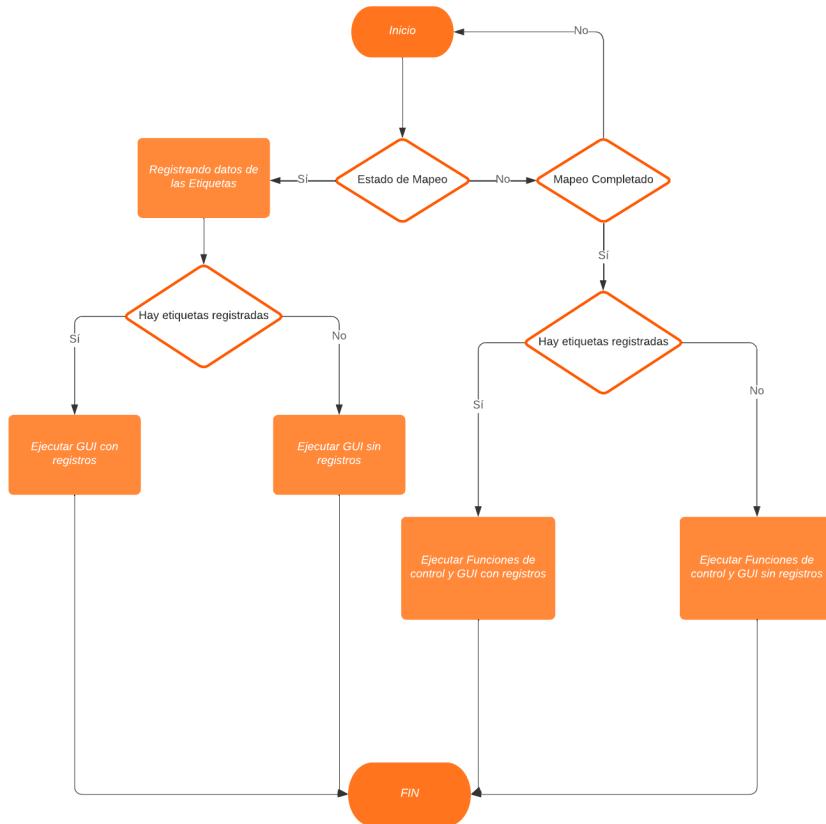


Figura 5.13: Control de flujo para verificar estado de Mapeo

El diagrama 5.13 representa un el flujo que controla las otras configuraciones mencionadas anteriormente:

- 1. Estado de Mapeo:** Se verifica si el dron está en el estado de mapeo.
 - Sí:** Si el dron está en estado de mapeo, el flujo continúa con "Registrando datos de las Etiquetas".
 - No:** Si no está en estado de mapeo, el flujo avanza hacia la verificación del "Mapeo Completado".

2. **Registrando datos de las Etiquetas:** En este paso, el dron registra los datos de las etiquetas detectadas durante el mapeo.
3. **¿Hay etiquetas registradas?:** Se verifica si hay etiquetas registradas.
 - **Sí:** Si hay etiquetas registradas, el flujo avanza hacia “Ejecutar GUI con registros”.
 - **No:** Si no hay etiquetas registradas, el flujo sigue hacia “Ejecutar GUI sin registros”.
4. **Ejecutar GUI con registros:** Si hay etiquetas registradas, se ejecuta la interfaz gráfica de usuario (GUI) utilizando los registros obtenidos.
5. **Ejecutar GUI sin registros:** Si no hay etiquetas registradas, se ejecuta la GUI sin utilizar registros.
6. **Mapeo Completado:** Si el estado de mapeo no está activo, se verifica si el mapeo ha sido completado.
 - **Sí:** Si el mapeo está completado, se verifica si hay etiquetas registradas.
 - **No:** Si el mapeo no está completado, el flujo no sigue hacia ningún paso posterior específico en este diagrama.
7. **¿Hay etiquetas registradas?:** En este punto, se realiza nuevamente una verificación para saber si hay etiquetas registradas después de que el mapeo ha sido completado.
 - **Sí:** Si hay etiquetas registradas, se procede a “Ejecutar Funciones de control y GUI con registros”.
 - **No:** Si no hay etiquetas registradas, se procede a “Ejecutar Funciones de control y GUI sin registros”.
8. **Ejecutar Funciones de control y GUI con registros:** Si hay etiquetas registradas, se ejecutan las funciones de control junto con la GUI utilizando los registros obtenidos.
9. **Ejecutar Funciones de control y GUI sin registros:** Si no hay etiquetas registradas, se ejecutan las funciones de control y la GUI sin utilizar registros.

El fragmento de código 6.9 muestra como se maneja el control del flujo principal.

5.6.5. Sistemas de alertas basado en Telegram

El sistema de alertas de nuestro proyecto se basa en un bot automatizado en Telegram que envía un video cuando Yolov5 detecta la presencia de una persona o un coche durante más de 2 segundos, continuando hasta que la detección cesa. Al establecer el límite en 2 segundos, se eliminan posibles errores que podrían surgir por el paso rápido de una persona o por detecciones erróneas de corta duración, asegurando así una mayor precisión en las alertas.

Pero primero las instrucciones para crear un bot en telegram [40]:

Paso 1: Crear un bot con BotFather

1. **Abre Telegram** y busca **@BotFather**, el cual es un bot oficial de Telegram que te permite crear y gestionar otros bots.
2. **Inicia una conversación** con **@BotFather** y usa el comando **/start** para ver las opciones disponibles.
3. Usa el comando **/newbot** para crear un nuevo bot. BotFather te pedirá que:
 - Le des un nombre a tu bot.
 - Le asigne un nombre de usuario (que debe terminar en **bot**, por ejemplo, **MiBotAlertaBot**).
4. Una vez que completes estos pasos, BotFather te proporcionará un **token** 5.14. Este token es esencial para interactuar con la API de Telegram.

Paso 2: Obtener el chat ID

1. Para enviar mensajes a un chat específico, necesitas el **chat_id** del usuario o grupo. Una manera sencilla de obtener tu propio **chat_id** es usar un bot ya creado para enviarte un mensaje y luego revisar la respuesta de la API.
2. Puedes enviar un mensaje de prueba a tu bot y luego visitar el siguiente enlace en tu navegador, reemplazando **BOT_TOKEN** con el token de tu bot:

https://api.telegram.org/bot<BOT_TOKEN>/getUpdates

Esto te mostrará las actualizaciones recientes que el bot ha recibido, incluyendo el **chat_id**.

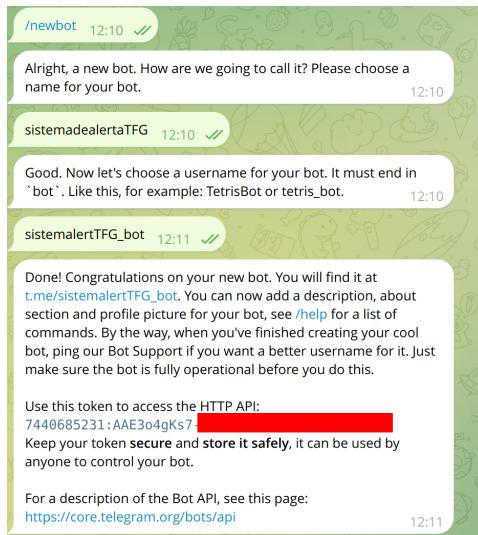


Figura 5.14: Crear bot de telegram

Paso 3: Usar la API HTTP de Telegram

Ahora que tienes el BOT_TOKEN y el CHAT_ID, puedes usar la API HTTP de Telegram para enviar mensajes y videos. Aquí tienes el código mencionado, con una breve explicación de cada parte.

El fragmento de código 6.10 muestra como se maneja la API para el envío de datos.

1. Enviar un mensaje de texto:

- Se construye la URL para la API sendMessage y se prepara el diccionario data con el chat_id y el texto del mensaje.
- Se envía una petición POST a la API de Telegram usando la función requests.post().

2. Enviar un video:

- La URL se construye para la API sendVideo.
- El archivo de video se abre en modo binario ('rb').
- El diccionario files se usa para enviar el archivo de video, mientras que data sigue conteniendo el chat_id.
- Se envía otra petición POST para subir el video.

Paso 4: Prueba del bot

1. Ejecuta el código y verifica en Telegram que recibes tanto el mensaje de alerta como el video.

2. Puedes personalizar el código para ajustarlo a tus necesidades específicas, como enviar diferentes tipos de archivos o manejar otros tipos de eventos.

El diagrama de flujo 5.15 representa la lógica de control de grabación para un sistema de detección de personas utilizando YOLOv5, reflejando cómo funciona el código presentado en 6.11.

Explicación del Diagrama de Flujo:

1. Inicio y Captura de Video:

- El proceso comienza con la captura de video en tiempo real desde una cámara utilizando YOLOv5 para detectar la presencia de personas.

2. Detección de Personas:

- **Condición:** Se evalúa si hay una persona detectada en el frame actual.
- **Acción:** Si no se detecta ninguna persona, el sistema continúa capturando video hasta que se detecte una.

3. Inicialización del Contador:

- **Condición:** Si se detecta una persona, se inicializa un contador de tiempo (2 segundos) para comprobar la persistencia de la detección.

4. Evaluación para Iniciar la Grabación:

- **Condición:** Si la persona ha estado presente durante al menos 2 segundos y la grabación aún no ha comenzado, se procede a iniciar la grabación.
- **Acción:** La grabación se inicia y se comienza a contar el tiempo de grabación.

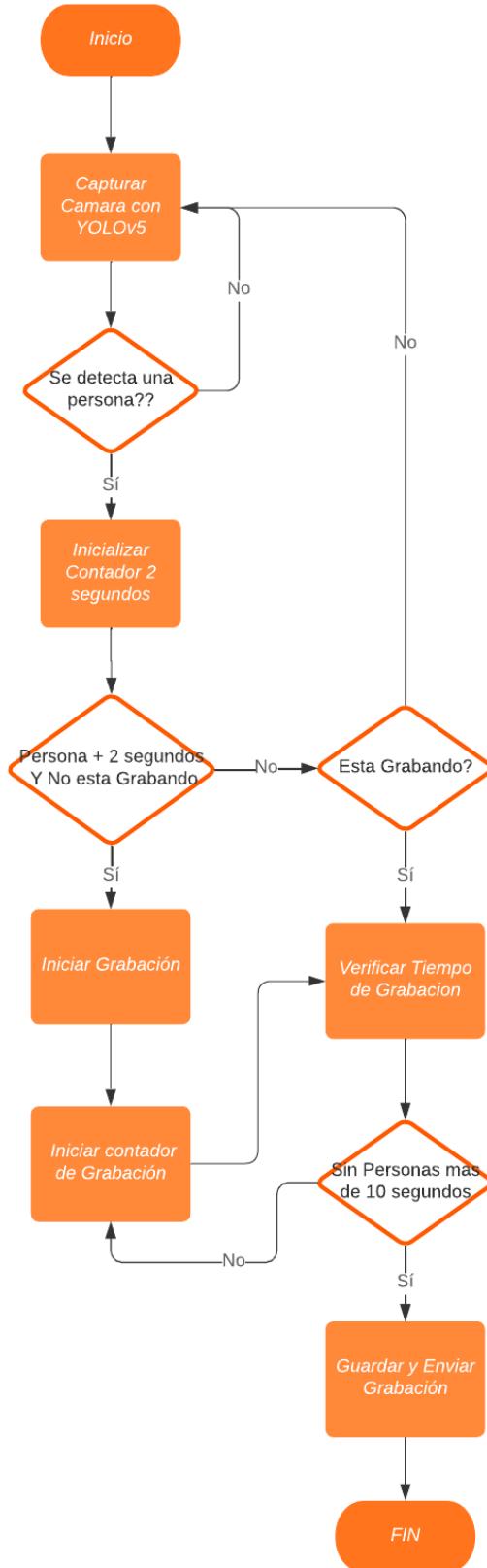


Figura 5.15: Control de flujo para el envio de alertas

5. Verificación del Estado de Grabación:

- **Condición:** Si ya se está grabando, el sistema verifica si la persona sigue presente o no.
- **Acción:** Si la persona sigue presente, se continúa la grabación.

6. Terminación de la Grabación:

- **Condición:** Si la persona no es detectada durante un período de más de 10 segundos, la grabación se detiene.
- **Final:** El video grabado se guarda y se envía una alerta o notificación con la grabación.

El fragmento de código 6.11 muestra como se maneja el sistema de alerta en el proyecto.

Capítulo 6

Evaluación

En este capítulo se realizarán experimentos detallados que incluyen la medición de errores en los sistemas de marcadores, con el objetivo de analizar su precisión y determinar sus límites operativos. Estos experimentos permitirán entender mejor el comportamiento de los marcadores bajo diversas condiciones. Además, se evaluarán todos los algoritmos descritos en el capítulo anterior, se probarán tanto en un entorno simulado como en un dron real. Esto permitirá comparar el desempeño de los algoritmos en escenarios controlados frente a condiciones del mundo real, proporcionando una visión de su efectividad y fiabilidad.

6.1. Experimentos de calibración y mediciones de error en ARUCO

Los errores en la detección de los marcadores ARUCO pueden ser casi insignificantes, tanto en términos de distancia como de ángulos. Para medir los errores en distancias, se utilizó un metro y se midieron las distancias reales en función de las proporcionadas por ARUCO de 40cm a 500cm en intervalos de 20cm. Los errores angulares se midieron de manera similar de -60° a 60° en intervalos de 10° . Se tomaron 200 datos para ambas mediciones y de forma aleatorizada para asegurar la validez de los resultados.

La tabla 6.1 presenta estadísticas clave para tres conjuntos de datos: **real_angle** (ángulo real medido), **aruco_angle** (ángulo estimado usando ArUco), y **error** (diferencia entre **aruco_angle** y **real_angle**).

Columnas

- **real_angle**: Muestra las estadísticas de los ángulos reales medidos.

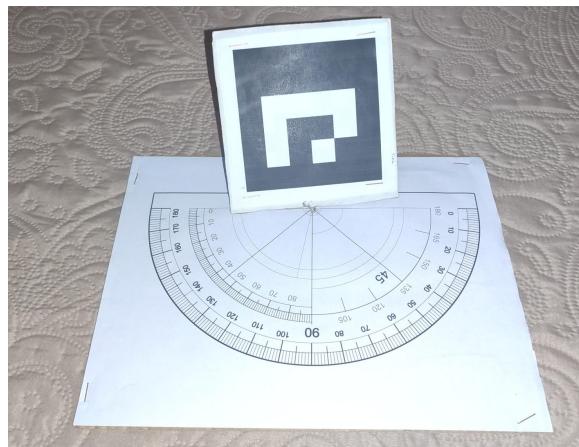


Figura 6.1: Herramienta para de medición de errores para los ángulos

- **aruco_angle:** Contiene las estadísticas de los ángulos estimados por ArUco.
- **error:** Presenta las estadísticas de la diferencia entre los ángulos estimados y los reales.

Filas (Estadísticas)

- **Count:** Número de observaciones en cada conjunto.
- **Mean:** Valor promedio de cada conjunto de datos.
- **Std:** Desviación estándar, que mide la variabilidad de los datos.
- **Min y Max:** Valores mínimos y máximos en cada conjunto.
- **25 %, 50 %, 75 %:** Los cuartiles, que dividen los datos en porcentajes, mostrando la distribución.

La tabla 6.1 muestra cómo se distribuyen los ángulos reales y estimados, y el tamaño del error de estimación, facilitando la comparación y comprensión de los resultados.

Las figuras 6.2, 6.3, 6.4 y 6.5 demuestran que los errores en la detección de ángulos por ARUCO son consistentes y controlados, lo que respalda la precisión y la fiabilidad de este sistema para aplicaciones de navegación autónoma.

	real_angle	aruco_angle	error
count	202.000000	202.000000	
mean	1.732673	2.672723	0.940049
std	37.475934	36.079559	2.249982
min	-60.000000	-66.772976	-6.772976
25 %	-30.000000	-28.095172	-0.693887
50 %	0.000000	2.813426	1.138701
75 %	30.000000	31.138620	2.535485
max	60.000000	66.259790	6.259790

Tabla 6.1: Resumen Estadístico de los Errores Angulares

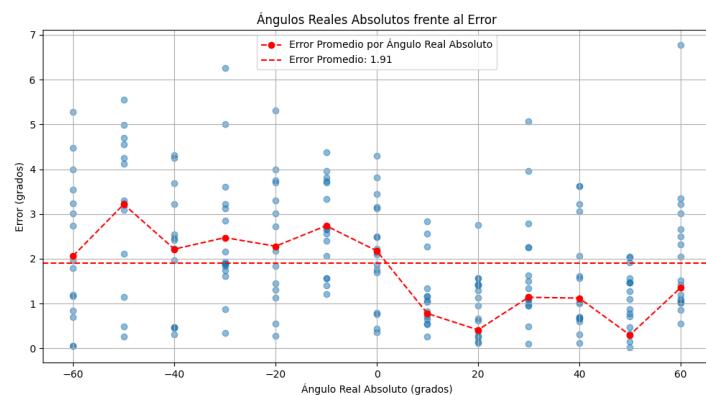


Figura 6.2: Errores en Ángulos Detectados por ARUCO

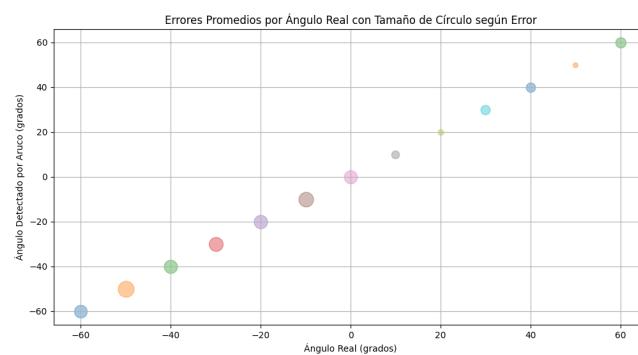


Figura 6.3: Distribución de Errores Absolutos por Ángulo Real

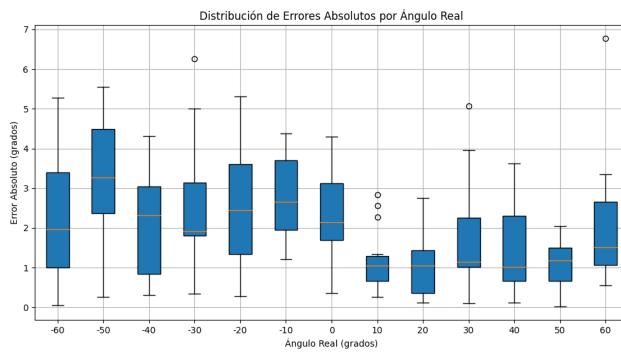


Figura 6.4: Errores Promedios por Ángulo Real con Tamaño de Círculo según Error

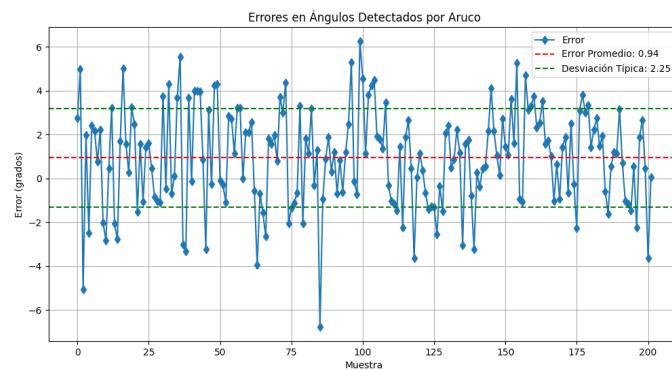


Figura 6.5: Ángulos Reales Absolutos frente al Error

A continuación, se presenta un resumen estadístico de los errores de distancia obtenidos:

	real_distance	aruco_distance	error
count	200.000000	200.000000	
mean	2698.000000	2690.202074	-7.797926
std	1605.892416	1618.957472	182.400848
min	200.000000	195.069631	-4.930369
25 %	1200.000000	1220.098551	-65.460062
50 %	2600.000000	2686.722210	-1.132192
75 %	4000.000000	3987.163999	53.108131
max	5400.000000	5994.369566	594.369566

Tabla 6.2: Resumen Estadístico de los Errores de Distancia

La tabla 6.2 muestra que el error promedio es de aproximadamente -7.80 mm, con una desviación estándar de 182.40 mm. Estos resultados indican que los errores en la detección de distancia son pequeños y manejables, aunque tienden a aumentar con la distancia real medida.

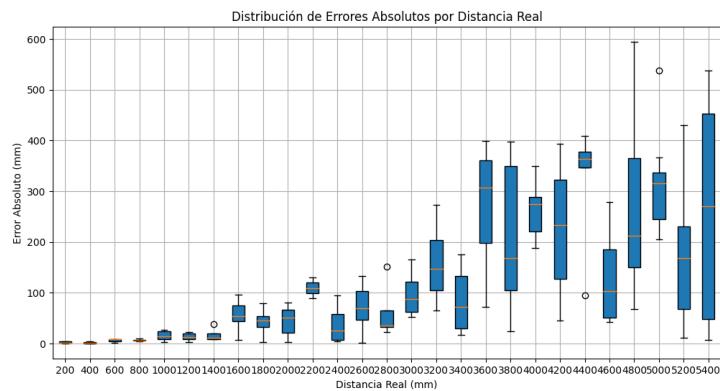


Figura 6.6: Errores en Distancias Detectadas por ARUCO

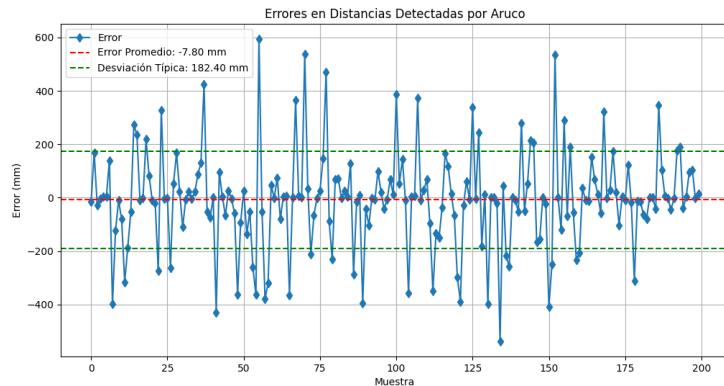


Figura 6.7: Distribución de Errores Absolutos por Distancia Real

Las figuras 6.6 y 6.7 demuestran que los errores en la detección de distancias por ARUCO son consistentes y controlados, con un aumento en el error a medida que aumenta la distancia. Sin embargo, este aumento es aceptable, ya que a mayores distancias se puede tolerar un poco más de error debido a la naturaleza de la medición.

6.1.1. Análisis de cómo ARUCO mejora la orientación y posicionamiento del dron

Gracias a los bajos errores generados por una correcta calibración, junto con los datos del propio dron, es posible determinar con precisión las coordenadas x , y , z y los ángulos de orientación (yaw, pitch, roll). Esto hace que los marcadores ARUCO sean indispensables para mejorar la navegación autónoma del dron, proporcionando un sistema de posicionamiento fiable y preciso en entornos donde otros métodos no son viables.

6.2. Experimentos en el Simulador

El uso de un simulador y sus beneficios ya han sido explicados en capítulos anteriores. Ahora, se procederá a implementar todas las funciones descritas en el capítulo anterior en el dron, asegurando de que su integración resulte en un funcionamiento óptimo y correcto del dispositivo. El mapa utilizado para la simulación y el recorrido realizado se pueden observar en la figura 6.8.



Figura 6.8: Mapa y recorrido en el simulador

La integración de todas estas configuraciones en un solo sistema permite

que el dron realice las siguientes acciones: primero, despega y entra en un estado de mapeo (Fig 6.9), donde realiza dos vueltas completas (720°)(Fig 6.10). Luego, intenta localizar los marcos de las puertas identificadas por etiquetas que terminan en 5 (Fig 6.11). Una vez encontrada una puerta, el dron se dirige hacia ella y verifica si está cerrada o abierta (Fig 6.12). Si la puerta está cerrada, el dron espera; si está abierta, pasa a través de ella (Fig 6.13) y regresa al estado de mapeo. Este proceso se repite sucesivamente (Fig 6.14).

6.2.1. Implementación en el Simulador

Uno de los mayores retos en la implementación es diseñar todas las funciones de manera que la transición al dron real sea lo más sencilla posible, y que la simulación se asemeje tanto como sea posible a la realidad. Esto implica integrar las funciones de control del dron, similares a las de la función `tellopy`, y combinar los diagramas de estado descritos anteriormente.

Para capturar el *roll*, *pitch* y *yaw* del robot en Webots, utilizamos los siguientes comandos:

```
roll = imu.getRollPitchYaw()[0]
pitch = imu.getRollPitchYaw()[1]
yaw = imu.getRollPitchYaw()[2]
```

Además, los métodos para obtener la altura y las distancias en el simulador difieren de los del dron real, lo cual será abordado en la siguiente sección.

Inicialmente, enfrentamos un problema donde el dron realizaba movimientos bruscos en el estado de seguimiento de etiquetas, lo que causaba caídas 6.1. Para mitigar esto, se implementó un factor de suavizado para las transiciones de velocidad, evitando cambios abruptos de 0.1 a 0.3, y en su lugar realizando incrementos graduales como 0.1, 0.12, 0.13, etc.

Listing 6.1: Suavización de la transición del pitch

```
1 # Suavizar la transición del pitch actual al pitch objetivo
2 suavizado = 0.05 # Factor de suavizado, ajustar según sea necesario
3 pitch = pitch_actual + (pitch_objetivo - pitch_actual) * suavizado
```

Sin embargo, el problema persistía debido a que la función de seguimiento solo se ejecutaba si el dron detectaba la etiqueta. Hubo casos en que el dron perdía visualmente la etiqueta por milisegundos, imperceptibles a simple vista. La solución fue implementar un tiempo mínimo de 0.5 segundos, durante el cual el dron continuaría con la última velocidad antes de perder la etiqueta 6.2.

Listing 6.2: Manejo del tiempo antes de considerar el marcador perdido

```
1 MARKER_LOST_TIMEOUT = 0.5 # Tiempo en segundos antes de considerar el
2 marcador perdido
3 if (current_time - last_marker_seen_time) < MARKER_LOST_TIMEOUT:
4     set_pitch(pitch)
5     print(f"Dron moviéndose con un pitch de {pitch:.3f}.)")
```

Esta solución mejoró significativamente la estabilidad del dron durante el seguimiento de etiquetas, como se puede ver a la figuras 6.9 - 6.15.

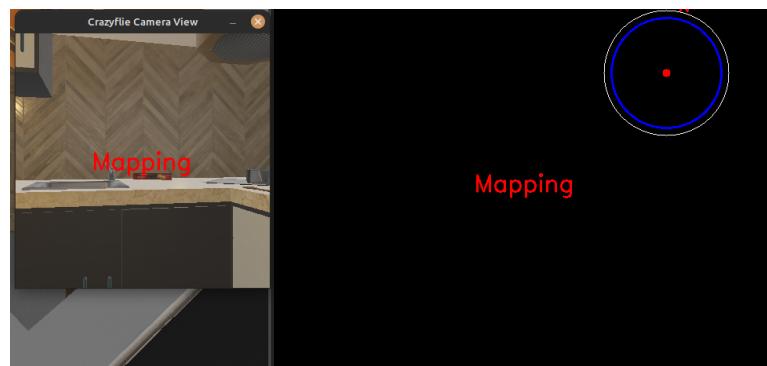


Figura 6.9: Estado Mapping



Figura 6.10: Finalizado Mapping

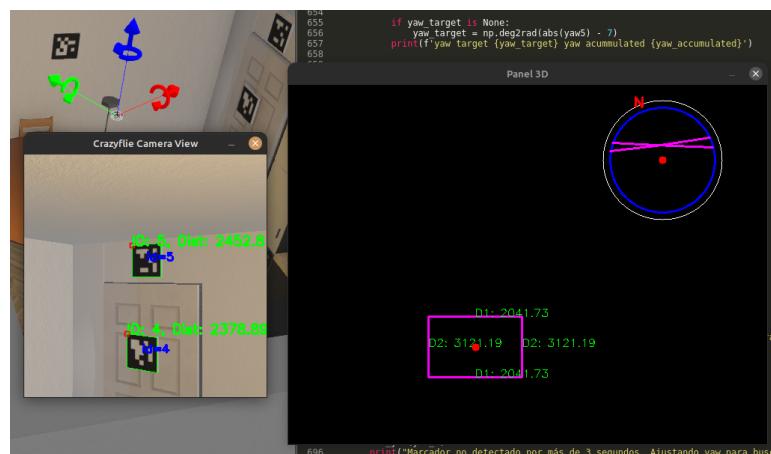


Figura 6.11: Estado de Seguimiento de Puerta (Etiqueta 5)



Figura 6.12: Estado de Seguimiento de Puerta (Stop Puerta Cerrada)

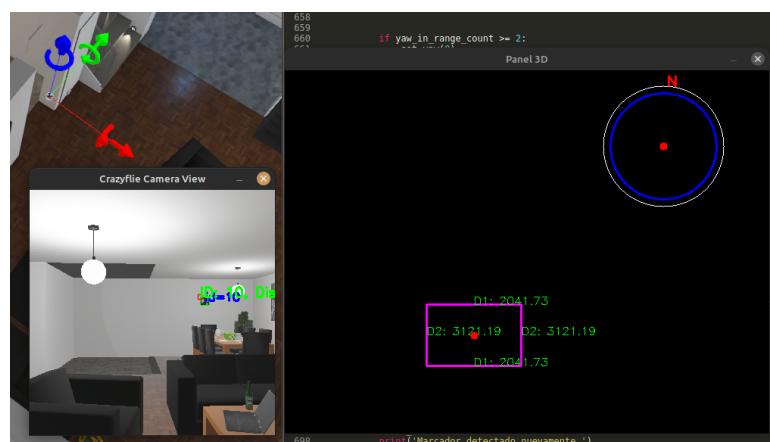


Figura 6.13: Estado de Pasado puerta (pass)

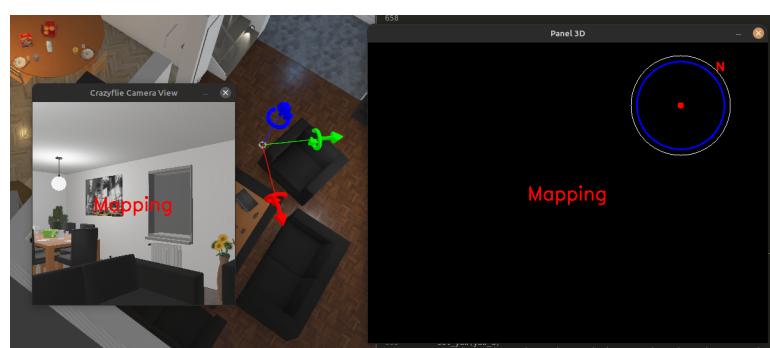


Figura 6.14: Vuelta a Estado de Mapping

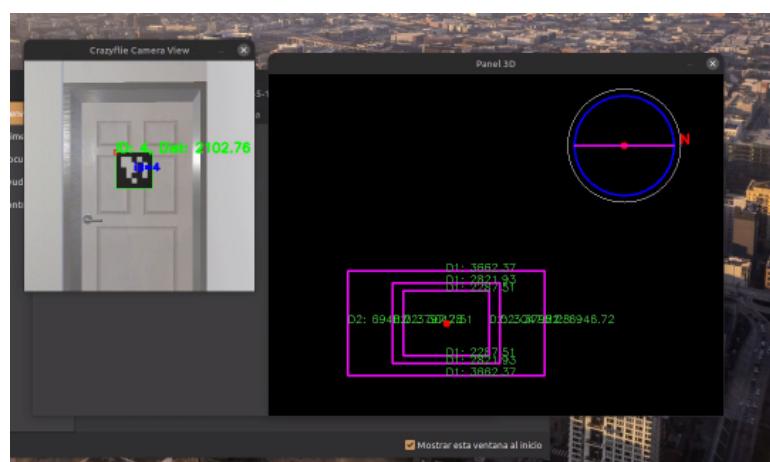


Figura 6.15: Final del recorrido

Después de realizar el experimento aproximadamente 15 veces, se logró el éxito en 13 ocasiones. Las dos veces en las que no se obtuvo éxito fueron atribuibles a errores del simulador y problemas en el ordenador. En las 13 ocasiones exitosas, el dron completó un recorrido de aproximadamente 20 metros, atravesando dos puertas, con un tiempo constante de 2:03.744 minutos, como se muestra en la figura 6.16. Cabe destacar que, dado que el simulador proporciona un escenario irreal pero ideal para pruebas, estos resultados son muy valiosos para evaluar el rendimiento en condiciones controladas.

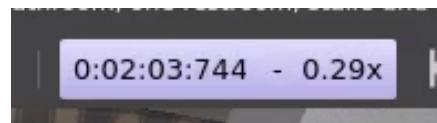


Figura 6.16: Tiempo en completar el recorrido.

6.2.2. Implementación en el dron real

Dado que utilizamos las mismas funciones de control del dron, no es necesario modificarlas para la implementación en la vida real. Sin embargo, sí es necesario ajustar los datos obtenidos del dron y la configuración de las distancias.

6.2.2.1. Estado de Mapeo

La única parte que se debe modificar es cómo se manejan y procesan los datos del *yaw* del dron para controlar su orientación. Esto se obtiene de la siguiente manera 6.3:

Listing 6.3: Capturador de datos del giroscopio

```

1 def handler(event, sender, data, **args):
2     global gyro_data
3     drone = sender
4     if event is drone.EVENT_LOG_DATA:
5         if hasattr(data, 'imu') and hasattr(data.imu, 'gyro_z'):
6             gyro_data = data.imu.gyro_z

```

Para procesar estos datos, se utiliza el siguiente fragmento de código 6.4, donde se convierten los valores a grados para minimizar el error, ya que trabajar con radianes genera más errores:

Listing 6.4: Actualización de la dirección basada en datos del giroscopio

```

1 # Actualizar la dirección actual basada en los datos del giroscopio
2 if gyro_data is not None:

```

```

3     if last_time is not None:
4         delta_time = start_time - last_time
5         current_direction += gyro_data * delta_time * 180 / np.pi
6         # Convertir radianes a grados
7         current_direction %= 360
8         # Mantener la dirección entre 0 y 360 grados
9         last_time = start_time

```

la variable se pasa a la función `draw_panel` como `yaw=current_direction` en grados para dibujar el norte de posicionamiento, en caso de que sea necesario dentro de la función de la GUI los datos en radianes se puede utilizar la función `np.deg2rad()`. Una vez estos ajustes hayan sido realizados se puede comprobar que la implementación del código simulado funciona y a su vez finlando los datos de la cámara con ARUCO y Yolov5 (Fig 6.17 6.18 6.19).

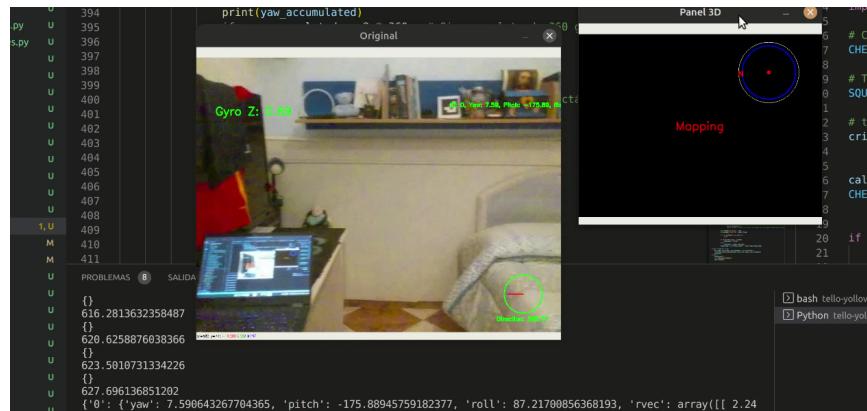


Figura 6.17: Implementación de Estado Mapeo en condiciones reales

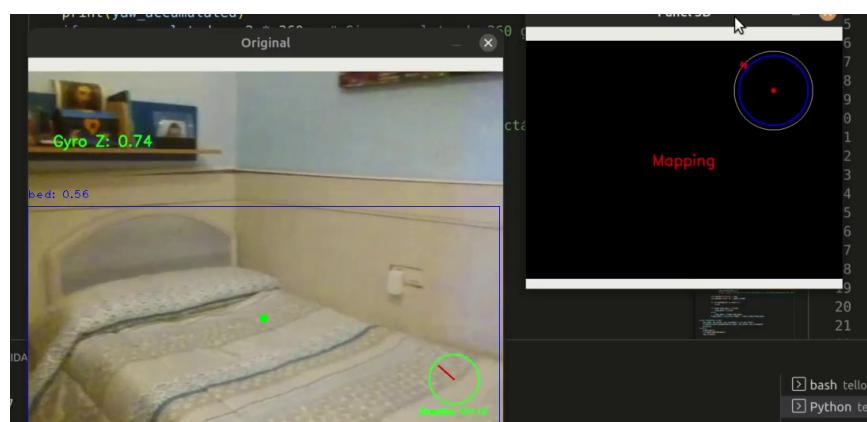


Figura 6.18: Estado Mapeo en condiciones reales con Yolo y ARUCO

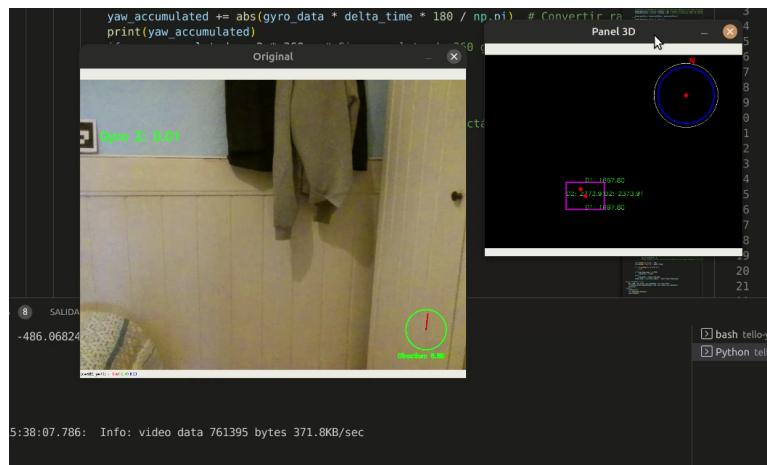


Figura 6.19: Implementación de Estado Mapeo en condiciones reales

6.3. Evaluación del sistema de alertas

Tras probar el código en diversas condiciones y someterlo a una amplia gama de situaciones aleatorias, se ha verificado que su funcionamiento es correcto y conforme a lo esperado. El temporizador de 2 segundos, diseñado para confirmar la presencia continua de una persona antes de iniciar la grabación, se hizo por las siguientes razones:

- **Evitar Falsos Positivos:** Este umbral reduce la probabilidad de que movimientos o sombras activen la grabación, asegurando que solo se graben eventos relevantes.
- **Estabilidad en la Detección:** Garantiza que la detección sea consistente y no fluctúe, evitando que la grabación se active y desactive repetidamente.
- **Optimización de Recursos:** Minimiza la grabación de detecciones breves e irrelevantes, ahorrando espacio de almacenamiento y facilitando la revisión posterior.

El sistema respondió adecuadamente, evitando falsos positivos y asegurando que la grabación solo se activara cuando realmente se detectó una persona de manera persistente 6.20. Este comportamiento confirma que la implementación del temporizador y la lógica de control de grabación funcionan de manera óptima, cumpliendo con los objetivos planteados.

El canal de Telegram donde se han realizado las pruebas es

t.me/sistemalertTFGDRON



Figura 6.20: Evaluación del sistema de detección

Conclusiones y Líneas Futuras

En este proyecto, se ha desarrollado un algoritmo de control y navegación para drones en entornos interiores, complementado con una interfaz gráfica diseñada para facilitar su uso por parte de los usuarios. Esta interfaz intuitiva permite que incluso aquellos con poca experiencia en programación puedan comprender y operar el sistema con mayor facilidad. El sistema navega por los diferentes espacios del edificio utilizando balizas visuales, y es capaz de detectar personas, integrando así un enfoque robusto para la navegación y seguridad. Se ha empleado el uso de balizas ARUCO, utilizando marcas fiduciales para facilitar la localización y navegación del dron, lo cual requirió un proceso de calibración y selección de tarjetas apropiadas para el entorno. Además, se ha usado YOLOv5 para la detección de personas, proporcionando un sistema eficiente y preciso para identificar la presencia humana en el espacio. El código se puede examinar en GitHub.

La integración de ambos sistemas, el de detección de balizas y el de personas, se realizó sobre un flujo continuo de imágenes, lo que permite al dron tomar decisiones de navegación en tiempo real. Además, se hizo uso de una biblioteca de bajo nivel para la comunicación con el dron, permitiendo el control directo sobre sus motores y sensores.

Se ha creado una API en el simulador Webots para el dron Tello EDU, que emula el comportamiento del dron real, lo cual facilita las pruebas y simulaciones sin incurrir en los riesgos o costos de experimentos físicos. Durante la fase de evaluación, se lograron resultados satisfactorios en cuanto a la detección de balizas y personas, aunque aún existen desafíos por resolver en la precisión de la navegación en entornos complejos.

Sin embargo, el desarrollo de este proyecto no estuvo exento de dificultades. Una de las principales complicaciones fue la implementación del control de navegación en el dron real. A pesar de haber conseguido un control efectivo sobre los parámetros de roll, yaw y throttle, el ajuste del pitch presentó un reto considerable. A pesar de los múltiples intentos por afinar este aspecto, no se logró el resultado esperado, dejando una sensación de

insatisfacción, aunque con la convicción de que es solo cuestión de tiempo para superar este obstáculo.

Además, se ha implementado un sistema de alertas utilizando Telegram y YOLOv5. Este sistema permite enviar notificaciones en tiempo real cuando se detectan personas en el entorno, brindando así una solución eficaz para la vigilancia o la detección de intrusos. El algoritmo basado en YOLOv5, al integrarse con Telegram, envía automáticamente alertas a los usuarios cuando el dron identifica la presencia humana, lo que proporciona un canal de comunicación instantáneo, eficiente y barato. Esta funcionalidad añade un componente crucial de interacción y respuesta rápida, mejorando la seguridad y la capacidad de reacción ante posibles amenazas o situaciones críticas en tiempo real.

En cuanto a las aplicaciones en la vida real, este proyecto abre un abanico de posibilidades. Por ejemplo, un dron equipado con este sistema podría servir como una herramienta de seguridad en el hogar, capaz de seguir a intrusos, o en escenarios más complejos, como la neutralización de amenazas en espacios concurridos, tales como aeropuertos, estadios o conciertos. Estas aplicaciones demuestran el potencial ilimitado de esta tecnología y cómo puede ser adaptada a diferentes necesidades, impulsando la innovación y la imaginación en el uso de drones.

Respecto a la continuación del proyecto, existen numerosas vías a explorar, especialmente en la integración del sistema en drones reales y su optimización para aplicaciones específicas. Aunque este proyecto no está orientado hacia un ámbito laboral específico en este momento, su desarrollo podría ser continuado con vistas a una implementación profesional en el futuro. La experiencia adquirida y las bases establecidas son un punto de partida sólido para futuros avances, y el desafío no resuelto del control de pitch en el dron real es una espina clavada que, sin duda, motiva a seguir trabajando hasta conseguir una solución definitiva.

6.3.1. Diferencias y Consideraciones entre la Implementación en Vida Real vs. Simulador

La implementación en simulador ofrece un ambiente controlado, seguro y flexible, permitiendo avanzar rápidamente y experimentar sin riesgos. Sin embargo, no refleja con precisión las condiciones del mundo real, donde factores como el desgaste del hardware y la imprevisibilidad del entorno afectan el comportamiento del dron. En la vida real, aunque el proceso es más lento y existe el riesgo de dañar el dron, se obtienen datos más relevantes y se pueden ajustar los sistemas para enfrentar problemas reales, como variaciones de luz y obstáculos imprevistos.

6.3.2. Evaluación de la Efectividad de la Simulación en la Preparación para Vuelos Reales

En este proyecto, se ha trabajado para que la simulación sea lo más parecida posible a la realidad. Esto ha permitido que la efectividad en vuelos reales se asemeje considerablemente a la simulación. Por lo tanto, se puede afirmar que los vuelos simulados preparan de manera eficiente y rápida para vuelos en el mundo real. La simulación ha demostrado ser una herramienta valiosa para el desarrollo y ajuste de los sistemas de control del dron, facilitando una transición suave y efectiva a la implementación práctica.

6.4. Lineas Futuras

6.4.1. Módulo ESP32

En este proyecto se utilizará el módulo ESP32 6.21, conocido por su bajo consumo y grandes capacidades de adaptabilidad. La ESP32 es un microcontrolador altamente versátil que permite la integración de una amplia variedad de módulos adicionales, como LoRa, 4G o 5G. Esto es particularmente útil para aplicaciones



Figura 6.21: ESP32.

Aunque en este proyecto específico no se ha utilizado el ESP32 debido a la funcionalidad actual del dron, se planea su uso en futuros casos donde se necesiten tecnologías adaptativas. Su eficiencia energética y flexibilidad hacen del ESP32 una opción popular en proyectos de automatización y control.

Este microcontrolador soporta múltiples interfaces de comunicación, como Wi-Fi y Bluetooth, permitiendo una conectividad robusta y variada. En aplicaciones futuras, la ESP32 se utilizará para gestionar las comunicaciones y el control del dron, aprovechando su capacidad para integrarse con diferentes tecnologías.

Además de sus capacidades de conectividad, la ESP32 incluye una serie de características avanzadas, como múltiples pines de entrada/salida (GPIO), como se puede ver en la figura 6.22, ADCs, DACs y capacidad de procesamiento dual-core, lo que permite realizar tareas complejas de forma eficiente. Su comunidad de desarrolladores y extensa documentación facilitan su implementación y personalización en una amplia gama de aplicaciones.

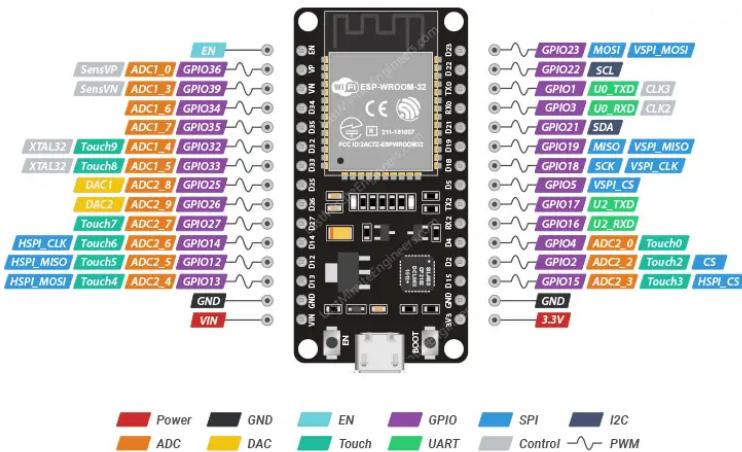


Figura 6.22: ESP32 Pinout.

Estas características hacen del ESP32 una opción ideal para futuros proyectos, buscando aprovechar su bajo consumo, adaptabilidad y el extenso soporte comunitario que respalda su uso. Con la ESP32, nuestro sistema podrá evolucionar para incluir nuevas tecnologías y mejorar sus capacidades, asegurando una solución flexible y de alto rendimiento para nuestras necesidades presentes y futuras.

6.5. Integración de SLAM con Tello

La interfaz de usuario del Tello permite controlar el dron utilizando un algoritmo SLAM o mediante (Roll, Pitch, Yaw). El sistema toma medidas tanto del sensor de altura del dron como del algoritmo SLAM para calibrar y ajustar las coordenadas en el mundo real. Esto garantiza una navegación precisa al utilizar datos de ambas fuentes para determinar la posición y el movimiento del dron [41] (Fig 6.23).

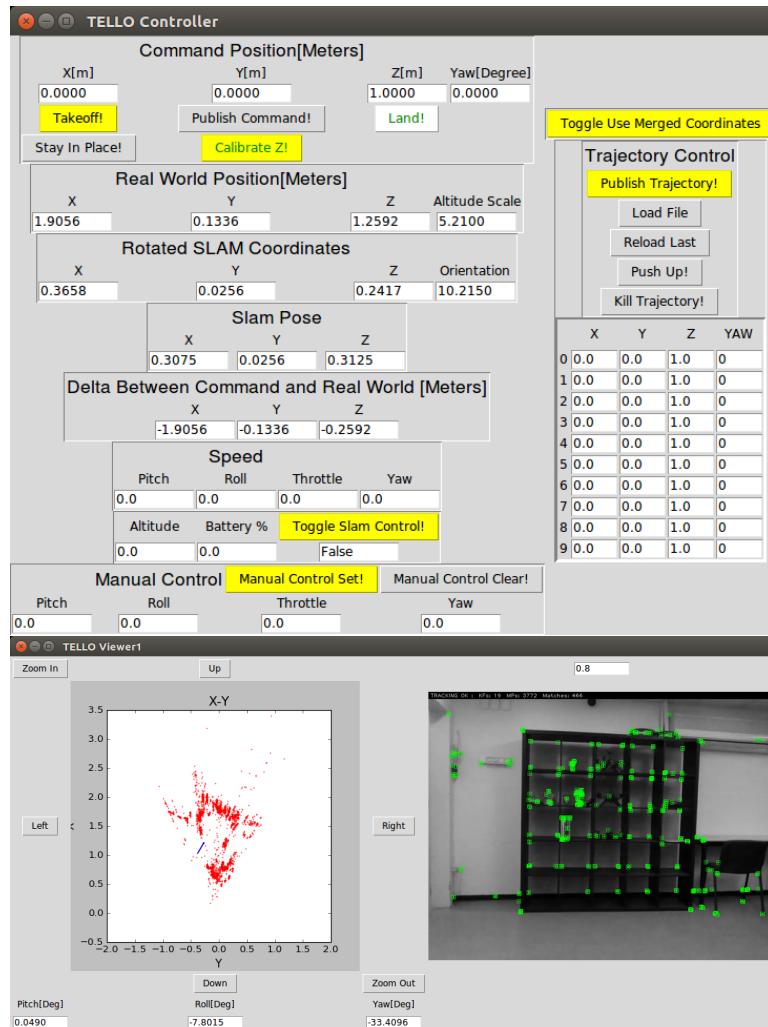


Figura 6.23: SLAM implementado en Tello EDU

6.6. Integración de SLAM con ARUCO en la Navegación Autónoma

La integración de ARUCO con SLAM mejora significativamente la velocidad y optimización del mapeo al fusionar los puntos obtenidos de las etiquetas, además de la localización y orientación de la cámara con los valores que se obtienen de las etiquetas de ARUCO[42] (Fig 6.24).

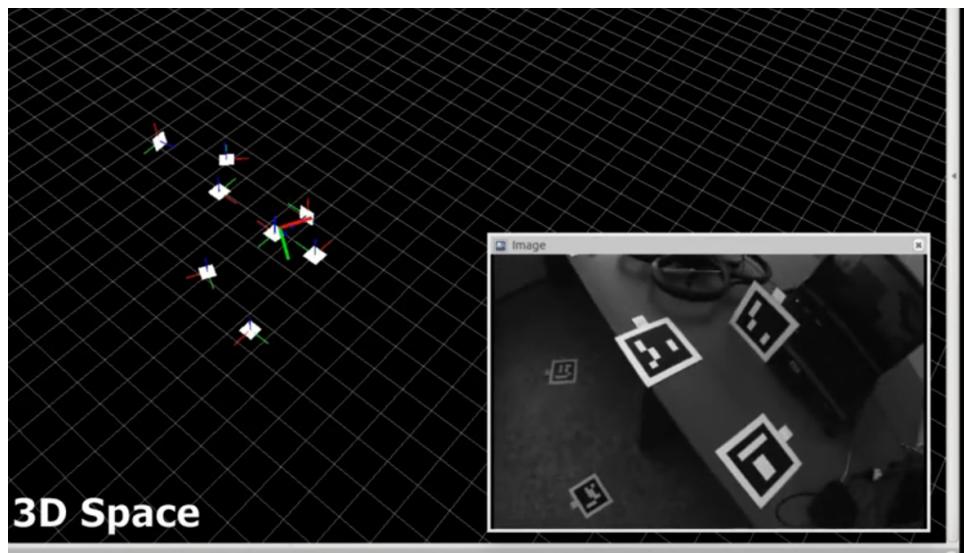


Figura 6.24: SLAM fusionado con ARUCO

6.6.1. Análisis de la mejora en la navegación y mapeo a través de esta integración

La fusión de SLAM con ARUCO, junto con los datos obtenidos del propio dron, mejora considerablemente la calidad del mapeo, la velocidad de creación y reduce los recursos necesarios. Esta integración permite un mapeo más eficiente y preciso, facilitando la navegación autónoma y el control de múltiples drones en un entorno compartido (fig 6.25).

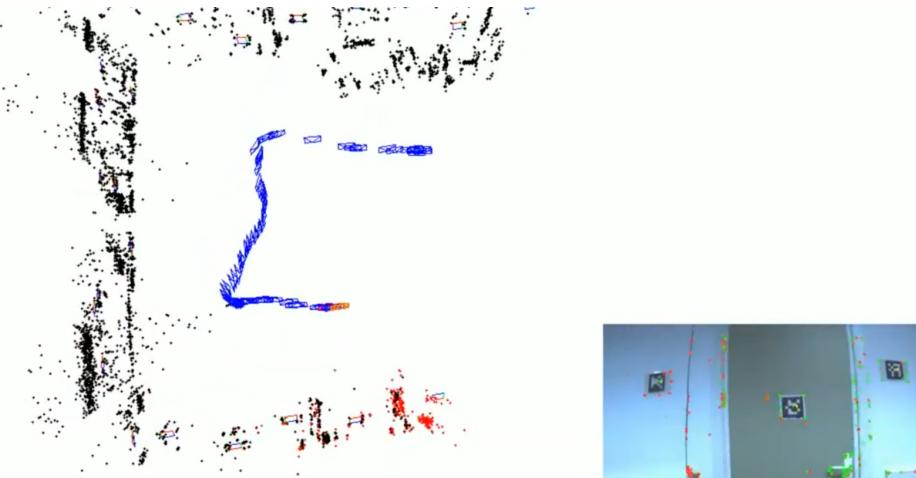


Figura 6.25: SLAM fusionado con ArUco en UAV

6.7. Edge Computing + ESP32

Gracias a la tecnología de Edge Computing y ESP32, se puede hacer que el dron sea semiautónomo. Esto significa que el ESP32, montado en el dron y programado con MicroPython (Fig 6.26), para conectar la ESP32 al dron se necesita imprimir en 3D una plataforma y conectar ambos polos de la batería del dron al ESP32 PINOUT Batería TELLO EDU (Fig 6.27)

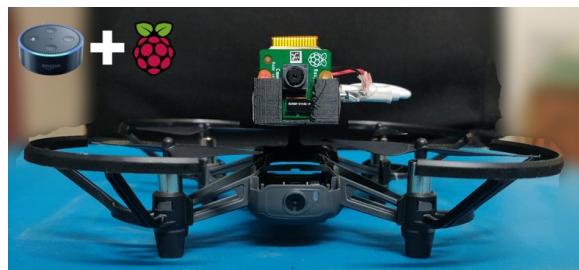


Figura 6.26: ESP32 onboard TELLO



Figura 6.27: PINOUT Batería de TELLO

Esto controlaría las funciones más básicas utilizando solo los datos inmediatos del dron. En caso de detectar algo extraño o fuera de lo normal, el ESP32 comunicaría esta información al sistema de Edge Computing, que tomaría decisiones más complejas y coordinadas. Además, el Edge Computing permite controlar múltiples drones dentro de un mapa completo, mejorando así la eficiencia y efectividad de las operaciones en tiempo real (Fig 6.28).

6.8. Gestión de flotas de UAVs en entornos confinados

La gestión de flotas de UAVs en entornos confinados requiere la implementación de protocolos y sistemas eficientes que permitan la coordinación y operación simultánea de múltiples drones. Estos protocolos aseguran que los drones puedan comunicarse entre sí y con una estación central, optimizando

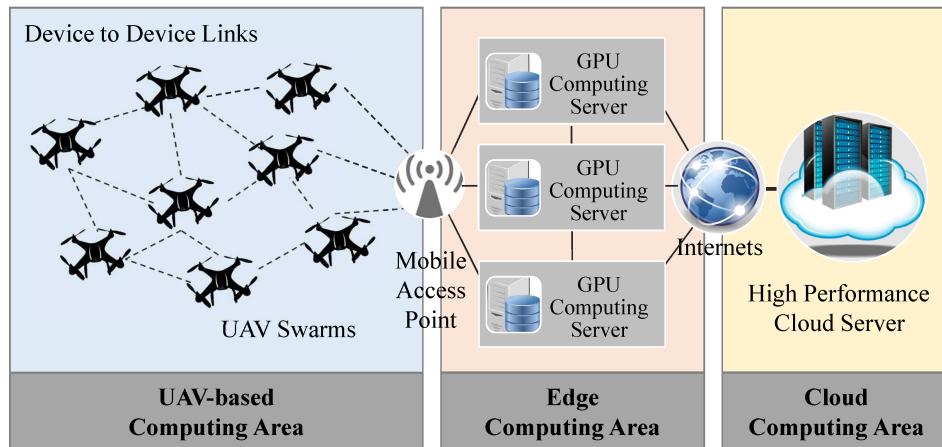


Figura 6.28: Diagrama de EdgeComputing con Drones

el flujo de trabajo y evitando colisiones. La eficiencia en la gestión de flotas es crucial para aplicaciones en áreas como la inspección de infraestructuras, la vigilancia y la entrega de paquetes en interiores.

6.8.1. Solución propuesta para la optimización del flujo de trabajo de la flota

En nuestro proyecto, hemos desarrollado una solución innovadora utilizando el módulo ESP32, que actúa como un gateway para los drones. Este módulo ESP32 puede ser equipado con varios módulos de comunicación, como LoRa, 4G, Bluetooth y WiFi, permitiendo una comunicación flexible y adaptativa con el sistema edge.

El dron Tello EDU, que utilizamos en nuestro proyecto, ya viene con la funcionalidad incorporada de conectarse a puntos de acceso WiFi, lo que facilita su integración en redes existentes sin necesidad de hardware adicional como se puede ver en la figura 3.1 [43]. Sin embargo, hemos considerado la posibilidad de utilizar el módulo ESP32 para drones que no tienen la capacidad de conectarse a puntos de acceso o para situaciones donde se requiere una interconexión a través de tecnologías como LoRa, 4G o Bluetooth.

La solución propuesta ofrece las siguientes ventajas:

- **Flexibilidad en la comunicación:** El módulo ESP32 permite la integración de diferentes tecnologías de comunicación, asegurando que los drones puedan conectarse al mejor medio disponible, ya sea WiFi, 4G, LoRa o Bluetooth.
- **Optimización del flujo de trabajo:** Al utilizar puntos de acceso



Figura 6.29: Diagrama para el control de colmenas mediante AP

comunes y gateways, se mejora la coordinación y gestión de la flota, permitiendo una operación más eficiente y reduciendo el riesgo de colisiones y fallos en la comunicación.

- **Adaptabilidad:** La posibilidad de equipar el ESP32 con diferentes módulos de comunicación permite su uso en una amplia variedad de drones y entornos, facilitando la escalabilidad del sistema.

Esta solución no solo mejora la gestión de flotas de UAVs en entornos confinados, sino que también asegura una operación más robusta y fiable, adaptándose a las necesidades específicas de cada aplicación y entorno.

Anexo

Peso y dimensiones			
Ancho	92 mm	Profundidad	98 mm
Anchura	98 mm	Altura	41 mm
Peso (con batería)	80 g	Diámetro rotor	7,62 mm
Cámara			
Megapíxeles	5 MP	Máxima resolución de video	HD: 1280 x 720 30p
Máximo tamaño imagen	2592 x 1936	Ángulo de campo de visión (FOV)	82,6º
Formato de video soportado	720p	Formato de video compatible	MP4
Batería			
Tipo	LiPo	Capacidad	1100 mAh
Voltaje	3,8 V	Energía	4,18 Wh
Potencia de carga máx.	10 W	Tiempo de funcionamiento máx.	13 min
Otras características			
Número de rotores	4	Velocidad máxima	8 m/s
Distancia máxima	100 m	WiFi	SI
Microprocesador	SI	Frecuencia de banda	2,4 a 2,8 GHz
GPS	NO	Sensores de distancia	NO

Tabla 6.3: Datasheet Drone DJI Tello EDU.

ID (Hex)	Tello Function	Dir	Comentarios
0x0001	Connect	→	Pseudo-comando enviado como texto plano
0x0002	Connected	←	Pseudo-comando recibido como texto plano
0x0011	Query SSID	↔	
0x0012	Set SSID	→	
0x0013	Query SSID Password	→	
0x0014	Set SSID Password	→	
0x0020	Set Video Bit-Rate	→	
0x0025	Request Video Start	→	
0x0032	Start Recording	→	
0x0045	Query Version	↔	
0x0046	Set Date & Time	↔	
0x0050	Set Sticks	→	Tello necesita estos regularmente como un 'latido'
0x0054	Take Off	↔	Despegue normal y ascenso a aproximadamente 1.8m agl
0x0055	Land	↔	

Tabla 6.4: Tello Message IDs and Meanings (Part 1)

ID (Hex)	Tello Function	Dir	Comentarios
0x0058	Set Height Limit	→	
0x005c	Flip	→	
0x005d	Throw Take Off	→	
0x005e	Palm Land	→	
0x0062	File Size	←	por ejemplo, para un fragmento de foto
0x0063	File Data	←	por ejemplo, fragmento de foto
0x0064	File Done	←	también conocido como EOF. por ejemplo, final de datos de foto
0x0080	Start Smart Video	→	
0x0081	Smart Video Status	←	
0x1050	Log Header	↔	
0x1051	Log Data	←	
0x1052	Log Config.	←	
0x1054	Calibration	↔	Payload (1 byte): 0 para calibrar IMU, 1 para calibrar el Centro de Gravedad (debe estar flotando)
0x1055	Set Low Battery Threshold	↔	Algunos comandos se ignoran por debajo del umbral
0x1056	Query Height Limit	↔	
0x1057	Query Low Battery Threshold	↔	
0x1058	Query Attitude (Limit?)	→	
0x1059	Set Attitude (Limit?)	→	

Tabla 6.5: Tello Message IDs and Meanings (Part 2)

Función	Descripción
<code>set_loglevel(level)</code>	Controla los mensajes de salida. Los niveles válidos son <code>LOG_ERROR</code> , <code>LOG_WARN</code> , <code>LOG_INFO</code> , <code>LOG_DEBUG</code> y <code>LOG_ALL</code> .
<code>get_video_stream()</code>	Prepara el objeto buffer para recibir datos de video del dron.
<code>connect()</code>	Envía la solicitud de conexión inicial al dron.
<code>wait_for_connection(timeout=None)</code>	Bloquea hasta que se establece la conexión.
<code>subscribe(signal, handler)</code>	Suscribe a un evento como <code>EVENT_CONNECTED</code> , <code>EVENT_FLIGHT_DATA</code> , <code>EVENT_VIDEO_FRAME</code> , etc.
<code>takeoff()</code>	Ordena al dron despegar y comenzar a volar.
<code>throw_and_go()</code>	Inicia una secuencia de lanzamiento y vuelo.
<code>land()</code>	Ordena al dron aterrizar.
<code>palm_land()</code>	Indica al dron que espere una mano debajo y luego aterrice.
<code>quit()</code>	Detiene los hilos internos.
<code>get_low_bat_threshold()</code>	Obtiene el umbral de batería baja.
<code>set_video_encoder_rate(rate)</code>	Establece la tasa de codificación de video del dron.
<code>take_picture()</code>	Ordena al dron tomar una foto.
<code>up(val)</code>	Ordena al dron ascender. El valor debe estar entre 0 y 100.
<code>down(val)</code>	Ordena al dron descender. El valor debe estar entre 0 y 100.

Tabla 6.6: Funciones y Descripciones de Tellopy

Función	Descripción
<code>set_throttle(throttle)</code>	Controla el movimiento vertical del dron. El valor debe estar entre -1.0 y 1.0 (positivo hacia arriba).
<code>set_yaw(yaw)</code>	Controla la rotación izquierda y derecha del dron. El valor debe estar entre -1.0 y 1.0 (positivo hacia la derecha).
<code>set_pitch(pitch)</code>	Controla la inclinación hacia adelante y atrás del dron. El valor debe estar entre -1.0 y 1.0 (positivo hacia adelante).
<code>set_roll(roll)</code>	Controla la inclinación lateral del dron. El valor debe estar entre -1.0 y 1.0 (positivo hacia la derecha).
<code>toggle_fast_mode()</code>	Alterna el modo rápido del dron.

Tabla 6.7: Funciones y Descripciones de Tellopy

Funcionalidad	Descripción
EVENT_CONNECTED	Evento de conexión establecida.
EVENT_WIFI	Evento de actualización de señal WiFi.
EVENT_LIGHT	Evento de cambio de estado de la luz.
EVENT_FLIGHT_DATA	Evento de actualización de datos de vuelo.
EVENT_LOG_HEADER	Evento de cabecera de registro.
EVENT_LOG	Evento de registro (alias para EVENT_LOG_HEADER).
EVENT_LOG_RAWDATA	Evento de datos crudos de registro.
EVENT_LOG_DATA	Evento de datos de registro procesados.
EVENT_LOG_CONFIG	Evento de configuración de registro.
EVENT_TIME	Evento de actualización de tiempo.
EVENT_VIDEO_FRAME	Evento de recepción de un cuadro de video.
EVENT_VIDEO_DATA	Evento de recepción de datos de video.
EVENT_DISCONNECTED	Evento de desconexión.
EVENT_FILE RECEIVED	Evento de archivo recibido.
STATE_DISCONNECTED	Estado desconectado.
STATE_CONNECTING	Estado conectando.
STATE_CONNECTED	Estado conectado.
STATE_QUIT	Estado de finalización.
LOG_ERROR	Nivel de log: Error.
LOG_WARN	Nivel de log: Advertencia.
LOG_INFO	Nivel de log: Información.
LOG_DEBUG	Nivel de log: Depuración.
LOG_ALL	Nivel de log: Todo.

Tabla 6.8: Eventos y Estados de Tellopy

Variable	Descripción
battery_low	Indicador de batería baja.
battery_lower	Indicador de batería muy baja.
battery_percentage	Porcentaje de batería.
battery_state	Estado de la batería.
camera_state	Estado de la cámara.
down_visual_state	Estado del sensor visual inferior.
drone_battery_left	Batería restante del dron.
drone_fly_time_left	Tiempo de vuelo restante del dron.
drone_hover	Indicador de suspensión del dron.
em_open	Indicador de emergencia abierta.
em_sky	Indicador de emergencia en el aire.
em_ground	Indicador de emergencia en tierra.
east_speed	Velocidad hacia el este.
electrical_machinery_state	Estado de la maquinaria eléctrica.
factory_mode	Modo de fábrica.
fly_mode	Modo de vuelo.
fly_speed	Velocidad de vuelo.
fly_time	Tiempo de vuelo.
front_in	Estado del sensor frontal.
front_lsc	Estado del sensor LSC frontal.
front_out	Estado del sensor frontal externo.
gravity_state	Estado del sensor de gravedad.
ground_speed	Velocidad sobre el suelo.
height	Altura.
imu_calibration_state	Estado de calibración del IMU.
imu_state	Estado del IMU.
light_strength	Intensidad de la luz.
north_speed	Velocidad hacia el norte.
outage_recording	Registro de apagones.
power_state	Estado de la alimentación.
pressure_state	Estado de la presión.
smart_video_exit_mode	Modo de salida de video inteligente.
temperature_height	Altura de la temperatura.
throw_fly_timer	Temporizador de lanzamiento.
wifi_disturb	Interferencia de WiFi.
wifi_strength	Intensidad de la señal WiFi.
wind_state	Estado del viento.

Tabla 6.9: Variables de la clase FlightData

Listing 6.5: Detectar objetos usando Yolov5

```

1 import cv2
2 import torch
3 import numpy as np
4 import matplotlib.path as mplPath
5
6 def get_center(bbox):
7     # xmin, ymin, xmax, ymax
8     center = ((bbox[0] + bbox[2]) // 2, (bbox[1] + bbox[3]) // 2)
9     return center
10
11 def load_model():
12     print("Loading model...")
13     model = torch.hub.load("ultralytics/yolov5", model="yolov5n",
14                             pretrained=True)
15     print("Model loaded successfully!")
16     return model
17
18 def get_bboxes(preds: object):
19     # xmin, ymin, xmax, ymax, confidence, name
20     df = preds.pandas().xyxy[0]
21     df = df[df["confidence"] >= 0.3]
22     return df[["xmin", "ymin", "xmax", "ymax", "confidence", "name"]]
23
24 def detector(frame, model):
25     print("Running inference...")
26     preds = model(frame)
27     print("Inference completed.")
28
29     bboxes = get_bboxes(preds)
30
31     detections = 0
32     for _, row in bboxes.iterrows():
33         box = row[["xmin", "ymin", "xmax", "ymax"]].astype(int)
34         xc, yc = get_center(box)
35
36         if is_valid_detection(xc, yc):
37             detections += 1
38
39         cv2.circle(img=frame, center=(xc, yc), radius=5, color
40                    =(0,255,0), thickness=-1)
41         cv2.rectangle(img=frame, pt1=(box[0], box[1]), pt2=(box[2],
42                                box[3]), color=(255, 0, 0), thickness=1)
43         label = f"{row['name']}: {row['confidence']:.2f}"
44         cv2.putText(img=frame, text=label, org=(box[0], box[1] - 10),
45                     fontFace=cv2.FONT_HERSHEY_PLAIN, fontScale=1, color
46                     =(255,0,0), thickness=1)
47
48     return frame

```

Listing 6.6: Flujo de la GUI

```

1 def draw_panel(img, markers_orientations=None, camera_matrix=None,
2 dist_coeffs=None, yaw=0, mapping_mode=False, stored_distances=None,
3 avg_distances=None):
4     # Código para dibujar el panel de control
5     # ...
6     # Dibuja el minimapa
7     map_center = (500, 100)

```

```
8 minimap_radius = 70
9
10 # Draw minimap circle
11 cv.circle(img, map_center, minimap_radius, (255, 0, 0), 2)
12 cv.circle(img, map_center, outer_radius, (255, 255, 255), 1)
13 # Draw a red dot at the center of the minimap
14 cv.circle(img, map_center, 5, (0, 0, 255), -1)
15
16 # Dibuja el indicador de norte
17 north_x = int(map_center[0] + outer_radius * cos(north_angle))
18 north_y = int(map_center[1] - outer_radius * sin(north_angle))
19 cv.putText(img, 'N', (north_x, north_y), cv.FONT_HERSHEY_SIMPLEX
20     , ...
21
22 # Dibujar el plano general basado en los datos de las etiquetas
23 if avg_distances:
24     # Código para dibujar el plano
25     # ...
26
27 if mapping_mode:
28     # Código para dibujar estado Mapping y guardar distancias
29
30 if not mapping_mode and avg_distances:
31     # Código para dibujar planos 2D
32
33 return avg_distances
34
35 # En Código para dibujar estado Mapping y guardar distancias
36 avg_distances = draw_panel(.....)
```

Listing 6.7: Flujo del Control de Seguimiento

```
1 def follow_marker(pitch_actual, roll_actual, throttle_actual,
2 marker_detected, tvec, estado, id, marker_data, yaw1):
3     global contador_pitch_cero, roll_contador_cero,
4         contador_throttle_cero,
5     last_marker_seen_time, last_marker_4_seen_time
6     global last_valid_pitch, last_valid_roll, last_valid_throttle,
7         yaw_desired,
8     initial_yaw1, yaw_accumulated, yaw_in_range_count, pass_start_time
9         , yaw_target
10    global mapping_mode, mapping_completed
11
12 pitch = 0
13 roll = 0
14 throttle = 0
15 yaw = 0
16 current_time = robot.getTime()
17 pi = np.pi
18
19 print(estado)
20
21 if '5' in marker_data:
22     # Extraer las coordenadas x, y y la distancia z del marcador 5
23     pass
24
25 if id == 4:
26     # Actualizar el tiempo de detección de la etiqueta 4
27     pass
```

```
26     if estado == "centrado_roll":
27         if marker_detected and (id & 0b111) == 0b101:
28             distancia_minima_roll = -200
29             distancia_maxima_roll = 150
30             roll_max = 0.3
31             roll_min = 0.1
32
33         if roll_contador_cero >= 16:
34             # Se mantiene el roll en 0 despues de 16 veces
35             # consecutivas
36             pass
37         else:
38             if x > distancia_maxima_roll:
39                 # Control proporcional suavizado si x esta fuera
40                 # del limite maximo
41                 pass
42             elif x < distancia_minima_roll:
43                 # Control proporcional suavizado si x esta fuera
44                 # del limite minimo
45                 pass
46             else:
47                 # Roll ajustado a 0
48                 pass
49
50             if distancia_minima_roll <= x <= distancia_maxima_roll
51                 :
52                 # Incrementar contador si el dron esta centrado en
53                 # el eje X
54                 pass
55             else:
56                 # Resetear contador si el dron no esta centrado
57                 pass
58
59             last_marker_seen_time = current_time
60             last_valid_roll = roll_objetivo # Guardar el ultimo roll
61             valido
62
63         # Suavizar la transicion del roll actual al roll objetivo
64         pass
65
66         if (current_time - last_marker_seen_time) <
67             MARKER_LOST_TIMEOUT:
68             # Ajustar el roll del dron
69             pass
70
71     elif estado == "centrado_throttle":
72         # Misma Filosofia que estado "centrado_roll" pero eje Y
73
74     elif estado == "control_pitch":
75         # Misma Filosofia que estado "centrado_roll" pero eje Z
76
77     elif estado == "control_yaw":
78         yaw_speed = 0.3
79
80         if '5' in marker_data:
81             yaw5 = marker_data['5']['yaw']
```

```
82     if initial_yaw1 is None:
83         # Establecer yaw inicial
84         pass
85
86     # Calcular delta de yaw
87     pass
88
89     yaw_accumulated += abs(delta_yaw)
90     initial_yaw1 = yaw1
91
92     if yaw_target is None:
93         # Establecer objetivo de yaw
94         pass
95
96     if -10 <= yaw5 <= 10:
97         # Incrementar contador si yaw esta en rango deseado
98         pass
99
100    if yaw_in_range_count >= 1:
101        # Detener el dron si yaw esta en rango deseado por mas
102        # de 10 veces
103        pass
104
105    if yaw_accumulated < yaw_target:
106        # Ajustar yaw del dron
107        pass
108    else:
109        # Ajustar yaw y cambiar estado
110        pass
111
112 elif estado == 'mapping':
113     # Ajustar yaw para buscar marcador
114     pass
115     if marker_detected and (id & 0b111) == 0b101:
116         # Cambiar estado si se detecta marcador
117         pass
118
119 elif estado == 'stop':
120     # Detener el dron
121     pass
122     if id == 5:
123         estado = "control_pitch_door"
124     elif id == 4:
125         # Cerrar puerta
126         pass
127
128 elif estado == "control_pitch_door":
129     # Misma Filosofia que estado "centrado_pitch" pero cambiando
130     Distancia max y
131     Distancia Min
132
133 elif estado == "centrado_roll_door":
134     # Misma Filosofia que estado "centrado_roll" pero cambiando
135     Distancia max y
136     Distancia Min
137
138 elif estado == "stop_pass":
139     if marker_detected and (id & 0b111) == 0b101:
140         # Ajustar throttle objetivo
141         pass
142         last_marker_seen_time = current_time
143         last_valid_throttle = throttle_objetivo # Guardar el
```

```

143         ultimo
144         throttle valido
145     else:
146         # Usar el ultimo throttle valido si no se detecta el
147         # marcador
148         pass
149         # Cambiar a estado 'centrado_roll_door' si la etiqueta 5
150         # no es
151         detectada por mas de 0.5 segundos
152         pass
153
154         # Suavizar la transicion del throttle actual al throttle
155         # objetivo
156         pass
157
158     elif estado == 'pass':
159         if last_marker_4_seen_time is not None and
160             (current_time - last_marker_4_seen_time) <
161                 MARKER_LOST_TIMEOUT:
162                 # Ajustar el throttle del dron
163                 pass
164
165         elif estado == 'pitch':
166             pitch_speed = 0.6
167
168         if pass_start_time is None:
169             # Establecer tiempo de inicio del pase
170             pass
171
172         if current_time - pass_start_time < 6:
173             # Mantener el pitch durante el tiempo deseado
174             pass
175         else:
176             # Completar el pase y cambiar de estado
177             pass
178
179     return pitch, roll, throttle, estado
180
181 #Codigo para dibujar estado Mapping y guardar distancias
182 pitch_actual, roll_actual, throttle_actual, estado = follow_marker
183     (...)
```

Listing 6.8: Control del Yaw

```

1 if mapping_completed = False and mapping_mode = True
2     set_yaw(-1)
3     if initial_yaw is not None:
4         delta_yaw = yaw1 - initial_yaw
5         if delta_yaw > pi:
6             delta_yaw -= 2 * pi
7         elif delta_yaw < -pi:
8             delta_yaw += 2 * pi
9         yaw_accumulated += abs(delta_yaw)
10        initial_yaw = yaw1
11
12        if yaw_accumulated >= 4 * pi: # Dos vueltas
13            mapping_mode = False
```

```

14         mapping_completed = True
15         set_yaw(0)
16         yaw_accumulated = 0
17         print("Mapping mode: OFF")

```

Listing 6.9: Control del estado de Mapeo

```

1 if mapping_mode:
2     if marker_detected:
3         ...
4         for i, aruco_id in enumerate(ids.flatten()):
5             ...
6                 avg_distances = draw_panel(....)
7             else:
8                 avg_distances = draw_panel(....)
9
10
11
12 if mapping_completed and not mapping_mode:
13     if marker_detected:
14         ...
15         for i, id in enumerate(ids):
16             ...
17                 pitch_actual, roll_actual, throttle_actual, estado =
18                     follow_marker(....)
19             avg_distances = draw_panel(....)
20         else:
21             pitch_actual, roll_actual, throttle_actual, estado
22             = follow_marker(....)
23             avg_distances = draw_panel(....)

```

Listing 6.10: Código para enviar datos usando API de telegram.

```

1 import requests
2
3 # Reemplaza estos valores con tu token y chat_id
4 BOT_TOKEN = "tu_token_aqui"
5 CHAT_ID = "tu_chat_id_aqui"
6
7 def send_alert(video_path):
8     # Enviar un mensaje de texto
9     url = f"https://api.telegram.org/bot{BOT_TOKEN}/sendMessage"
10    data = {
11        "chat_id": CHAT_ID,
12        "text": " Alerta ! Se detecto una persona."
13    }
14    requests.post(url, data=data)
15
16    # Enviar un video
17    url = f"https://api.telegram.org/bot{BOT_TOKEN}/sendVideo"
18    with open(video_path, 'rb') as video:
19        data = {
20            "chat_id": CHAT_ID
21        }
22        files = {
23            "video": video
24        }
25        requests.post(url, data=data, files=files)

```

Listing 6.11: Control del sistema de Alertas

```

1 def detector(frame, model, last_detection_time, start_record_time,
2             recording, video_writer):
3     preds = model(frame)
4     bboxes = get_bboxes(preds)
5
6     current_time = time.time()
7     person_detected = False
8
9     # Verifica si hay detección de persona en el frame actual
10    for _, row in bboxes.iterrows():
11        box = row[["xmin", "ymin", "xmax", "ymax"]].astype(int)
12        xc, yc = get_center(box)
13
14        if row["name"] == "person":
15            person_detected = True
16            # Solo actualizamos last_detection_time si es la primera
17            # detección
18            if not recording and last_detection_time == 0:
19                last_detection_time = current_time
20
21            # Dibujar las cajas y centros
22            cv2.circle(img=frame, center=(xc, yc), radius=5, color
23                      =(0,255,0), thickness=-1)
24            cv2.rectangle(img=frame, pt1=(box[0], box[1]), pt2=(box[2],
25                          box[3]), color=(255, 0, 0), thickness=1)
26            label = f"{row['name']}: {row['confidence']:.2f}"
27            cv2.putText(img=frame, text=label, org=(box[0], box[1] - 10),
28                        fontFace=cv2.FONT_HERSHEY_PLAIN, fontScale=1, color
29                        =(255,0,0), thickness=1)
30
31    # Lógica para iniciar la grabación
32    if not recording and person_detected:
33        if last_detection_time == 0:
34            last_detection_time = current_time # Inicializamos el
35            # tiempo de detección
36        elif (current_time - last_detection_time) >=
37            DETECTION_THRESHOLD:
38            # Si se detecta una persona por m s de
39            # DETECTION_THRESHOLD segundos, empieza a grabar
40            recording = True
41            start_record_time = current_time
42            video_writer = cv2.VideoWriter('detection.mp4', cv2.
43                VideoWriter_fourcc(*'mp4v'), 10, (frame.shape[1],
44                frame.shape[0]))
45
46    # Lógica para mantener la grabación
47    if recording:
48        video_writer.write(frame)
49        if person_detected:
50            last_detection_time = current_time # Reinicia el tiempo
51            de detección si sigue detectando personas
52        else:
53            # Si no se detecta a una persona, verifica si el tiempo
54            # transcurrido est dentro de la tolerancia
55            if (current_time - last_detection_time) >
56                RECORDING_DURATION_AFTER_LOSS:
57                # Termina la grabación si no se detectan personas
58                # durante m s del tiempo permitido
59                recording = False
60                video_writer.release()
61                last_detection_time = 0

```

```
47         send_alert('detection.mp4')
48
49     return frame, last_detection_time, start_record_time, recording,
        video_writer
```

Glosario

BW Bandwidth o Ancho de banda. 44

EASA Agencia de la Unión Europea para la Seguridad Aérea. 12

FAA Federal Aviation Administration. 42

GPS Global Positioning System o Sistema de Posicionamiento Global. 12

GUI graphical User Interface o Interfaz Gráfica de Usuario. 75

IDE Integrated Development Environment o Entorno de Desarrollo Integrado. 58

LIDAR Acrónimo de *Light Detection and Ranging*, es una tecnología de medición remota que utiliza pulsos de luz láser para medir distancias precisas hacia un objeto o superficie. Se emplea en cartografía, geodesia, vehículos autónomos, entre otras aplicaciones. 64

R-CNN Region-based Convolutional Neural Networks, una familia de redes neuronales convolucionales para la detección de objetos. 40

SDK Software Development Kit, un conjunto de herramientas de desarrollo de software. 20, 48

SSD Single Shot MultiBox Detector, un método para la detección de objetos en imágenes. 40

UAVs Unmanned Aerial Vehicles o Vehículos Aéreos no Tripulados. 2–4, 11, 15, 42

YOLO You Only Look Once, un algoritmo de detección de objetos en tiempo real. 2, 33, 36

Bibliografía

- [1] Toton Chandra Mallick, Mohammad Ariful Islam Bhuyan, and Mohammed Saifuddin Munna. Design & implementation of an uav (drone) with flight data record. pages 1–6, 2016.
- [2] Anduril. Anduril industries. <https://www.anduril.com/>.
- [3] bostondynamics. Workplace Safety and Security — Boston Dynamics. <https://bostondynamics.com/solutions/site-management/workplace-safety-security/>.
- [4] AlexeyAB. YOLOv4 / Scaled-YOLOv4 / YOLO - Neural Networks for Object Detection. <https://github.com/AlexeyAB/darknet>, 2017.
- [5] WongKinYiu. Implementation of paper - YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. <https://github.com/WongKinYiu/yolov7>, 2022.
- [6] Pedro Cubria de Gorostegui. El exitoso uso de los drones de Ikea que grandes empresas como Amazon no supieron ver — elconfidencial.com. https://www.elconfidencial.com/tecnologia/2024-09-02/drones-ikea-verity-amazon-robots-almacen-inventario_3947940/.
- [7] Anthony Morgan and Christopher Dowling. Physical security: Video surveillance, equipment, and training. In *Encyclopedia of Security and Emergency Management*, pages 766–775. Springer, 2021.
- [8] Nancy G La Vigne, Samantha S Lowry, Joshua A Markman, and Allison M Dwyer. Evaluating the use of public surveillance cameras for crime control and prevention. *Washington, DC: US Department of Justice, Office of Community Oriented Policing Services. Urban Institute, Justice Policy Center*, pages 1–152, 2011.
- [9] Gustav Alexandrie. Surveillance cameras and crime: A review of randomized and natural experiments. *Journal of Scandinavian Studies in Criminology and Crime Prevention*, 18(2):210–222, 2017.

- [10] EASA. Open Category - Low Risk - Civil Drones. <https://www.easa.europa.eu/en/domains/civil-drones-rpas/open-category-civil-drones>, 2020.
- [11] Prosegur. Yellow, el perro robot de Prosegur. <https://www.prosegur.es/media/articulo/prensa/yellow-perro-robot-prosegur-salta-nuevamente-a-mutua-madrid-open>, 2023.
- [12] Tello. Spark. https://dl.djicdn.com/downloads/Spark/Spark_User_Manual_V1.2_ES.pdf.
- [13] bitcraze.io. Crazyflie. https://www.bitcraze.io/documentation/hardware/crazyflie_2_1/crazyflie_2_1-datasheet.pdf.
- [14] Parrot. Bebop 2. https://www.parrot.com/assets/s3fs-public/2021-09/bebop-2_user-guide_uk.pdf.
- [15] PinguSoft. Low-Level Protocol. <https://tellopilots.com/wiki/protocol/>, 2019.
- [16] Hanyazou. GitHub - hanyazou/TelloPy: DJI Tello drone controller python packag. <https://github.com/hanyazou/TelloPy>, 2019.
- [17] Francisco Romero-Ramirez, Rafael Muñoz-Salinas, and Rafael Medina-Carnicer. Speeded up detection of squared fiducial markers. *Image and Vision Computing*, 76, 06 2018.
- [18] Sergio Garrido-Jurado, Rafael Muñoz-Salinas, Francisco Madrid-Cuevas, and Rafael Medina-Carnicer. Generation of fiducial marker dictionaries using mixed integer linear programming. *Pattern Recognition*, 51, 10 2015.
- [19] Edwin Olson. Apriltag: A robust and flexible visual fiducial system. In *2011 IEEE international conference on robotics and automation*, pages 3400–3407. IEEE, 2011.
- [20] Mark Fiala. Comparing artag and artoolkit plus fiducial marker systems. In *IEEE International Workshop on Haptic Audio Visual Environments and their Applications*, pages 6–pp. IEEE, 2005.
- [21] Quentin Bonnard, Severin Lemaignan, Guillaume Zufferey, Andrea Mazzei, Sébastien Cuendet, Nan Li, Ayberk Özgür, and Pierre Dillenbourg. Chilitags 2: Robust fiducial markers for augmented reality and robotics. *CHILI, EPFL, Switzerland.-2013*, 2013.
- [22] Alexander Panov Sergio Garrido. Detection of ArUco Markers. https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html, 2016.

- [23] Rafael Muñoz Salinas. ArUco markers and boards detection for robust camera pose estimation. https://docs.opencv.org/4.x/de/d67/group__objdetect__aruco.html#ga2ad34b0f277edebb6a132d3069ed2909, 2017.
- [24] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolo: Optimal speed and accuracy of object detection, 2020.
- [25] Camillo Lugaressi, Jiuqiang Tang, Hadon Nash, Chris McClanahan, Esha Ubweja, Michael Hays, Fan Zhang, Chuo-Ling Chang, Ming Guang Yong, Juhyun Lee, et al. Mediapipe: A framework for building perception pipelines. *arXiv preprint arXiv:1906.08172*, 2019.
- [26] Ashwani Kumar, Zuopeng Justin Zhang, and Hongbo Lyu. Object detection in real time based on improved single shot multi-box detector algorithm. *EURASIP Journal on Wireless Communications and Networking*, 2020(1):204, 2020.
- [27] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE transactions on pattern analysis and machine intelligence*, 39(6):1137–1149, 2016.
- [28] Glenn Jocher. YOLOv5 by Ultralytics. <https://github.com/ultralytics/yolov5>, 5 2020.
- [29] Khaled R. Ahmed. Smart pothole detection using deep learning based on dilated convolution. *Sensors*, 21, 12 2021.
- [30] Ahmed Jamal Abdullah Al-Gburi, Zahriladha Zakaria, Hussein Alsariera, Muhammad Firdaus Akbar, Imran Mohd Ibrahim, Khalid Subhi Ahmad, Sarosh Ahmad, and Samir Salem Al-Bawri. Broadband circular polarised printed antennas for indoor wireless communication systems: A comprehensive review. *Micromachines*, 13(7):1048, 2022.
- [31] Cyberbotics. Cyberbotics: Robotics simulation with Webots. <https://cyberbotics.com/>.
- [32] Gazebo. Gazebo. <https://gazebosim.org>.
- [33] Microsoft. Flight Simulator - The next generation of one of the most beloved simulation. <https://www.flightsimulator.com/>.
- [34] Community. Developing for the Tello. <https://tellopilots.com/wiki/development/>, 2019.
- [35] ryzerobotics. SDK 2.0 Tello EDU. <https://dl-cdn.ryzerobotics.com/downloads/Tello/Tello%20SDK%202.0%20User%20Guide.pdf>.

- [36] DJI Tello Drone Forum — tellopilots.com. <https://tellopilots.com/>.
- [37] Anne Steenbeek. Cnn based dense monocular visual slam for indoor mapping and autonomous exploration, May 2020.
- [38] YM Wang, Y Li, and JB Zheng. A camera calibration technique based on opencv. In *The 3rd International Conference on Information Sciences and Interaction Sciences*, pages 403–406. IEEE, 2010.
- [39] Ultralytics. Train Custom Data — docs.ultralytics.com. https://docs.ultralytics.com/yolov5/tutorials/train_custom_data/#before-you-start.
- [40] Telegram. Telegram Bot API — core.telegram.org. <https://core.telegram.org/bots/api>.
- [41] tau adl. full and whole framework for using Drones in general, and the DJI Tello specifically. https://github.com/tau-adl/Tello_ROS_ORBSLAM?tab=readme-ov-file, 2018.
- [42] Rafael Muñoz-Salinas and R. Medina-Carnicer. Ucoslam: Simultaneous localization and mapping by fusion of keypoints and squared planar markers. *Pattern Recognition*, page 107193, 2020.
- [43] JA Jimenez Cavadas. Using drone tello edu for educational purposes. *MSc Theses, Universidad Politécnica de Cataluña*, pages 35 – 50, 2019.