Universidad de la República


# Inteligencia computacional aplicada a vigilancia con una flota de drones

Proyecto de grado
Ingeniería en Computación

Autores

José Martín Pérez González
Juan Manuel Roquero Bravo
Timothy Daniel Peraza Geist

Supervisores

Sergio Nesmachnow
Santiago Iturriaga

Montevideo, Uruguay — Noviembre, 2019

*El éxito no es un accidente. Es trabajo duro, perseverancia, aprendizaje, estudio, sacrificio y, sobre todo, el amor por lo que estás haciendo o aprendiendo a hacer* - Edson Arantes do Nascimento, "Pele".

# Agradecimientos

*En primer lugar, damos las gracias a nuestras familias que nos han dado su apoyo incondicional durante toda la carrera universitaria. Así como también a nuestros amigos/compañeros de carrera por ser parte fundamental de este viaje transitado. Finalmente darle las gracias a nuestros supervisores, por brindarnos su ayuda y permitirnos ser parte de este proyecto.*

# Resumen

Un drone (Unmanned Aerial Vehicle, UAV) es un vehículo aéreo no tripulado capaz de volar de forma autónoma o controlado remotamente. En comparación con emplear un único UAV para misiones de vigilancia y/o reconocimiento, un mecanismo que utilice una flota tiene importantes ventajas. Una flota de UAV utilizando un enfoque colaborativo permite aumentar el área o reducir el tiempo requerido de una misión. Sin embargo, la coordinación con un propósito colaborativo introduce complejidades inherentes al control de vuelo, en especial cuando los UAVs tiene que actuar de forma autónoma. Este proyecto aborda el diseño de algoritmos de inteligencia computacional para la programación de vuelos autónomos de una flota de drones, particularmente en una misión de vigilancia en un terreno conocido y limitado. La solución presentada hace foco en la coordinación de los UAV, la toma de decisiones ante eventos externos e internos, el reconocimiento de imágenes y el trabajo colaborativo en la búsqueda de intrusos. El análisis experimental demuestra la eficacia de la solución propuesta al problema de la vigilancia de un terreno conocido con una flota autónoma de UAVs.

**Palabras clave:** Drones, Flota, Vuelos autónomos, *UAV*, Inteligencia computacional, Intrusos.

# Índice general

**Índice de cuadros** xi

**Índice de figuras** xiii

**1. Introducción** 1

**2. El problema de vigilancia con una flota de drones** 5
2.1. Introducción . . . . . 5
2.2. Descripción del problema . . . . . 7
2.3. Trabajos relacionados . . . . . 8

**3. Materiales y tecnologías** 19
3.1. Aspectos básicos de los cuadricópteros . . . . . 19
3.1.1. Componentes de los cuadricópteros . . . . . 20
3.1.2. Primitivas de movimiento . . . . . 22
3.1.3. Parrot Bebop 2 . . . . . 24
3.2. ARM y Raspberry . . . . . 26
3.2.1. ARM . . . . . 26
3.2.2. Raspberry Pi 2 Modelo B . . . . . 28
3.3. Elección del lenguaje de programación . . . . . 32
3.3.1. Lenguaje C . . . . . 32
3.3.2. Lenguaje Python . . . . . 34
3.4. Procesamiento de imágenes . . . . . 36
3.4.1. OpenCV . . . . . 36
3.4.2. Instalación de OpenCV en los drones . . . . . 38
3.4.3. Support Vector Machine . . . . . 39

3.5. Descripción de la biblioteca PyParrot . . . . . 40
3.6. El simulador Sphinx . . . . . 42
3.7. Otras herramientas utilizadas . . . . . 45
3.7.1. Oracle VM VirtualBox . . . . . 45
3.7.2. Ubuntu 16.04 . . . . . 46
3.8. Ambiente de desarrollo . . . . . 46

**4. Implementación** **49**
4.1. Arquitectura de la solución . . . . . 49
4.2. Módulo de comunicación . . . . . 54
4.2.1. Medio de comunicación . . . . . 54
4.2.2. Protocolo de comunicación . . . . . 56
4.3. Navegación . . . . . 62
4.3.1. Escenario . . . . . 62
4.3.2. Algoritmos de navegación . . . . . 62
4.3.3. Coordinación entre dos o más drones . . . . . 71
4.3.4. Control de vuelo . . . . . 73
4.3.5. Manejo de error . . . . . 81
4.3.6. Optimización del uso de la batería . . . . . 84
4.4. Detección y seguimiento . . . . . 87
4.4.1. Detección de intrusos . . . . . 87
4.4.2. Corrección de falsos positivos . . . . . 90
4.4.3. Seguimiento de intrusos . . . . . 91
4.5. Configuración y depuración . . . . . 95

**5. Análisis experimental** **97**
5.1. Navegación del predio . . . . . 97
5.1.1. Metodología de evaluación . . . . . 98
5.1.2. Escenarios de prueba . . . . . 100
5.1.3. Resultados obtenidos . . . . . 102
5.2. Detección de intrusos . . . . . 117
5.2.1. Metodología de evaluación . . . . . 117
5.2.2. Videos de prueba . . . . . 120
5.2.3. Resultados . . . . . 121

**6. Conclusiones y trabajo futuro** **127**
6.1. Conclusiones . . . . . 127
6.2. Trabajo futuro . . . . . 129

**Glosario** **131**

**Bibliografía** **133**

# Índice de cuadros

5.1. Beneficio de vigilancia con un drone. . . . . 106
5.2. Escenario 1 con dos drones. . . . . 110
5.3. Escenario 2 con dos drones. . . . . 112
5.4. Escenario 3 con dos drones. . . . . 115
5.5. Valores relativos de beneficio con respecto al algoritmo scan para un drone. . . . . 116
5.6. Valores relativos de beneficio con respecto al algoritmo scan para una flota de drones. . . . . 116
5.7. Resultados del video A. . . . . 121
5.8. Resultados del video B. . . . . 122
5.9. Resultados del video C. . . . . 124
5.10. Resultados de todos los videos. . . . . 125

# Índice de figuras

3.1. Motor BLDC (imagen de uso público de Jjmontero9 CC BY-SA 3.0, de Wikimedia Commons) . . . . . 20
3.2. Comparación entres distintos tipos de hélices. (imagen de uso público de Tubezlob y Chaagii 0817 CC BY-SA, de Wikimedia Commons) . . . . . 21
3.3. Control electrónico de velocidad (imagen de uso público de Avsar Aras CC BY-SA 3.0, de Wikimedia Commons) . . . 22
3.4. Ejes de movimiento de aeronaves (imagen de uso público de Quirón5 CC BY-SA 3.0, de Wikimedia Commons) . . . . . 23
3.5. Comportamiento de los rotores para cada movimiento. . . . 23
3.6. Parrot Bebop 2 (imagen de uso público de Hunini CC BY-SA 4.0, de Wikimedia Commons) . . . . . 24

4.1. Arquitectura de la solución propuesta. . . . . 50
4.2. Máquina de estados. . . . . 52
4.3. Red ad-hoc de una flota de drones. . . . . 55
4.4. Escenarios de envíos de mensajes. . . . . 58
4.5. Planificación de trayectoria. . . . . 63
4.6. Algoritmo scan. . . . . 65
4.7. Matriz con prioridades y matriz con beneficios de vigilancia. 68
4.8. Matriz con beneficios de vigilancia máximos por fila y matriz con trayectoria final. . . . . 68
4.9. Algoritmo de la trayectoria con restricción . . . . . 70
4.10. Intersección de caminos. . . . . 73
4.11. Ejes relativos del drone. . . . . 74

4.12. Distancia entre la posición actual y la posición destino y sus componentes distanciaLatitud y distanciaLongitud paralelas a los ejes latitud y longitud respectivamente. . . . . 75
4.13. Marco de referencia del drone y marco de referencia de la tierra. . . . . 76
4.14. Ángulo de giro. . . . . 78
4.15. Ángulo GPS según la dirección de la distancia real entre posición actual y destino (rojo) determinada por sus componentes, distancia latitud (azul) y distancia longitud (verde). 79
4.16. Desigualdad triangular. . . . . 82
4.17. Batería en función del tiempo. . . . . 86
4.18. Imagen ilustrativa del rectángulo de detección. . . . . 89
4.19. Diferencia entre imagen sin supresión y con supresión (imagen realizada con el software de código abierto PyImageSearch). . . . . 89
4.20. Support Following. . . . . 94
4.21. Mapa de ruta calculada. . . . . 96

5.1. Subdivisión del predio. . . . . 98
5.2. Prioridades del escenario 1 . . . . . 100
5.3. Prioridades del escenario 2 . . . . . 101
5.4. Prioridades del escenario 3 . . . . . 101
5.5. Algoritmos en el escenario 1. . . . . 103
5.6. Algoritmos en el escenario 2. . . . . 104
5.7. Algoritmos en el escenario 3. . . . . 105
5.8. Algoritmo scan con dos drones en el escenario 1. . . . . 108
5.9. Algoritmo trayectoria con dos drones en el escenario 1. . . . 108
5.10. Algoritmo trayectoria con restricción con dos drones en el escenario 1. . . . . 109
5.11. Algoritmo scan con dos drones en el escenario 1. . . . . 111
5.12. Algoritmo trayectoria con dos drones en escenario 2. . . . . 111
5.13. Algoritmo trayectoria con restricción con dos drones en escenario 2. . . . . 112
5.14. Algoritmo scan con dos drones en el escenario 3. . . . . 113
5.15. Algoritmo trayectoria con dos drones en escenario 3. . . . . 114

5.16. Algoritmo trayectoria con restricción con dos drones en escenario 3. . . . . 114
5.17. Diferentes resultados en la detección de intrusos. . . . . 118
5.18. Matrices de confusión del video A. . . . . 122
5.19. Matrices de confusión del video B. . . . . 123
5.20. Matrices de confusión del video C. . . . . 124

# Capítulo 1

# Introducción

La idea de las aeronaves no tripuladas se remonta aproximadamente al año 1849, cuando los austríacos, a través de pruebas de ensayo y error, bombardearon la ciudad italiana de Venecia usando globos aerostáticos no tripulados con cargas explosivas. A partir de ese momento y tras un siglo y medio de avances tecnológicos, impulsados por hechos como las guerras mundiales y la guerra de Vietnam, se llegó a lo que hoy se conocen como vehículos aéreos no tripulados (*unmanned aerial vehicles* ,UAVs) [1].

La evolución en la tecnología de las aeronaves no tripuladas en los últimos nueve años, desde que se introdujo el primer UAV comercial en la Feria de Electrónica de Consumo (*Consumer Electronics Show*, CES) en el año 2010, ha provocado que muchas personas y empresas los adopten para diversos fines [2]. El uso de las aeronaves no tripuladas está creciendo rápidamente en muchos dominios de aplicación civil, incluido el monitoreo en tiempo real, el suministro de cobertura inalámbrica, la detección remota, la búsqueda y el rescate, la entrega de bienes, la seguridad y la vigilancia, la agricultura de precisión y la inspección de infraestructura civil. Los UAVs inteligentes son la próxima gran revolución en la tecnología que promete brindar nuevas oportunidades en diferentes aplicaciones, especialmente en infraestructura civil, en términos de reducción de riesgos y costos [3].

El conocimiento del contexto actual de los UAVs motivó la propuesta de este proyecto, que tiene por objetivo la investigación, el diseño y la implementación de una solución de vigilancia en un predio determinado mediante una flota de drones autónomos. Para lograr el objetivo planteado, se definieron una serie de problemas a resolver. El primer problema es construir una solución para la navegación de los drones dentro de un predio determinado, donde cada drone conozca de que forma y dentro de que límites debe desplazarse. El segundo problema es la comunicación entre los drones de la flota, definir una vía y un protocolo de comunicación para eventualmente coordinar los drones. El tercer problema es obtener una solución para la coordinación, donde cada drone pueda conocer información de los restantes drones, algunos de los datos de interés son, el estado en el que se encuentran, la posición en la que están y las acciones que van a tomar. El último problema a resolver es implementar una solución para la interpretación de imágenes obtenidas mediante las cámaras de los drones, donde los drones sean capaces de detectar intrusos u objetos de interés, realizar el seguimiento, así como también comunicar la información de los puntos anteriores.

Para cumplir con el problema de la navegación, se diseñó una solución que combina una grilla de cuadrantes de tamaño fijo y la ubicación de posicionamiento global (*Global Positioning System*, GPS). La grilla es una representación matricial del predio donde se desplazan los drones y cada celda es ponderada según el grado de importancia que posea. Aquellos cuadrantes que tienen una prioridad más alta, son visitados con mayor frecuencia. El GPS proporciona a esta solución un mayor grado de precisión en la ubicación de cada drone de la flota, dado que en cada iteración se corrigen errores ocurridos en el desplazamiento de los drones provocados tanto por agentes internos como externos.

En lo referente al segundo problema, se diseñó una solución similar al protocolo de control de transmisión (*Transmission Control Protocol*, TCP) en lo que refiere a la fiabilidad, pero enviando mensajes con el protocolo de datagrama de usuario (*User Datagram Protocol*, UDP). UDP proporciona una mayor velocidad de comunicación respecto a TCP, pero como es bien conocido, no implementa ningún tipo de control de flujo, siendo así menos

fiable. La pérdida de mensajes podría ser catastrófica para un sistema de seguridad mediante una flota de drones autónomos por la alta cohesión que hay entre los drones. Por lo tanto, se implementó un sistema de mensajes de reconocimiento (*acknowledgement*, ACK) sobre la comunicación UDP. Los drones cuentan con un puerto específico, común a todos ellos, a través del cual se tiene una vía sencilla de comunicación sin la necesidad de crear puertos para cada uno de los drones [4].

Estrechamente relacionado al problema de la comunicación se encuentra el de cooperación. Para cumplir este problema se diseñó como solución una estructura de mensajes mediante la cual los drones de la flota pueden interactuar, compartir información y trabajar en conjunto. En esta estructura se envía información como la posición y estado actual, la ruta que están siguiendo y la cantidad de batería con la que cuentan.

Finalmente, con respecto al problema de procesamiento de imagen, se diseñó una solución basada en la biblioteca OpenCV [5] y la técnica de histograma de gradientes orientados (*Histogram of Oriented Gradients*, HOG). En esta solución se implementó un método de detección de personas y otro de rastreo para permitir el seguimiento de intrusos. Además de los métodos nombrados, se implementaron algoritmos de corrección de errores y predicción de movimientos para el seguimiento cooperativo.

El resultado final del trabajo se evaluó con una serie de vídeos que simulan escenarios de vigilancia. A partir de estas simulaciones se optimizó el algoritmo para que tuviera un comportamiento satisfactorio con respecto al comportamiento esperado en un sistema de seguridad con drones. En el proyecto se crearon archivos de configuración que permiten inicializar el ambiente de trabajo y parametrizar el comportamiento de la flota de drones. Las configuraciones definidas en los archivos de configuración debieron ser ajustadas a medida que se realizaron las pruebas, con el fin de lograr el objetivo de este proyecto. Los resultados finales indican que el algoritmo de navegación se desempeño correctamente. La cooperación y comunicación, entre al menos dos drones, fue exitosa. La detección y seguimiento de intrusos funcionó de manera esperada y aceptable.

El documento se estructura del modo que se describe a continuación. El capítulo 2 describe el problema a resolver y se analizan los trabajos relacionados. El capítulo 3 da un marco teórico de los cuadricópteros e introducen las herramientas a usar. En el capítulo 4 se presenta el diseño de la arquitectura de la solución junto a los detalles de la implementación. En el capítulo 5 se describe la metodología usada para la evaluación experimental, se describen los distintos casos de pruebas planteados y se analizan los resultados obtenidos. Por último, en el capítulo 6 se realiza el planteo de las conclusiones obtenidas en base a los resultados de la experimentación, incluyendo posibilidades de trabajo futuro teniendo como punto de partida la investigación realizada.

# Capítulo 2

# El problema de vigilancia con una flota de drones

En este capítulo se presenta el problema de vigilar un predio conocido utilizando una flota de drones, para el cual se propuso desarrollar una solución en éste proyecto de grado y se analizan los principales trabajos relacionados.

## 2.1. Introducción

La evolución de los sistemas de seguridad está fuertemente relacionada con el avance tecnológico. Gracias a la capacidad de introducir avances tecnológicos en los sistemas de seguridad, se pueden distinguir entre dos tipos de sistemas, los sistemas de vigilancia tradicional y los sistemas de vigilancia inteligente [6].

Los sistemas de seguridad tradicional son aquellos que están conformados por un conjunto de personas. Cuando involucran más de una

persona, dan uso a intercomunicadores para comunicarse entre sí y usualmente vigilan el perímetro del terreno y/o los posibles ingresos al mismo. Uno de los inconvenientes que presenta este tipo de sistemas es la existencia de errores cometidos por los seres humanos. También se observa una clara dificultad para vigilar el terreno, ocasionando que el sistema se limite a controlar solamente el perímetro y los posibles accesos.

Por el contrario, la vigilancia inteligente consiste en un sistema interconectado que cuenta con la capacidad no sólo de grabar el escenario, sino también de procesar información de forma autónoma y en caso de detectar un objeto sospechoso, tomar las acciones pertinentes. Adicionalmente, los sistemas de vigilancia inteligente cuentan con la capacidad de compartir toda la información a los participantes del sistema y de esta forma realizar un trabajo colaborativo logrando un resultado eficiente y eficaz.

Se define un sistema de vigilancia óptimo como aquel en el que se cumplen las siguientes condiciones; el tiempo en el que un punto del predio está sin vigilar debe ser el mínimo posible dependiendo de su prioridad. Se debe evitar que el predio quede sin ser vigilado cuando se tiene más de un drone, es decir, en caso de ser posible los drones deben cargar en diferentes momentos. Las rutas que tomen los drones deben ser las más cortas posibles pero obteniendo el mayor beneficio de vigilancia (mayor cantidad de celdas con mayor prioridad).

Gracias al avance tecnológico de los vehículos aéreos no tripulados, se puede enfrentar el problema de la vigilancia de un terreno mediante una flota de drones, dando solución a los problemas usuales que presentan las vigilancias tradicionales. Este enfoque permite desarrollar una solución autónoma y que trabaje de forma colaborativa, con la capacidad de cubrir en su totalidad y de forma óptima el terreno a vigilar así como también detectar los objetos sospechosos y seguirlos.

A diferencia del sistema de vigilancia tradicional, la vigilancia mediante una flota de drones presenta notorias ventajas [7], algunas de ellas son:

- Una flota de drones puede recoger información simultáneamente en múltiples lugares y compartirla con los restantes agentes, lo cual permite generar un modelo para luego tomar las decisiones con un mayor grado de certeza.
- Crear rutas de vuelo teniendo en cuenta las rutas que están siguiendo los restantes drones, lo cual permite optimizar el tiempo en el que es visitado cada punto del predio.
- Con un sistema de vigilancia conformado por múltiples drones, se puede diseñar un algoritmo que se adapte a la falla de vehículos, esto brinda al sistema de vigilancia una mayor tolerancia a fallas.
- La velocidad horizontal que alcanzan los drones parrot bebop 2 (los drones utilizados en este proyecto de grado), pueden ser de hasta 16 metros por segundo, esto permite transportarse entre dos puntos cualesquiera del terreno en poco segundos (asumiendo que el terreno a vigilar es reducido).
- Debido a la altura a la que vuelan los drones y la cámara de alta definición que tienen, se logra obtener una gran amplitud de visión y definición.

Combinando todas las ventajas mencionadas, se obtiene un sistema de vigilancia basado en una flota de drones que satisfacen los requisitos de un sistema de seguridad.

## 2.2. Descripción del problema

El problema a resolver consiste en vigilar un predio determinado que posee puntos de interés de forma eficiente y eficaz utilizando una flota de drones que funcionen de forma autónoma, es decir que los drones sean capaces de tomar decisiones o realizar acciones por su propios medios sin necesidad de la intervención de un tercero. Los drones deben de contar con la inteligencia suficiente para determinar qué momento es el indicado para

cargar su batería, cuándo retomar su vuelo, así como también determinar cuál será su próximo paso a seguir a partir del estado actual.

Del predio se conocen las coordenadas GPS y las prioridades de cada punto, así como también sus límites y tamaño. Dado que los puntos de interés tienen diferente prioridad, es de importancia que los de mayor prioridad sean visitados más frecuentemente que aquellos con menor prioridad.

Los drones también deben ser capaces de calcular sus rutas teniendo en cuenta múltiples factores como, su estado actual, el estado y las rutas que están siguiendo los restantes drones y la prioridad de los puntos de interés. Para cumplir con esto, es fundamental contar con un protocolo de comunicación para que los drones puedan compartir información.

Para vigilar de forma eficiente el territorio, no sólo basta con que los drones recorran de forma óptima el predio, sino que también, es de crucial importancia que una vez que se detecte la presencia de un intruso en el predio, el drone lo siga mientras que el intruso se encuentra dentro del terreno.

## 2.3. Trabajos relacionados

Existe una gran cantidad de artículos relacionados a la temática que se aborda en este proyecto, muchos de ellos muy recientes debido a la popularidad que han tomado los drones y particularmente a su uso en misiones del tipo de vigilancia ó exploración. A continuación se reseñan algunos trabajos relacionados de los que se tomaron ideas para este proyecto. Se hizo especial énfasis en artículos que traten la planificación de vuelo con una flota de drones aplicando inteligencia computacional, artículos sobre los distintos problemas que presenta mantener la comunicación en una flota de drones y artículos sobre la detección de personas y procesamiento de imágenes aplicados en UAVs.

### Planificación de misión con Múltiples UAVs

La planificación de una misión para una flota de UAVs es un problema de optimización que tiene como objetivo encontrar un conjunto de caminos de modo de maximizar beneficios de vigilancia satisfaciendo ciertas restricciones como por ejemplo, los UAVs deben visitar todos los puntos de interés y tener suficiente batería para regresar al punto de carga de la batería.

Shang et al. [8] plantearon una solución híbrida basada en algoritmos genéticos (*Genetic Algorithm*, GA) y algoritmos de optimización de colonia de hormigas (*Ant Colony Optimization Algorithms*, ACO) para el problema de planificación de una misión de múltiples UAVs. En un predio a vigilar existe un conjunto de puntos de interés, cada punto tiene un beneficio de vigilancia indicado por el grado de importancia del punto, y un conjunto de UAVs consigue el beneficio cuando visita el punto. En el trabajo de Shang et al. los puntos necesitan ser visitados una única vez y los UAVs deben poder regresar a la base antes de que la batería agote su carga. Por lo tanto, el objetivo de la misión es conseguir la máxima cantidad de beneficio de vigilancia cumpliendo las restricciones de batería y que cada punto sea visitado una sola vez. El problema está planteado como un problema de orientación de equipo (*Team Orienteering Problem*, TOP) donde los puntos de interés son representados por los nodos y las aristas representan el camino entre los puntos de interés. El problema de planificación de una misión de vigilancia para múltiples UAVs consiste en encontrar $m$ caminos empezando en el origen y finalizando en el destino, de modo que los UAVs visiten la máxima cantidad de objetivos para maximizar el beneficio de vigilancia respetando las restricciones. De este trabajo se tomó para el proyecto la idea de los puntos de interés y del beneficio de vigilancia de visitar los puntos de interés aplicando la restricción de batería, pero omitiendo la restricción de visitar cada punto de interés solo una vez. Para obligar a que se visiten todos los puntos y al mismo tiempo evitar que un punto sea visitado de forma reiterativa en poco tiempo, en el proyecto se agregó un sistema de envejecimiento al valor otorgado a los puntos de interés.

Grøtli et al. [9] también estudiaron el problema de la planificación de caminos para múltiples UAVs utilizando restricciones por ejemplo, que el camino sea seguro con respecto a colisiones, que se cumplan ciertos criterios para las comunicaciones y que los caminos sean eficientes con respecto a la batería. Los autores utilizaron programación lineal entera mixta (*Mixed Integer Linear Programming*, MILP) para encontrar el camino y la herramienta de análisis de propagación, pérdida y terreno de la señal (*Signal Propagation, Loss and terrain Analysis Tool*, SPLAT) para obtener predicciones más precisas de la capacidad de comunicación de esos caminos. Sea el camino entre dos UAVs el camino que se debe tomar para llegar desde la posición de uno a la posición del otro, se planifica de modo de que haya comunicación posible entre ambos para que uno de ellos pueda servir como nodo de apoyo al otro cuando no tiene alcance de comunicación con la base. Para este proyecto de grado se utilizó un algoritmo más simple de planificación que no se considera la comunicación como restricción. La razón es que en el contexto de un terreno limitado, conocido y considerablemente pequeño no resulta necesario agregar este tipo de restricciones al algoritmo. Tampoco se tuvo en cuenta la restricción de que el camino sea seguro a colisiones. El terreno seleccionado tiene como precondición de que no tenga obstáculos con los que el drone pueda colisionar y para evitar la colisión entre drones se optó por configurar la altura de vuelo de cada drone diferente a los demás, una solución similar a la utilizada por aviones comerciales.

Debido a limitaciones relacionadas con el simulador utilizado en este proyecto se omitió el estudio de algunos factores que son influyentes en situaciones reales. Uno de ellos es el viento, Cole y Wickenheiser [10] propusieron una solución al problema del cambio de la trayectoria inicial debido a este factor. Se propuso un controlador que tiene dos modos: normal y drift. En modo normal el viento no afecta lo suficiente al drone para cambiar su trayectoria y el drone es capaz de contrarrestar los efectos del viento por su cuenta. En modo drift el vehículo mantiene el control generando nuevas trayectorias a medida que son necesarias. El Parrot Bebop 2 contrarresta el viento (si este no es muy fuerte) inclinándose automáticamente al detectarlo. En caso de fuertes ráfagas en este proyecto de grado se diseño una estrategia basada en la desigualdad triangular con el mismo objetivo que

la planteada por Cole y Wickenheiser para calcular nuevas trayectorias si existe una desviación de la trayectoria original.

Lograr que una flota de UAVs viajen de manera rápida sin colisiones entre ellos a causa de factores como el viento es una tarea compleja. Shiri et al. [11] en su investigación del problema de la planificación de un camino para múltiples UAVs atacaron este problema utilizando un método de juego de campo medio (*Mean-Field Game*, MFG) de control teórico que requiere que los UAVs intercambien estados solo una vez al comienzo. Luego cada UAV puede controlar su aceleración localmente resolviendo dos ecuaciones en derivadas parciales (*Partial Differential Equation*, PDE), conocidas como ecuaciones de Hamilton-Jacobi-Bellman (HJB) y ecuaciones de Fokker-Planck-Kolmogorov (FPK). Este enfoque tiene como desventaja que conlleva un alto costo computacional para resolver las PDE. Para mejorar este problema Shiri et al. propusieron utilizar un método de aprendizaje automático (*Machine Learning*, ML) donde dos modelos de ML distintos aproximaron las soluciones de las ecuaciones de HJB y FPK. Como resultado los autores llegaron a obtener costos computacionales aceptables para este tipo de problema.

### Comunicacion entre múltiples UAVs

Mantener la comunicación entre una flota de UAVs es fundamental para el éxito de una misión de vigilancia o para cualquier otro tipo de misión que involucre coordinación para su desarrollo. El problema de mantener la comunicación y la calidad de la misma en un contexto de múltiples UAVs es otro de los problemas más estudiados en el campo de los UAVs. Si bien se optó por usar una red wifi local como enlace de comunicación para los drones, se estudiaron otras posibilidades como por ejemplo redes ad hoc (ver sección 4.2.1).

Bekmezci et al. [12] propusieron una red ad hoc aérea (*Flying Ad Hoc Network*, FANET) para la comunicación entre UAVs, dado que es una de las arquitecturas de multicomunicación más efectivas por su capacidad

de transferir datos simultáneamente sin ninguna infraestructura adicional. Por otro lado, dada la movilidad de los UAVs, la topología de la FANET cambia frecuentemente y la interconectividad durante todo el tiempo se convierte en una importante restricción. Una asignación de tareas asegurando la coordinación temporal y espacial entre los UAVs es esencial para las FANET. Bekmezci et al. presentaron una heurística para FANETs con el objetivo de visitar todos los puntos de interés en un tiempo mínimo preservando en todo momento la conectividad entre los UAVs.

Hanna et al. [13] propusieron un interesante uso de los UAVs en el contexto de las redes de comunicación. Se planteó utilizar UAVs como nodos que extienden el alcance de un enlace y mejoran su capacidad. La principal ventaja contra los clásicos nodos fijos es que los UAVs pueden cambiar de posición para optimizar la capacidad o el alcance del enlace a demanda. El artículo plantea usar un conjunto de UAVs como un nodo de múltiple entradas y salidas (*Multiple-input Multiple-output*, MIMO) para proveer conectividad entre dos puntos.

Otra característica importante en una red de comunicación de UAVs, especialmente para misiones del tipo de vigilancia y más especialmente aún para misiones en zonas de guerra, es la seguridad de los mensajes. Kanth et al. [14] propusieron una comunicación encriptada para proteger los mensajes entre un centro de operaciones y un UAV. El centro de operaciones y el UAV tienen un conjunto de claves de encriptación y desencriptación único. El procedimiento empieza con una autenticación por UDP donde el UAV recibe la clave de encriptación. Los datos encriptados y una firma son enviados desde el centro de operaciones al UAV encriptados con la clave de encriptación del centro de operaciones, luego el UAV desencripta los datos y verifica la firma. Para este proyecto se omitió la seguridad en los mensajes entre los UAVs porque la información compartida es principalmente posicional, que puede ser conocida a simple vista dado que el terreno es limitado y considerablemente pequeño.

Una de las claves para que un sistema de vigilancia basado en UAVs sea exitoso es que las imágenes captadas por la cámara del UAV sean enviadas eficientemente a un equipo para realizar el procesamiento y detectar

intrusos en ellas. Adaptar el ratio de bits del vídeo para evitar la congestión en la red es crítico, dado que las congestiones resultan en retardo autoinfligido y pérdida de paquetes lo cual deteriora la calidad del flujo del video. Dai et al. [15] exploraron los principios subyacentes del retardo autoinfligido y concluyeron que el retardo de congestión es determinado por el ratio de envio, el ratio de recepción y el estado de la red, por lo que decidieron controlar el ratio de bits del video usando un modelo de control de flujo de aprendizaje estadístico. La clave de la propuesta es forzar todos los flujos a converger a la misma cola de carga y ajustar el ratio de bits al modelo. Todos los flujos mantienen una cantidad de paquetes pequeña y fija haciendo cola en la red, por lo que se logra el reparto justo de la banda ancha y la baja latencia.

Los UAVs normalmente se comunican mediante redes wifi como se propone en este proyecto, pero otras opciones como una red celular 4G pueden proveer una mayor cobertura de área, que es interesante para misiones de vuelos autónomos que se extienden más allá de la línea de visión. Sin embargo, las redes de celulares no están optimizadas para dispositivos aéreos. Por ejemplo las antenas apuntan hacia abajo para servir a los usuarios en el suelo y no a dispositivos voladores. Para poder usar la red 4G son necesarias mejoras para optimizar la conectividad entre UAVs. Raffelsberger et al. [16] introdujeron una herramienta para analizar y evaluar el rendimiento de las redes 4G denominada herramienta de medida de drones celulares (*Cellular Drone Measurement Tool*, CMDT). La herramienta integra varias características importantes para evaluar las comunicaciones celulares en redes aéreas, tales como monitorear la fuerza de una señal, evaluar el rendimiento de TCP y UDP y rastrear las coordenadas GPS. Los autores concluyeron que TCP muestra un mejor rendimiento a la altura del suelo pero UDP mejora a TCP en alturas de vuelo.

### Detección de objetos con UAVs

La detección de objetos, formas y patrones es muy útil en el contexto de los UAV para misiones de vigilancia, reconocimiento y búsqueda.

Kyrkou y Theocharides [17] estudiaron el uso de UAVs para monitorear un área donde haya ocurrido un desastre natural. Utilizaron una estrategia de aprendizaje profundo (*Deep Learning*, DL) para analizar las imágenes obtenidas de forma autónoma y alertar en tiempo real sobre la presencia de elementos como fuego o inundaciones obteniendo buenos resultados como por ejemplo hasta tres veces mayor eficiencia utilizando menos memoria pero manteniendo una precisión similar a los modelos actuales.

Mittal et al. [18] también se enfocaron en misiones de rescate pero más específicamente en casos de edificios colapsados. Los autores propusieron un algoritmo para detectar el mejor lugar de aterrizaje en un escenario de terrenos desbalanceados a causa de escombros. El enfoque es interesante porque a diferencia de otras soluciones propuestas para este tipo de problemas, en este caso no se pueden usar mapas preexistentes porque pueden haber ocurrido cambios estructurales en el terreno. Además los autores determinaron que los algoritmos existentes no identificaban correctamente sitios de aterrizaje seguros en un terreno cubierto de escombros. El algoritmo propuesto toma en cuenta el riesgo de aterrizaje en un sitio evaluando la planicie e inclinación del terreno, la profundidad y la energía requerida para aterrizar en dicho sitio.

La detección de objetos y patrones también puede ser usada como asistencia en la navegación. Díaz y Garate [19] implementaron una navegación mediante la identificación de marcadores posicionados estratégicamente en un predio o zona de vuelo. Los marcadores son reconocidos por los drones mediante su cámara, con el fin de estimar su posición y ser capaces de seguir rutas preconfiguradas. Los autores concluyeron que el sistema diseñado presenta una solución viable a pesar de algunos errores en estimación de la posición y de que es necesario contar con una cantidad de marcadores importante.

Detectar personas en imágenes es una tarea desafiante debido a las diferentes apariencias y el variado rango de poses que pueden adoptar. Lo primero que se necesita es un conjunto robusto que permita que la forma humana pueda ser discriminada, incluso en escenarios con varios objetos extraños o con mala iluminación. Dalal y Triggs [20] demostraron que la

técnica de HOG provee un excelente rendimiento relativo a otras técnicas existentes para detectar personas.

Otra alternativa para la detección de personas basada en un clasificador de Haar en cascada fue propuesta por Arreola et al. [21]. Dado que los clasificadores de Haar son considerados como clasificadores débiles, un entrenamiento de cascada es aplicado para obtener una clasificación más robusta. Una vez que un objeto, en este caso una persona, es detectado, se determina la posición del objeto con respecto al drone. Luego esta posición es procesada por el algoritmo para determinar las instrucciones que deben darse al control de vuelo.

Para este proyecto se utilizó la técnica de HOG propuesta por Dalal y Triggs para detectar personas sumado a la metodología de Arreola et al, particionando la entrada de la cámara en cuadros para procesarla, detectar personas y en caso positivo calcular la posición y pasar esta información al control de vuelo.

Otros estudios del problema como el de Afifi et al. [22] se concentraron en utilizar algoritmos livianos que puedan ser utilizados en procesadores que tienen limitado poder computacional, porque necesitan ser lo suficientemente simples para ser embebidos en un UAV y no afectar su peso. Se propuso un algoritmo para detección de peatones basado en la red *You Only Look Once*, (YOLO) de detección de objetos. El framework usa *Deep Learning* para operaciones robustas y un modelo pre-entrenado. Un problema de este enfoque que también se vio en este proyecto (aún al usar otro framework) es que no se considera la relación temporal entre dos cuadros de video consecutivos, por lo que regularmente se detecta un objeto en un número de cuadros y luego ese mismo objeto no se detecta en los cuadros siguientes. La detección de un objeto retorna una caja de detección que enmarca al objeto detectado en el cuadro estableciendo su posición. El enfoque también tiene problemas para detectar precisamente peatones cuando se superponen (problema que se omite en ese proyecto al asumir un solo intruso en el cuadro). Para resolver estos problemas, en el proyecto se propone mantener las posiciones detectadas en el siguiente cuadro y cuando dos cajas de detección se superponen se fusionan. Estas

detecciones se clasifican con alto o bajo puntaje de confianza, las de alto puntaje son consideradas correctas y agregadas a un conjunto de objetos detectados y las de bajo puntaje son solo consideradas si se superponen con detecciones en el cuadro anterior.

El siguiente paso a la detección es el seguimiento o tracking de la persona. Byun et al. [23] resolvieron el problema de calcular una trayectoria para perseguir vehículos terrestres respetando ciertas restricciones. La solución planteada incorpora datos del pasado para predecir la posición del vehículo terrestre y usa esos datos para generar puntos de referencia que el UAV puede seguir. Se asume un modelo cinético con estados delimitados para construir un conjunto de puntos de posiciones predecidas y el centro de Chebyshev de este conjunto es usado como estimación de la futura posición del vehículo. Para crear los puntos de referencia que el UAV debe seguir se utiliza un control predictivo por modelo. En este proyecto se utilizó una estrategia simplificada inspirada en la de Byun et al. donde al momento de detectarse un intruso se pasa la posición GPS al control de vuelo y luego de obtener un conjunto de posiciones del intruso detectado el control de vuelo traza una trayectoria y calcula un punto donde se espera que el intruso esté en dos segundos dada su trayectoria y velocidad aproximada.

| Artículos de planificación | | |
|---|---|---|
| **Autor** | **Año** | **Concepto Clave** |
| Shang et al. | 2014 | Planificación de una misión con múltiples UAVs. Beneficio de vigilancia obtenido al visitar puntos de interés aplicando restricciones. |
| Grotli et al. | 2012 | Planificación de caminos teniendo en cuenta posibles colisiones, perdida de comunicación entre la flota y eficiencia de la batería. |
| Cole y Wickenheiser | 2017 | Cálculo de nuevas trayectorias en vuelo debido a desviaciones por efecto del viento. |
| Shiri et al. | 2019 | Planficación de trayectorias de una flota de UAVs intercambiando estados solo al comienzo. |

| Artículos de comunicación | | |
|---|---|---|
| **Autor** | **Año** | **Concepto Clave** |
| Bekmezci et al. | 2014 | Red ad hoc para la comunicación de UAVs. |
| Hanna et al. | 2019 | Utilizar UAVs como nodos que extienden el alcance de un enlace y mejoran la capacidad de una red. |
| Kanth et al. | 2019 | Encriptación de mensajes en una red de comunicación de UAVs. |
| Dai et al. | 2019 | Optimizar el ratio de trasmisión de bits de un vídeo evitando la congestión de la red. |
| Raffelsberger et al. | 2019 | Comunicación de UAVs mediante una red celular 4G. |

| Artículos de procesamiento de imágenes | | |
|---|---|---|
| **Autor** | **Año** | **Concepto Clave** |
| Kyrkou et al. | 2019 | Monitorear un área donde ocurrió un desastre natural con una flota de UAVs utilizando deep learning para el procesamiento de imágenes. |
| Mittal et al. | 2019 | Detectar mejor lugar de aterrizaje en un terreno de desastre natural, sin el uso de mapas preexistentes. |
| Díaz y Garate | 2018 | Navegación a través de identificación de marcadores. |
| Dalal y Triggs | 2005 | Histogramas de gradientes orientados para detección de personas. |
| Arreola et al. | 2019 | Detección de personas basada en un clasificador de Haar en cascada. |
| Afifi et al. | 2019 | Algoritmos livianos para ser utilizados en procesadores que tienen limitado poder computacional. |
| Byun et al. | 2019 | Tracking y seguimiento de vehículos mediante posiciones predichas. |

# Capítulo 3

# Materiales y tecnologías

Este capítulo presenta los materiales empleados a lo largo del proyecto y las tecnologías utilizadas. La sección 3.1 describe las características generales de los cuadricópteros, en particular las del Parrot Bebop 2. La sección 3.2 introduce la arquitectura ARM. En la sección 3.3 se analizan los posibles lenguajes ha utilizar para la implementación del proyecto. La sección 3.4 describe la herramienta a utilizar para el procesamiento de imágenes. La sección 3.5 describe la biblioteca PyParrot. La sección 3.6 introduce el simulador utilizado en el proyecto. En la sección 3.7 se realiza una mención a otras herramientas que fueron utilizadas y finalmente en la sección 3.8 se describe el ambiente de desarrollo.

## 3.1. Aspectos básicos de los cuadricópteros

En esta sección se describen los componentes y las primitivas de movimiento que tienen los cuadricópteros y en particular se detallan las características del Parrot Bebop 2.

### 3.1.1. Componentes de los cuadricópteros

Los cuadricópteros son vehículos capaces de volar impulsados por hélices y rotores. Ciertos componentes mecánicos y eléctricos son esenciales, independientemente de la marca, modelo o cual sea su uso. Estos componentes se describen a continuación.

- **Rotores**: los rotores transforman la energía eléctrica en movimiento circular que pasa a transmitirse a las hélices del cuadricóptero y causa un empuje que permite que el cuadricóptero se mueva. Los rotores pueden ser de distintos tamaños, velocidades y potencias, así como trifásicos o bifásicos. Debido a la alta velocidad con la cual giran, suelen utilizarse motores de corriente continua sin escobillas (brushless direct current motors, BLDC) (ver figura 3.1).

Figura 3.1: Motor BLDC (imagen de uso público de Jjmontero9 CC BY-SA 3.0, de Wikimedia Commons)

- **Estructura**: la estructura es el cuerpo o cuadro del cuadricóptero, donde se ensamblan y apoyan todos los componentes. Diversos materiales son utilizados para su construcción, como por ejemplo: titanio, magnesio, aluminio, fibra de carbono, fibra de vidrio, plástico, entre otros. Algunas de las características buscadas en la elección del material son fortaleza, rigidez y ligereza. En función del uso del cuadricóptero, se establece cuáles son prioritarias.

- **Controladores**: la placa controladora es la encargada del procesamiento de la información recolectada por los sensores, las órdenes recibidas por los comandos y la generación de instrucciones. Su principal función es asegurar la estabilidad de vuelo transmitiendo la información al control electrónico de velocidad.

- **Hélices**: las hélices son giradas por la potencia que transmiten los motores y elevan al cuadricóptero en el aire. Habitualmente, las hélices están compuestas por fibra de carbono, plástico o nylon, y pueden ser de dos o tres aspas (ver figura 3.2). Las hélices de dos aspas son más comunes y consumen menos energía que las de tres aspas, pero ofrecen un menor desempeño en la estabilidad del cuadricóptero. Ambos tipos de hélices dependen, de igual manera, del diámetro de sus aspas; a mayor longitud, mayor es el empuje pero mayor es el consumo de energía.

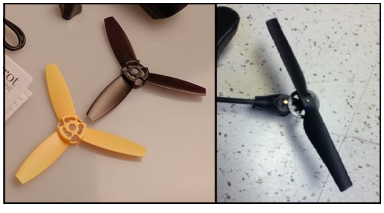

Figura 3.2: Comparación entres distintos tipos de hélices. (imagen de uso público de Tubezlob y Chaagii 0817 CC BY-SA, de Wikimedia Commons)

- **Batería**: la batería alimenta de energía todos los componentes electrónicos utilizados en la aeronave. Existe una gran variedad de baterías, y en ellas se buscan dos características fundamentales: en primer lugar, que posea una gran capacidad de almacenamiento de carga, y en segundo lugar, que posean una gran capacidad de descarga, por estos motivos generalmente son utilizadas las baterías químicas de polímero de litio (LiPo).

- **Controles electrónicos de velocidad**: el control de velocidad electrónico (Electronic Speed Control, ESC) (ver figura 3.3) es un circuito

eléctrico que interpreta la información del procesador, y es el encargado de controlar los rotores de modo que giren a la velocidad y dirección necesaria.

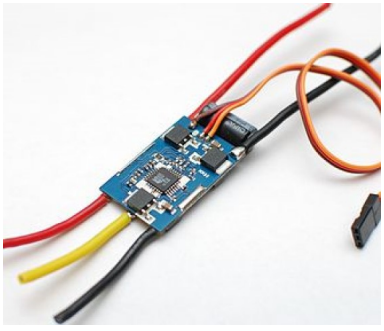

Figura 3.3: Control electrónico de velocidad (imagen de uso público de Avsar Aras CC BY-SA 3.0, de Wikimedia Commons)

- **Sensores**: los cuadricópteros ejecutan su plan de vuelo gracias a múltiples sensores que cumplen la función de adquirir datos que posteriormente serán procesados y analizados. Generalmente, los sensores son utilizados para mantener y mejorar la estabilidad, así como también determinar su orientación, posición y altitud.

### 3.1.2. Primitivas de movimiento

Como fue mencionado en el capítulo 2, los cuadricópteros son vehículos capaces de volar impulsados por hélices y rotores. Para el caso de los cuadricópteros (cuatro rotores), dos rotores giran en sentido horario y dos en sentido antihorario. Si todos los rotores giran en el mismo sentido, el torque ejercido por los rotores provocaría que el drone rote de forma continua sobre su eje vertical.

Existen cuatro primitivas de movimiento (ver Figura 3.4):

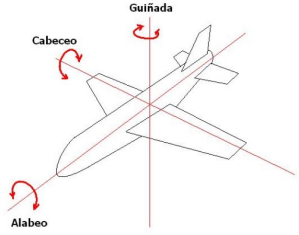


Figura 3.4: Ejes de movimiento de aeronaves (imagen de uso público de Quirón5 CC BY-SA 3.0, de Wikimedia Commons)

- **Guiñada** (yaw): rotación sobre el eje vertical.
- **Alabeo** (roll): rotación sobre el eje longitudinal.
- **Cabeceo** (pitch): rotación sobre el eje transversal.
- **Velocidad** (throttle): desplazamiento sobre el eje vertical.

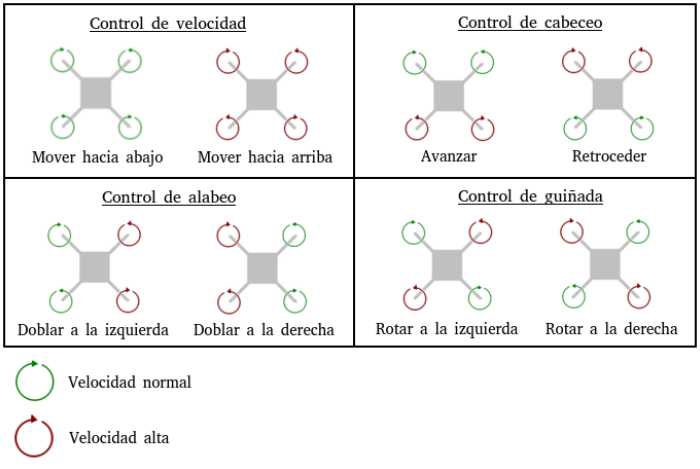


Figura 3.5: Comportamiento de los rotores para cada movimiento.

Al realizar distintas combinaciones de las cuatro primitivas de movimiento (ver Figura 3.5), un drone puede desplazarse en las tres dimensiones.

### 3.1.3. Parrot Bebop 2

El drone Parrot Bebop 2 (ver figura 3.6) es un modelo particular de cuadricóptero civil diseñado y fabricado por la empresa Parrot en el año 2015. Parrot, de origen francés, fue fundada en 1994. Originalmente se dedicaba a la fabricación de dispositivos manos libres, pero desde el año 2010 gracias a un programa de diversificación de sus productos, comenzó la fabricación de cuadricópteros. El Parrot Bebop 2 es sucesor directo del Bebop Parrot producido un año antes.

Figura 3.6: Parrot Bebop 2 (imagen de uso público de Hunini CC BY-SA 4.0, de Wikimedia Commons)

El Parrot Bebop 2 está conformado por una estructura en forma de cruz central fabricada en Poliamida, en color blanco y negro, reforzada con fibra de vidrio para garantizar mayor resistencia y ligereza. Cuenta con unas dimensiones de 38 centímetros de largo por 33 centímetros de ancho y 9 centímetros de altura. El peso del drone ronda los 500 gramos, esto lo hace un vehículo bastante ligero. Es propulsado por cuatro rotores que giran hélices de 3 hojas de alrededor de 5 centímetros de diámetro,

los cuales están compuestos de plástico flexible, son bastante resistentes y fácilmente reemplazables. El Bebop 2 puede alcanzar velocidades máximas de 18 metros por segundo en horizontal, la cual tarda 14 segundos en conseguirla si no se cuenta con oposición del viento, y 6 metros por segundo de ascenso y descenso, llegando a una altura de 100 metros en 20 segundos durante condiciones favorables. La altura máxima a la que puede llegar este vehículo es de unos 300 metros. La gran autonomía es una de sus principales cualidades con una batería de 2700 miliamperios, de ion de litio, hace que el Bebop 2 tenga 25 minutos de vuelo. En el frente del drone se ubica una cámara de 14 megapíxeles, con capacidad de grabar vídeos en formato full-HD (1080p) a 30 frames por segundo. Además la cámara cuenta con estabilización de imagen y puede hacer un desplazamiento de 180 grados. Finalmente, el drone tiene una memoria interna de unos 8 giga-bytes accesible por usb para el almacenamiento de vídeo e imagen.

Internamente a nivel de hardware, el drone cuenta con siete sensores que transmiten la información a su ordenador integrado:

- **Cámara de estabilización**: la cámara inferior saca fotos del suelo cada 16 milisegundos y las compara con las anteriores para determinar la velocidad de la aeronave.
- **Sensor de ultrasonido**: analiza la altitud de vuelo hasta los 5 metros.
- **Sensor de presión**: mide la presión del aire y analiza la altitud de vuelo pasados los 5 metros.
- **Giroscopio de tres ejes**: mide el ángulo de inclinación del drone.
- **Acelerómetro**: permite medir la posición del drone y su velocidad lineal.
- **Magnetómetro de tres ejes**: ayuda a definir la posición como una brújula.

- **Chipset GNSS** (Global Navigation Satellite System) (GPS + GLONASS): geolocaliza al cuadricóptero y ayuda a medir la velocidad para estabilizarlo en altitudes elevadas.

La placa base del drone cuenta con un procesador ARM cortex A9 de núcleo dual con una CPU de cuatro núcleos, dicho procesador esta basado en una arquitectura ARMv7-A cuyos detalles se presentan en la sección 3.2.

## 3.2. ARM y Raspberry

Uno de los aspectos claves para que la solución propuesta en este proyecto pueda ser ejecutable en el drone es que la computadora embebida tenga la capacidad de procesamiento suficiente para resolver los cálculos del algoritmo de vuelo, el procesamiento de imágenes y el envío y recepción de mensajes. En está sección se describe el procesador embebido en los drones Parrot Bebop 2 (ARMv7-A), así como también la principal alternativa considerada, la computadora Raspberry Pi 2 Modelo B.

### 3.2.1. ARM

Los drones Bebop Parrot 2 cuentan con un hardware interno de arquitectura mecánica de grupo reducido de instrucciones para computadoras (Reduced Instruction Set Computer, RISC) avanzada (Advanced RISC machine, ARM). Específicamente, los Bebop Parrot 2 usan la arquitectura ARMv7-A implementada a través del procesador ARM Cortex A9 [24].

La arquitectura ARM surgió en 1983, con la empresa Arcorn Computers Ltd. Roger Wilson y Steve Furder lideraron un equipo con el objetivo de desarrollar un procesador avanzado, que contará con una arquitectura similar a la del MOS 6502, procesador diseñado por MOS Technology en

1975. En 1985, el equipo de investigación terminó el diseño y realizó algunos prototipos del procesador, eventualmente le dan el nombre de ARM1. Un año después, Arcorn sacó a la venta una versión comercial de este prototipo con el nombre de ARM2 y posteriormente la empresa realizó el ARM3 dándole algunas mejoras con respecto a la versión anterior. En 1990, surgió la ARM6 como resultado del trabajo en conjunto con Apple y a partir de ese momento el uso de los procesadores ARM tuvo un crecimiento notorio en los años siguientes. La mayor utilización de la tecnología ARM se alcanzó con el procesador ARM7TDMI, con millones de unidades en teléfonos móviles y sistemas de videojuegos portátiles. Actualmente, la arquitectura ARM se ha convertido en una de las más usadas del mundo, desde discos duros hasta juguetes [25].

La arquitectura ARM tiene un juego de instrucciones similar al del MOS 6502, pero incluye características adicionales que le permiten conseguir un mejor rendimiento en su ejecución. Para mantener el concepto tradicional de RISC, se estableció la ejecución de una instrucción por ciclo de reloj. La característica más interesante es el uso de 4 bits como código de condición, haciendo que cualquier instrucción pueda ser condicional. Estos 4 bits reducen el espacio para algunos desplazamientos en el acceso a la memoria, pero permiten evitar perder ciclos de reloj en el pipeline al ejecutar pequeños trozos de código con ejecución condicional. Otra característica única del juego de instrucciones es la posibilidad de añadir shifts y rotar en el procesamiento de datos (aritmético, lógico y movimiento de registros). Todo lo mencionado ocasiona que se necesiten menos operaciones de carga y almacenamiento, mejorando el rendimiento. Además la arquitectura ARM también tiene algunas características que son inusuales en otras arquitecturas también consideradas RISC, como el direccionamiento relativo, y el pre y post incremento en el modo de direccionamiento [25].

Este proyecto plantea un sistema autónomo para el cual es de interés la ejecución de código dentro de los drones. Por lo tanto, ejecutar código en la arquitectura ARM con la que cuentan los drones Bebop Parrot 2. Para lograrlo, se presentaron dos posibles soluciones: utilizar alguna herramienta de desarrollo que tuviera el mismo tipo de arquitectura y mediante esa herramienta pasar el software a los drones, o usar el método de compilación

cruzada (cross compile) donde se usa alguna herramienta con una arquitectura distinta pero se genera el código o las bibliotecas en la arquitectura deseada.

Finalmente, debido a las dificultades con la compilación cruzada se optó por trabajar con una herramienta que tuviera el mismo tipo de arquitectura. La herramienta en cuestión fue la Raspberry Pi 2 modelo B, una micro computadora que se describe en detalle en la siguiente sección.

### 3.2.2. Raspberry Pi 2 Modelo B

Como se mencionó en la sección 3.2.1, es de gran importancia la posibilidad de ejecutar código dentro de los drones para cumplir con el objetivo de crear un sistema autónomo. Los drones deben poder ejecutar la lógica de navegación implementada en algún lenguaje y procesar las imágenes para la detección de intrusos. Para implementar esta lógica de navegación se plantearon dos posibles soluciones: utilizar el lenguaje C o utilizar el lenguaje Python. Por diferentes ventajas y desventajas que se comentan en la sección 3.3, se decidió utilizar Python. Para realizar la implementación del procesamiento de imágenes, se optó usar la biblioteca OpenCV, muy conocida por su desempeño en el área de visión artificial y que se presenta con mayor profundidad en la sección 3.4. Para poder desarrollar el sistema utilizando Python y OpenCV, es necesario instalar ambos productos de software en los drones. El método que se decidió usar para lograr esto, fue trabajar con la placa Raspberry Pi 2 modelo B.

Las Raspberry Pi son computadoras de una sola placa (Single Board Computer, SBC) fabricadas por la fundación de caridad Raspberry Pi Foundation, en Reino Unido [26]. Esta fundación surgió en el año 2009 con la promesa de promover el avance de la educación de adultos y niños, particularmente en el ámbito de las computadoras, la ciencias de la computación y temas relacionados.

Las SBC son computadores personales donde el procesador, la memoria y algunos tipos de E/S (USB, tarjeta de vídeo, tarjeta de red, HDMI, etc) están integrados en la placa madre. Esta disposición del hardware difiere de las placas madres tradicionales en las ranuras de expansión que tienen para muchos de los periféricos adicionales, como tarjeta de audio, vídeo y red. Hoy en día, la mayoría de las placas madre de consumo se consideran SBC ya que la mayoría de las funciones necesarias existen en una sola placa base, con la ventaja adicional de poder actualizar la funcionalidad existente mediante el uso de tarjetas adicionales. Actualmente, el término SBC se refiere a una placa basada en microprocesador [27].

En particular, la Raspberry Pi 2 Modelo B es un ordenador de bajo costo de la segunda generación de Raspberry Pi, lanzada en febrero de 2015 y reemplazando al modelo Raspberry Pi 1 Modelo B+. Este ordenador, a diferencia de sus predecesores, cuenta con una placa Broadcom BCM2836 con cuatro núcleos a 900 MHz. Además, la Raspberry Pi 2 modelo B tiene 1 GB de memoria RAM, lo que ofrece hasta seis veces más de potencia con respecto al modelo predecesor. La Raspberry Pi 2 soporta diversos sistemas operativos como Windows y Linux [26]. Las características generales de este ordenador son:

- Una CPU ARM Cortex-A7 de cuatro núcleos a 900 MHz con arquitectura ARMv7-A de 32 bit
- Un sistema en chip (System on a chip, Soc) Broadcom BCM2836
- 1GB de memoria RAM
- Una GPU VideoCore IV 250MHz (Núcleo de gráficos 3D)
- 100 Base Ethernet (Ethernet rápido)
- 4 puertos USB
- 40 pines de entrada y salida de proposito general (General Purpose Input Output, GPIO)
- Puerto HDMI completo

- Jack de audio combinado de 3.5mm y video compuesto
- Interfaz de cámara (CSI)
- Interfaz de pantalla (DSI)
- Ranura para tarjeta micro SD

Además, la Raspberry Pi 2 modelo B cuenta con unas dimensiones de 8.56 centímetros de largo, 5.6 centímetros de ancho y un peso de 45 gramos.

Las Raspberry Pi vienen con un sistema operativo gratuito de nombre Raspbian, que está optimizado para su hardware. Raspbian es una versión no oficial del sistema operativo Debian con configuraciones de compilación ajustadas para producir un código "flotante fuerte"(hard float) optimizado que se ejecuta en la Raspberry Pi. El código proporciona un rendimiento significativamente mayor para las aplicaciones que hacen un uso intensivo de las operaciones aritméticas de punto flotante. Las demás aplicaciones también obtienen una mejora de rendimiento mediante el uso de instrucciones avanzadas de la CPU ARMv6 en Raspberry Pi. Aunque Raspbian es principalmente el esfuerzo de Mike Thompson y Peter Green, desarrolladores del sistema en 2012, también se ha beneficiado enormemente del apoyo entusiasta de los miembros de la comunidad de Raspberry Pi que desean obtener el máximo rendimiento de sus dispositivos [28].

También es importante destacar que Raspbian viene con más de 35000 paquetes de software precompilado, optimizados para un mejor rendimiento, en un formato que permite una fácil instalación en las Raspberry Pi. Raspbian todavía está en desarrollo activo con énfasis en mejorar la estabilidad y el rendimiento de la mayor cantidad posible de paquetes tomados de Debian. Raspbian no está afiliado a Raspberry Pi Foundation [28].

Con respecto al trabajo realizado sobre la Raspberry Pi 2 modelo B, se logró instalar con éxito las bibliotecas de Python en los drones mediante la Raspberry, sin embargo con OpenCV no hubo éxito. El procedimiento para instalar el lenguaje Python en los drones fue el siguiente: en primer lugar se generó un ejecutable del lenguaje Python en la Raspberry, posteriormente se transfirieron esos archivos al sistema interno de los drones (donde está instalado el sistema operativo). Luego, basta con simplemente pasar el código implementado en Python al sistema interno de los drones y ejecutarlo usando las bibliotecas de Python. Para el caso de OpenCV se intentó seguir el mismo procedimiento. OpenCV cuenta con una versión funcional para Python, se instaló esa versión en la Raspberry y luego al tratar de transferir los archivos al sistema interno. Por causa de las restricciones internas que tiene el sistema operativo de los drones y el poco espacio de almacenamiento donde se encuentra instalado ese sistema operativo, no se logró la instalación de OpenCV. En la tarea de instalar la biblioteca OpenCV se invirtió un tiempo aproximado de dos meses.

Finalmente, los problemas mencionados en el párrafo anterior fueron la razón de que se desistiera de trabajar y de crear el sistema autónomo embebido en los drones comerciales Parrot Bebop 2. Aun así se continuó el proyecto utilizando un simulador de la empresa Parrot de nombre Sphinx, el cual se describe en la sección 3.6.

## 3.3. Elección del lenguaje de programación

Otra de las decisiones importantes fue la selección del lenguaje de programación a utilizar. En esta sección se discuten características, propiedades, ventajas y desventajas de los lenguajes C y Python.

### 3.3.1. Lenguaje C

Este proyecto de grado trata de seguir, en parte, la línea de trabajo del proyecto "Planificación e instrumentación de vuelo de una flotilla de drones"[19] intentando profundizar y continuar algunos aspectos e ideas que se plantearon previamente. Por lo tanto, una buena manera de continuar el trabajo anterior era iterar sobre el sistema de drones autónomos desarrollado. Dicho sistema está implementado en el lenguaje C y ejecuta sobre el simulador VRep, por ende, fue de importancia definir si se utilizaba o no este lenguaje como herramienta de trabajo.

El lenguaje de programación C es un lenguaje de propósito general desarrollado por Dennis Ritchie a principios de la década de 1970. C fue diseñado como un lenguaje de implementación de sistemas para el nacimiento del sistema operativo Unix. C es una evolución del lenguaje B el cual deriva del lenguaje carente de tipos: lenguaje de programación básico combinado (Basic Combined Programming Language, BCPL). En contraste con los dos anteriores, C evolucionó a una estructura de tipos. Fue creado como una herramienta para mejorar los entornos de programación pobres y se ha convertido en uno de los lenguajes dominantes de la actualidad. En 1983, el instituto americano de estándares nacionales (American National Standars Institute, ANSI) estableció un comité para proporcionar una moderna y compatible definición de C. La primera estandarización del lenguaje C fue con el estándar X3 159-1989, conocido como "ANSI C". En 1990, "ANCI C" fue ratificado como estándar por la organización internacional de estandarización (International Organization for Standardization, ISO) [29].

Algunas de las características principales del lenguaje C son [30]:

- "Bajo nivel" de programación. C trata con el mismo tipo de objetos que la mayoría de las computadoras, como los caracteres, los números y las direcciones.
- No proporciona operaciones para tratar directamente con objetos compuestos, tales como cadenas de caracteres, conjuntos, listas o arreglos.
- No define facilidades para asignación de almacenamiento, salvo la de definición estática y la disciplina de pilas provista por las variables locales de funciones; tampoco emplea heap ni recolector de basura.
- No proporciona capacidades de entrada y salida. No hay proposiciones READ o WRITE ni métodos propios de acceso a archivos. Todos esos mecanismos de alto nivel deben ser proporcionados por funciones llamadas explícitamente.
- Únicamente ofrece un control de flujo franco y lineal como condiciones, ciclos, agrupamientos y subprogramas pero no multiprogramación, operaciones paralelas, sincronización ni corrutinas.

Alguna de las ventajas que proporcionaba el lenguaje C para la realización del proyecto son que este lenguaje, al ser de "bajo nivel", cuenta con grandes ventajas en la eficiencia. Adicionalmente, los drones Bebop 2 tienen incorporado un sistema operativo Linux de base y el lenguaje C se desempeña mejor en este tipo de sistemas. Además, como se destacó anteriormente en esta sección, el trabajo realizado en el proyecto "Planificación e instrumentación de vuelo de una flotilla de drones" fue implementado en C, por lo que seguir esa línea de trabajo era favorable. Por el lado de las desventajas, utilizar el lenguaje C implicaba que se debería hacer compilación cruzada para insertar el código dentro de los drones (debido a que las herramientas con las que se desarrollaba el sistema, son de distinta arquitectura que los drones).

### 3.3.2. Lenguaje Python

Cuando se desarrolló la etapa de investigación de este trabajo, se encontró la biblioteca PyParrot. PyParrot está implementado en Python, para ser utilizado también en dicho lenguaje, esto hizo que se analizara su utilización y se investigaran las posibilidades de Python para cumplir con los objetivos propuestos en este proyecto.

Python es un lenguaje de programación interpretado de "alto nivel" y propósitos generales, desarrollado por Guido van Rossum en el año 1991. Python fue creado en el Centro para las Matemáticas y la Informática (*Centrum Wiskunde Informatica*, CWI) en Holanda, como sucesor del lenguaje llamado ABC. En 1995, Guido Van Rossum continuó su trabajo en Python en la Corporación para Iniciativas Nacionales de Investigación (*The Corporation for National Research Initiatives*, CNRI) en Reston, Virginia, donde lanzó varias versiones del software [31]. Para el desarrollo de este proyecto se utilizó la versión 3.4.

Algunas de las características con las que cuenta Python son [32]:

- Es un lenguaje dinámicamente tipado: en Python no se define el tipo de las variables (enteros, cadena de caracteres, etc) sino que se le fija un tipo a cada variable según el valor que se quiera asignar.
- Es un lenguaje fuertemente tipado: aunque Python es dinámicamente tipado, no permite realizar operaciones entre variables de tipo distinto (por ejemplo, sumar un entero con una cadena de caracteres).
- Es un lenguaje multiparadigma: aunque Python está principalmente pensado para la programación orientada a objetos, soporta otros tipos de paradigmas como programación imperativa, programación funcional y programación procedural.
- Es un lenguaje interpretado: en Python, el código fuente no es compilado a código de máquina sino que hay un intérprete que es el que ejecuta el programa basándose en el código directamente.

- Es un lenguaje con recolector de basura (garbage collector): Python cuenta con recolector de basura dinámico, que permite que el programador no tenga que preocuparse de liberar la memoria sin uso, Python lo hace por sí mismo.

- Es un lenguaje de "baterías incluidas" (batteries included): Python es considerado un lenguaje de "baterías incluídas" debido a su completa biblioteca estándar, que permite a los usuarios desarrollar sin la necesidad de descargar ningún paquete de desarrollo por separado.

Algunas de las ventajas que proporciona el lenguaje Python para la realización de este proyecto son: en primer lugar, permite el uso de la biblioteca PyParrot (ideal para la programación de drones Bebop 2); por otra parte, aunque es un lenguaje de más alto nivel en comparación a C, es muy eficiente y por sus filosofías cuenta con un estilo de código legible y fácil de entender para cualquier desarrollador; por último, no es necesario hacer compilación cruzada (a diferencia de C), lo cual reduce en gran medida el tiempo de trabajo del equipo de desarrollo. En cuanto a las desventajas, para poder hacer uso de Python era necesario la instalación del lenguaje dentro de los drones, tarea que no era trivial. En contraste, el lenguaje C no era necesario instalarlo ya que se encontraba instalado dentro del sistema de los drones. Otra desventaja es que el trabajo hecho en el proyecto "Planificación e instrumentación de vuelo de una flotilla de drones" está implementado en C.

Luego de analizar detenidamente las ventajas y desventajas se decidió usar el lenguaje de programación Python para la implementación de este proyecto. Las ventajas que proporciona el lenguaje sumado a la posibilidad de utilizar la biblioteca PyParrot hicieron que se optara por ésta opción.

## 3.4. Procesamiento de imágenes

En esta sección se describe la biblioteca OpenCV, la cual es utilizada para el procesamiento de imágenes. Posteriormente se describen los pasos seguidos para la instalación de la biblioteca dentro de los drones y se describe la maquina de vectores de soporte (SVM).

### 3.4.1. OpenCV

Después de haber definido que se utilizaría Python como el lenguaje de trabajo y la biblioteca PyParrot, fue necesario definir qué biblioteca se iba a utilizar para el procesamiento de imagen (en caso de utilizar alguna). En este proyecto se optó por la biblioteca de visión de computadoras abierta (Open Computer Vision Library, OpenCV), que es utilizada por la biblioteca PyParrot para realizar algunas operaciones vinculadas con el uso de la cámara del drone y además existe una versión en lenguaje Python de la biblioteca OpenCV. Estas dos razones fueron determinantes para el uso de OpenCV.

OpenCV es una biblioteca multiplataforma de visión de computadoras y aprendizaje automático de código abierto creada en el año 1999 por la corporación Intel junto a un grupo de investigadores [33]. OpenCV fue construida para proporcionar una infraestructura común para las aplicaciones de visión artificial y acelerar el uso de la percepción de máquina en los productos comerciales. Al ser un producto con licencia de distribución de software de Berkeley (Berkeley Software Distribution, BSD), OpenCV facilita que las empresas utilicen y modifiquen el código. OpenCV tiene interfaces para C/C++, Python, Java y MATLAB, además es compatible con los sistemas Windows, GNU/Linux, Mac OS X y Android [5]. En este proyecto se utilizó la versión 4.0 de OpenCV para Python.

La biblioteca OpenCV cuenta con más de 2500 algoritmos optimizados, que incluyen un conjunto completo de algoritmos clásicos y de

vanguardia, de visión de computadora y de aprendizaje automático. Estos algoritmos se pueden usar para detectar y reconocer rostros, identificar objetos, clasificar acciones humanas en vídeos, rastrear movimientos de cámara, rastrear objetos en movimiento, extraer modelos 3D de objetos, unir imágenes para producir una alta resolución imagen de una escena completa, encontrar imágenes similares de una base de datos de imágenes, reconocer paisajes y establecer marcadores para superponerlos con realidad aumentada, entre otros [5].

Las funciones proporcionadas por la biblioteca OpenCV se pueden agrupar en los siguientes bloques [34]:

- Estructuras y operaciones básicas: matrices, grafos, árboles, etc.
- Procesamiento y análisis de imágenes: filtros, momentos, histogramas, etc.
- Análisis estructural: geometría, procesamiento del contorno, etc.
- Análisis del movimiento y seguimiento de objetos: plantillas de movimiento, seguidores, flujo óptico, etc.
- Reconocimiento de objetos: objetos propios (eigen objects), modelos HMM, etc.
- Calibración de la cámara: morphing, geometría epipolar, estimación de la pose, etc.
- Reconstrucción tridimensional (funcionalidad experimental): detección de objetos, seguimiento de objetos tridimensionales, etc.
- Interfaces gráficas de usuarios y adquisición de video.

En cuanto a las ventajas de utilizar la biblioteca OpenCV para este proyecto se puede indicar que facilita significativamente el procesamiento de imágenes, desde la lectura de archivos hasta el análisis de las imágenes. Por otra parte, las operaciones de la biblioteca son sencillas de utilizar y

hay una cantidad considerable de documentación y guías de cómo realizar el procesamiento de imágenes. Sin embargo, una gran desventaja es que se debe instalar la biblioteca dentro de los drones, que es una tarea considerablemente compleja.

### 3.4.2. Instalación de OpenCV en los drones

Para poder realizar el sistema de drones autónomos usando los drones Bebop Parrot 2 es necesario instalar la biblioteca OpenCV. Instalar OpenCV requiere incluir en el sistema operativo de los drones la versión de la biblioteca para Python y poder ejecutarla dentro. El sistema operativo de los drones está restringido (dispone de poca memoria y sitios del disco sin acceso de propietario) y no proporciona ninguna herramienta avanzada de empaquetado como para poder instalar la biblioteca, por lo cual se buscó hacer un procedimiento análogo al de Python.

Se trató de usar la Raspberry Pi como intermediaria para la instalación de OpenCV. Los pasos realizados fueron: primero se instaló OpenCV para Python en la Raspberry y luego se intentó transferir todos los archivos generados en la instalación al sistema interno de los drones; no se tuvo un resultado satisfactorio debido a que las restricciones del sistema operativo no permitieron copiar los archivos (de OpenCV) en las rutas correspondientes dentro del sistema de archivos; otra limitante fue el disco interno (donde está instalado el sistema operativo) que cuenta con poco espacio y la biblioteca OpenCV excedía el espacio disponible. Luego de dos meses de trabajo se decidió descartar la idea de instalar OpenCV dentro de los drones.

A pesar de que se había descartado la idea de usar OpenCV dentro de los drones, se estudiaron otras opciones para intentar desarrollar un sistema de drones autónomos. La opción que tomó mayor relevancia fue la de incorporar hardware externo a los drones Bebop Parrot 2, en especial usar la micro computadora Raspberry Pi 2. Sin embargo, esta idea fue descartada por las siguientes razones:

- Problemas de consumo de energía: usar un hardware externo implica tener que proporcionar alguna fuente de energía para alimentarlo, por lo tanto las únicas opciones posibles son usar la batería de los drones o usar una batería externa. Usar una batería externa fue inviable debido a que aumentaba considerablemente el peso de los drones y no les permite maniobrar o desplazarse correctamente. Usar la batería de los drones también era inviable debido a que se veía afectada en gran medida la autosuficiencia energética de los drones.
- Inestabilidad de movimiento: al ser los drones vehículos de pequeño tamaño, era necesario seleccionar una ubicación correcta para el hardware externo debido a que podía afectar la estabilidad o el vuelo de los drones. Además de lo mencionado, agregar un hardware externo a los drones repercute en su peso, por lo tanto los rendimientos en sus velocidades, su aceleración y su aerodinámica no iban a ser los esperados con respecto a un drone de igual modelo que no tuviese hardware externo.
- Daños por agentes externos: al estar por fuera de la carcasa de los drones, el hardware externo es vulnerable al clima, al ambiente, objetos peligrosos que vuelan por el viento o choques del vehículo con otros objetos, que podrían llegar a dañar el hardware y de esa manera destruir el sistema de vigilancia.

Como conclusión, se decidió no continuar con la idea de hacer un sistema de drones autónomos usando los Bebop Parrot 2. Las diferentes dificultades que se presentaron al momento de armar el ambiente de trabajo, hicieron que se declinara la idea de usar los Bebop y en su lugar dar uso del simulador proporcionado por la empresa Parrot, de nombre Sphinx, que se describe en la sección 3.6.

### 3.4.3. Support Vector Machine

Para poder detectar intrusos se necesita de un sistema previamente entrenado para clasificar personas. En este proyecto de grado se utilizó

(Suppor Vector Machine, SVM). La máquina de vectores de soporte (SVM) es un conjunto afín de métodos de aprendizaje supervisado que son populares por su performance en clasificación y en análisis de regresión al usar análisis de datos y patrones de reconocimiento. Los métodos varían en los atributos y la estructura del clasificador. El más conocido de los SVM es un clasificador lineal (usado en este proyecto) que predice la clase miembro de cada elemento de entrada entre dos posibles clasificaciones. Una definición más precisa es que el SVM construye un hiper-plano o un conjunto de hiper-planos para clasificar todas las entradas en un espacio de alta dimensión o incluso infinitas dimensiones. Los valores más cercanos al margen de clasificación se conocen como vectores de soporte. El objetivo de SVM es maximizar el margen entre el hiper-plano y los vectores de soporte [35].

## 3.5. Descripción de la biblioteca PyParrot

En la búsqueda de opciones para implementar el sistema de drones autónomos se encontró la biblioteca PyParrot. PyParrot es una biblioteca desarrollada y diseñada en el lenguaje Python por Amy McGovern. Esta biblioteca fue desarrollada para programar drones Parrot de los modelos Minidrone Mambo FPV, Minidrone Swing, Bebop 1 y Bebop 2. La razón para desarrollar esta biblioteca fue para enseñar a los niños de todas las edades los conceptos de programación y matemática al hacer que programen un dron para volar de forma autónoma. La liberación de PyParrot con la que se trabajó para este proyecto fue la 1.5.3.

Como toda biblioteca desarrollada por terceros, PyParrot proporcionó ciertas ventajas y desventajas. Como principales ventajas, PyParrot permite usar el lenguaje de programación Python, la extensa variedad de métodos de la biblioteca facilita en gran medida la implementación de un sistema de drones autónomos permitiendo realizar diferentes movimientos y utilizar los diversos sensores con los que cuenta un drone, y finalmente los tiempos de desarrollo eran menores utilizando la biblioteca a que si no se usara, debido a que si no se usara la biblioteca se tendría que haber implementado, por ejemplo, un gran cantidad de funciones para ejecutar

distintos movimientos en el espacio con los drones. Como principales desventajas, con PyParrot no se podía reutilizar el código implementado en el proyecto "Planificación e instrumentación de vuelo de una flotilla de drones", debido a que la biblioteca PyParrot solo se encontraba implementada en lenguaje Phyton y el proyecto mencionado está desarrollado en el lenguaje C; a su vez esta biblioteca permite la manipulación de los drones a "alto nivel", por lo tanto, como desventaja, se debió modificar la biblioteca para hacer algunas revisiones en el desplazamiento de los drones.

Existe una documentación de PyParrot con explicaciones detalladas de cómo instalar la biblioteca, con qué métodos cuenta, cómo se utilizan dichos métodos, entre otras funcionalidades. Algunas de las funcionalidades utilizadas en este proyecto fueron:

- Métodos de conexión: PyParrot cuenta con un par de métodos que permiten a los desarrolladores conectarse y desconectarse a los drones.
- Métodos de configuración: PyParrot cuenta con algunos métodos que permiten parametrizar el vuelo de los drones como por ejemplo limitar la distancia de vuelo máxima, determinar la máxima altitud que pueden alcanzar o establecer el máximo de inclinación y balanceo en grados.
- Métodos de consulta de estado: PyParrot ofrece algunos métodos que permiten consultar el estado del drone en todo momento. Se puede obtener el estado completo del drone o se pueden verificar diferentes atributos del drone, como saber si el drone ha aterrizado.
- Metodos de vuelo y desplazamiento: PyParrot dispone de distintos métodos para el vuelo y desplazamiento de los drones, uno para realizar movimientos relativos donde se le indica al drone un vector de movimiento a partir de la posición actual y otro de vuelo directo donde se indica la posición a la que debe llegar el drone y el mismo se mueve hacia ahí. Además, la biblioteca cuenta con movimientos para dar vuelta hacia una dirección o estacionar.
- Métodos de lectura de sensores: PyParrot tiene métodos para actualizar los sensores y tener lectura de su información.

- Métodos para la manipulación de cámara: PyParrot provee varios métodos para el uso de la cámara integrada que tiene los drones. Alguno de los metodos son para inclinar la cámara en un numero especifico de grados o para establecer el formato de imagen.

## 3.6. El simulador Sphinx

Ante la realidad de no poder trabajar sobre los drones Bebop 2, se buscó un simulador para poder desarrollar el sistema de seguridad con drones autónomos. Como resultado de la búsqueda, se encontró Sphinx (https://developer.parrot.com/docs/sphinx/whatissphinx.htm), un software creado por Parrot que permite simular un escenario y diversos vehículos aéreos de la misma marca.

Sphinx es una herramienta de simulación pensada para cubrir las necesidades de los desarrolladores Parrot. La idea principal de este software es poder ejecutar un firmware de Parrot en una computadora, utilizando varias funcionalidades del software Gazebo para simular el entorno físico y visual de los drones.

Gazebo (http://gazebosim.org) es un software de simulación de entornos de alta fidelidad desarrollado en el año 2002 en la universidad de california del sur. Los creadores originales fueron Andrew Howard y Nate Koenig. El concepto de un simulador de alta fidelidad surgió de la necesidad de simular robots en entornos al aire libre en diversas condiciones.

Sphinx cuenta con funcionalidades de ayuda para el desarrollo con simulación de aeronaves como: visualizar la información de vuelo en tiempo real, modificar el comportamiento del drone en tiempo de ejecución, tener un sistema y entorno totalmente programable, permitir construir escenarios de vuelo propios, simular sensor de vídeo, permitir el uso de todo tipo de controles de vuelo como el FreeFlight y permitir ejecución remota.

Para poder ejecutar Sphinx se necesita obligatoriamente un sistema operativo Linux 64bits, más precisamente se requiere de un Ubuntu 18.04 (Bionic) o Ubuntu 16.04 (Xenial) o Debian 9 (Stretch) o Debian 8 (Jessie). Además se necesita 1 GB de almacenamiento para su instalación y que la computadora soporte OpenGL en su versión 3.0 o superior. Open Graphic Library es una API multilenguaje y multiplataforma para desarrollar aplicaciones que produzcan gráficos 2D y 3D.

Cabe destacar que los requerimientos mínimos no son suficientes para acceder a todas las funcionalidades de Sphinx. Para poder acceder a todas las funcionalidades hay unos requerimientos recomendados como: tener Ubuntu 18.08 con Linux kernel 4.15, tener una tarjeta gráfica NVIDIA con los últimos drivers, tener conexión a Internet, tener bluetooth, tener Wifi y adaptador USB Wifi.

Durante el uso del simulador se presentaron algunos inconvenientes. A pesar de ello, se decidió seguir utilizando dicho simulador ya que se consideraban mayores las ventajas que ofrecía frente a las desventajas. Los problemas presentados se detallan a continuación:

- Vientos en modo de ráfagas: Si bien se logró simular viento, solamente se pudo lograr que sea lineal uniforme. Durante el periodo de implementación se logró la simulación del viento, para esto se agrega un vector que indica la velocidad en las direcciones $X\ Y\ Z$ en el archivo *world* que representa el escenario simulado. Los valores se miden en metros por segundo según el estándar de ROS [36]. Si bien al levantar vuelo el drone muestra una dificultad interesante porque se produce un desvío en el vuelo hacia un lado, logra recuperarse por sí solo y una vez que empieza a realizar el recorrido lo hace volando de forma inclinada contrarrestando el viento, por lo que no se aprecia ningún error en el recorrido. Este vuelo inclinado es automático. Si se aumenta la velocidad del viento a una cantidad de, por ejemplo 10 metros por segundo, el drone en un primer momento se desvía, presumiblemente, unos 10 metros. Al suceder esto el drone se posiciona contra el viento e intenta volver a la celda objetivo. No lo consigue por ser

el viento muy fuerte y continúa desviándose de forma uniforme, es decir siempre a la misma velocidad, en dirección del viento.

No se consiguió agregar al simulador viento en modo de ráfagas para visualizar otros comportamientos más naturales y frecuentes en el mundo real. Dado que el drone no presenta dificultades con poco viento y con mucho viento la solución es inviable, se decidió hacer todos los casos de prueba sin viento.

- Simulación múltiple: Si bien se logró una simulación correcta para un drone, no se pudo obtener los mismos resultados para varias instancias de drones dentro de la misma simulación. Este error es conocido (https://forum.developer.parrot.com/t/running-multiples-drones-in-a-simulation/7646/18), pero en el transcurso del desarrollo del proyecto no hubo actualización que resuelva este inconveniente, por lo que las pruebas con múltiples drones se realizaron en múltiples simuladores. Esto es posible porque la posición y estado de otros drones es conocida mediante distintos mensajes que pueden ser enviados entre distintos dispositivos. Además fuera de estos mensajes, el comportamiento de un drone no esta relacionado con los movimientos de los otros, por lo que es viable ejecutar el simulador en dos máquinas distintas. A pesar de que no se ven en la misma pantalla ni están en el mismo escenario simulado, la comunicación entre los drones se realiza de forma correcta, y a los efectos del algoritmo desarrollado, se toman las mismas decisiones que si se simularan los drones en la misma máquina.

- Cambio de coordenadas GPS y magnetismo: Para que la simulación sea más realista, se decidió cambiar las coordenadas GPS por defecto del escenario (París, Francia) a las del predio seleccionado para hacer las pruebas de este proyecto (INCO, Fing, Montevideo). Para ésto se modificó el archivo *world*. Éste archivo es un archivo de formato XML que contiene las especificaciones y características del escenario simulado. Además se agregaron "modelos" (i.e. objetos modelados dentro del mundo simulado) al *world* simulando, por ejemplo los muros del predio.

Al cambiar los atributos latitud y longitud en el archivo *world* se notó que en una nueva simulación el drone se movió de manera errática. Por ejemplo, al ordenarle ir en línea recta, realizó una diagonal con una inclinación entre 5 y 10 grados. Investigando, se encontró que no era suficiente con cambiar las coordenadas GPS sino que también hay que cambiar los vectores del campo magnético en el archivo *world* (https://developer.parrot.com/docs/sphinx/worldfile.html). El simulador tiene en cuenta valores del campo magnético, por lo que si no corresponden con los reales en el lugar del mundo que indican las coordenadas GPS, los objetos (sobre todo los voladores) pueden tener comportamientos no esperados en su movimiento. Para encontrar el vector magnético adecuado se utilizó el magnetic field calculator del National Centers For Environmental Information (NCEI).

## 3.7. Otras herramientas utilizadas

En el desarrollo del sistema de drones autónomos se usaron otras herramientas como Ubuntu 16.04 y Oracle VM VirtualBox para su implementación. En esta sección se realiza una breve reseña sobre estas herramientas.

### 3.7.1. Oracle VM VirtualBox

Oracle VM VirtualBox (https://www.virtualbox.org) es un potente producto de virtualización x86 y AMD64/Intel64 para empresas y uso doméstico, desarrollado por la corporación Oracle en el año 2007. VirtualBox no solo es un producto realmente completo en funciones y alto rendimiento sino que también es una de las únicas soluciones profesionales que está disponible gratuitamente como software de código abierto. VirtualBox soporta diversos sistemas operativos como GNU/Linux, Mac OS X, OS/2 Warp, Genode, Windows y Solaris.

Durante este proyecto se intentó usar una máquina virtual para tener un ambiente homogéneo de trabajo, ejecutar el simulador sphinx y realizar el desarrollo del sistema de drones autónomos. Por distintas razones que se comenta en la sección 3.8, se descartó esta posibilidad.

### 3.7.2. Ubuntu 16.04

Ubuntu (https://ubuntu.com) es un sistema operativo de código abierto para computadoras, creado en 2004 por Canonical Ltd. y posteriormente mantenido por la fundación Ubuntu. Es una distribución de Linux basada en la arquitectura Debian. Ubuntu surge ante la realidad de que Linux estaba fragmentado en ediciones propietarias e incompatibles ediciones comunitarias, donde el software libre no era parte de la vida cotidiana de los usuarios. Mark Shuttleworth y un equipo de desarrolladores de Debian (fundadores de Canonical) se propusieron crear este sistema operativo Linux de escritorio.

Como se mencionó en la sección 3.6, para poder utilizar el simulador Sphinx era requisito utilizar alguno de los siguientes sistemas operativos: Ubuntu 18.04 (Bionic) o Ubuntu 16.04 (Xenial) o Debian 9 (Stretch) o Debian 8 (Jessie). Se decidió utilizar el sistema Ubuntu 16.04 debido a que el equipo de desarrollo ya estaba familiarizado con este sistema operativo y funcionaba correctamente en todas las computadoras de trabajo. Además, la versión 16.04 de Ubuntu tiene menores requerimientos que la versión 18.04.

## 3.8. Ambiente de desarrollo

En esta sección se detalla el análisis realizado para la elección del entorno de desarrollo en el cual se trabajó para la implementación del sistema de vigilancia con una flota de drones.

En primera instancia se probó realizar una máquina virtual, y en ella instalar Linux Ubuntu 16.04 con una distribución de 64 bits, luego se instaló Gazebo y Sphinx (requeridos por el simulador a utilizar), y por último se instaló pyhton en su versión 3.6, dado que es la versión recomendada por la biblioteca utilizada para la navegación (pyparrot).

El primer inconveniente que surgió al utilizar la máquina virtual es que al ejecutar cualquier código, la función de pyparrot que utiliza la función primitiva del SDK para conectarse al drone retornaba un error indicando de que no era posible conectarse con el drone del simulador. Esto se debe a que el simulador se apropia de una de las interfaces wifi de la computadora donde se ejecuta el simulador y luego el drone simulado crea su punto de acceso de forma similar que el drone real lo hace con su interfaz wifi. El problema de esta funcionalidad del drone simulado es que la máquina virtual no tiene acceso a la interfaz wifi del host (la maquina física) por lo que el drone simulado no encuentra ninguna interfaz wifi de la cual apropiarse y crear su punto de acceso. Se encontró que la máquina virtual si es capaz de detectar interfaces wifi si éstas están en un adaptador wifi USB. Agregando este hardware a la máquina y mediante ciertas configuraciones en la máquina virtual, se logra que el drone simulado se apropie de la interfaz wifi del adaptador USB y de este modo poder ejecutar el código y que el drone reciba las órdenes del mismo.

Una vez solucionado el problema de la conexión con el drone, se notó que el rendimiento del simulador en la máquina virtual era ineficiente (la simulación no era fluida) debido a la gran cantidad de recursos que necesita. Se decidió terminar con los intentos de usar el simulador en la máquina virtual y se optó por instalar Ubuntu directamente en las computadoras personales.

Por último, otro problema que se encontró fue al momento de implementar y sumar el módulo de comunicación al sistema. Cómo se menciona en la sección 4.2, el módulo esta compuesto por un servidor y un cliente que necesitan una conexión a la red wifi para poder intercambiar mensajes con otros drones. El drone simulado se apropia de la interfaz wifi, por lo que impide que el servidor y cliente puedan conectarse a la red wifi local.

El adaptador wifi USB resulta de utilidad ya que al usarlo se agrega otra interfaz wifi que el drone simulado puede utilizar, dejando libre la interfaz wifi de la máquina para que sea usada por el servidor y el cliente para conectarse a la red wifi.

# Capítulo 4

# Implementación

En este capítulo se presentan las características del sistema de vigilancia implementado. La sección 4.1 describe la arquitectura y sus componentes. La sección 4.2 describe el módulo de comunicación. La sección 4.3 describe los procedimientos necesarios para la navegación. La sección 4.4 describe el algoritmo de detección y seguimientos de intrusos. Finalmente, en la sección 4.5 se presenta aspectos de configuración y depuración.

## 4.1. Arquitectura de la solución

La arquitectura que se diseñó para resolver la problemática planteada en este proyecto consta de cuatro componentes fundamentales: controlador, navegación, comunicación y procesamiento de imágenes (ver figura 4.1). La integración de los componentes se basa en una máquina de estados (ver figura 4.2). Se buscó que los componentes tengan bajo acoplamiento y alta cohesión debido a la complejidad de cada uno de ellos. Cuanto más independiente son los componentes, la solución es más mantenible y escalable.

### Componentes de la Arquitectura

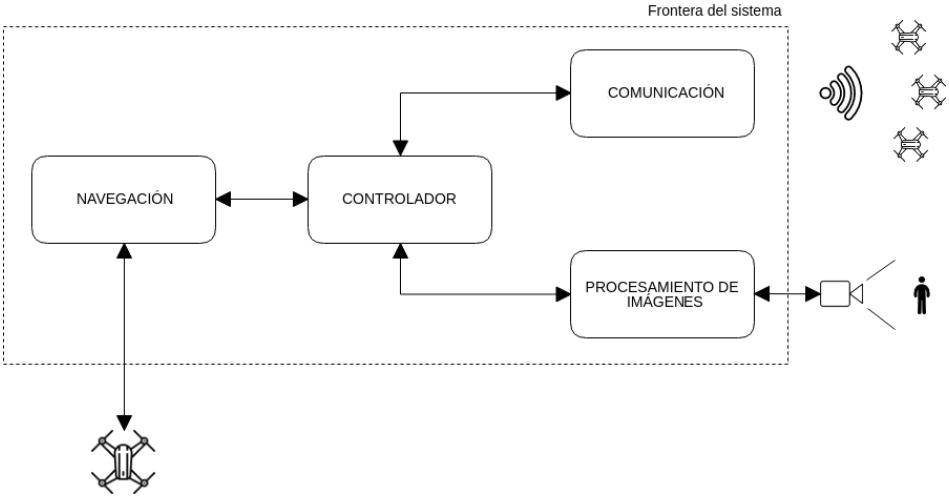


Figura 4.1: Arquitectura de la solución propuesta.

El componente controlador es el encargado de la toma de decisiones, y para ello se basa en la máquina de estados que determina a qué estados puede proceder dependiendo cuál sea el actual. Las transiciones posibles son determinadas dependiendo de la información recibida desde los restantes drones mediante el componente comunicación, la información relevada por los sensores del propio drone y el componente de procesamiento de imágenes.

El componente comunicación es el encargado de la comunicación entre los drones de la flota. Este componente se divide en dos módulos; el servidor, que es el encargado de escuchar todos los mensajes recibidos en un puerto acordado previamente y de descifrar el mensaje, y el cliente, que es el encargado de cifrar el mensaje a enviar y enviarlo a quien corresponda (ver capítulo 4.2).

La principal función del componente navegación es dar las órdenes al drone para que éste se mueva. Este componente se divide en tres módulos y cada uno de éstos módulos trabaja sobre un nivel de abstracción diferente. En primer lugar se encuentra la biblioteca pyparrot, que es la encargada de la comunicación directa con el kit de desarrollo de software (*Software Development Kit*, SDK) de parrot. En segundo lugar se encuentra un módulo *drone*, que es el encargado de mantener el estado actual del drone y de realizar las primitivas de movimiento (aterrizar, despegar y moverse a una coordenada GPS específica), invocando a las funciones brindadas por la biblioteca pyparrot. Por último, y a más alto nivel, se cuenta con un módulo de *cálculo de ruta*, que es el encargado de calcular los movimientos que debe de realizar el drone teniendo en cuenta no sólo su información, sino también la de los restantes drones de la flota (ver capítulo 4.3.2).

Las principales funciones del componente de procesamiento de imágenes son detectar intrusos, realizar el seguimiento y comunicarle al controlador los datos relevantes de cada detección (ver capítulo 4.4).

Para implementar la arquitectura propuesta se decidió hacer una separación a nivel de hilos de procesamiento (threads). Cada componente (comunicación, procesamiento de imágenes y controlador) ejecuta en su propio thread. La principal ventaja de esta implementación es que los componentes están permanentemente ejecutando en paralelo, permitiendo detectar un intruso independientemente del estado en que se encuentra, así como informar y analizar los datos al mismo tiempo que se envían o reciben otros.

### Máquina de estados del problema

La máquina de estados proporciona información del estado en el que se encuentra cada drone, además de poder determinar las acciones a tomar dependiendo de los datos de entrada como pueden ser mensajes de otros drones ó eventos externos detectados por los sensores.

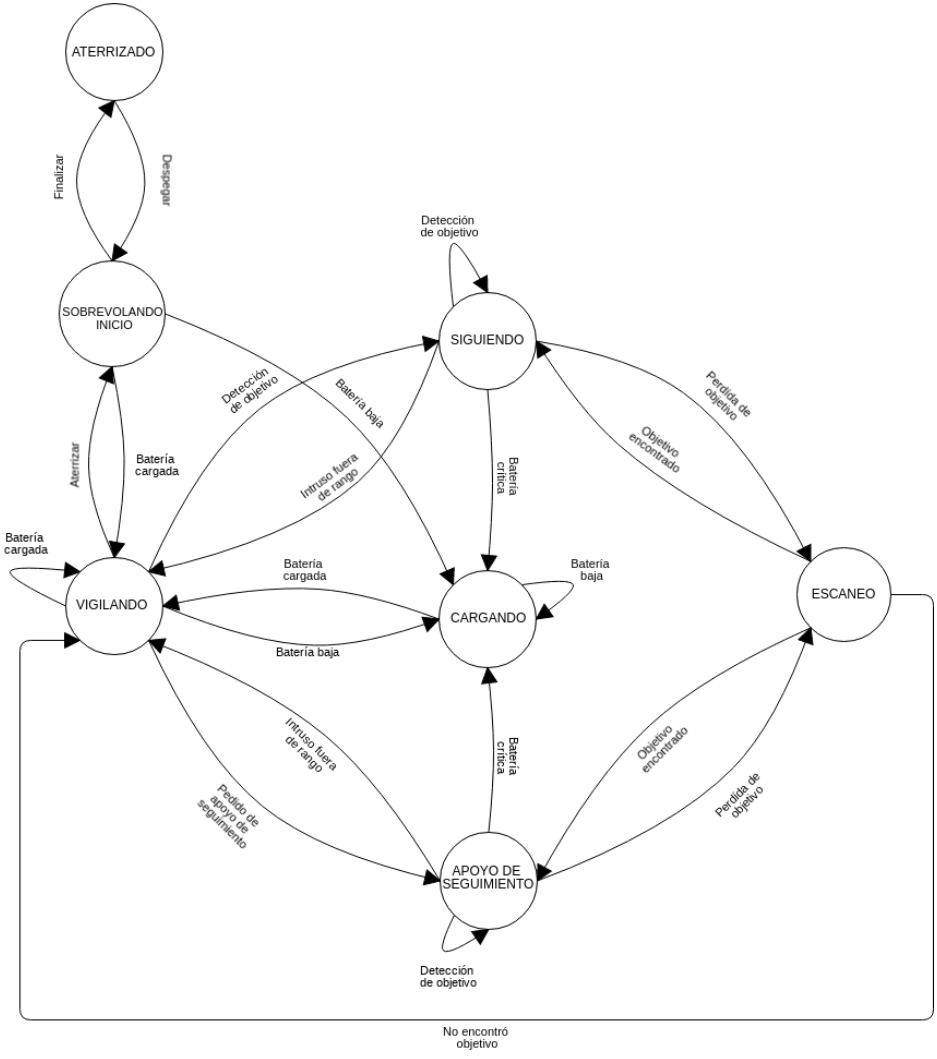


Figura 4.2: Máquina de estados.

Los posibles estados y transiciones del sistema son:

- **Aterrizado:** El drone se encuentra en la posición inicial aterrizado.
- **Sobrevolando inicio:** El drone se encuentra "flotando" luego de haber despegado sobre la posición inicial, posteriormente comprueba el estado actual de la batería; en caso que tenga batería baja procede al estado *Cargando*, en caso contrario, cambia al estado *Vigilando* donde se calcula la primer trayectoria que debe tomar (se explicará en detalle en la sección 4.3).
- **Cargando:** El drone se encuentra cargando la batería. Existen dos transiciones posibles para que el drone cambie de estado, en primer lugar, cuando la batería del drone se carga en su totalidad y el drone pasa al estado *Vigilando*. En segundo lugar, en caso de que la batería se haya cargado hasta el cincuenta por ciento y otro UAV requiera la intervención del drone, éste pasa al estado *Apoyo de seguimiento*.
- **Vigilando:** El drone se encuentra realizando la vigilancia sobre un camino previamente calculado. Permanecerá en este estado siempre y cuando tenga batería, no detecte ningún intruso y no le soliciten ayuda de seguimiento. En caso de detectar un intruso, el drone pasará al estado *Siguiendo*, si le solicitan ayuda de seguimiento cambiará al estado *Apoyo de seguimiento* y por último si no cuenta con batería suficiente, tendrá que ir a cargar y pasará al estado *Cargando*.
- **Seguimiento:** El drone realiza el seguimiento del intruso detectado por él, y lo seguirá siempre y cuando esté dentro del predio a vigilar y cuente con batería suficiente (el nivel de batería es superior al que requiere para hacer el próximo paso y volver a la estación de carga). En caso de que la batería llegue a su estado crítico (batería mínima necesaria para llegar a la estación de carga) el drone procede al estado *Cargando*. Si por algún motivo el drone pierde el rastro del intruso durante un lapso de tiempo, cambia al estado *Escaneo*, y finalmente, si el intruso detectado está fuera del predio a vigilar, vuelve al estado *Vigilando* donde calcula un nuevo camino a seguir.

- **Apoyo de seguimiento:** El drone realiza seguimiento de un intruso reportado por otro drone. Los criterios para que el drone cambie de estado son los mismos que para el estado *Seguimiento*. El drone irá a la posición indicada por el drone que solicitó el apoyo y se posicionará en un ángulo de 90 grados a éste, detectando al intruso de modo de que entre ambos drones se obtenga una mayor cobertura visual.

- **Escaneo:** El drone se mueve de lado a lado buscando una mayor cobertura del campo de visión tratando de detectar el intruso que previamente perdió; si lo detecta, vuelve al estado *Siguiendo* y si por un lapso de tiempo no lo detecta, procede al estado $Vigilando$.

## 4.2. Módulo de comunicación

La comunicación en un sistema de seguridad autónomo es de gran importancia ya que permite la coordinación y sincronización entre los drones. Por esta razón se creó un módulo de comunicación mediante el cual los drones puedan enviarse mensajes de forma directa, sin necesidad de agregar equipamiento externo a la arquitectura del drone. Para logralo, es necesario definir un protocolo que establezca cuándo y de qué forma se intercambia la información, así como también el contenido y formato de los mensajes.

### 4.2.1. Medio de comunicación

Se evaluaron varias alternativas como medio de comunicación. Inicialmente se eligió trabajar con una red ad-hoc, ya que no dependen de una infraestructura preexistente para su funcionamiento, sino que los drones están conectados entre sí sin la necesidad de usar un punto de acceso intermedio. Para el caso particular de un sistema de vigilancia mediante una flota de drones, las redes ad-hoc presentan dos grandes ventajas. En primer lugar, la posibilidad de que cada drone pueda seguir operando sin necesidad de estar conectado a toda la red. En segundo lugar, si dos dro-

nes se encuentran fuera de alcance entre sí, puede haber un tercer drone que funcione como intermediario. A modo de ejemplo (ver figura 4.3), si el Drone1 no puede establecer conexión directa con el Drone2, el Drone3 puede funcionar como intermediario.

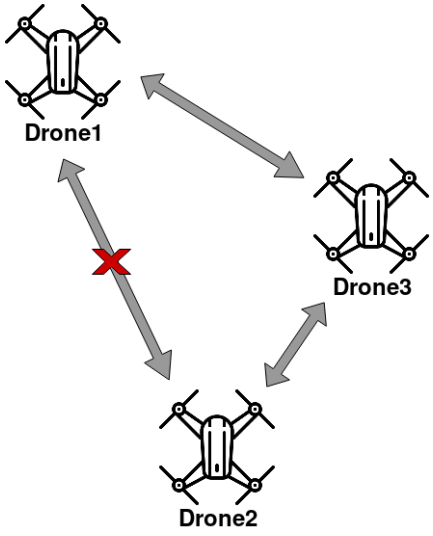


Figura 4.3: Red ad-hoc de una flota de drones.

Sin embargo, el problema con los drones Parrot Bebop 2 es que la tarjeta de red no incluye drivers para el modo ad-hoc. Los drones poseen una tarjeta de red integrada Broadcom BCM4360 [37] que solo incluye de fábrica drivers para los modos maestro y gestionado. Una posible opción para utilizar redes ad-hoc es incorporar el modo ad-hoc a los drivers que traen los Bebop 2, pero hacerlo requiere efectuar operaciones complejas y riesgosas sobre el sistema operativo que pueden dañarlo. Debido a este inconveniente, se decidió descartar el uso de una red ad-hoc.

Como alternativa a una red ad-hoc se propone usar una red WiFi, este enfoque presenta varias ventajas. En primer lugar, utilizar una red WiFi simplifica la implementación del sistema de comunicación ya que lo drones Parrot Bebop 2 cuentan con los drivers para el modo WiFi y se evita la etapa de configuración sobre la tarjeta de red. En segundo lugar,

su alcance es superior respecto al de las redes ad-hoc, lo cual incrementa el área que los drones pueden cubrir y por tanto el área que pueden vigilar de forma coordinada. Sin embargo, presenta la desventaja de que los drones dependen de una infraestructura externa que mantenga la red. A pesar de ello, cómo el predio a vigilar es conocido y de tamaño reducido, es razonable obviar ésta problemática e imponer como requisito previo para la utilización del sistema que el lugar a vigilar deba poseer una red WiFi.

Se analizaron dos posibles arquitecturas para la red WiFi. La primera es la arquitectura maestro-esclavo [38], en redes de computadoras, maestro-esclavo es un modelo para un protocolo de comunicación en el cual un dispositivo (conocido como el maestro) controla uno o mas dispositivos (conocidos como esclavos). Una vez que la relación maestro-esclavo se encuentra establecida, la dirección de control es siempre desde el maestro a los esclavos. Si bien la implementación es relativamente sencilla, se tiene como principal desventaja que el sistema genera dependencia del drone maestro, si por algún motivo se provoca una falla en éste, el sistema no funcionaría de manera correcta ya que se vería afectada la comunicación entre los drones, lo cual impediría la sincronización entre ellos. La segunda arquitectura analizada es una arquitectura distribuida [39], en redes de computadoras, un modelo distribuido es aquel en el que los datos y el procesamiento es distribuido entre todos los dispositivos. Con esta arquitectura no se depende de ningún drone en particular para el funcionamiento correcto del sistema, porque en caso de que un drone falle, el resto del sistema puede seguir funcionando de forma correcta. A pesar de que la implementación es más compleja, se decidió utilizar ésta arquitectura porque provee mayor fiabilidad al sistema.

### 4.2.2. Protocolo de comunicación

Para la elección del protocolo de comunicación se impusieron dos requisitos: en primer lugar, que no se pierdan mensajes, y en segundo lugar, que los mensajes lleguen tan pronto como sea posible. Se consideraron necesarios éstos requisitos porque en un sistema de vigilancia autónomo

con drones es imprescindible que los drones estén sincronizados. Si ocurre perdida de mensajes, un drone podría perder información fundamental sobre el estado del sistema y los caminos que están tomando otros drones afectando su toma de decisiones. Por otra parte, es deseable que los mensajes lleguen tan pronto como sea posible para que el drone receptor pueda tomar las decisiones en tiempo real, considerando las acciones que están realizando los restantes drones de la flota.

Para el envío de mensajes entre los drones se evaluaron dos protocolos de comunicación: TCP y UDP [4]. Una de las grandes diferencias que poseen éstos protocolos es que TCP es orientado a la conexión, maneja control de congestión y fiabilidad, por otro lado UDP es un protocolo de comunicación más liviano en el cual no hay conexiones, rastreos, ni ordenamiento de mensajes.

Si bien para este proyecto de grado se cuenta con dos drones, el sistema de vigilancia está construido para soportar una flota de N drones. El protocolo TCP garantiza que todos los mensajes llegarán a destino, sin embargo es necesario que cada drone establezca $N-1$ conexiones, una por cada drone de la flota, y al contar con un control de congestión, los mensajes podrían ser demorados si la red está muy congestionada. Por otra parte, el protocolo UDP enviará los mensajes tan pronto como sea posible, pero al no contar con un control de congestión, los mensajes podrían perderse y no llegar a destino.

Para el envío de mensajes entre los drones se eligió el protocolo de difusión (broadcast). Un protocolo de difusión se utiliza para enviar paquetes a todos los hosts en la red usando la dirección de broadcast para la red. Es decir, un nodo emisor envía información a una multitud de nodos receptores de manera simultánea, sin necesidad de reproducir la misma transmisión nodo por nodo.

Por las características de ambos protocolos, se consideró más apropiado utilizar el protocolo UDP ya que los mensajes llegarán a destino tan pronto como sea posible sin la necesidad de establecer una conexión. Además si se quiere enviar un mensaje a todos los drones de la flota usando

TCP se debe copiar y enviar el mensaje para cada conexión perteneciente a un drone de la flota, en cambio con UDP si todos los drones escuchan en el mismo puerto es suficiente con enviar un mensaje hacia ese puerto con la dirección de red broadcast para que el mensaje les llegue a todos los drones de la flota [4]. Sin embargo se hace necesario implementar una solución para el problema de la perdida de mensajes.

La solución propuesta para la perdida de mensajes es la siguiente: luego de que un drone envía un mensaje, esperará una respuesta por el drone receptor del mensaje en el que se le notificará al drone emisor que el mensaje enviado fue recibido correctamente, éste mensaje se denomina acknowledgement (ACK) [4]. En caso de que el drone emisor no reciba el ACK del drone receptor, luego de un intervalo de tiempo (timeout), se considera que el mensaje enviado fue perdido, en consecuencia el drone emisor reenviará el mensaje.

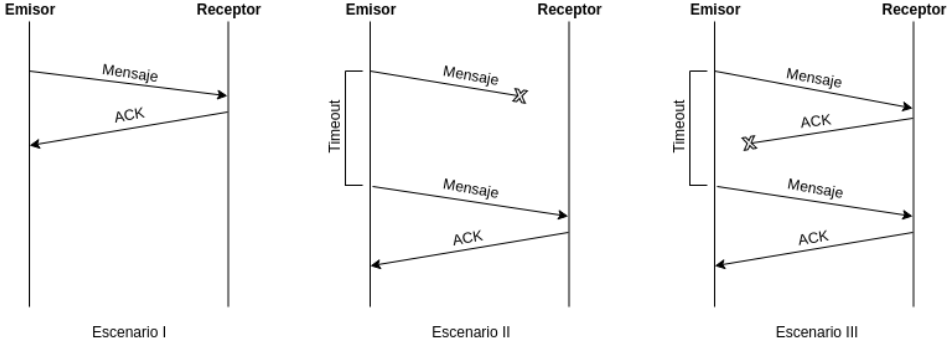


Figura 4.4: Escenarios de envíos de mensajes.

Existen tres escenarios posibles (ver figura 4.4). El escenario I es cuando no ocurre ningún error y el drone emisor recibe el mensaje ACK antes del timeout, por lo que se interpreta que el mensaje fue enviado correctamente. El escenario II es cuando se pierde el mensaje enviado por el drone emisor, para éste caso el drone receptor no enviará el mensaje ACK, por lo tanto, cuando ocurra timeout en el drone emisor reenviará el

mensaje. El escenario III es cuando se pierde el mensaje ACK que envía el drone receptor, por consiguiente, cuando ocurra timeout en el drone emisor reenviará el mensaje, sin embargo el mensaje reenviado por el emisor ya fue recibido por el receptor, por lo cual el receptor ignorará el mensaje (identificado por *messageId*) y reenviará el mensaje ACK.

Cada dos segundos el drone comunica su estado a los demás drones de la flota. De igual forma, al recibir un mensaje de actualización de estado por parte de otros drones, se actualiza localmente la información recibida por parte de los emisores, esta información permite que cada drone conozca el estado actual de los restantes integrantes de la flota. Dado que todos los drones de la flota utilizan el mismo canal de comunicación, en cada mensaje se incluye toda la información necesaria para el correcto funcionamiento del sistema. Dicha información es la siguiente:

- messageId: identificador del mensaje de tipo numérico.
- originDroneId: identificador de tipo numérico del drone que emite el mensaje.
- destinationDroneId: identificador de tipo numérico del drone que recibe el mensaje.
- type: enumerado que identifica el tipo de contenido que se envía en el mensaje.
- priority: asigna una prioridad de tipo numérico al mensaje que se envía.
- data: conjunto de datos de tipo JSON con el contenido del mensaje.

El campo *messageId* es el identificador del mensaje y es utilizado para evitar mensajes duplicados. El campo *originDroneId* se utiliza para identificar al emisor del mensaje con el objetivo de que los receptores puedan actualizar de forma local la información relacionada al drone emisor. En caso de que el receptor y el emisor sean el mismo, el mensaje sera descartado por el drone, esto sucede debido a que la comunicación se realiza

por el broadcast. El campo *destinationDroneId* se utiliza para identificar al receptor del mensaje. Cuando un drone recibe un mensaje, si el valor del campo *destinationDroneId* es el *id* de él o el valor *broadcast* (lo que quiere decir que el mensaje está dirigido a todos los drones de la flota), entonces el mensaje se agrega a la cola de mensajes para ser procesado, de no ser así el mensaje se descarta. El campo *priority* asigna una prioridad al mensaje enviado por el emisor. Este campo es usado por el receptor para ordenar de forma descendente todos los mensajes que no han sido procesados, de tal forma que se procesen primero los mensajes de mayor prioridad.

Se decidió usar una estructura homogénea para todos los tipos de mensajes con el fin de facilitar el procesamiento de los mismos. Los tipos de mensajes enviados son:

- UPDATE_POSITION: es enviado para notificar que el drone emisor actualizó su posición actual. El mensaje se envía con el valor *broadcast* en el campo *destinationDroneId*. El objetivo principal de éste tipo de mensajes es que cada drone conozca en todo momento donde se encuentra el resto de la flota.
- UPDATE_ROUTE: es enviado para notificar que el drone emisor actualizó la ruta actual que está siguiendo. El mensaje es enviado por cada drone luego de crear una nueva ruta a seguir. De forma similar al tipo de mensaje UPDATE_POSITION, el objetivo del mensaje es que cada drone conozca la ruta que realizarán los restantes drones en el corto plazo, de modo de planificar su ruta acorde a los futuros movimientos de toda la flota.
- ACK_MESSAGE: se utiliza para confirmar la recepción de un mensaje. Cada vez que se recibe un mensaje, se envía un ACK para que el drone emisor sepa si el mensaje fue recibido por el receptor.
- GO_HOME: es enviado por un agente externo y es utilizado cuando se desea finalizar con el sistema de vigilancia. Cada drone, al recibir este mensaje, vuelve a su punto de partida y aterriza en él.
- LOGOUT_SYSTEM: se envía para notificar que un drone se va del sistema y no estará vigilando. Cuando un drone recibe este mensaje

elimina de su información local la posición y ruta actual del emisor del mensaje. El motivo para eliminar la información es no tenerla en cuenta al momento de calcular una nueva ruta.

- UPDATE_ROUTE_INTERSECTION: es enviado para notificar que se provocó una intersección entre dos o más rutas.
- CHANGE_STATE_NOTIFICATION: se utiliza para notificar que el drone cambió de estado, por ejemplo cuando el drone va a cargar o cuando detecta un intruso.
- START_FOLLOWING: es enviado para notificar que el drone detectó un intruso y lo comenzó a seguir.
- REQUEST_SUPPORT: lo envía un drone que detectó un intruso y solicita soporte para la vigilancia.
- UPDATE_DETECTION_OBJECT: es enviado por aquellos drones que están siguiendo a un intruso. La información que se envía es la posición estimada del intruso.

## 4.3. Navegación

En esta sección se describen los componentes y procedimientos necesarios para la navegación. En la subsección 4.3.1 se describe el escenario. En la subsección 4.3.2 se desarollan los algoritmos de navegación. La subsección 4.3.3 explica la coordinación y comunicación entre varios drones. En la subsección 4.3.4 se describe la implementación y los cálculos a realizar para el control de vuelo. La subsección 4.3.5 explica el funcionamiento del manejo de error en la navegación. Finalmente, en la subsección 4.3.6 se describe la optimización del uso de la batería y su simulación.

### 4.3.1. Escenario

Al predio a vigilar se lo divide en celdas cuadradas de igual medida las cuales forman una grilla. La grilla está representada en el sistema como una matriz de tuplas compuestas por las coordenadas GPS correspondientes al punto medio de cada celda y un valor que indica la prioridad. La prioridad de una celda es la importancia de vigilancia que tiene la misma, es decir, celdas con mayor prioridad deberán ser visitadas mas frecuentemente que aquellas celdas con menor prioridad.

El algoritmo de vigilancia construido es exportable a cualquier predio siempre y cuando tenga las siguientes características: sea un predio limitado y conocido, se conozcan sus coordenadas GPS, no contenga obstáculos dentro con los que el drone pueda colisionar y pueda ser representado como una matriz $M$ de $m * n$ celdas.

### 4.3.2. Algoritmos de navegación

Una flota de drones autónoma debe ser capaz de operar por si sola sin la asistencia de terceros. Para la navegación esto significa que la flota de drones debe ser capaz de construir y elegir por si sola los caminos a seguir.

Inicialmente se construyó un algoritmo básico en el cual los drones realizan una recorrida en barrido para la exploración, el cual se denomina algoritmo scan. Luego se construyó un algoritmo denominado algoritmo trayectoria, el cual elige la mejor secuencia de celdas y navega por ella, una vez que termina el ultimo paso de la secuencia de celdas, se elige una nueva secuencia de celdas. Por último, al algoritmo de trayectoria se le realizó una modificación para que no tome en cuenta ciertas celdas al momento de elegir la trayectoria, este algoritmo se denomina como algoritmo de trayectoria con restricción.

Se denomina trayectoria al conjunto de celdas que el drone recorre desde que parte desde uno de los límites del terreno hasta que llega hasta el límite opuesto. En la figura 4.5 se puede ver una trayectoria tomada por un drone de la flota, comienza en la fila 0 (uno de los limites del terreno) y tiene como final la fila 7 (límite opuesto del terreno). Ésta trayectoria está compuesta por las celdas: (0,0), (1,0), (2,0), (2,1), (2,2), (3,2), (4,2), (5,2), (5,3), (5,4), (6,4), (7,4), donde el primer valor de la dupla hace referencia a la fila, y el segundo valor a la columna.

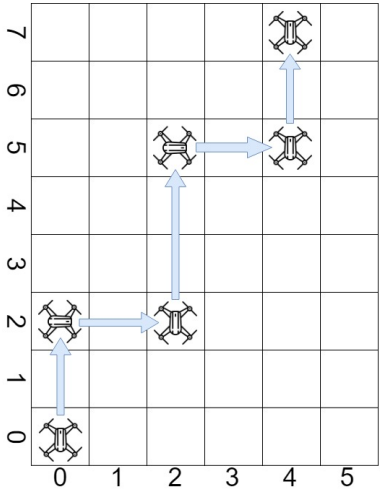


Figura 4.5: Planificación de trayectoria.

Se define beneficio de vigilancia de una celda como la importancia de esa celda, sea entonces $bvc_{ij}$ el beneficio de vigilancia de la celda $ij$ de la matriz $M$ y sea $P_{ij}$ el valor de prioridad de la celda $ij$, entonces:

$$bvc_{ij} = P_{ij} \; donde \; ij \in M$$

Se define beneficio de vigilancia de una trayectoria a la sumatoria de los beneficios de vigilancia de cada una de las celdas que forma la trayectoria, por lo tanto: sea $bvt$ el beneficio de vigilancia de una trayectoria, sea $bvc_{ij}$ el beneficio de vigilancia de la celda $ij$ de la matriz $M$ y sea $T$ el conjunto de celdas que forma la trayectoria, entonces:

$$bvt = \sum bvc_{ij} \; donde \; ij \in T$$

**Algoritmo scan**

Este algoritmo implica partir desde un extremo del terreno, navegar por una columna de la matriz hasta llegar al extremo contrario (linea 2 y 3 del Algoritmo 1) y volver por la columna contigua como se muestra en la figura 4.6. Ésta sencilla solución no cumple con uno de los objetivos principales de la vigilancia, que las celdas con mayor prioridad deben ser visitadas más frecuentemente que aquellas con menor prioridad.

**Algoritmo 1** Algoritmo scan

```
1: function obtenerCaminoScan()
2:     columna = obtenerColumnaARecorrer()
3:     camino = agregarPasosHastaExtremoOpuesto(columna)
4:     return camino
5:
6: procedure main()
7:     while sistemActivo do
8:         camino = obtenerCaminoScan()
9:         avanzar(camino)
10:    end
```

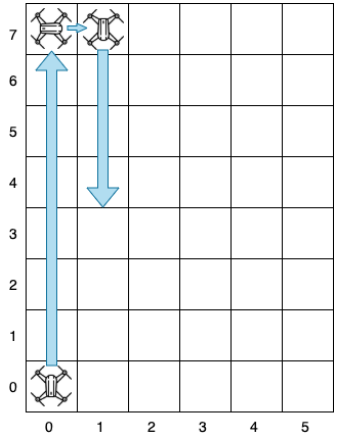


Figura 4.6: Algoritmo scan.

### Algoritmo de trayectoria

Se busca que el drone calcule la trayectoria con mayor beneficio de vigilancia cuyo comienzo es la fila 0 y su final la fila $m$, siendo $m$ la última fila de la matriz. Una vez que el drone llega a destino se realiza el mismo cálculo pero en sentido contrario, comenzando en la fila $m$ y con final en la fila 0 de la matriz.

Un problema que se detectó inmediatamente al realizar algunas pruebas con este algoritmo, es que algunas celdas con menor prioridad no son visitadas nunca si tienen celdas contiguas con mayor prioridad porque estas últimas siempre son elegidas por el cálculo de la trayectoria. Para resolver este problema, se agregó un valor a cada celda que indica el tiempo en el que fue visitada la última vez. En consecuencia, sea $t$ el tiempo desde que la celda fue visitada por última vez, se define el beneficio de vigilancia de una celda con envejecimiento como:

$$bvce_{ij} = (P_{ij} * t) \; donde \; ij \in M$$

De este modo, a la prioridad inicial de cada celda dada por la importancia del lugar, se agrega una prioridad dinámica conformada por el tiempo que ha pasado desde que fue visitada por última vez (último acceso). Para este algoritmo se utiliza el beneficio de trayectoria con envejecimiento que se define a continuación.

$$bvte = \sum bvce_{ij} \; donde \; ij \in T$$

Se denomina trayectoria continua a una lista de celdas donde cada celda es contigua a la anterior y a la siguiente, es decir, sea la celda $c_{ij}$ en el lugar $n$ de la lista, entonces las celdas $n-1$ y $n+1$ son de la forma $c_{hk}$ donde $h \in [i-1, i+1]$ con $k = j$ ó $k \in [j-1, j+1]$ con $h = i$. Para encontrar la trayectoria con mayor beneficio de vigilancia se realiza el siguiente procedimiento: para cada fila de la matriz se busca la celda cuyo valor de beneficio de vigilancia sea el máximo (linea 2 a 5 del Algoritmo 2). Con los máximos de cada fila se crea la trayectoria ordenando las celdas por el índice de fila a la que pertenece, de forma creciente o decreciente según el sentido de la trayectoria (linea 6 del Algoritmo 2). Notar que con este procedimiento se puede obtener una trayectoria donde un par de celdas adyacentes en la lista no sean contiguas entre sí en la matriz, provocando que la trayectoria no sea continua. Para que la trayectoria sea continua se busca entre cada par de celdas adyacentes de la trayectoria no contiguas

en la matriz, el camino de mayor beneficio de vigilancia entre ambas, y se agregan las celdas de este camino intermedio a la trayectoria, logrando así una trayectoria continua (linea 7 del Algoritmo 2).

**Algoritmo 2** Algoritmo trayectoria

```
1:  function obtenerCaminoTrayectoria()
2:      maximos = []
3:      for todas las filas do
4:          max = obtenerMaximoDeLasColumnas()
5:          maximos.add(max)
6:      ordenarSegunPosicionActual(maximos)
7:      camino = unirPuntosConMayorBeneficio(maximos)
8:      return camino
9:
10: procedure main()
11:     while sistemActivo do
12:         camino = obtenerCaminoTrayectoria()
13:         avanzar(camino)
14:     end
```

A modo de ejemplo, se muestra un escenario posible. En la figura 4.7 a la izquierda se muestra la matriz con una dupla en cada celda que contiene la prioridad inicial y el tiempo que pasó desde la última vez que un drone de la flota pasó por ella. A la derecha se muestra la matriz con los valores de beneficio de vigilancia de cada celda.

En la figura 4.8 a la izquierda se muestran resaltados los valores máximos de cada fila. Se puede ver que existen celdas con beneficio de vigilancia máximo no contiguas en filas adyacentes, por lo tanto se toma el camino entre ellas con mayor beneficio de vigilancia y se agrega a la trayectoria como se puede ver en la matriz de la derecha. Notar que el drone no se mueve de una celda a otra directamente en diagonal si no que debe dirigirse primero a una de las celdas contiguas y luego a la celda en diagonal.

| | | | | | |
|---|---|---|---|---|---|
| (1, 13s) | (2, 10s) | (1, 21s) | (3, 8s) | (1, 1s) | (1, 6s) |
| (1, 5s) | (2, 24s) | (5, 9s) | (1, 0s) | (3, 10s) | (1, 120s) |
| (3, 0s) | (2, 106s) | (1, 7s) | (1, 52s) | (1, 18s) | (5, 2s) |
| (3, 0s) | (1, 7s) | (4, 0s) | (1, 10s) | (1, 6s) | (5, 0s) |
| (3, 2s) | (2, 4s) | (1, 14s) | (1, 40s) | (2, 5s) | (1, 0s) |
| (1, 30s) | (3, 27s) | (2, 4s) | (1, 10s) | (1, 26s) | (2, 8s) |
| (1, 20s) | (1, 25s) | (1, 22s) | (3, 120s) | (1, 1s) | (3, 14s) |
| (1, 10s) | (3, 0s) | (5, 14s) | (5, 0s) | (1, 13s) | (1, 0s) |

| | | | | | |
|---|---|---|---|---|---|
| 13 | 20 | 21 | 24 | 1 | 6 |
| 5 | 48 | 45 | 0 | 30 | 120 |
| 0 | 212 | 7 | 52 | 18 | 10 |
| 0 | 7 | 0 | 10 | 6 | 0 |
| 6 | 8 | 14 | 40 | 10 | 0 |
| 30 | 81 | 8 | 10 | 26 | 16 |
| 20 | 25 | 22 | 360 | 1 | 42 |
| 10 | 0 | 70 | 0 | 13 | 0 |

Figura 4.7: Matriz con prioridades y matriz con beneficios de vigilancia.

| | | | | | |
|---|---|---|---|---|---|
| 13 | 20 | 21 | **24** | 1 | 6 |
| 5 | 48 | 45 | 0 | 30 | **120** |
| 0 | **212** | 7 | 52 | 18 | 10 |
| 0 | 7 | 0 | **10** | 6 | 0 |
| 6 | 8 | 14 | **40** | 10 | 0 |
| 30 | **81** | 8 | 10 | 26 | 16 |
| 20 | 25 | 22 | **360** | 1 | 42 |
| 10 | 0 | **70** | 0 | 13 | 0 |

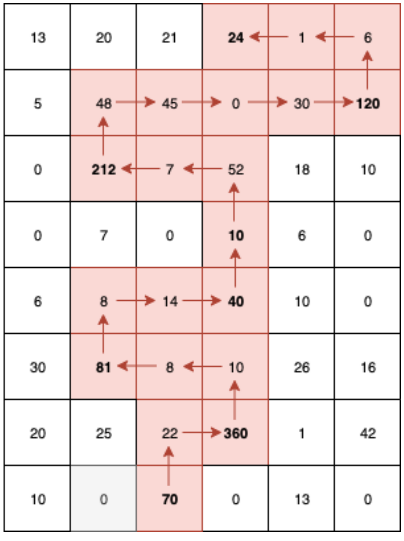


Figura 4.8: Matriz con beneficios de vigilancia máximos por fila y matriz con trayectoria final.

**Algoritmo de trayectoria con restricciones**

Al tener dos o mas drones realizando la vigilancia lo ideal es que se repartan el terreno, esto no sucede con el algoritmo de trayectoria ya que los drones pueden tomar trayectorias en el mismo lado del predio dejando el resto sin vigilar hasta que no existan celdas con la prioridad suficiente para ser consideradas. Para resolver este problema se diseñó la siguiente solución: las celdas que un drone puede agregar a su trayectoria deben estar a un máximo de $K$ columnas de distancia de la columna donde se encuentra el drone actualmente. Para implementarlo basta con multiplicar por cero todos los beneficios de vigilancia de las celdas de las columnas que tienen distancia mayor a $K$ de la columna donde se encuentra el drone actualmente, de este modo esas celdas no serán tenidas en cuenta para la trayectoria del drone por tener el mínimo valor posible de beneficio de vigilancia (linea 2 a 5 del Algoritmo 3). Se define el beneficio de vigilancia de una celda con envejecimiento y restricción de la siguiente manera:

$$bvcer_{ij} = (P_{ij} * t * q) \ donde \ ij \in M \ y \ q = \begin{cases} 1 \ si \ D \leq K \\ 0 \ si \ D > K \end{cases}$$

Siendo $D$ la distancia entre la columna a considerar y la columna actual donde se encuentra el drone, $q$ equivalente a 1 si la celda en consideración pertenece a una columna a una distancia $D$ menor a $K$ y equivalente a 0 en otro caso. Notar que este nuevo cálculo del beneficio de vigilancia de una celda produce valores diferentes dependiendo del lugar donde se encuentra el drone, por lo que distintos drones en distintas posiciones pueden tener distinto beneficio de vigilancia para la misma celda en el mismo momento. Con esta nueva implementación se obtiene un comportamiento como se muestra en la figura 4.9.

**Algoritmo 3** Algoritmo trayectoria con restricciones

```
1:  function obtenerCaminoTrayectoriaConRestricciones()
2:      maximos = []
3:      for todas las filas do
4:          max = obtenerMaximoDeLasColumnasCercanas()
5:          maximos.add(max)
6:      ordenarSegunPosicionActual(maximos)
7:      camino = unirPuntosConMayorBeneficio(maximos)
8:      return camino
9:
10: procedure main()
11:     while sistemActivo do
12:         camino = obtenerCaminoTrayectoriaConRestricciones()
13:         avanzar(camino)
14:     end
```

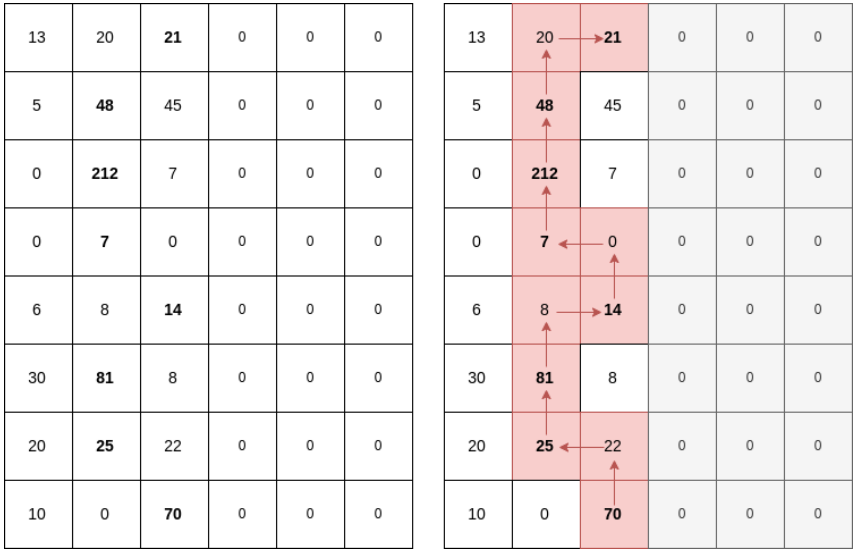


Figura 4.9: Algoritmo de la trayectoria con restricción

### 4.3.3. Coordinación entre dos o más drones

Para que la matriz de prioridades tenga los mismos valores para todos los drones de la flota, es necesario que exista la comunicación y coordinación entre estos. Luego de realizar un movimiento desde una celda a otra, se debe enviar un mensaje a los restantes drones de la flota indicando que se visitó esta última celda y que el último acceso de la misma debe ser actualizado. Cuando un drone recibe un mensaje de éste tipo, actualiza la celda de la matriz local indicada en el mensaje con el tiempo de último acceso. El objetivo de esta funcionalidad es que todos los drones de la flota estén coordinados en cuanto al estado del sistema.

Un problema encontrado que si bien no interrumpe el correcto funcionamiento el sistema, lo hace menos eficiente, sucede cuando existe algún punto en común entre dos o más trayectorias de distintos drones. Esto afecta la eficiencia de la vigilancia ya que al menos dos drones irían a vigilar el mismo punto. Por este motivo, se decidió optimizar el algoritmo de exploración para evitar que esto ocurra.

El procedimiento que se realiza se describe a continuación: en primer lugar, una vez que un drone calcula su próxima trayectoria comprueba si algunos de los pasos de su trayectoria se encuentra en la trayectoria de otro drone de la flota (linea 6 a la 12 del Algoritmo 4), si no se encuentra sigue su trayectoria normal, de no ser así procede a realizar el cambio de las trayectorias.

Dado que las trayectorias calculadas poseen el mayor beneficio de vigilancia, se opta por hacer un intercambio entre las trayectorias de ambos drones sin perder ninguno de los pasos, el responsable de realizar el intercambio es el drone que detecta la intersección. Entonces, la nueva trayectoria del drone que detecta la intersección será desde el origen de su trayectoria hasta el punto de intersección inclusive, y luego desde el punto de intersección hasta el destino de la trayectoria del otro drone. La trayectoria del otro drone es el resultado de realizar el proceso inverso, es decir, desde el origen de su trayectoria hasta el punto intersección (sin incluir

este punto), y luego desde éste punto hasta el destino de la trayectoria del primer drone (linea 15 y 16 del Algoritmo 4). Una vez realizado el intercambio de la trayectoria, se envía un mensaje al drone con el que se provocó la intersección con la nueva ruta a seguir (linea 17 del Algoritmo 4).

**Algoritmo 4** Algoritmo intercambio de rutas

```
 1: function verificarCamino(camino)
 2:     intersecciona = false
 3:     pasoInterseccion = null)
 4:     droneInterseccion = null
 5:
 6:     for cada drone en el sistema do
 7:         ruta = obtenerRutaDrone(drone)
 8:         for cada paso de mi camino do
 9:             if (perteneecePasoEnCamino(paso, ruta)) then
10:                 intersecciona = true
11:                 pasoInterseccion = paso
12:                 droneInterseccion = drone
13:
14:     if (intersecciona) then
15:         miNuevaRuta = miRutaNueva(pasoInterseccion)
16:         nuevaRutaOtroDrone = nuevaRutaOtroDrone(pasoInterseccion)
17:         droneInterseccion.notificarRutaNueva(nuevaRutaOtroDrone)
18:         return miNuevaRuta
19:     else
20:         return camino
21:
22: procedure main()
23:     while sistemActivo do
24:         camino = obtenerCamino()
25:         caminoValidado = verificarCamino(camino)
26:         avanzar(caminoValidado)
27:     end
```

A modo de ejemplo, sea D1 el drone que detecta la intersección, D2 el drone con el que se produce la intersección, [(2,2), (3,2), (3,3), (4,3), (5,3), (6,3)] la trayectoria de D1 y [(6,2), (5,2), (4,2), (4,3), (4,4), (3,4), (2,4)] la trayectoria de D2, el punto en el que se produce la intersección es (4,3) (ver figura 4.10), por lo tanto D1 intercambia las trayectorias de tal forma que (4,3) sea sólo visitado por él, entonces su nueva trayectoria será: [(2,2), (3,2), (3,3), (4,3), (4,4), (3,4), (2,4)], y para el drone D2: [(6,2), (5,2), (4,2), (5,3), (6,3)].

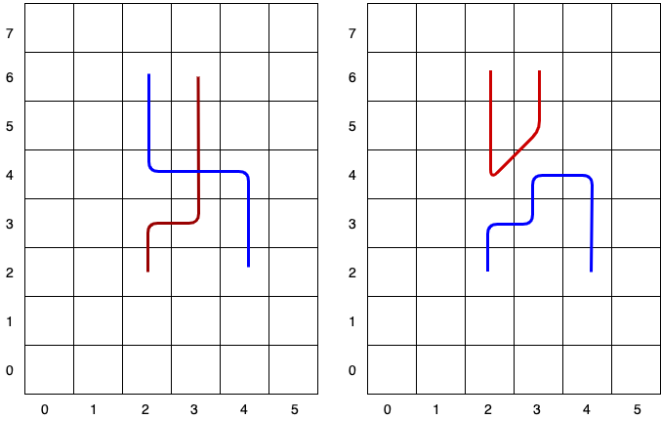


Figura 4.10: Intersección de caminos.

### 4.3.4. Control de vuelo

Una trayectoria está formada por una lista de pasos que contienen la prioridad y la posición GPS (latitud y longitud) de una de las celdas que forman la matriz que representa el terreno. De cada paso se toman las coordenadas GPS de la celda y se le indica al drone que se desplace desde la posición actual hacia esa posición denominada posición destino. Una vez que el drone se encuentra en la posición destino, se toma el siguiente paso de la trayectoria y se avanza hacia la posición GPS que se indica en él. Éste proceso de moverse desde un punto a otro por sencillo que parezca conlleva cierta complejidad que se detalla a continuación.

Para realizar un movimiento básico, es decir, mover el drone desde la posición actual a la posición destino (desde una celda del predio a otra), se decidió utilizar la función *move_relative* de la biblioteca PyParrot. Esta función consume dos parámetros $x$ e $y$ que representan los valores en el marco de referencia del drone que éste debe moverse para desplazarse desde la posición actual a la posición destino. Se define marco de referencia del drone como el marco de referencia cuyo origen es el drone, formado por el $Eje\ x$, el eje donde el drone se mueve hacia adelante y hacia atrás, y el $Eje\ y$, el eje donde el drone se mueve hacia la derecha o hacia la izquierda (ver figura 4.11). Entonces, los parámetros $x$ e $y$ representan los valores en metros que el drone debe moverse sobre el $Eje\ x$ y sobre el $Eje\ y$ respectivamente. Como es esperado, el drone no se mueve primero sobre su $Eje\ x$ y luego sobre su $Eje\ y$ para llegar a la posición destino si no que calcula la componente de $x$ e $y$ y toma el camino que indica la misma. Para poder usar esta función de movimiento, el problema reside en obtener los valores de $x$ e $y$ necesarios para que el drone se mueva de la posición actual a la posición destino a partir de los valores conocidos de latitud y longitud de la posición actual y los valores conocidos de latitud y longitud de la posición destino. Para los siguientes cálculos se denomina $distanciaX$ a $x$ y $distanciaY$ a $y$.

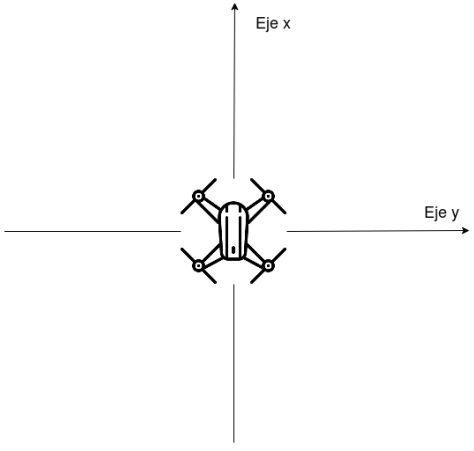


Figura 4.11: Ejes relativos del drone.

Con las coordenadas GPS (latitud y longitud) de la posición destino se forma una tupla que se denomina *posicionDestino*. Luego se obtiene del sensor GPS la latitud y la longitud actual del drone formando un tupla denominada *posicionActual*. Posteriormente se calcula la distancia entre las tuplas *posicionActual* y *posicionDestino* que se denomina distancia real. Notar que estas posiciones están en medidas de distancia angular (por estar compuestas por valores de latitud y longitud).

Se tienen dos ejes que llamaremos *ejeLatitud* y *ejeLongitud*, estos ejes son paralelos a la latitud y longitud de la Tierra respectivamente y representan el marco de referencia de la Tierra. Se descompone la distancia real en dos componentes: *distanciaLatitud* y *distanciaLongitud*, siendo estas componentes paralelas a los ejes *ejeLatitud* y *ejeLongitud* respectivamente como indica la figura 4.12.

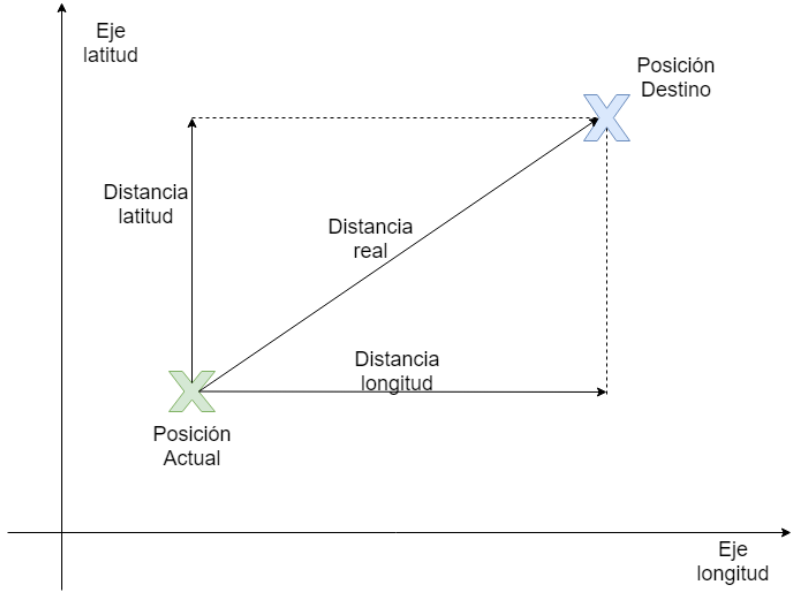


Figura 4.12: Distancia entre la posición actual y la posición destino y sus componentes distanciaLatitud y distanciaLongitud paralelas a los ejes latitud y longitud respectivamente.

Los valores de las componentes *distanciaLatitud* y *distanciaLongitud* están en medidas de distancia angular (grados sexagesimales), mientras que los valores de *distanciaX* y *distanciaY* se encuentran en metros. Para obtener los valores buscados de *distanciaX* y *distanciaY* en el marco de referencia del drone a partir de los valores de *distanciaLatitud* y *distanciaLongitud* en el marco de referencia de la tierra (ver figura 4.13), primero se debe convertir *distanciaLatitud* y *distanciaLongitud* de medidas de distancia angular a metros. Se define *distanciaLatMetros* y *distanciaLongMetros* como las componentes medidas en metros de la distancia real en los ejes latitud y longitud.

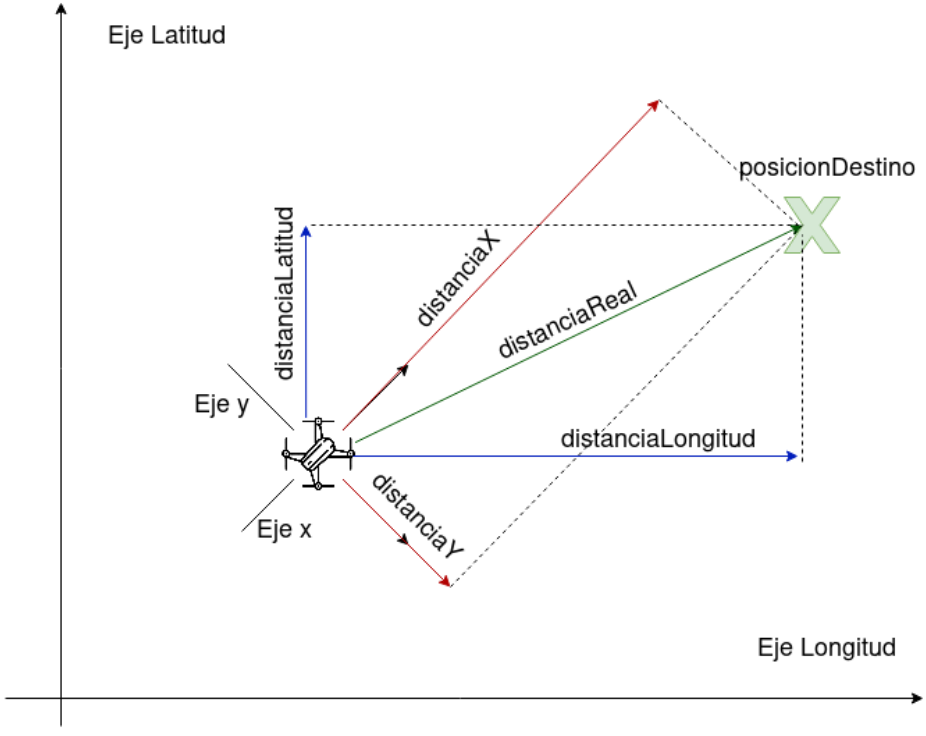


Figura 4.13: Marco de referencia del drone y marco de referencia de la tierra.

Se define $anguloGPS$ al ángulo entre la distancia real y la componente $distanciaLatMetros$ (que se encuentra sobre el eje latitud). Para hallar el $anguloGPS$ se realiza el siguiente cálculo:

$$anguloGPS = \arctan\left(\frac{distanciaLongMetros}{distanciaLatMetros}\right)$$

Se estudian por separado los casos donde la $distanciaLatMetros$ es cero ya que en estos casos no se puede realizar el cálculo anterior. Si ambos puntos (posición actual y posición destino) están sobre la misma latitud, es decir, $distanciaLatMetros$ es cero, la recta entre un punto y otro es paralela a la componente longitud, por lo que se forma un ángulo de 90 (o -90 según la dirección) con la componente latitud. Se tienen entonces los siguientes casos:

- si $distanciaLongMetros$ es positiva, entonces $anguloGPS$ es 90.
- si $distanciaLongMetros$ es negativa, entonces $anguloGPS$ es -90.

Se define $anguloOrientacionActual$ como el ángulo dado por el sensor de orientación del drone que indica cuantos grados desviado desde el norte se encuentra apuntando el drone en ese momento.

Se define $anguloDeGiro$ como el ángulo que el drone debe girar para apuntar en la dirección correcta (i.e. la que apunta a la posición destino) el cuál está dado por la diferencia entre $anguloGPS$ y $anguloOrientacionActual$ (ver figura 4.14).

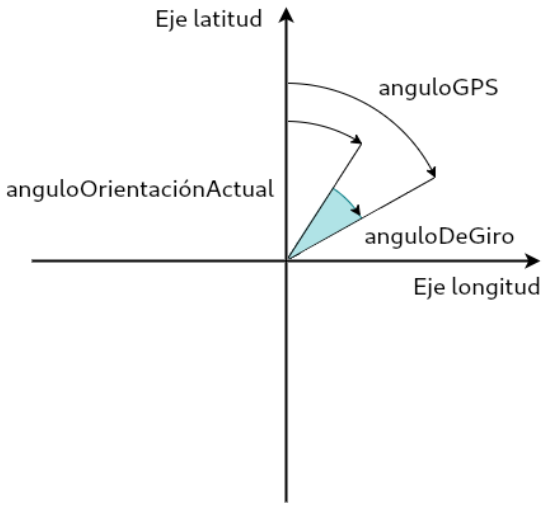


Figura 4.14: Ángulo de giro.

Antes de realizar la resta se debe convertir el rango del $anguloGPS$, ya que por ser el resultado de una función arco tangente su rango es [-180,180], por otro lado el $anguloOrientacionActual$ es proporcionado por el drone en un rango de [0,360]. Para la conversión del rango de $anguloGPS$ se estudian los siguientes casos que se ven en la figura 4.15

- Si $distanciaLatMetros$ y $distanciaLongMetros$ son positivas, entonces el punto destino esta en el primer cuadrante, por lo tanto: $anguloGPS = anguloGPS$

- Si $distanciaLatMetros$ es positiva y $distanciaLongMetros$ negativa, entonces el punto destino esta en el segundo cuadrante, por lo tanto: $anguloGPS = 360 - anguloGPS$

- Si $distanciaLatMetros$ y $distanciaLongMetros$ son negativas, entonces el punto destino esta en el tercer cuadrante, por lo tanto: $anguloGPS = 180 + anguloGPS$

- Si $distanciaLatMetros$ es negativa y $distanciaLongMetros$ es positiva, entonces el punto destino esta en el cuarto cuadrante, por lo tanto: $anguloGPS = 180 - anguloGPS$

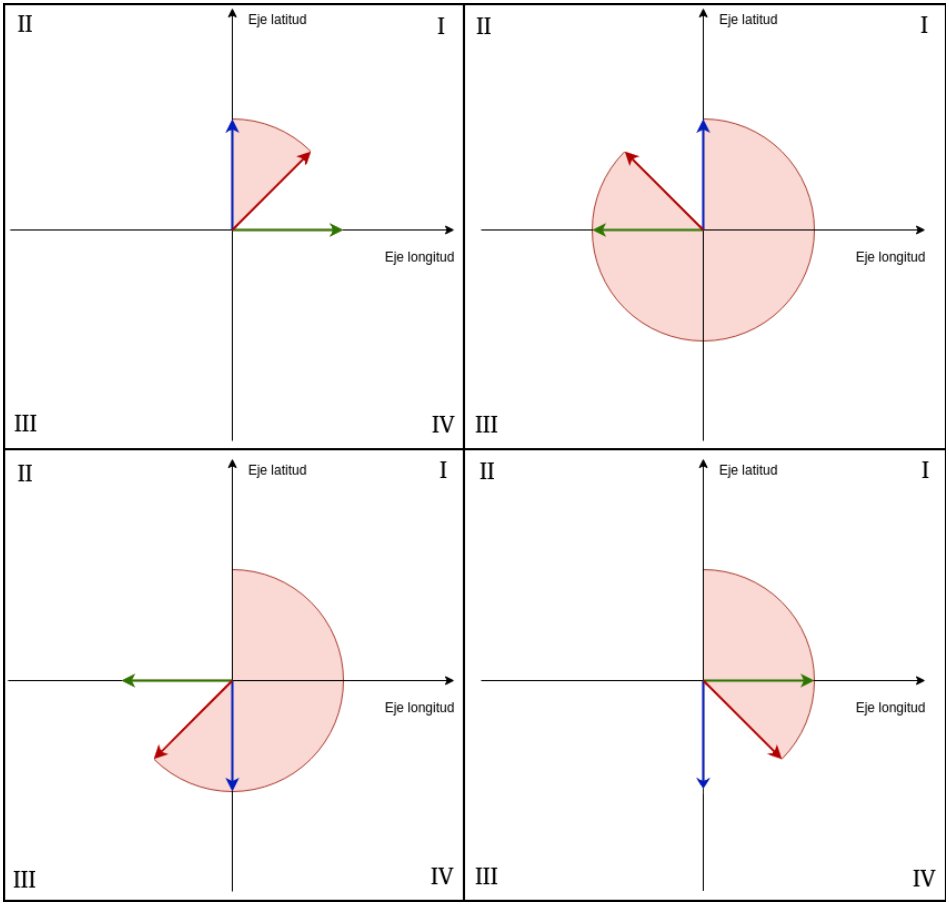


Figura 4.15: Ángulo GPS según la dirección de la distancia real entre posición actual y destino (rojo) determinada por sus componentes, distancia latitud (azul) y distancia longitud (verde).

Una vez convertido $anguloGPS$, ambos ángulos comparten rango y se procede a realizar la diferencia para obtener el ángulo que el drone debe girar.

$$anguloDeGiro = anguloGPS - anguloOrientacionActual$$

Se define $distanciaRealMetros$ como el valor en metros que separa a la $posicionActual$ de la $posicionDestino$ (la distancia real). Sabiendo que $distanciaLatMetros$ es la componente en metros de la distancia real sobre el eje latitud y que $distanciaLongMetros$ es la componente en metros de la distancia real sobre el eje longitud se tiene que:

$$distanciaRealMetros = \sqrt{(distanciaLongMetros)^2 + (distanciaLatMetros)^2}$$

Se sabe que $distanciaRealMetros$ puede ser también descompuesta en $distanciaX$ y $distanciaY$ que son las componentes sobre los ejes del marco de referencia del drone. Además se sabe que el $anguloDeGiro$ es el ángulo comprendido entre $distanciaRealMetros$ y $distanciaX$ ya que es el ángulo que el drone debe girar para apuntar desde donde apunta actualmente ($distanciaX$ sobre el $ejeX$ del marco de referencia del drone) hacia la posición destino (indicada por la dirección de $distanciaRealMetros$). Entonces por trigonometría se tiene que:

$$\begin{aligned} distanciaX &= distanciaRealMetros * \cos(anguloDeGiro) \\ distanciaY &= distanciaRealMetros * \sin(anguloDeGiro) \end{aligned} \tag{4.1}$$

Finalmente se obtienen los valores de las componentes $distanciaX$ y $distanciaY$ en el marco de referencia del drone que se necesitan para pasar como parámetro a la función *move_relative* y desplazarse de la posición actual a la posición destino.

Por otro lado, se debe proporcionar a la función *move_relative* la orientación en la que el drone debe navegar, de lo contrario el drone se mueve apuntando para la misma orientación que está apuntando en el momento de iniciar el movimiento. Esto puede llevar a que el drone se mueva de costado o hacia atrás, apuntando su cámara a orientaciones distintas a las que se dirige el movimiento, lo cual no es para nada deseable para el proyecto de vigilancia, es deseable que la cámara del drone apunte siempre en la dirección en la que el drone se mueve. Esta orientación es el *anguloDeGiro* calculado.

### 4.3.5. Manejo de error

A modo de corrección de posibles desviaciones en el eje vertical debido a factores externos como el viento, antes de comenzar el movimiento, se corrige (en caso de ser necesario) la altitud del drone. Para ello cada drone cuenta con una altura predefinida a la que debe volar y un margen de tolerancia. Mediante el sensor de altitud se obtiene la altura actual del drone y si la diferencia entre ésta y la altura predefinida es superior a $T$, siendo $T$ la tolerancia predefinida de desviación de altura, el drone aumenta o disminuye su altitud según corresponda hasta que la diferencia sea menor a $T$.

Por el lado del eje horizontal, dado que cada movimiento a un punto GPS de la ruta (un paso) es independiente de los otros, se tiene la ventaja de que los errores no se acumulan. Esto se debe a que para ir a un punto del camino solo se necesita la posición actual y la posición destino, si bien la posición actual puede ser incorrecta, es decir fuera de la ruta, no se arrastra error ya que el sistema encuentra correctamente las componentes $x$ e $y$ necesarias para llegar a la posición destino independientemente de si la posición actual es correcta o no.

De todos modos esta corrección de errores casi automática no es suficiente. En el caso de que se produzca un error en el cálculo del punto destino, el drone esperará hasta llegar al punto destino incorrecto para

tomar otro punto de la ruta y retomar la trayectoria correcta, esto conlleva a una perdida de tiempo mientras se navega hacia un punto que no está en la trayectoria calculada. Para solucionar este problema se decidió modificar la función de PyParrot de movimiento relativo.

La función *move_relative* de PyParrot utiliza una función primitiva de movimiento del SDK del drone. Los fabricantes del drone a través del SDK ofrecen comandos para mover el drone de forma relativa a su posición actual, PyParrot utiliza estos comandos primitivos y para el caso del movimiento relativo invoca una función de movimiento llamada *moveBy*. Luego de llamar a éste comando primitivo la función de PyParrot se queda bloqueada esperando hasta que un sensor de movimiento le notifique que el drone ya no se mueve porque llegó a destino. Se modificó esta parte y se agregó la siguiente lógica para efectuar validaciones de que el camino tomado es correcto: en lugar de esperar bloqueado a que termine el movimiento, cada medio segundo se obtienen las coordenadas GPS del punto actual, se agrupan en parejas y se realiza la desigualdad triangular con el punto destino de la siguiente manera. (Ver Figura 4.16)

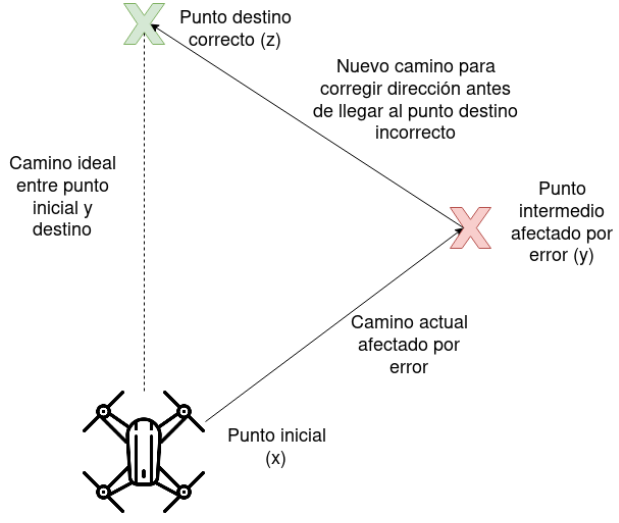


Figura 4.16: Desigualdad triangular.

Sea $x$ el punto GPS obtenido en el instante $tx$, y sea $y$ el punto GPS obtenido en el instante $ty$ (siendo $ty \geq tx$) y sea $z$ el punto GPS que representa la posición destino, se tiene que $xyz$ es el triángulo formado por estos 3 puntos. Entonces por desigualdad triangular se cumple:

$$d(x, z) \leq d(x, y) + d(y, z)$$

Siendo que el camino más corto entre $x$ y $z$ es la recta entre ambos, entonces si el punto $y$ pertenece a la recta no hubo desviación en la trayectoria, por lo que:

$$d(x, z) = d(x, y) + d(y, z)$$

Al desviarse el punto $y$ se forma un triángulo $xyz$ con:

$$d(x, z) \leq d(x, y) + d(y, z)$$

Por lo que para detectar si hubo error se compara la siguiente desigualdad:

$$margenError + d(x, z) \geq d(x, y) + d(y, z)$$

Donde *margenError* es un margen de tolerancia establecido previamente. Al detectar una desviación se interrumpe el vuelo del drone y se fuerza un nuevo cálculo de los metros que el drone debe moverse en los ejes relativos para llegar directamente desde la nueva posición actual a la posición destino correcta.

### 4.3.6. Optimización del uso de la batería

Para lograr que el uso de la batería sea eficiente, se consideró necesario contar con distintos estados para la toma decisiones:

- CRITICAL: cuando la batería es la mínima necesaria para regresar al punto de carga.
- LOW: cuando la batería se encuentra entre 10 y 20 por ciento.
- NORMAL: cuando la batería se encuentra entre 20 y 50 por ciento.
- HIGH: cuando la batería es superior al 50 por ciento.

Estos estados se usan para la toma de decisiones de la máquina de estado del sistema de vigilancia. Ante un estado de batería HIGH, un drone está habilitado a realizar todas las funcionalidades del sistema como desplazarse, patrullar el terreno, seguir a un intruso o asistir a otro drone en el seguimiento de un intruso. Cuando el nivel de batería hace que el estado cambie a NORMAL, el sistema limita las funcionalidades del drone. Esta limitación impide que el drone puedan asistir a otro en el seguimiento de un intruso. Es importante aclarar que en caso de que la batería baje a nivel del estado NORMAL y el drone se encuentre asistiendo en el seguimiento a otro drone, continuará asistiendo en el seguimiento de intruso.

Nuevamente, cuando el nivel de batería baje y cambie a estado LOW se volverán a limitar las funcionalidades del drone. Si el drone se encuentra patrullando el terreno vuelve a base para recargar la batería, en cambio si se encuentra siguiendo un intruso ó asistiendo en el seguimiento de un intruso, continuará asistiendo hasta que sus niveles lleguen a CRITICAL. Ante un estado de batería CRITICAL, el sistema envía al drone a cargar en la base sin importar que esté haciendo (siguiendo, asistiendo en el seguimiento o patrullando), de lo contrario, el drone se apagaría y aterrizaría en el lugar del predio que se encuentre en ese momento.

Previamente a cada acción a realizar, por ejemplo, moverse de una celda del predio a otra, se calcula si la batería que posee es superior a $X + Y + Z$, donde $X$ es la batería que requiere para realizar la acción, $Y$ es la batería que requiere para trasladarse desde el punto actual al punto de carga, y $Z$ la batería que requiere para realizar el aterrizaje.

Luego, el estado de la batería en conjunto a la información del sistema (por ejemplo, estado de vigilancia ó posición actual) son utilizados para determinar los comportamientos que los UAVs deban de seguir.

### Simulación de batería

En una primera instancia, se consideró usar la simulación de batería que brinda Sphinx, pero por los motivos que se detallan a continuación se optó por realizar una implementación propia. En primer lugar, al obtener el valor de la batería dado por el simulador, el resultado obtenido era de 100 %, independientemente de que acciones realizaran los drones o de cuanto tiempo de vuelo tuvieran. Por otra parte, para poder realizar pruebas con diversos cambios de estados, se consideró apropiado la implementación dado que facilitaba la modificación de los tiempo de carga y descarga, lo cual repercute directamente en la simplicidad de reproducir ciertos escenarios de prueba.

Inicialmente, tanto la carga como la descarga, se realizaron de forma lineal. Es decir, considerando que el drone Parrot Bebop 2 posee una autonomía de 25 minutos, se asumió que en agotarse un 1 % se tarda 15 segundos. Análogamente para la carga, un drone tarda 105 minutos en tener carga completa por lo tanto demora 63 segundos en cargar un 1 % (ver figura 4.17).

Se modificó el algoritmo de simulación de batería para que éste se asemeje más a la realidad, para la carga se siguió implementado de la misma forma, sin embargo para la descarga se tiene en cuenta en que estado se encuentra el drone. Cuando está explorando se mueve a una velocidad baja

y constante por lo que en este momento la descarga es normal. Sin embargo cuando está siguiendo a un intruso generalmente los rotores se mueven a una velocidad muy superior, lo cual tiene un mayor consumo de batería. Por lo tanto, se decidió que el intervalo de tiempo en el que se tarda en agotar un uno por ciento sea menor. (ver figura 4.17).

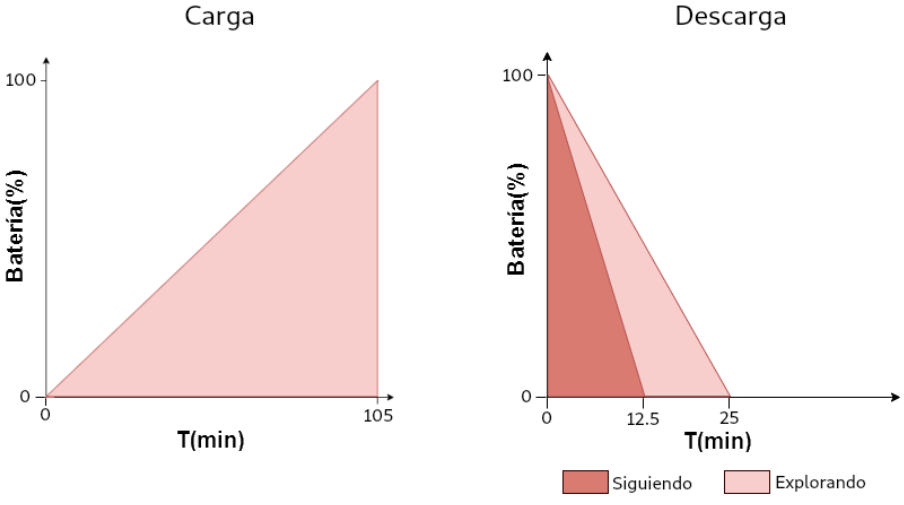


Figura 4.17: Batería en función del tiempo.

## 4.4. Detección y seguimiento

En un sistema de seguridad con drones autónomos es de vital importancia poder detectar y tener un seguimiento de los intrusos, para ello se implementó un sistema de procesamiento de imágenes y seguimiento utilizando la biblioteca OpenCV. En ésta sección se describen: el método mediante el cual los drones detectan intrusos, la corrección de falsos positivos al momento de detectar intrusos y el proceso de seguimiento del intruso.

### 4.4.1. Detección de intrusos

OpenCV proporciona diversas funcionalidades básicas que facilitan el trabajo de interpretación y reconocimiento de imágenes.

Para este trabajo, la base para la detección de intrusos reside en el histograma de gradientes orientados (Histogram of Oriented Gradients, HOG). El histograma de gradientes orientados es un descriptor de características el cual tiene como objetivo generalizar los objetos que aparecen en una imagen de manera que la apariencia y la forma (de los objetos o personas) se pueda describir mediante la distribución de los gradientes de intensidad o las direcciones de borde. La imagen se divide en pequeñas regiones conectadas llamadas celdas y para los píxeles dentro de cada celda, se compila un histograma de direcciones de gradiente y el descriptor es la concatenación de estos histogramas [40] [20].

La biblioteca OpenCV ofrece un conjunto de funcionalidades relacionadas al algoritmo HOG a través de la función llamada $HOGDescriptor$, la cual crea una instancia del descriptor y detector HOG para su uso. Con la ayuda de otras funciones disponibles en la biblioteca OpenCV, como $setSVMDetector$ y $getDefaultPeopleDetector$, se puede pre-configurar el algoritmo para la detección de personas. La primera función ($setSVMDetector$) establece los coeficientes para el clasificador lineal de la máquina de vec-

tores de soporte (Support Vector Machine, SVM). La segunda función ($getDefaultPeopleDetector$) devuelve los coeficientes del clasificador capacitado para la detección de personas.

Una vez inicializado el histograma de gradientes orientados, se hace una captura del vídeo que se va a analizar y se realiza la correspondiente separación en cuadros (frames) para eventualmente procesarlos. En cada procesamiento de cuadro se usa una función de HOG para detección, de nombre: $detectMultiScale$. Esta función permite detectar objetos a partir de una imagen según una serie de entradas que recibe por parámetro. En el caso de este proyecto se utilizaron $WinStride$, $Padding$ y $Scale$ [20].

- $WinStride$: es una tupla que dicta el "tamaño de los pasos" (en el algoritmo HOG) en el eje X y en el eje Y de la ventana deslizante. Se llama ventana deslizante a: seleccionar un rectángulo dentro de la imagen (ventana) y aplicar sobre él un conjunto de operaciones, luego desplazar (deslizar) la ventana un píxel hacia la derecha y repetir el proceso, esto se usa principalmente para la búsqueda de características [41].
- $Padding$: es una tupla que indica el número de píxeles en las direcciones X e Y donde la región de interés (Region of interest, ROI) de la ventana deslizante se "rellena" antes de la extracción de la característica HOG.
- $Scale$: controla el factor en el que la imagen se redimensiona en cada capa de la pirámide de imágenes, influyendo en última instancia en el número de niveles de la pirámide de imágenes.

Es relevante destacar que los parámetros $WinStride$ y $Scala$ son parámetros importantes y deben configurarse correctamente. Estos parámetros tienen implicaciones no solo en la precisión con la que se detecta, sino que también en la velocidad con la que funciona el detector. Basado en algunas pruebas hechas con el simulador y en [42] se decidió usar la configuración: $winStride$=(8,8), $padding$=(32,32), $scale$=1.05.

Para concluir el proceso de detección, se procesa el conjunto de "rectángulos de detección" (ver imagen 4.18) generados por la función *detectMultiScale*; Este conjunto de rectángulos de detección representan personas encontradas por el algoritmo. Durante el procesamiento del conjunto de rectángulos, se realiza una "supresión" de los datos a través de la función *non_max_supression* para corregir la superposición de los rectángulos de detección, el resultado de esa supresión se puede ver en la figura 4.19. La superposición de rectángulos se debe a que, como el algoritmo HOG junto al SVM funcionan correctamente, se generan varias detecciones sobre un mismo objeto (ver en la imagen 4.19) entonces para poder indicar que se ha encontrado una sola ocurrencia del intruso en sí, se necesita aplicar la supresión anteriormente mencionada.

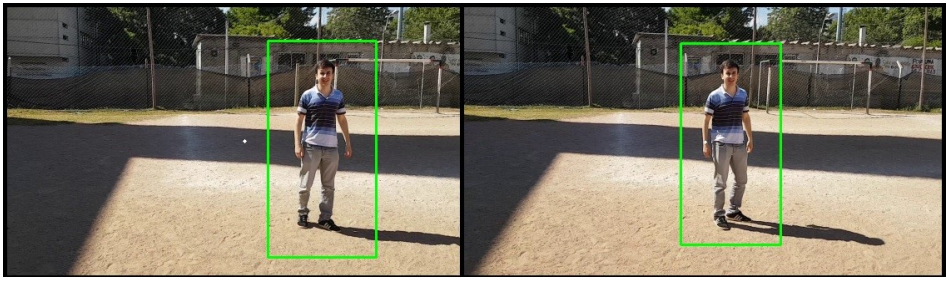

Figura 4.18: Imagen ilustrativa del rectángulo de detección.

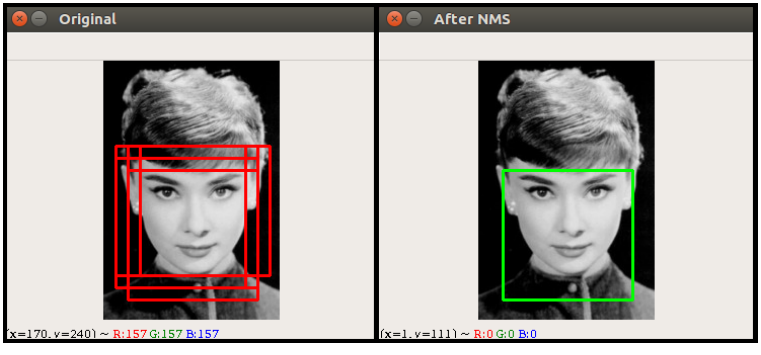


Figura 4.19: Diferencia entre imagen sin supresión y con supresión (imagen realizada con el software de código abierto PyImageSearch).

### 4.4.2. Corrección de falsos positivos

Aunque en términos generales la funcionalidad de detección funciona correctamente y logra detectar personas quietas o en movimiento, ocasionalmente pueden ocurrir falsos positivos (objetos detectados que tienen la propiedades de tamaño y forma de una persona pero no lo son) en cuadros de imágenes tomados durante el seguimiento y/o detección. Para los casos donde hay falsos positivos es necesario hacer una corrección de los datos obtenidos en el algoritmo HOG, para evitar que por una incorrecta interpretación de la información, la flota de drones tome decisiones equivocadas.

Los pasos que se siguieron para resolver el problema de falsos positivos son: primero, a partir de los "rectángulos de detección"de cada frame se determina el punto medio de dicho rectángulo y se guarda temporalmente la información. Cada punto medio está determinado por un ancho (width) y un alto (height) en píxeles. Luego se compara la posición entre el centro del último rectángulo de detección obtenido ($npX$, $npY$) y el centro del rectángulo de detección anterior ($opX$,$opY$), si la diferencia de la posición es menor a un cierto épsilon previamente escogido, entonces el objeto es el mismo, en caso contrario se ignora el objeto detectado debido a que es un falso positivo.

La distancia entre los centros de los recuadros de cada marco se determina por el valor absoluto de la diferencia de los anchos ($|opX - npX|$) y el valor absoluto de la diferencia de los largos ($|opY - npY|$). El épsilon elegido fue determinado en base a ensayos y errores al trabajar con un número comprensible de vídeos con los que se realizaban detecciones de personas. Dicho épsilon se definió como una décima parte del ancho y el largo del marco capturado por la cámara. La condición que determina los falsos positivos queda de la siguiente forma:

$$(|opX - npX| \leq \epsilon) \text{ y } (|opY - npY| \leq \epsilon)$$

Como solución al caso borde de que el primer marco detectado era un falso positivo se implementó una condición de seguimiento. El seguimiento de una persona comienza si se supera un número arbitrario de detecciones de la misma persona. Si bien eligiendo un número de detecciones como por ejemplo 5 éste problema se soluciona, se notó que se volvió más improbable que se produzca una detección sobre una persona lo que resultaba en un aumento de falsos negativos, es decir, en marcos donde hay personas estas no se detectan. Realizando un análisis experimental (ver sección 5.2) se resolvió que el mejor resultado en general se da cuando el número de detecciones es una, es decir, cuando no se utiliza esta restricción.

### 4.4.3. Seguimiento de intrusos

Una vez que se identifica un intruso empieza el sistema de seguimiento del objetivo. En el sistema de seguimiento del objetivo, los drones actualizan sus estados y utilizan el sistema de comunicación para enviarse la posición actual del drone que esta detectando un intruso, posición del objetivo, estado actual del drone, entre otros. El sistema de seguimiento continua ejecutándose hasta el momento en que se da por perdido al objetivo. El objetivo se da por perdido luego de que el sistema de seguimiento pierde contacto visual con él, y a pesar de intentar recuperarse, no logra detectarlo mas.

El sistema de seguimiento comienza verificando el estado actual del drone, si el estado actual es “explorando” (Exploring) y detecta un intruso entonces el sistema actualiza su estado a “seguimiento” (Following). Después de que el sistema cambia el estado del drone, envía un mensaje a los otros drones de la flota indicando que ha comenzado el seguimiento. Una vez enviado el mensaje, el sistema actualiza variables internas que almacenan marcas de tiempo (Timestamps) y un contador de drones en modo de seguimiento y finalmente busca el mejor drone para hacer apoyo de seguimiento.

**Algoritmo 5** Algoritmo intercambio de rutas

```
1:  procedure BUSCARMEJORDRONE()
2:      mejorDrone = null
3:
4:      for cada drone en el sistema do
5:          enSeguimiento = (drone.status == "seguimiento" ||
6:                           drone.status == "apoyo de seguimiento")
7:          if (!enSeguimiento && drone.bateria > mejorDrone.bateria) then
8:              mejorDrone = drone
9:
10:     if (mejorDrone.bateria > 50) then
11:         solicitarAyuda(mejorDrone)
```

Para buscar el drone que mas se adecua para hacer el apoyo de seguimiento, el sistema chequea uno por uno el estado y la carga de batería del resto de los drones de la flota, luego busca cual es el drone con más batería y que no esté en el estado de "seguimiento" ni en el estado de "apoyo de seguimiento" (linea 2 a 8 del Algoritmo 5). Por último, si el drone con mayor carga de batería y estado diferente a seguimiento y apoyo de seguimiento tiene un porcentaje de batería mayor al 50 por ciento, el sistema le indica a ese drone que tiene que hacer apoyo de seguimiento al drone que detectó al intruso (linea 10 y 11 del Algoritmo 5). En caso de que ningún drone cumpla con las condiciones mencionadas, no se solicitará ayuda en el momento pero se preguntará periódicamente si hay algún drone disponible para hacer apoyo de seguimiento.

Una vez determinado el mejor drone para el apoyo de seguimiento, el sistema continúa con el proceso de seguimiento. El sistema almacena en una lista toda la información obtenida hasta ese momento sobre el objetivo detectado. En la lista se guardan los siguientes datos: cantidad de drones en estado de seguimiento o apoyo de seguimiento, posición del centro del objeto detectado respecto al campo de visión del drone, distancia al objetivo, longitud y latitud del objetivo, orientación hacia el objetivo y una marca de tiempo de cuando fue guardada la información.

Luego de guardar la información, el sistema actualiza su información para centrarse al objetivo y si hay algún drone de apoyo, el sistema le envía la información para que pueda comenzar a seguir al objetivo. En caso de que no haya ningún drone de apoyo, el sistema vuelve a buscar el mejor drone para apoyo de seguimiento con el procedimiento que se mencionó anteriormente.

Antes de enviarle la información al drone de apoyo de seguimiento, el sistema estima la próxima posición en la que va a estar el objetivo. Para lograr estimar la próxima posición, el sistema usa todas las posiciones de latitud y longitud del objetivo que ha almacenado a lo largo del proceso de seguimiento y calcula el polinomio de grado uno que mejor se ajusta a todos esos puntos (para esto se asume que el intruso se desplaza en una linea recta). Para calcular el polinomio que mejor se ajusta se hace uso de la función $polyfit$ de Python. En base al polinomio obtenido se estima la próxima latitud y longitud del objetivo en dos segundos.

Finalmente, después del cálculo de estimación de posición, el sistema envía un mensaje al drone de apoyo de seguimiento con la información obtenida. Al terminar de enviar el mensaje se vuelve a comenzar un nuevo ciclo en el procesamiento de imagen y en el sistema de seguimiento.

Desde la perspectiva del drone de apoyo de seguimiento, el drone se encuentra previamente, en estado explorando y recibe un mensaje del drone que está realizando el seguimiento. Al recibir el mensaje, el drone cambia su estado a apoyo de seguimiento e inmediatamente se mueve a la posición que el otro drone le indica donde estará el intruso. Cuando el drone de apoyo de seguimiento detecta al intruso, el drone se posiciona a noventa grados (ver figura 4.20) con respecto a la línea central de visión del drone de seguimiento. Esta acción se realiza para poder mejorar el campo de cobertura de visión del sistema de seguridad con drones. Luego el drone de apoyo de seguimiento continua en comunicación con el drone de seguimiento para mantener vigilado el intruso manteniendo los ángulos de visión.

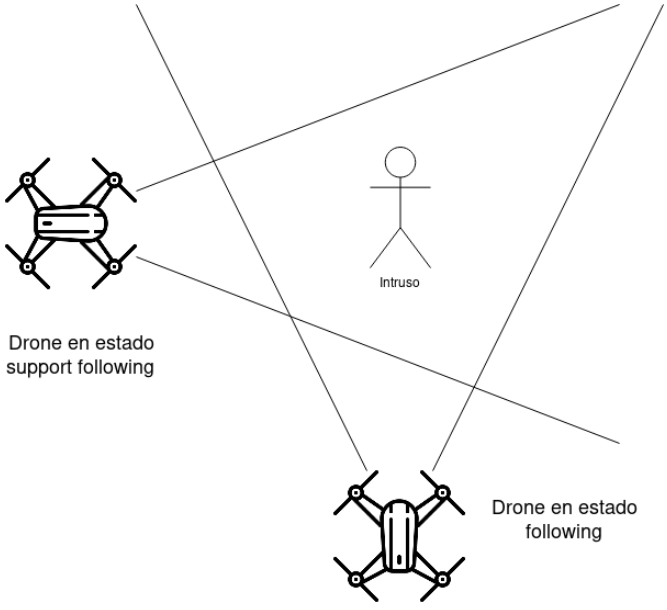


Figura 4.20: Support Following.

Es importante mencionar que durante el proceso de seguimiento y detección puede surgir que el sistema pierda contacto visual con el objetivo. Ante el caso de que se pierda contacto visual con el objetivo y el sistema no pueda detectarlo, el sistema cambia su estado a "escanear"(Scan) y el drone comienza a girar en su propio eje en busca del objetivo. En caso de que el sistema lo detecte nuevamente, el sistema cambia su estado a seguimiento y continua con el proceso mencionado en esta sección. En caso contrario, el sistema cambia su estado a explorando y vuelve a su ruta de navegación, descripta en la sección 4.3.

## 4.5. Configuración y depuración

En la etapa inicial del proyecto, todas las variables eran cargadas directamente en el código, lo que resultaba complicado y engorroso cuando se deseaba realizar alguna actualización sobre éstos valores, ya que era necesario actualizar en todos los lugares donde se usaba. Por dicho motivo, se consideró apropiado contar con un archivo de configuración y en él tener todas las variables, tanto las que son usadas para la calibración (para el procesamiento de imágenes) como las que están relacionadas directamente con el drone (identificador dentro de la flota, altura de vuelo, posición inicial).

Contar con un archivo de configuración también facilitó el proceso de agregar un nuevo drone a la flota, dado que alcanza con actualizar las variables en dicho archivo (por ejemplo, indicar cuál será el identificador del nuevo drone, a qué altura tendrá que volar normalmente, cuáles son las coordenadas donde se encuentra su estación de carga, entre otras).

Para facilitar la depuración (debugging), se implementaron mecanismos de registros (logging). El logging permite visualizar y analizar el procesamiento que hizo el drone, permitiendo luego la comparación del resultado efectuado con el resultado esperado. Se realizaron dos sistemas de registros:

- **mapa:** ilustra de forma gráfica cada camino calculado por el drone, donde cada punto es representado de la forma DI_J (por ejemplo D1_4) siendo $I$ el identificador del drone y $J$ el número del paso (ver figura 4.21).
- **log:** se genera un log en el que se registran todas las acciones que el drone realiza, por ejemplo, el estado de la batería, los cambios de transiciones en la máquina de estado, los mensajes recibidos por los restantes drones de la flota, los mensaje que son enviados, entre otros.

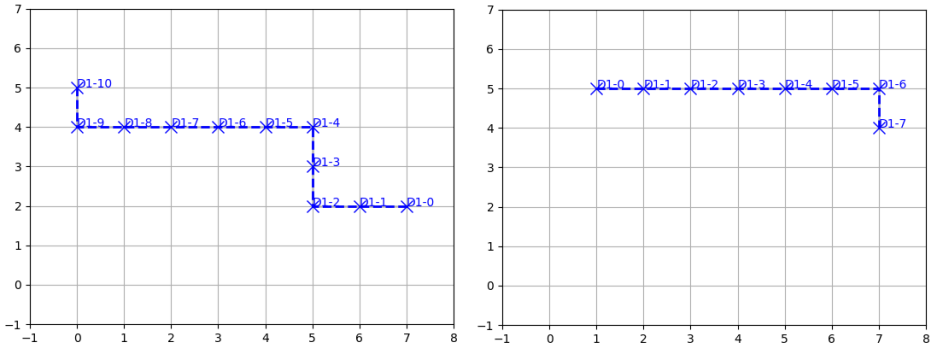


Figura 4.21: Mapa de ruta calculada.

La combinación de los sistemas de logging mencionados permite detectar posibles problemas que estén ocurriendo en el sistema y otorga un mayor dinamismo dado que se puede ver de forma gráfica los caminos que están realizando. Tener dibujados los caminos que tomó en el mapa cada drone en cada momento de la simulación, acompañado con los cambios de estado, mensajes enviados y recibidos escritos en el archivo de logging permite realizar un análisis post mortem más preciso y completo sobre el vuelo de la flota de drones.

# Capítulo 5

# Análisis experimental

En este capítulo se presenta un análisis de la solución implementada. Se divide el análisis experimental en dos partes, en la sección 5.1 se evalúa el sistema de navegación y en la sección 5.2 se evalúa el sistema de detección de intrusos y procesamiento de imágenes.

## 5.1. Navegación del predio

Esta sección presenta un análisis de la solución implementada para la navegación. En la sección 5.1.1 se describen las metodologías de evaluación, la sección 5.1.2 explica en detalle los escenarios de pruebas, mientras que en la sección 5.1.3 se exponen los resultados obtenidos.

### 5.1.1. Metodología de evaluación

El predio en el que se basan las pruebas realizadas de este proyecto de grado, es el terreno adjunto al Instituto de Computación de la Facultad de Ingeniería de la Universidad de la República (ver figura 5.1). La subdivisión de este predio es de 40 metros de largo por 30 metros de ancho y está dividida en celdas de 5 metros de largo por 5 metros de ancho, las cuales forman la grilla.

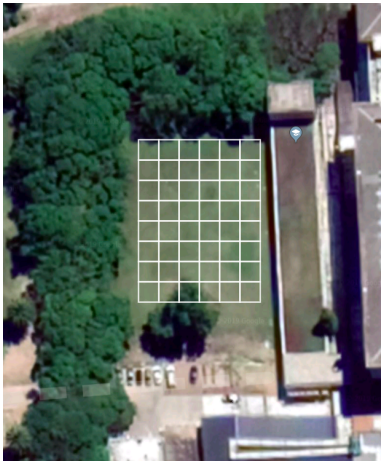

Figura 5.1: Subdivisión del predio.

Las pruebas experimentales de navegación se realizaron en el simulador Sphinx de Parrot. Dado que la autonomía de los drones Parrot Bebop 2 es de 25 minutos, se ejecutaron las pruebas del sistema de vigilancia por un tiempo de 20 minutos. Se diseñaron tres escenarios de prueba que son descriptos en detalle en la sección 5.1.2. En cada uno de ellos se ejecutó el sistema de vigilancia con un solo drone y luego con una flota de dos drones. Se decidió también ejecutar el algoritmo de navegación en distintas etapas de su construcción, con el objetivo de detectar si las mejoras o restricciones agregadas a éste aumentaban la eficiencia del sistema de vigilancia o la disminuían. Las etapas del algoritmo que se eligieron para realizar las

pruebas son desarrolladas en la sección 4.3.2, éstas son: scan, trayectoria y trayectoria con restricción. El algoritmo scan es un algoritmo básico sin inteligencia computacional, se realizan pruebas con él principalmente para utilizarlo como referencia con los otros algoritmos. Para éstas pruebas del sistema de navegación se asume que no se detectan intrusos en ningún momento.

Para medir la eficiencia y eficacia de las distintas soluciones se utilizan dos métricas. La primera es contar cuantas veces se visita cada celda del predio. Se espera que celdas con mayor prioridad sean visitadas con mayor frecuencia que celdas con menor prioridad. Para visualizar esta métrica, en cada ejecución del sistema se crea un mapa de calor de la matriz que representa el predio, indicando la cantidad de veces que fue visitada cada celda. Se crea también un mapa de calor de la matriz de prioridades donde en cada celda se indica su prioridad con el objetivo de usarla como referencia. Cuanto más similares sean estos mapas de calor entre sí, mejor será la eficacia del sistema ya que implica que el drone o la flota de drones visitó una mayor cantidad de veces las celdas con mayor prioridad. La segunda métrica utilizada se basa en el beneficio de vigilancia, para cada drone se obtiene el beneficio de vigilancia acumulado durante la ejecución. A modo de simplificar los resultados se continua utilizando el beneficio de vigilancia con envejecimiento para los cálculos de las trayectorias en la ejecución, pero se utiliza el beneficio de vigilancia sin envejecimiento para mostrar los resultados.

### 5.1.2. Escenarios de prueba

Se crearon tres escenarios para realizar las pruebas de exploración. Cada uno de ellos presenta distintas características, lo cual permite evaluar el desempeño del algoritmo en distintas situaciones. En el primer escenario (ver figura 5.2) las prioridades se encuentran distribuidas de forma casi uniforme, es decir, no existe ninguna sección con una clara preferencia sino que la vigilancia sobre éste predio será uniforme.

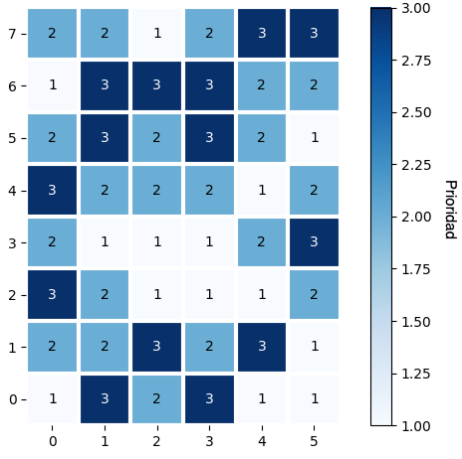


Figura 5.2: Prioridades del escenario 1

En el segundo escenario (ver figura 5.3) se cuenta con dos zonas en las que se presenta una diferencia de prioridad relevante respecto al resto del predio, una casi central y una sobre el limite del predio que representa un ingreso.

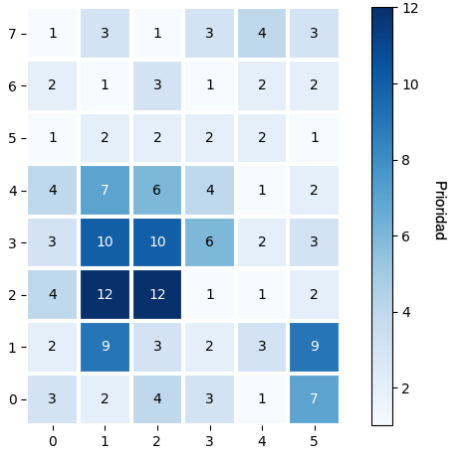


Figura 5.3: Prioridades del escenario 2

En el tercer escenario (ver figura 5.4) uno de los laterales presenta una zona de alta prioridad respecto al resto del predio.

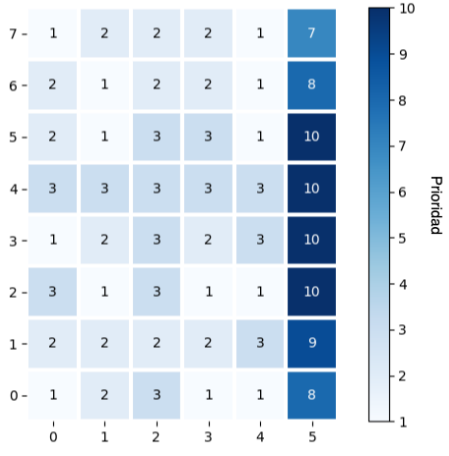


Figura 5.4: Prioridades del escenario 3

### 5.1.3. Resultados obtenidos

A continuación se analizarán los resultados obtenidos a partir de dos tipos de pruebas: las pruebas realizadas con un drone y las pruebas realizadas con una flota de dos drones. Finalmente se presenta un cuadro con el objetivo de comparar los algoritmos.

#### Pruebas de navegación con un drone

Para las pruebas siguientes se considera únicamente un drone, el cual parte del punto $(0, 0)$ de la matriz que representa el predio.

#### Escenario 1

En la figura 5.5 se muestran los resultados de ejecutar los distintos algoritmos con un drone sobre el escenario 1 así como también el mapa de prioridades del escenario 1 (arriba al centro). Debajo a la izquierda se muestra el mapa de calor resultante luego de ejecutar el algoritmo scan, debajo al centro el resultado de ejecutar el algoritmo de trayectoria sin restricción, y debajo a la derecha el resultado de ejecutar el algoritmo de trayectoria con restricción.

Como es esperado, en el algoritmo scan la cantidad de visitas a cada celda es uniforme ya que el algoritmo no tiene en cuenta las prioridades. Por el contrario, en el algoritmo de trayectoria sin restricciones visita diferente cantidad de veces las celdas del predio. En el algoritmo trayectoria con restricción se puede ver que el drone se desplaza principalmente por el centro del predio porque aunque sea un escenario uniforme la mayoría de las celdas de mayor prioridad se encuentran allí. Además, al tener la restricción de las columnas, el drone visita poco los laterales y se mueve principalmente por el centro del predio.

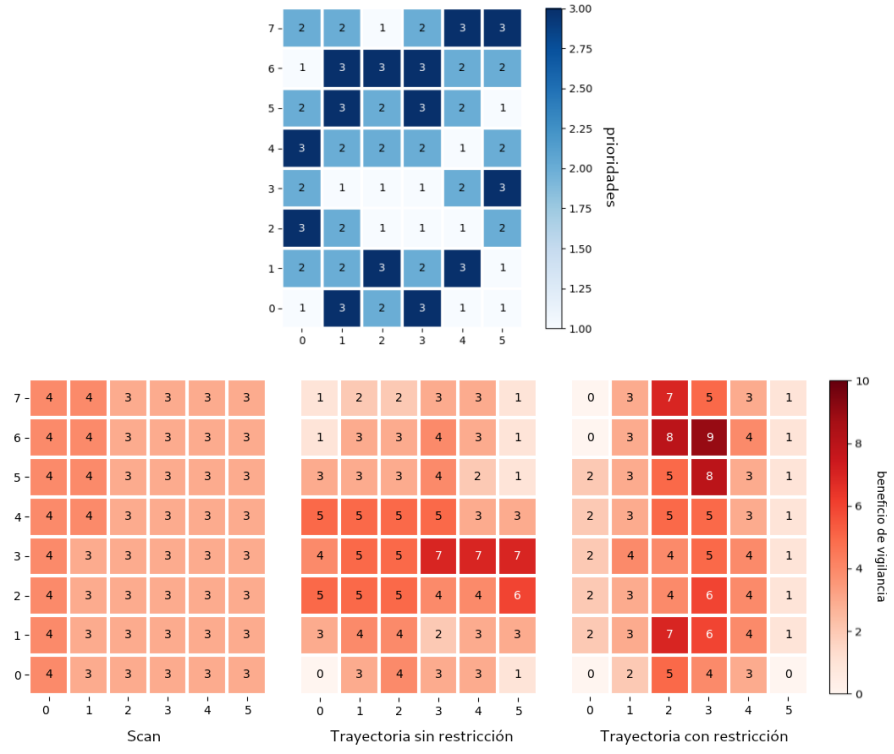


Figura 5.5: Algoritmos en el escenario 1.

### Escenario 2

En la figura 5.6 debajo a la izquierda se muestra el mapa de calor resultante de aplicar el algoritmo scan al escenario 2, se puede ver que es exactamente el mismo que el del escenario 1 dado que el algoritmo scan sigue una trayectoria fija. Luego de ejecutar el algoritmo de trayectoria sin restricciones sobre el escenario 2 se obtiene el mapa de calor que se muestra en la figura 5.6 debajo al centro. Se puede ver que éste mapa de calor coincide mayoritariamente con el mapa de calor de las prioridades del escenario 2 (ver figura 5.6). El algoritmo de trayectoria sin restricciones tiene gran cantidad de visitas en celdas que se encuentran sobre la zona de

alta prioridad. En la figura 5.6 (debajo a la derecha) se muestra el mapa de calor obtenido luego de ejecutar el algoritmo de trayectoria con restricción sobre el escenario 2. De manera similar a lo que sucede en el escenario 1 al usar este algoritmo, el drone recorre principalmente el centro del mapa porque allí se encuentran las celdas con mayor prioridad, descuidando las columnas de los límites del predio incluyendo las celdas de alta prioridad de la zona de ingreso al predio. Análogamente al escenario anterior, la diferencia de la ejecución con el algoritmo de trayectoria sin restricciones es que las visitas a las celdas son menos uniformes.

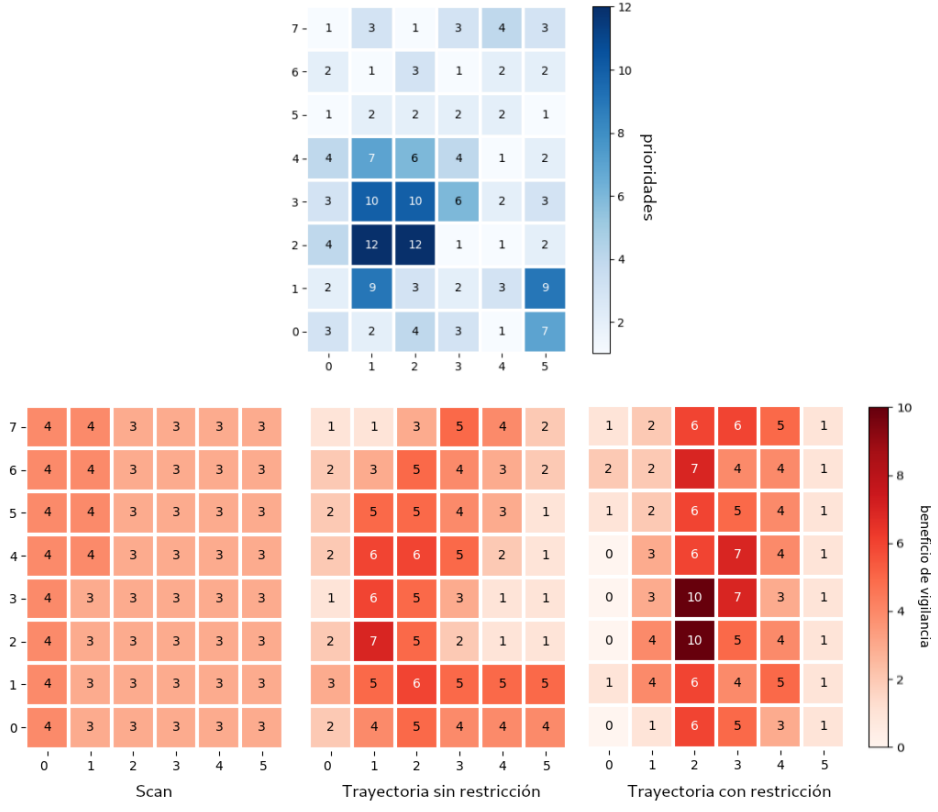


Figura 5.6: Algoritmos en el escenario 2.

**Escenario 3**

En la figura 5.7 debajo a la izquierda se muestra el mapa de calor resultante de aplicar el algoritmo scan al escenario 3, el cual es el mismo que el del escenario 1 y escenario 2. Debajo al centro se muestra el mapa de calor resultante de ejecutar el algoritmo de trayectoria sobre el escenario 3. El drone recorre el predio visitando en mayor medida las celdas con mayor prioridad, sin dejar de visitar en buena medida otras celdas con menor prioridad.

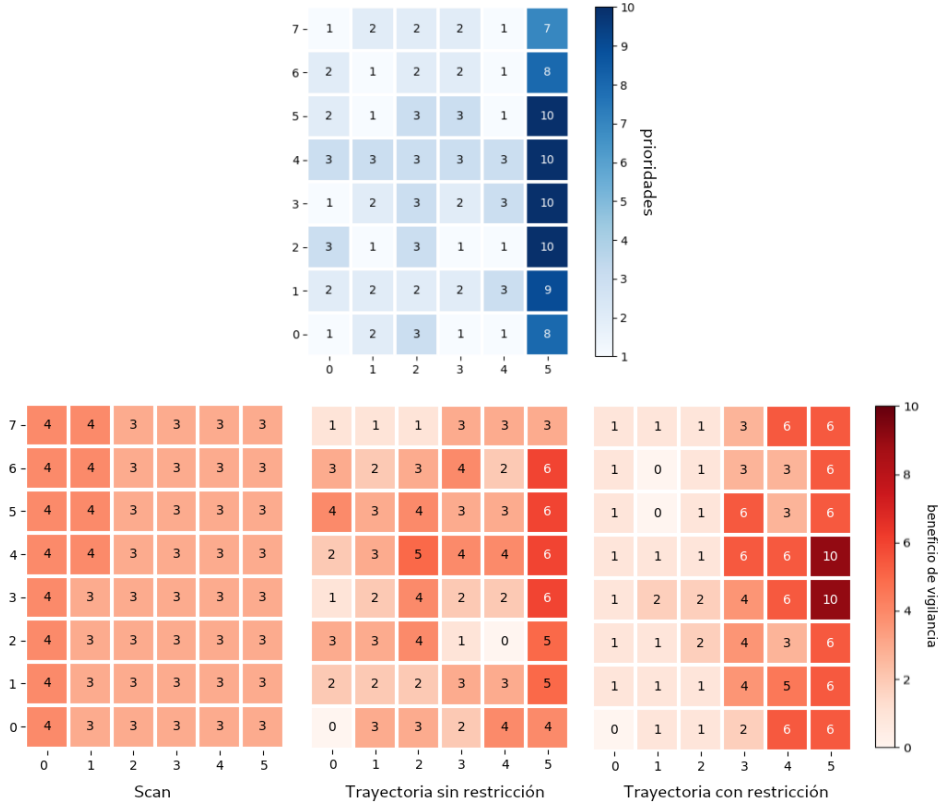


Figura 5.7: Algoritmos en el escenario 3.

Luego de ejecutar el algoritmo de trayectoria con restricción sobre el escenario 3 se obtiene el mapa de calor de la figura 5.7 debajo a la derecha. Este mapa de calor coincide en gran medida con el mapa de calor de las prioridades de éste escenario (ver figura centro arriba). El drone recorre principalmente la zona del lateral derecho porque allí se encuentran las celdas con mayor prioridad de la matriz. Debido a la restricción de columnas, el drone se mantiene la mayor parte de la ejecución sobre la zona del lateral derecho y solo visita una o dos veces las celdas del otro lado del predio.

### Comparación de resultados

Por último, en el cuadro 5.1 se muestra los valores de beneficio de vigilancia acumulado para cada algoritmo en cada escenario. Para los valores del escenario 1 en particular se puede notar que el valor es muy similar entre los tres ya que al ser el escenario un predio con prioridades uniformes, se obtiene un beneficio acumulado similar independientemente de la estrategia con la que se recorra el predio.

En el escenario 1 y 2 no se puede apreciar una diferencia considerable entre el algoritmo de trayectoria con restricción y el algoritmo de trayectoria sin restricción.

| Métrica | Algoritmos | | |
|---|---|---|---|
| | Scan | Trayectoria | Trayectoria con restricción |
| Escenario 1 | 325 | **343** | 339 |
| Escenario 2 | 534 | **737** | 721 |
| Escenario 3 | 526 | 603 | **729** |

Cuadro 5.1: Beneficio de vigilancia con un drone.

La principal conclusión que se obtiene es que para escenarios donde existen prioridades diferenciadas, los algoritmos de trayectoria son más eficientes que el scan dado que obtienen mayor beneficio de vigilancia. En el

caso del escenario 3 el algoritmo de trayectoria con restricción muestra un mayor beneficio de vigilancia y su mapa de calor es más parecido al mapa de calor de las prioridades del escenario. Esto se explica por la particularidad del escenario, el tener una columna con prioridades muy altas es más beneficioso para el algoritmo de trayectoria con restricción ya que visita mas seguido ésta columna descuidando el resto del predio, mientras que el algoritmo de trayectoria sin restricciones suele repartir mas sus visitas sin descuidar el resto del predio, es decir visita regularmente todas las celdas aunque visitando con mayor frecuencia las celdas de mayor prioridad.

### Pruebas de navegación con una flota de drones

Para las pruebas siguientes se considera una flota de dos drones. Se espera que el beneficio acumulado de la flota sea mayor que el de las pruebas anteriores, y que la flota visite las celdas prioritarias de la matriz una mayor cantidad de veces. El Drone1 comienza su recorrido en la celda $(0, 0)$ mientras que el Drone2 lo hace en la celda $(7, 5)$.

### Escenario 1

En la figura 5.8 se muestran los mapas de calor de cada drone y la suma de ambos, es decir el mapa de calor de la flota luego de ejecutar el algoritmo scan en el escenario 1. También se vuelve a mostrar el mapa de priorirdades del escenario 1 para poder comparar fácilmente los resultados con él. Del mismo modo que con un drone, se puede ver que la cantidad de visitas a cada celda es uniforme dado que no es un algoritmo inteligente.

En la figura 5.9 se muestra los mapas de calor de la flota luego de ejecutar el algoritmo de trayectoria sin restricciones en el escenario 1. El mapa de calor de la flota muestra una distribución uniforme como se espera, sin embargo la mayoría de las celdas con mayor prioridad son visitadas más frecuentemente. Los drones se distribuyen la vigilancia del predio equitativamente dado que se tiene una grilla con prioridad uniforme.

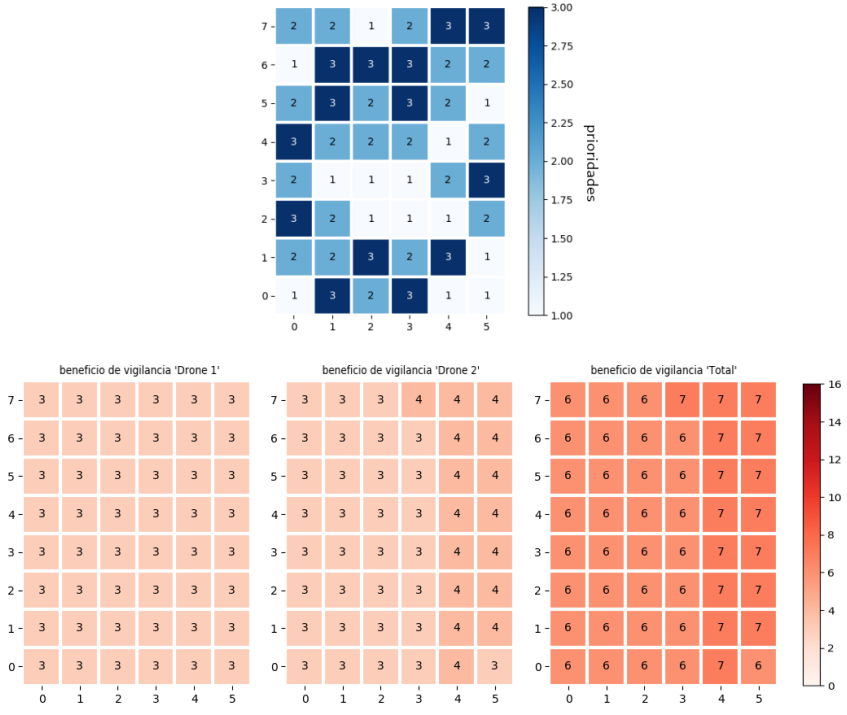


Figura 5.8: Algoritmo scan con dos drones en el escenario 1.

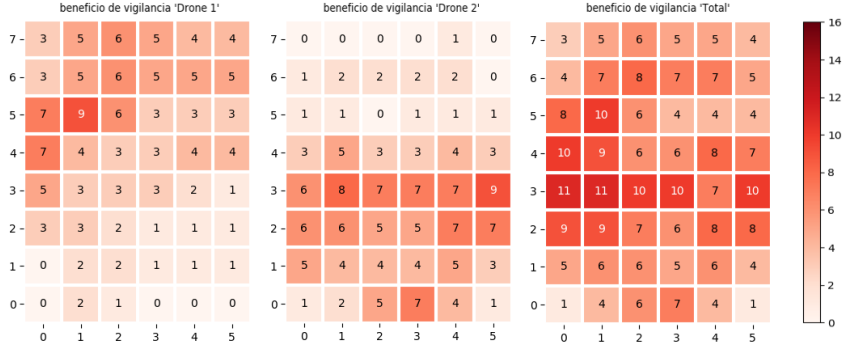


Figura 5.9: Algoritmo trayectoria con dos drones en el escenario 1.

La figura 5.10 muestra los mapas de calor de cada drone y la flota luego de ejecutar el algoritmo de trayectoria con restricción. Se puede ver como los drones de la flota se reparten el predio dado que las prioridades de las celdas son uniformes y además, con la restricción de columnas es más improbable que un drone se desplace de un lado a otro del predio. El mapa de calor de la flota es similar al obtenido en la ejecución del algoritmo de trayectoria, variando levemente la cantidad de veces que son visitadas algunas celdas, pero sin diferencias importantes.

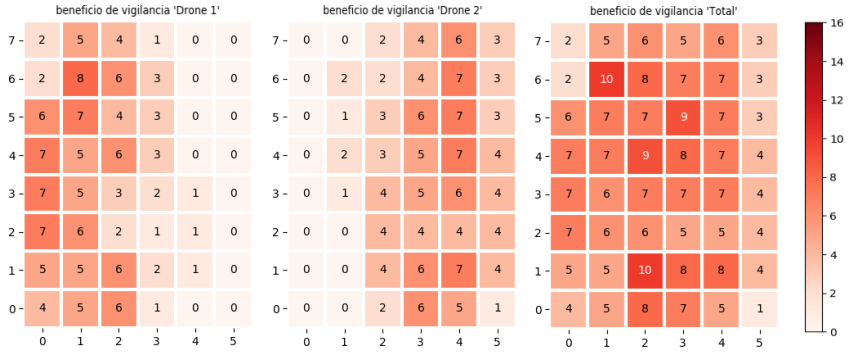


Figura 5.10: Algoritmo trayectoria con restricción con dos drones en el escenario 1.

Finalmente el cuadro 5.2 muestra los beneficios de vigilancia acumulados por la flota durante las ejecuciones con los distintos algoritmos. Del mismo modo que recorriendo el predio con un drone, no se obtienen diferencias considerables entre los beneficios de vigilancia de los distintos algoritmos por ser el predio casi uniforme. Otra conclusión que se desprende luego de ver el cuadro es que con la ejecución de cualquier algoritmo no se obtienen grandes diferencias entre los beneficios de vigilancia acumulados por cada drone, esto se debe principalmente a que los drones se reparten el predio y éste es casi uniforme.

| Métrica | Algoritmos | | |
|---|---|---|---|
| | Scan | Trayectoria | Trayectoria con restricción |
| Beneficio drone 1 | 287 | 322 | 314 |
| Beneficio drone 2 | 302 | 313 | 306 |
| Beneficio total | **589** | **635** | **620** |

Cuadro 5.2: Escenario 1 con dos drones.

**Escenario 2**

En la figura 5.11 se muestran el mapa de prioridades y los mapas de calor obtenidos luego de recorrer el escenario 2 con una flota de dos drones con el algoritmo scan, se puede ver que son iguales a los mapas de calor obtenidos luego de haber recorrido el escenario 1 con el algoritmo scan, ya que éste siempre sigue la misma trayectoria independientemente de las prioridades del predio.

La figura 5.12 muestra los mapas de calor que se obtienen al ejecutar el algoritmo de trayectoria sobre el escenario 2. Al igual que en el escenario 1, con éste algoritmo los drones recorren todo el terreno. Sin embargo se puede notar que un drone visita en mayor medida las celdas de las filas 4, 5, 6 y 7 y el otro las celdas de las filas 0, 1, 2 y 3. Esto se debe a que la funcionalidad de cruce de trayectorias intercambia las trayectorias de los drones si éstas se cruzan, lo cual dificulta el cambio de zona de exploración.

En la figura 5.13 se muestra los mapas de calor resultantes de ejecutar el algoritmo de trayectoria con restricción con una flota de drones sobre el escenario 2. Similar al escenario 1, los drones se reparten el terreno debido a la restricción de columnas. Si se compara el mapa de calor de la flota utilizando el algoritmo de trayectoria con restricción y el mapa de calor de la flota utilizando el algoritmo de trayectoria (sin restricción) con el mapa de calor de las prioridades, se puede ver que si bien los dos primeros mapas son similares al mapa de prioridades, el mapa del algoritmo de trayectoria se parece más al mapa de prioridades.

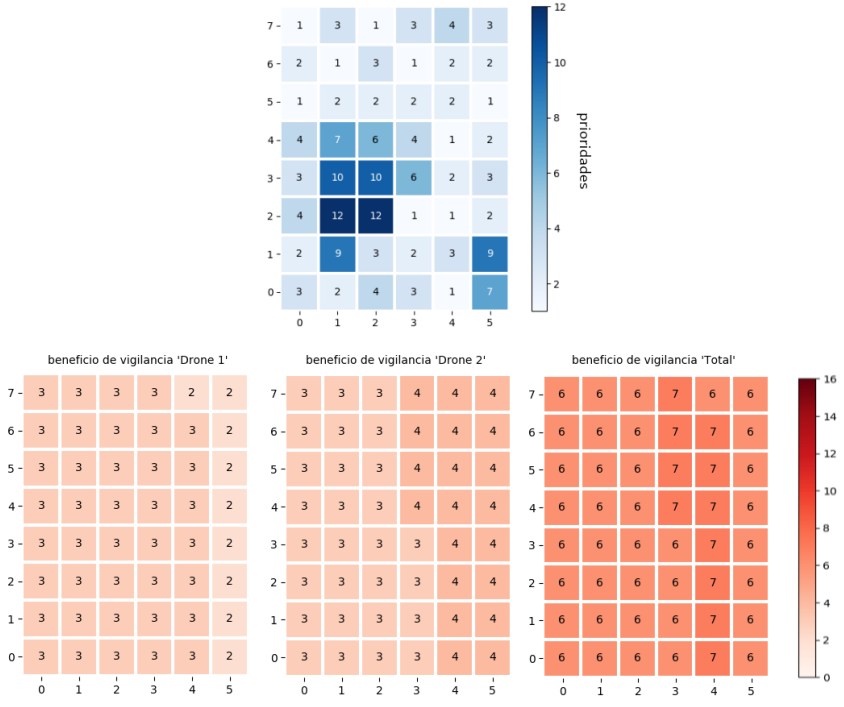


Figura 5.11: Algoritmo scan con dos drones en el escenario 1.

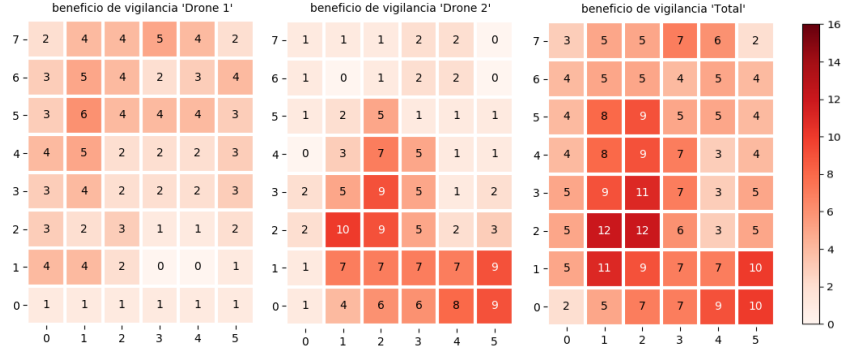


Figura 5.12: Algoritmo trayectoria con dos drones en escenario 2.

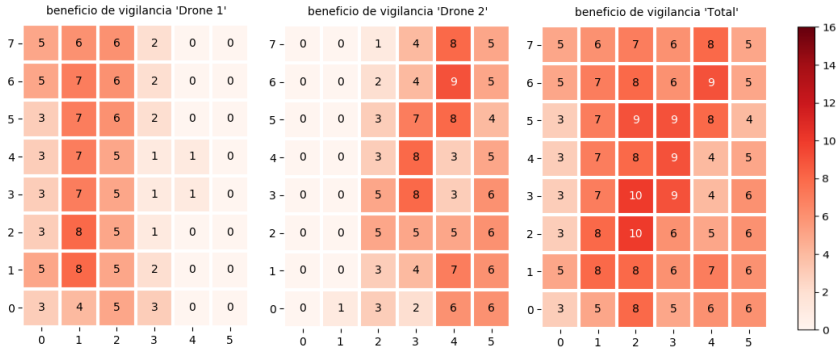


Figura 5.13: Algoritmo trayectoria con restricción con dos drones en escenario 2.

En el cuadro 5.3 se puede ver que el algoritmo de trayectoria obtiene el mayor beneficio de vigilancia, lo cual es consistente a los resultados de los mapas de calor. Otra característica interesante es que la diferencia de beneficio entre un drone y otro varia en mayor medida con respecto a las obtenidas en el escenario 1. Esto se debe a que el mapa no es uniforme, por lo tanto, aquel drone que visite zonas con mayor prioridad obtendrá mayor beneficio.

| Métrica | Algoritmos | | |
|---|---|---|---|
| | Scan | Trayectoria | Trayectoria con restricción |
| Beneficio drone 1 | 489 | 506 | 654 |
| Beneficio drone 2 | 577 | 828 | 565 |
| Beneficio total | **1066** | **1334** | **1219** |

Cuadro 5.3: Escenario 2 con dos drones.

**Escenario 3**

Luego de recorrer el escenario 3 con el algoritmo scan y una flota de dos drones se obtienen los mapas de calor que se muestran en la figura 5.14. También en esta figura se puede ver el mapa de prioridades del escenario 3 con el fin de compara los resultados con él.

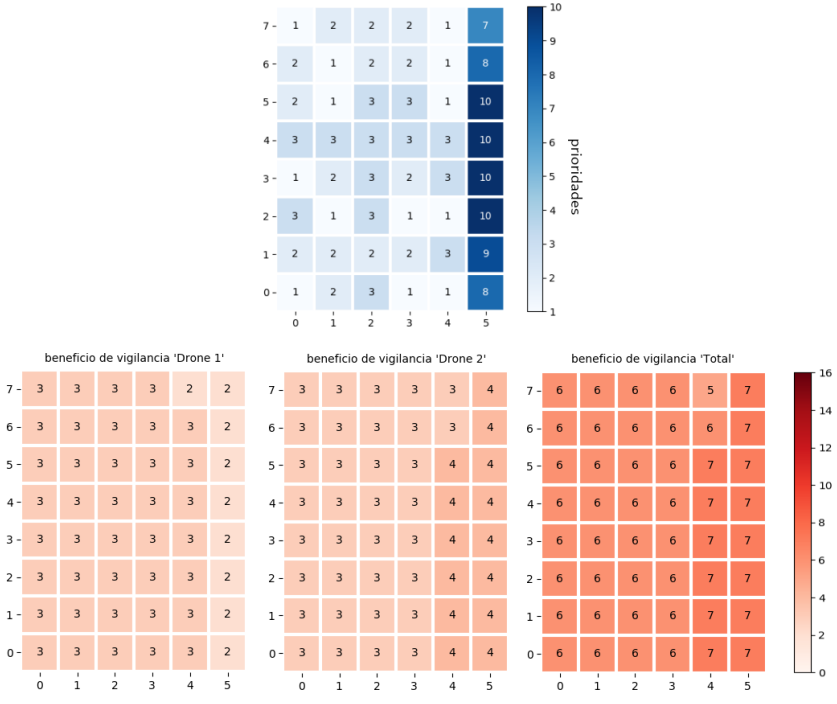


Figura 5.14: Algoritmo scan con dos drones en el escenario 3.

Luego de aplicar el algoritmo de trayectoria sin restricciones se obtienen los mapas de calor que se muestran en la figura 5.15. Aquí nuevamente se puede ver que con este algoritmo ambos drones navegan por todo el predio. El mapa de calor de la flota es similar en gran medida al mapa de calor de las prioridades del escenario (ver figura 5.14).

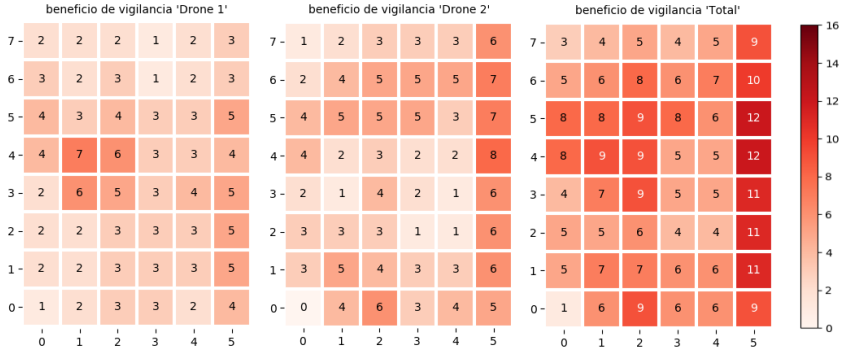


Figura 5.15: Algoritmo trayectoria con dos drones en escenario 3.

La figura 5.16 muestra el mapa de calor de cada drone y el de la flota luego de aplicar el algoritmo de trayectoria con restricción sobre el escenario 3. De igual forma a los otros escenarios los drones se reparten el predio por la restricción de las columnas. Dada la naturaleza del escenario que contiene altas prioridades sobre un lado del predio y bajas prioridades del otro lado, el drone que recorre el lado con altas prioridades termina acumulando una mayor cantidad de beneficio de vigilancia. El mapa de calor de la flota es muy similar al mapa de calor de las prioridades del escenario.

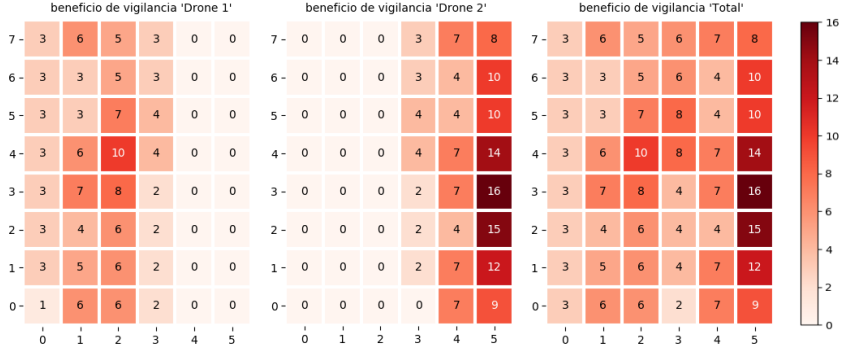


Figura 5.16: Algoritmo trayectoria con restricción con dos drones en escenario 3.

En el cuadro 5.4 se muestran los beneficios de vigilancia acumulados de cada drone y de la flota en total luego de ejecutar cada algoritmo sobre el escenario 3. Se puede notar que para este escenario el algoritmo de trayectoria con restricción obtiene un mayor beneficio de vigilancia que el algoritmo de trayectoria (sin restricción). El principal motivo es que con la restricción uno de los drones se mantiene la mayor parte de la ejecución sobre la columna con las altas prioridades obteniendo siempre en cada trayectoria un gran beneficio de vigilancia.

| Métrica | Algoritmos | | |
|---|---|---|---|
| | Scan | Trayectoria | Trayectoria con restricción |
| Beneficio drone 1 | 383 | 505 | 328 |
| Beneficio drone 2 | 540 | 573 | 949 |
| Beneficio total | **923** | **1078** | **1277** |

Cuadro 5.4: Escenario 3 con dos drones.

### Comparación de resultados

Finalmente se presentan las mejoras relativas de cada algoritmo en relación con los resultados obtenidos por el algoritmo scan en cada escenario, en el cuadro 5.5 utilizando un drone y en el cuadro 5.6 ejecutado con una flota de dos drones.

Se puede afirmar que los algoritmos inteligentes diseñados presentan una considerable mejora de la eficiencia de la exploración frente a algoritmos no inteligentes (como el scan) cuando se tienen puntos del predio con diferentes prioridades.

En cuanto a si es mejor utilizar el algoritmo de trayectoria con o sin la restricción de las columnas la respuesta esta condicionada por el escenario. En el escenario 2 obtiene mejores resultados el algoritmo de trayectoria sin restricción mientras que para el escenario 3 la mejor solución es utili-

zar el algoritmo de trayectoria con restricción. Por este motivo se decidió implementar la restricción de forma de que sea configurable al iniciar la ejecución del sistema y que el usuario decida si utilizarla o no dependiendo de las características del escenario en el que se ejecute el sistema de vigilancia.

| Escenarios | Algoritmos | |
|---|---|---|
| | Trayectoria | Trayectoria con restricción |
| Escenario 1 | **5,5 %** | 4,3 % |
| Escenario 2 | **38,1 %** | 35,1 % |
| Escenario 3 | 14,6 % | **38,6 %** |

Cuadro 5.5: Valores relativos de beneficio con respecto al algoritmo scan para un drone.

| Escenarios | Algoritmos | |
|---|---|---|
| | Trayectoria | Trayectoria con restricción |
| Escenario 1 | **7,2 %** | 5,2 % |
| Escenario 2 | **26,1 %** | 14,4 % |
| Escenario 3 | 16,8 % | **38,3 %** |

Cuadro 5.6: Valores relativos de beneficio con respecto al algoritmo scan para una flota de drones.

## 5.2. Detección de intrusos

Esta sección presenta un análisis de la solución implementada para la detección de intrusos. En la sección 5.2.1 se describen las metodologías de evaluación, la sección 5.2.2 explica en detalle los videos seleccionados para realizar las pruebas, mientras que en la sección 5.2.3 se exponen los resultados obtenidos.

### 5.2.1. Metodología de evaluación

Para evaluar el módulo de procesamiento de imágenes se realizaron pruebas en tres videos donde se observan intrusos en distintos escenarios. Las características de cada video de prueba se describen en detalle en la sección 5.2.2. Sobre cada video se realizan tres pruebas cambiando la sensibilidad de la detección en cada prueba. La sensibilidad de detección implica cuantas detecciones seguidas son necesarias para considerar a un objeto como un intruso. Las configuraciones de sensibilidad escogidas para realizar estas pruebas son: una detección (alta), tres detecciones (moderada) y cinco detecciones (baja). Para medir la eficiencia y eficacia de la detección se utilizan las medidas precisión (Precision), exhaustividad (Recall) y medida F (F-Score) [43]. Para ello, primero es necesario definir los resultados: verdadero positivo (VP), falso positivo (FP), verdadero negativo (VN) y falso negativo (FN) (ver figura 5.17) [43].

- Verdadero positivo: un resultado que debería ser positivo es etiquetado como positivo. Es decir, en un frame que contiene un intruso se detecta el intruso correctamente.

- Falso positivo: un resultado que debería ser negativo es etiquetado como positivo. En un frame se detecta un intruso donde no lo hay, ya sea porque el frame no contiene un intruso o porque la detección se produce por otro objeto que no es el intruso.

- Verdadero negativo: un resultado que debería ser negativo es etiquetado como negativo. Esto es, en un frame que no contiene un intruso no hay detección.
- Falso negativo: un resultado que debería ser positivo es etiquetado como negativo. En un frame que contiene un intruso no hay detección.

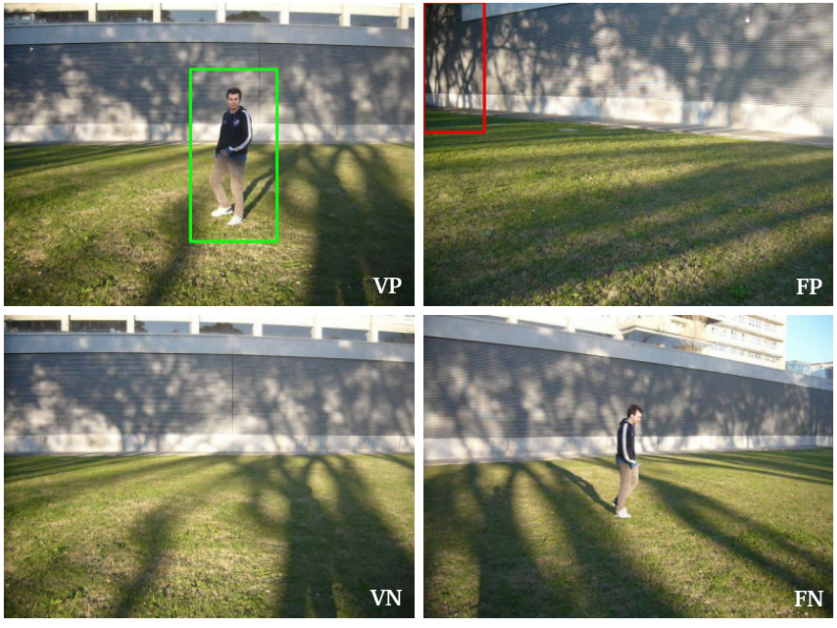


Figura 5.17: Diferentes resultados en la detección de intrusos.

La precisión indica la proporción de resultados positivos que el sistema detectó correctamente. Esto es, la cantidad de detecciones de intrusos que se hicieron correctamente, sobre todas las detecciones de intrusos que se realizaron (sean correctas o no). La precisión se calcula de la siguiente manera:

$$Precision = \frac{Verdaderos\ Positivos}{Verdaderos\ Positivos + Falsos\ Positivos}$$

La exhaustividad es la habilidad de encontrar todos los resultados positivos. En el sistema de detección, es la proporción entre todas las detecciones de intrusos correctas, sobre todas las detecciones de intrusos correctas mas los frames donde se deberían haber producido detecciones pero no lo hicieron. La exhaustividad se calcula de la siguiente forma:

$$Recall = \frac{Verdaderos\ Positivos}{Verdaderos\ Positivo + Falsos\ Negativos}$$

Se busca tener el mayor porcentaje de precisión y exhaustividad posible. Tener una alta precesión implica que el sistema cuando detecta un intruso lo hace correctamente, es decir, las detecciones se producen por personas y no por otros objetos del terreno. Una alta exhaustividad implica que cuando hay un intruso es correctamente detectado por el sistema. Si se tiene alta precisión pero baja exhaustividad se tiene un sistema que cuando detecta un intruso lo hace correctamente, pero existe una alta cantidad de frames donde existen intrusos y no se produjo una detección. Una alta exhaustividad pero baja precisión implica que el sistema produce muchas detecciones, detectando siempre los intrusos pero también otros objetos del terreno que no son personas. Por lo tanto, para medir la eficacia y eficiencia del sistema se utiliza la medida f. La medida f es la media armónica de la precisión y la exhaustividad, y se calcula de la siguiente forma:

$$F\ Score = 2 * \frac{precision * recall}{precision + recall}$$

Para visualizar mejor los resultados y verificar si el sistema de detección confunde resultados positivos con negativos o resultados negativos con positivos, se crea a partir del conjunto de resultados obtenidos y el conjunto de resultados esperados, la matriz de confusión [44] para cada video y cada nivel de sensibilidad. Se decidió normalizar la matriz para visualizar que proporción de resultados obtenidos es correctamente predecido como positivo o negativo según corresponda.

### 5.2.2. Videos de prueba

Se utilizaron tres videos para realizar las pruebas del sistema de detección donde se busco tener diferentes segundos planos, diferentes movimientos de los intrusos y frames donde se observe un intruso y frames donde no.

El primer video denominado “video A” es un video de dos minutos en el que un intruso entra y sale del cuadro, obteniendo así frames del video donde se observa un intruso y frames donde solo se ve el predio. Cuando se observa un intruso en el video, éste se mueve de perfil por momentos y por momentos se queda estático de frente a la cámara. El video comienza en una posición del predio donde el segundo plano es un muro plano, luego se detecta un intruso y se lo sigue hacia otra posición del predio donde se puede ver un segundo plano con árboles y otros objetos.

El segundo video que se denomina “video B” tiene una duración de 30 segundos. Se grabó en otro predio con el fin de obtener distintos segundos planos para probar el sistema de detección sobre ellos. En este video se ven muchos más objetos sobre el fondo como árboles, un arco y postes. Se sigue al intruso en un desplazamiento en el cual se puede ver la variedad de objetos en el fondo.

En el último video denominado “video C” un intruso entra en cuadro de perfil y posteriormente cambia su posición para pararse de frente a la cámara, se mantiene un tiempo corto en esa posición y luego corre para salir de cuadro. Cuando el intruso deja el cuadro, el video continúa con el predio vacío por unos 15 segundos. El total del video es de 40 segundos.

### 5.2.3. Resultados

En el cuadro 5.7 se ven los resultados del procesamiento del video A. La cantidad de verdaderos negativos se debe a que en gran parte del video no se observan intrusos y solo se observa el predio. Se tiene una baja cantidad de falsos positivos dado que en la mayor parte del video el fondo es un muro plano por lo que el sistema no detecta personas incorrectamente. Los falsos negativos se dan en mayor medida cuando el intruso se encuentra de perfil a la cámara y uno de sus brazos o ambos no son completamente visibles. Las tres configuraciones obtienen una alta precisión, sin embargo la exhaustividad es baja para la configuración de sensibilidad alta y continúa disminuyendo a medida que se aumenta la cantidad de detecciones a considerar como válidas, es decir, disminuye la sensibilidad. Esto se debe a que al ser más estricto en considerar una detección como valida, existen frames donde se encuentra el intruso y no se producen detecciones, lo cual genera una mayor cantidad de falsos negativos.

| *Sensibilidad* | *FP* | *FN* | *VP* | *VN* | *Precision* | *Recall* | *F-Score* |
|---|---|---|---|---|---|---|---|
| Alta | 13 | 304 | 1003 | 555 | 98,7 % | 76,7 % | 86,3 % |
| Moderada | 1 | 446 | 866 | 562 | 99,8 % | 66 % | 79,5 % |
| Baja | 0 | 536 | 775 | 564 | 100 % | 59,1 % | 74,3 % |

Cuadro 5.7: Resultados del video A.

La figura 5.18 muestra las matrices de confusión del video A para las distintas configuraciones. Se puede ver en gran medida como el sistema de detección no confunde negativos con positivos, es decir, no se detectan intrusos donde no los hay. Sin embargo, el sistema confunde positivos con negativos en una cantidad considerable, esto significa que en frames donde existen intrusos el sistema falla en detectarlos. Estos errores aumentan a medida que aumenta el número de detecciones necesarias para considerar una detección como válida ya que el sistema de detección es mas estricto para detectar intrusos.

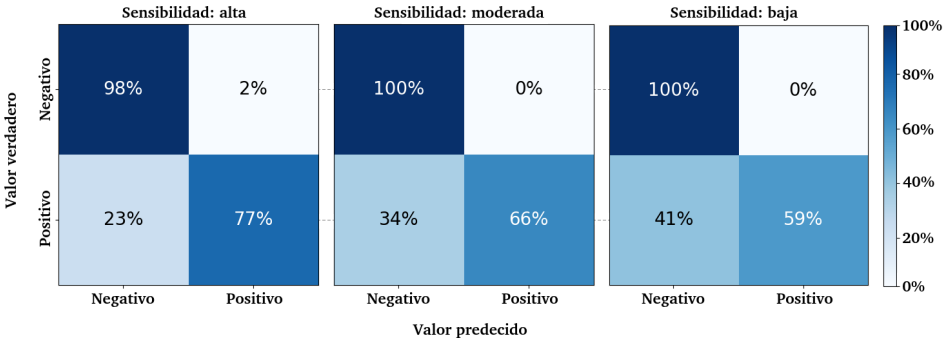


Figura 5.18: Matrices de confusión del video A.

El cuadro 5.8 muestra los resultados de aplicar el sistema de detección sobre el video B. En el segundo plano de este video existe una variabilidad de objetos a los que el sistema confunde con personas, por lo que la cantidad de falsos positivos es mayor al video anterior. Otra característica que se diferencia del caso anterior es que en este video el intruso no se posiciona de perfil a la cámara por muchos frames por lo que la cantidad de falsos negativos en proporción al video anterior es menor, por lo que se obtiene una buena medida de exhaustividad. Al aumentar el número de detecciones para considerar válida una detección se consigue una disminución de falsos positivos y por lo tanto un aumento de la precisión. Sin embargo, también se obtiene un aumento de falsos negativos y por lo tanto baja la exhaustividad.

| *Sensibilidad* | $FP$ | $FN$ | $VP$ | $VN$ | *Precision* | *Recall* | *F-Score* |
|---|---|---|---|---|---|---|---|
| Alta | 31 | 78 | 366 | 181 | 92,1 % | 82,4 % | 87 % |
| Moderada | 13 | 126 | 329 | 188 | 96,1 % | 72,3 % | 82,5 % |
| Baja | 7 | 152 | 305 | 192 | 97,8 % | 66,7 % | 79,3 % |

Cuadro 5.8: Resultados del video B.

La figura 5.19 muestra las matrices de confusión para el video B. Se puede ver como al obtener mayor cantidad de falsos positivos hay un considerable porcentaje de negativos que dan como resultado positivo. Al igual que el video anterior, a medida que baja la sensibilidad, aumenta la cantidad de negativos reconocidos como negativos pero disminuye la cantidad de positivos reconocidos como positivos.

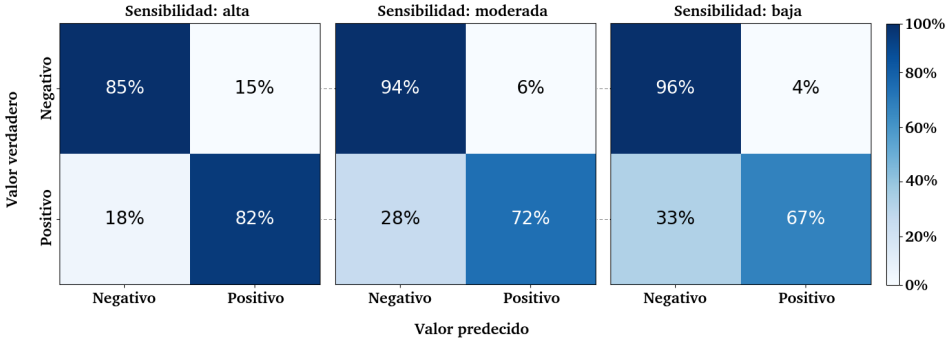


Figura 5.19: Matrices de confusión del video B.

En el cuadro 5.9 se muestran los resultados del procesamiento del video C. La característica mas interesante es que muestra altos porcentajes de precisión y bajo porcentaje de recall independientemente de la configuración. La alta precisión es resultado de la baja cantidad de falsos positivos, esto se debe a que no hay otros objetos en el video que puedan ser confundidos por el sistema como personas. El principal motivo de la baja exhaustividad es que gran parte del tiempo que el intruso aparece en el video lo hace de perfil ocultando uno de sus brazos. El sistema demuestra tener problemas para detectar personas cuando alguno o ambos brazos del intruso no se ven claramente, independientemente del nivel de sensibilidad. Este problema aumenta cuando se aumenta la cantidad de detecciones necesarias para ser considerada como válida.

| *Sensibilidad* | *FP* | *FN* | *VP* | *VN* | *Precision* | *Recall* | *F-Score* |
|---|---|---|---|---|---|---|---|
| Alta | 7 | 68 | 261 | 309 | 97,3 % | 79,3 % | 87,4 % |
| Moderada | 0 | 89 | 244 | 312 | 100 % | 73,3 % | 84,6 % |
| Baja | 0 | 98 | 235 | 312 | 100 % | 70,6 % | 82,7 % |

Cuadro 5.9: Resultados del video C.

La matriz de confusión del video C (ver figura 5.20) indica que el sistema de detección identifica correctamente los casos negativos dada la baja o nula cantidad de falsos positivos, pero al igual que en los casos anteriores, confunde en gran proporción y a medida que aumenta la sensibilidad los casos positivos con casos negativos, es decir, en frames donde deberían haber detecciones, no las hay, debido principalmente a que el intruso se muestra de perfil la mayor parte del video sin mostrar ambos brazos a la vez.

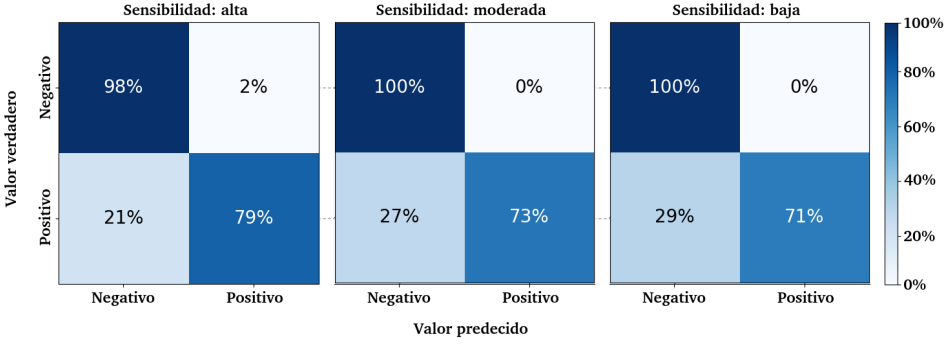


Figura 5.20: Matrices de confusión del video C.

Para el mejor funcionamiento del sistema de vigilancia se busca una alta precisión y una alta exhaustividad, en consecuencia una alta medida f. Una baja precisión provocaría que un drone detecte y persiga objetos del predio que no son intrusos, lo que conllevaría a que el drone se mantenga vigilando este objeto suponiendo que es un intruso y por lo tanto, descuidando el resto del predio innecesariamente. Una baja exhaustividad provocaría que un drone no detecte intrusos, provocando una baja en la

confiabilidad del sistema de vigilancia. En el cuadro 5.10 se puede ver que el sistema de detección obtiene en todos los casos una alta precisión, es decir, la mayoría de las veces que se produce una detección se debe a la presencia de un intruso. Por otro lado la exhaustividad general del sistema es buena excepto en casos donde el intruso se muestra de perfil a la cámara durante un tiempo prolongado. Se puede concluir a través del cuadro que para todos los videos de prueba la mejor medida f es obtenida por la configuración de sensibilidad alta, por lo que se eligió esta configuración por defecto para el sistema de detección.

| *Videos* | *Sensibilidad* | *Precision* | *Recall* | *F-Score* |
|---|---|---|---|---|
| Video A | Alta | 98,7 % | 76,7 % | 86,3 % |
| | Moderada | 99,8 % | 66 % | 79,5 % |
| | Baja | 100 % | 59,1 % | 74,3 % |
| Video B | Alta | 92,1 % | 82,4 % | 87 % |
| | Moderada | 96,1 % | 72,3 % | 82,5 % |
| | Baja | 97,8 % | 66,7 % | 79,3 % |
| Video C | Alta | 97,3 % | 79,3 % | 87,4 % |
| | Moderada | 100 % | 73,3 % | 84,6 % |
| | Baja | 100 % | 70,6 % | 82,7 % |

Cuadro 5.10: Resultados de todos los videos.

# Capítulo 6

# Conclusiones y trabajo futuro

En este capítulo se presenta las conclusiones obtenidas por los autores a partir del trabajo realizado y los potenciales cambios o mejoras en un trabajo futuro. En la sección 6.1 se mencionan los objetivos logrados y se exponen las conclusiones. En la sección 6.2 se describen nuevas líneas de trabajo.

## 6.1. Conclusiones

La implementación de un sistema de drones autónomos para un sistema de vigilancia es una tarea compleja. Este proyecto continua la línea de trabajo propuesta en el proyecto "Planificación e instrumentación de vuelo de una flotilla de drones", para ello se comenzó verificando la viabilidad de embeber el sistema de vigilancia en los drones, concretamente en drones Parrot Bebop 2. Se propuso una solución a los problemas planteados, en primer lugar la navegación autónoma la cual involucra el calculo de las

rutas, carga de batería y la coordinación entre drones, en segundo lugar, la comunicación y coordinación entre agentes de la flota y en tercer lugar la detección y seguimiento de intrusos.

El enfoque de embeber el sistema de vigilancia en la arquitectura de los drones Parrot Bebop 2 resultó inviable debido a que no se puede instalar la biblioteca OpenCV dentro del sistema operativo de los mismos, principalmente porque los drones cuentan con capacidad limitada de espacio de memoria. Se buscaron alternativas, como el uso de hardware externo, pero luego de un análisis de las ventajas y desventajas, se entendió que usar hardware externo no era una opción factible por lo que finalmente se descartó este objetivo del proyecto. Como alternativa se implementó el sistema para ser ejecutado en un equipo externo el cual se comunica con los drones mediante su API.

En términos generales el simulador Sphinx, el lenguaje Python, y las bibliotecas PyParrot y OpenCV, son herramientas que funcionan correctamente y ofrecen las suficientes funcionalidades para implementar un sistema de este tipo. Aún así, surgieron diversos problemas para los cuales se les encontró una solución alternativa.

En lo que refiere a la navegación autónoma, se considera que el sistema diseñado presenta una solución viable. Sobre la solución realizada en este trabajo, las pruebas de vuelo mostraron el correcto seguimiento del plan de vuelo y un buen cubrimiento del predio por parte de la flota de drones donde las zonas de mayor prioridad son mayormente visitadas.

Sobre la colaboración entre agentes de la flota, el equipo implementó un sistema de comunicación robusto mediante el protocolo UDP. El desempeño del sistema de comunicación fue sobresaliente. Los drones permanecen siempre coordinados y se envían correctamente su información entre si, de manera rápida y sin pérdida de mensajes.

Finalmente acerca del sistema de detección y seguimiento que fue implementado en este proyecto, se puede concluir que existieron ciertos inconvenientes, por ejemplo una cantidad baja pero considerable de falsos negativos y falsos positivos. De todos modos cabe destacar que se obtuvieron resultados aceptables con valores de f-score mayores a 85 % en la mayoría de los casos.

Para concluir, el trabajo se encuentra abierto a varias lineas de investigación y mejora, pero este proyecto logró un acercamiento más profundo en la investigación de una solución eficiente para un sistema de vigilancia con una flota de drones autónomos en un predio conocido.

## 6.2. Trabajo futuro

Existen varias líneas de investigación que pueden seguirse para mejorar el sistema de vigilancia planteado en este proyecto de grado, las de mayor interés son las descriptas a continuación.

Exportar el sistema de vigilancia a otros tipos de drones que tengan mayor autonomía de vuelo. La autonomía de vuelo de los drones con los que se contó para este proyecto es aproximadamente cinco veces menor que el tiempo de carga, lo cual implica que para que en todo momento se tenga al menos un drone activo explorando se necesita una flota de al menos cinco drones. Además para hacer valer todo el potencial del sistema es de interés que en todo momento se tenga una flota de al menos dos drones explorando el predio. Por lo que exportar el sistema de vigilancia a una flota de drones que tengan mayor autonomía de vuelo y/o menor tiempo de carga permitiría que con una menor cantidad de drones, se pueda lograr que siempre haya al menos uno (o dos) drones vigilando.

Mejorar el sistema de detección para detectar personas en posiciones anormales y otro tipo de objetos. Si bien se demostró que la detección de personas funciona de manera correcta, el sistema presenta dificultades con algunas posiciones de la persona. Una posible línea de investigación es entrenar un mejor clasificador utilizando HOG, para mejorar la detección de personas. También puede ser útil contar con un sistema que detecte otros objetos como árboles u otros posibles obstáculos dentro del predio con el objetivo de hacer el sistema de vigilancia más robusto.

Enviar mensajes a un sistema central cuando se detecte un intruso. El sistema actual no cuenta con un módulo de notificación a un agente externo a la flota cuando se detecta un intruso. Se podría crear un nuevo módulo que envíe mensajes a un agente externo y que los mismos contengan imágenes del intruso detectado así como también el tiempo en el que ocurrió la detección y el lugar exacto del predio donde ocurrió la detección.

# Glosario

**UAV:** (Unmanned Aerial Vehicle) Son aeronaves que vuelan sin tripulación, comúnmente denominados dron.

**VANT:** es un vehículo sin tripulación reutilizable, capaz de mantener de manera autónoma un nivel de vuelo controlado y sostenido

**Ubuntu:** es una distribución Linux que ofrece un sistema operativo predominantemente enfocado a ordenadores aunque también proporciona soporte para servidores

**GNU/Linux:** es un sistema operativo libre tipo Unix, multiplataforma, multiusuario y multitarea

**GNSS:** (Global Navigation Satellite System) es el acrónimo que se refiere al conjunto de tecnologías de sistemas de navegación por satélite que proveen de posicionamiento geo-espacial con cobertura global de manera autónoma

**GPS:** (Global Positioning System) sistema que permite determinar posiciones y velocidades tanto en la Tierra como en el espacio.

**GLONASS:** (Global'naya Navigatsionnaya Sputnikovaya Sistema) es un GNSS desarrollado por la Unión Soviética.

**ARM:** es una arquitectura RISC de 32 bits.

**HMM:** (Hidden Markov Model) es un modelo estadístico en el que se asume que el sistema a modelar es un proceso de Márkov de parámetros desconocidos.

**SBC:** (Single Board Computer) es una computadora completa de un solo circuito.

# Bibliografía


[1] Philip Kaplan. *Naval Aviation in the Second World War.* Pen and Sword, 2013.

[2] Ben Nassi, Asaf Shabtai, Ryusuke Masuoka, and Yuval Elovici. Security and privacy in the age of drones: Threats, challenges, solution mechanisms, and scientific gaps. *Cornell, University of New York, Ithaca*, 2019.

[3] Hazim Shakhatreh, Ahmad Sawalmeh, Ala Al-Fuqaha, Zuochao Dou, Eyad Almaita, Issa Khalil, Noor Shamsiah Othman, Abdallah Khreishah, and Mohsen Guizani. Unmanned aerial vehicles (UAVs): A survey on civil applications and key research challenges. *IEEE Access*, 7:48572–48634, 2019.

[4] James Kurose and Keith Ross. *Computer Networking: A Top-Down Approach 5 Edition.* Addison Wesle, 2007.

[5] Alberto Fernandez. *Mastering OpenCV with Python.* Pack Publishing, 2019.

[6] U.M Prakash and V.G Thamaraiselvi. Detecting and tracking of multiple moving objects for intelligent video surveillance systems. *Second International Conference on Current Trends In Engineering and Technology*, pages 253–257, 2014.

[7] Kimon Valavanis and George Vachtsevanos. *Handbook of Unmanned Aerial Vehicles.* Springer, 2015.

[8] Ke Shang, Stephen Karungaru, Zuren Feng, Liangjun Ke, and Kenji Teradao. A ga-aco hybrid algorithm for the multi-UAV mission planning problem. *14th International Symposium on Communications and Information Technologies*, pages 243–248, 2014.

[9] Esten Grøtli and Tor Johansen. Path planning for UAVs under communication constraints using splat! and MILP. *Journal of Intelligent Robotic Systems*, 2012.

[10] Kenan Cole and Adam Wickenheiser. Reactive trajectory generation for multiple vehicles in unknown environments with wind disturbances. *IEEE Transactions on Robotics*, 34:1333–1348, 2018.

[11] Hamid Shiri, Jihong Park, and Mehdi Bennis. Massive autonomous UAV path planning: A neural network based mean-field game theoretic approach. *Cornell, University of New York, Ithaca*, 2019.

[12] Ilker Bekmezci, Murat Ermis, and Sezgin Kaplan. Connected multi UAV task planning for flying ad hoc networks. *IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom)*, pages 28–32, 2014.

[13] Samer Hanna, Enes Krijestorac, Han Yan, and Danijela Cabric. UAV swarms as amplify-and-forward mimo relays. *IEEE 20th International Workshop on Signal Processing Advances in Wireless Communications*, pages 1–5, 2019.

[14] Nagesh Chandrakanth, Hemanth Rao, and Anjan Koundinya. Secure handshake mechanism for autonomous flying agents using robust cryptosystem. *2nd International Conference on Computational Systems and Information Technology for Sustainable Solution*, pages 1–5, 2017.

[15] Tongyu Dai, Xinggong Zhang, Yihang Zhang, and Zongming Guo. Statistical learning based congestion control for real-time video communication. *Cornell, University of New York, Ithaca*, 2019.

[16] Christian Raffelsberger, Raheeb Muzaffar, and Christian Bettstetter. A performance evaluation tool for drone communications in 4g cellular networks. *16th International Symposium on Wireless Communication Systems*, pages 218–221, 2019.

[17] Seonghyun Kim, Wonjae Lee, Young-su Park, Hyun-Woo Lee, and Yong-Tae Lee. Forest fire monitoring system based on aerial image. *IEEE 3rd International Conference on Information and Communication Technologies for Disaster Management*, pages 1–6, 2016.

[18] Mayank Mittal, Rohit Mohan, Wolfram Burgard, and Abhinav Valada. Vision-based autonomous UAV navigation and landing for urban search and rescue. *Cornell, University of New York, Ithaca*, 2019.

[19] Santiago Díaz and Bruno Garate. Planificación e instrumentación de vuelo de una flotilla de drones. Proyecto de grado, Facultad de Ingeniería, UDELAR, 2018.

[20] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. *IEEE Conference on Computer Vision and Pattern Recognition*, pages 886–893, 2005.

[21] Luis Arreola, Gesem Gudiño, and Gerardo Flores. Object recognition and tracking using haar-like features cascade classifiers: Application to a quad-rotor UAV. *DeepAI*, 2019.

[22] Mohamed Afifi, Yara Ali, Karim Amer, Mahmoud Shaker, and Mohamed ElHelw. Robust real-time pedestrian detection in aerial imagery on jetson TX2. *Cornell, University of New York, Ithaca*, 2019.

[23] Jaeseung Byun, Karan Jain, Siddharth Nair, Haoyun Xu, and Jiaming Zha. Predictive control for chasing a ground vehicle using a UAV. *Cornell, University of New York, Ithaca*, 2019.

[24] Especificaciones de ARM cortex-A9. `https://developer.arm.com/ip-products/processors/cortex-a/cortex-a9`. Accedido: 06-2019.

[25] Joseph Yiu. *The Definitive Guide to the ARM Cortex-M3*. Elsevier, 2010.

[26] Warren Gay. *Raspberry Pi Hardware Reference*. Apress, 2014.

[27] Matt Richardson and Shawn Wallace. *Raspberry Pi User Guide*. Maker Media, 2016.

[28] William Harrington. *Learning Raspbian*. Packt Publishing, 2015.

[29] Brian Keringhan and Dennis Ritchie. *The C Programming Language*. Prentice Hall, 2012.

[30] Dennis Ritchie Brian Kernighan. *El lenguaje de programacion C*. Pearson Educación, 1991.

[31] Guido Van Rossum. *Python Reference Manual*. CWI (Centre for Mathematics and Computer Science), 1995.

[32] Luciano Ramalho. *Fluent Python*. O'Reilly Media, Inc., 2015.

[33] Monico Manuel Ponce Gonzalez. Vision por computadora para UAS. Proyecto de grado, Escuela Técnica Superior de Ingenieros Aeronáuticos, Universidad Politécnica de Madrid, 2012.

[34] Gregorio Ambrosio Vicente Arévalo, Javier González. La librería de visión artificial opencv aplicación a la docencia e investigación. Technical report, 2004.

[35] Atif Memon. *Advances in Computers*. Elsevier, 2012.

[36] ROS enhancement proposal, 103. `https://www.ros.org/reps/rep-0103.html`. Accedido: 08-2019.

[37] Siva Murthy and Shukla Manoj. *Ad Hoc Wireless Networks: Architectures and Protocols*. Prentice Hall, 2004.

[38] Jorge Luis Ortega. *Patterns for Parallel Software Design*. Wiley, 2010.

[39] Maarten van Steen and Andrew Tanenbaum. *Distributed Systems*. CreateSpace Independent Publishing Platform, 2017.

[40] Subrina Akter Muhammed Patwary, Shahnaj Parvin. Significant hog-histogram of oriented gradient feature. *International Journal of Computer Applications*, 2015.

[41] Junseong Lee, Jinsuand Bang and Seong-II Yang. Object detection with sliding window in images including multiple similar objects. *International Conference on Information and Communication Technology Convergence*, pages 803–806, 2017.

[42] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. *Computer Society Conference on Computer Vision and Pattern Recognition*, 1:886–893, 2005.

[43] David Powers. Evaluation: From precision, recall and f-factor to ROC, informedness, markedness correlation. *Maching Learning Technology*, 2008.

[44] Tom Fawcett. An introduction to ROC analysis. *Elsevier*, 2006.