UNIVERSIDAD
DE LA REPUBLICA
URUGUAY

Agentes cooperativos para vuelo inteligente de vehículos autónomos

Santiago Behak

Giovani Rondán

Martín Zanetti

Programa de Grado en Ingeniería en Computación
Facultad de Ingeniería
Universidad de la República

Montevideo – Uruguay
Noviembre de 2018

UNIVERSIDAD
DE LA REPUBLICA
URUGUAY

Agentes cooperativos para vuelo inteligente de vehículos autónomos

Santiago Behak
Giovani Rondán
Martín Zanetti

Informe de Proyecto de Grado presentado al Tribunal Evaluador como requisito de graduación de la carrera Ingeniería en Computación.

Directores:
Prof. Santiago Iturriaga
Prof. Sergio Nesmachnow

Montevideo – Uruguay
Noviembre de 2018

Behak, Santiago - Rondán, Giovani - Zanetti, Martín

Agentes cooperativos para vuelo inteligente de vehículos autónomos / Montevideo: Universidad de la República, Facultad de Ingeniería, 2018.

VII, 70 p. 29, 7cm.

Directores:

Santiago Iturriaga

Sergio Nesmachnow

Tesis de Grado – Universidad de la República, Programa en Ingeniería en Computación, 2018.

Referencias bibliográficas: p. 68 – 70.

1. Vuelo autónomo, 2. Vehículo aéreo no tripulado, 3. Cooperación. I. Iturriaga, Santiago, Nesmachnow, Sergio, . II. Universidad de la República, Programa de Grado en Ingeniería en Computación. III. Título.

INTEGRANTES DEL TRIBUNAL DE DEFENSA DE TESIS

Prof. Federico Rivero

Prof. Ernesto Dufrechu

Prof. Fernando Fernández

Montevideo – Uruguay
Noviembre de 2018

RESUMEN

El presente trabajo constituye el estudio de algoritmos colaborativos de exploración y misiones de vigilancia sobre puntos predefinidos mediante una flota de vehículos aéreos no tripulados capaces de operar de forma autónoma sobre escenarios con diversas características. La implementación de la solución al problema se basa en el paradigma de la computación basada en agentes. La misma está constituida por una máquina de estados encargada de resolver las distintas etapas por las que atraviesan los drones cuando ejecutan la misión, teniendo como objetivos el máximo cubrimiento de la zona a explorar en un tiempo óptimo de acuerdo a la duración de la batería de los drones, manteniendo vigilancia sobre puntos de interés previamente definidos e intentando mantener comunicación constante entre los drones de la flota.

Se utilizan drones Parrot Bebop 2 para la investigación. Si bien la solución tiene en cuenta cuestiones específicas de las características de este modelo de dron, la misma es adaptable a cualquier otro.

Con el objetivo de validar el algoritmo implementado para la solución del problema se realizan pruebas sobre distintos escenarios de exploración de áreas abiertas mediante simulaciones computacionales utilizando el simulador Sphinx.

A su vez, se realiza la instalación y ejecución del algoritmo directamente sobre los drones de modo de comprobar su correcto funcionamiento.

El trabajo realizado contribuye a presentar un desarrollo aplicable en la práctica que le permite a una flota de drones poder ejecutar tareas de exploración y vigilancia de puntos predefinidos de forma colaborativa y autónoma, maximizando el cubrimiento del territorio así como también optimizando la vigilancia sobre los puntos de interés elegidos.

Palabras claves:
Vuelo autónomo, Vehículo aéreo no tripulado, Cooperación.

Tabla de contenidos

Capítulo 1

Introducción

Los vehículos aéreos no tripulados (Unmanned Aerial Vehicle, UAV) son una herramienta cada vez más usada en campos muy diversos. Dentro de las principales aplicaciones para el uso de drones no tripulados, se encuentran las desarrolladas para tareas agrícolas [15], tanto en fertilización de campos como en mediciones, análisis del estado del suelo, de los cultivos, entre otros. También se encuentran aplicaciones muy controvertidas, pero que cada vez son más utilizadas e importantes, como son las aplicaciones militares [33]. Los drones no tripulados en aplicaciones militares, son utilizados en misiones de reconocimiento de zonas peligrosas, vigilancia [1] y apoyo en tareas de rescate [6], incluyendo también misiones de ataque de objetivos específicos, para lo cual se equipa a los drones con armamento.

En la actualidad la mayoría de las aplicaciones existentes se basan en controlar drones que no vuelan de forma completamente autónoma. En general los drones requieren que haya un piloto controlándolos directamente, o que al menos realice tareas de seguimiento. La falta de avances en relación al vuelo autónomo impide que se puedan desarrollar aplicaciones que hagan uso de flotas compuestas de múltiples drones, ya que a un operador humano le resulta muy difícil controlar varios patrones de vuelo simultáneamente. El proyecto que se describe en este informe surge en el marco de una serie de estudios realizados por el Centro de Cálculo del Instituto de Computación de la Facultad de Ingeniería, perteneciente a la Universidad de la República (Uruguay), que tienen por objetivo estudiar y desarrollar la tecnología de los UAV.

Uno de los principales factores que limita el vuelo de drones, ya sea autónomo o pilotado por una persona, es su autonomía de vuelo. La autonomía de vuelo

de un dron es el tiempo máximo que puede volar con una carga completa de su batería. En la actualidad, la escasa autonomía de vuelo que presentan los drones es la principal limitante a la hora de realizar vuelos, ya que la autonomía no supera los 30 minutos para la mayoría de los modelos de drones comerciales. Además se debe tener en cuenta que realizar una recarga completa de la batería del dron suele ser un proceso lento que por lo general toma aproximadamente una hora. Los dos factores mencionados, tanto la escasa autonomía de vuelo como el extenso tiempo que lleva recargar la batería del dron, limitan enormemente el volumen de tareas que puede realizar un único dron y hace que sea imposible realizar trabajos que requieran una presencia constante, como por ejemplo la vigilancia de un terreno. Estos problemas se pueden mitigar si en lugar de emplear un único dron, se utiliza una flota compuesta de múltiples drones. Una flota mejora la eficiencia del sistema.Por estas razones es que resulta interesante estudiar el patrón de vuelo de una flota de drones y evaluar posibles estrategias, para determinar cuales son más eficientes.
Este trabajo busca implementar un sistema que permita controlar una flota de drones de forma completamente autónoma, mientras se realiza una tarea concreta que combina dos de los objetivos más comunes en las aplicaciones de los UAV, las cuales son la exploración y la vigilancia. En particular, el problema a resolver cuenta con las siguientes características:

- Los drones que componen la flota deben manejarse de forma completamente autónoma, es decir que no pueden ser pilotados por una persona ni tampoco pueden ser controlados desde un servidor central.
- Los drones deben poder comunicarse entre sí para intercambiar información.
- se debe maximizar la cantidad de área explorada en la menor cantidad de tiempo posible.
- Existen puntos de interés en la zona a explorar (Points Of Interest, POI) que deben tener prioridad sobre el resto del terreno y deben ser vigilados con regularidad.

Para implementar el sistema, en primera instancia se crea un ambiente de desarrollo que permita programar drones individuales, de forma tal que los drones puedan ser completamente autónomos. Esto se debe a que muchos modelos comerciales de drones no cuentan con este tipo de ambientes o solo permiten desarrollar programas que controlan al dron a distancia. Mas adelante se

implementan los sistemas necesarios para realizar funciones básicas como la comunicación entre los drones, resolver su ubicación en el espacio, establecer un control de altitud de vuelo, entre otras. Después se combinan estos sistemas para implementar el producto final, basado en máquinas de estados, planteándose algunas variantes que aplican diferentes tipos de algoritmos. Por último se realiza un análisis estadístico, por medio de estudiar el desempeño en simulaciones de las diferentes variantes del programa final implementado, tomando en cuenta varias variables como la presencia de obstáculos en la zona a explorar, el tamaño de la zona a explorar y diferentes variaciones de requerimientos en los puntos de interés.
Los resultados en el análisis experimental muestran que el sistema desarrollado resulta efectivo para cumplir con los objetivos del problema planteado. Se destaca la capacidad del sistema para cumplir con las metas de exploración sobre un territorio, además del bajo tiempo de respuesta en el que los puntos de interés son vigilados. También se consigue que el algoritmo implementado sea capaz de reaccionar a la información recabada en tiempo real, lo que permite tener en cuenta los factores que varían durante la ejecución del sistema, como son las zonas del territorio ya exploradas y los puntos de interés que requieren ser visitados.
Las principales contribuciones realizadas en el marco de este proyecto son:

- La implementación de un ambiente de desarrollo, basado en el lenguaje de programación Python, que permite desarrollar con facilidad programas para arquitecturas ARM, muy comunes en diversos modelos de drones.
- El desarrollo de un sistema escalable, que permite cumplir simultáneamente con objetivos de exploración y vigilancia, además de poder ser utilizado en una flota de drones de cualquier tamaño y adaptarse para usarse en una gran variedad de modelos.
- El estudio comparativo de diversos algoritmos por medio de un análisis estadístico de resultados de simulaciones.

El informe se organiza de la siguiente manera. El capítulo 2 describe el problema a resolver y presenta antecedentes que pretenden resolver problemas similares. En el capítulo 3 se presentan las características de hardware y software de los drones Parrot Bebop 2 y se discuten soluciones a varios de los problemas para implementar un sistema autónomo sobre ellos como lo son la geo-localización y la conectividad a la red, además El capítulo 4 detalla las

características de las herramientas utilizadas para el desarrollo del proyecto y se describe la máquina de estados implementada para la solución al problema. 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 exploración con drones autónomos

En este capítulo se presenta el problema a investigar en el proyecto, se realiza una descripción de los conceptos generales tenidos en cuenta y se concluye con el estudio de investigaciones realizadas por otros autores que se emplean como referencia para la implementación de la solución.

2.1. Mecanismos de exploración para una flota de vehículos no tripulados

Esta sección describe el problema abordado en el proyecto, donde se considera la exploración de un territorio utilizando drones que operan de forma autónoma, estableciendo un canal de comunicación que permita la transmisión de datos entre los miembros de la flota de exploración.

2.1.1. Exploración de un territorio priorizando objetivos por una flota autónoma de drones

El problema a resolver consiste en explorar de manera eficiente un territorio por parte de una flota de drones de forma autónoma, donde además se priorizan distintos puntos de interés (Points Of Interest, POI) del territorio los cuáles serán recorridos regularmente. Explorar de forma eficiente un territorio significa que los drones maximicen la cantidad de territorio explorado en el menor tiempo posible debido a, como se mencionó en el capítulo anterior, la

escasa autonomía de vuelo.
Se plantea como el primer problema a resolver el de investigar posibles alternativas de algoritmos de exploración que permitan a la flota de drones explorar un territorio de forma eficiente teniendo en consideración el área explorada por los demás miembros de la flota. A su vez los algoritmos deberán tener en cuenta los POI's que se encuentran en el territorio para decidir las acciones de exploración de la flota.
La exploración debe ser realizada de forma autónoma por la flota de drones, sin tener un servidor central que controle a los drones ni que tenga visibilidad global de la flota, por lo tanto también se presenta la necesidad de investigar posibles alternativas que permitan la ejecución de un programa de software directamente en el hardware de los drones de la flota. El programa a implementar será el encargado decidir las acciones realizadas por el dron que lo esté ejecutando.
Los drones deben poder comunicarse entre sí para que el programa que están ejecutando pueda decidir de forma inteligente, con el objetivo de realizar la exploración de forma eficiente, cual es la siguiente acción a realizar. La necesidad de comunicación entre los drones plantea el problema de investigar posibles técnicas de comunicación entre drones que puedan ser implementadas con el hardware disponible y que permita la coordinación de movimientos entre los drones para realizar la tarea de exploración. Se plantea como una posible línea de estudio establecer una red ad-hoc entre los drones, de forma tal que la comunicación sea independiente de una infraestructura de red externa.
El problema de encontrar alternativas que permitan la ejecución de un programa de software directamente en el hardware de los drones es el primero resuelto, debido a que su solución impacta de forma directa en la implementación del algoritmo de exploración y en la posibilidad o no de implementar una red ad-hoc. Entre las consideraciones mas importantes, la forma en que se logre ejecutar código sobre los drones Parrot se determina el lenguaje de programación a utilizar, las bibliotecas a incorporar y el mecanismo de control de posicionamiento del dron.
El principal inconveniente al momento de desarrollar un sistema que se pueda ejecutar sobre el hardware del dron es la arquitectura del procesador con el que cuentan los drones Parrot Bebop 2. Los drones tienen un procesador con arquitectura ARM, más específicamente ARMv7, la cual se clasifica como una arquitectura con conjunto de instrucciones reducido (Reduced Instruction Set

Computer, RISC). La diferencia entre la arquitectura de los drones y la de las computadoras tradicionales, que siguen la filosofía de diseño con un conjunto de instrucciones complejo (Complex Instruction Set Computer, CISC), genera conflictos al momento de crear los archivos ejecutables.
Por otra parte, tanto para la elección del lenguaje de programación junto a sus compiladores como para la utilización de librerías existentes que permitan desarrollar programas capaces de comunicarse con los distintos componentes del dron, se tienen en cuenta las limitantes generadas a nivel de software por el sistema operativo que se ejecuta sobre el dron. Los drones ejecutan Busybox como sistema operativo, una distribución de Linux que cuenta solo con funcionalidades básicas para funcionar en el dron y que no permite, entre otras cosas, acceso a repositorios desde donde instalar distintos programas Linux como pueden ser compiladores de lenguajes de programación. Además, la distribución instalada en los drones de Busybox ha sido modificado por Parrot para evitar que los usuarios accedan de forma libre al sistema de archivos y ejecuten operaciones privilegiadas.

2.2. Planificación de vuelo

Esta sección describe la estrategia establecida para el estudio del problema descrito. La estrategia consiste en dividir el estudio del problema en una serie de etapas, detalladas a continuación, ejecutadas de forma secuencial.

2.2.1. Etapas del proyecto

El proyecto se organizó en cinco etapas, que se describen a continuación.

- **Primera etapa:** esta etapa se consideró como la fase inicial del proyecto donde se define el trabajo a realizar y donde se plantean los objetivos a alcanzar. Se realizó la investigación inicial sobre algoritmos de exploración colaborativos, las características de los drones Parrot Bebop 2 a utilizar durante el resto del proyecto, la implementación de un mecanismo de comunicación entre los drones y el estudio del paradigma de la programación orientada a agentes. A su vez se definieron las tecnologías a utilizar y las herramientas de hardware necesarias.

- **Segunda etapa:** en esta etapa se implementó una solución que permitió la ejecución de instrucciones de vuelo sobre los drones Parrot Bebop 2 de forma nativa. Para esto se definió el lenguaje de programación a utilizar y las bibliotecas necesarias para el envío de instrucciones al dron. Se implementó un prototipo que permite ejecutar en el dron instrucciones de movimiento de forma autónoma preestablecidas.
- **Tercera etapa:** en esta etapa se estableció un protocolo de intercambio de información entre los drones por medio de un canal de comunicación implementado por el grupo de trabajo. Para esto fue necesario contemplar las características de hardware de los drones Parrot Bebop 2 de forma tal que el método utilizado pudiese ser trasladado a los mismos de forma eficaz.
- **Cuarta etapa:** en esta etapa se implementó el sistema de exploración colaborativa entre los drones de acuerdo a las características establecidas en la etapa inicial. Para facilitar el desarrollo del sistema se definió el uso de un simulador que permitió realizar pruebas de verificación del funcionamiento del mismo.
- **Quinta etapa:** en esta etapa se realizó la evaluación experimental del sistema construido y se redactó el informe final del proyecto.

2.3. Revisión de trabajos relacionados

Existen diversos antecedentes de investigaciones para la coordinación de un equipo de UAVs con el fin de completar tareas, motivados principalmente por fines militares. En general, los trabajos previos centran la formulación del problema en la monitorización de uno o varios objetivos determinados a priori como es el caso del estudio presentado en la sección 2.3.1.

2.3.1. Coordinación de sistemas de agentes

Mufalli et al. [19] y su equipo realizaron una investigación para resolver el problema de optimizar tareas de investigación de instalaciones fijas por parte de una flota de drones. Se conoce de antemano la ubicación de las instalaciones fijas en el territorio a investigar. Los drones van equipados con distintos tipos de sensores cada uno. Las instalaciones fijas producen mejores o peores beneficios en función del sensor que lleva equipado el dron que la investiga. La solución

propuesta en el artículo utiliza como base para su desarrollo el problema de orientación de equipos (Team Orienteering Problem, TOP), el cual a su vez es una generalización del problema de orientación (Orienteering Problem, OP), que propone resolver el problema en el cual un vehículo partiendo desde una ubicación de origen debe llegar a un destino previamente establecido. Si bien los drones con los que se cuenta en el proyecto no poseen la característica de estar integrados con distintos sensores cada uno lo que permitiría a cada punto de interés producir mejores o peores beneficios dependiendo de qué dron fue el encargado de visitarlo, los mecanismos de selección para determinar qué dron de la flota debe encargarse de un punto de interés específico pueden adaptarse al problema a resolver en el proyecto ya que se puede considerar como un beneficio el que un dron vigile con mayor regularidad un punto de interés de más prioridad que otro con menos prioridad.

Cesare [7] en su trabajo propuso un algoritmo para la coordinación de un equipo de UAVs con el fin de explorar un territorio previamente desconocido, teniendo en cuenta las limitaciones que surgen por el tiempo de vuelo limitado de los drones por su batería y de los problemas que pueden surgir por problemas en la red que se utiliza para la comunicación del equipo. Se basa en la premisa de que al finalizar la exploración no es necesario que todos los drones retornen al punto de partida. El algoritmo definido por el autor se basa en una máquina de estados compuesta de cinco estados, denominados explore, meet, sacrifice, relay y go home. Explore es el estado inicial de todos los UAV que forman parte del equipo y es en el donde se define la estrategia de exploración del dron, que se basa en definir fronteras entre el espacio explorado y el espacio desconocido. Después de un tiempo constante previamente definido se cambia al estado meet, donde el dron busca comunicarse con otro miembro del equipo para transmitir su información y a su vez actualizar la información interna que posee sobre las posiciones del resto de los drones. A su vez cuando un dron se comunica con otro miembro del equipo se negocia quién pasa al estado relay y quién al estado sacrifice. Si se determina que el dron será sacrificado, pasa al estado de exploración. A su vez se controla el tiempo de vuelo restante que tiene el dron. Cuando el tiempo de vuelo restante del dron a ser sacrificado solamente es suficiente para encontrarse con el dron que se pasó a estado relay entonces el dron a ser sacrificado pasa al estado sacrifice y busca encontrarse con el dron en estado relay para transmitirle la información recolectada. En cambio el dron que fue seleccionado inicialmente

para pasar a estado relay continúa su viaje volviendo al estado exploración y cuando determina que le queda tiempo de vuelo suficiente solo para volver a la base aterriza para esperar al dron en estado sacrifice. Luego de que el dron en estado sacrifice llega a encontrarse con el dron que lo estaba esperando, éste último retorna a la base cambiando al estado go home. Este artículo provee un interesante enfoque sobre una arquitectura distribuida para la exploración de un territorio y basada en máquinas de estado que puede servir de guía para la implementación de la arquitectura de la solución al problema a resolver.
Grøtli and Johansen [13] en su trabajo presentó un algoritmo, basado en los utilizados para resolver un problema de tipo MILP, para resolver el problema del offline path planning para múltiples UAVs. MILP es un tipo de problema en el cual deben optimizarse una serie de variables basado en una formulación matemática de un problema de optimización con restricciones. El objetivo del problema abordado por Grotli es desplegar los drones en una zona conocida para que sirvan de relés de comunicación entre estaciones aisladas. La parte central del algoritmo desarrollado se basa en resolver un MILP. Los objetivos del problema se modelan con un funciones las cuales determinan rutas para los drones optimizando la conectividad entre ellos y minimizando el consumo de combustible. En resumen, los cuatro componentes principales de la función objetivo del problema describen: el consumo de combustible para mantener la velocidad crucero, el consumo de combustible para acelerar y frenar, el consumo de combustible para cambiar de altitud y el nivel de conectividad entre los UAV de la flota. A su vez, las restricciones definidas en el MILP se usan para evitar colisiones, que los UAV no salgan de las zonas de vuelo autorizadas, entre otros controles. El algoritmo propuesto por Grotli de optimización es ejecutado en sucesivas iteraciones, en cada una alimentándose con datos de SPLAT! [32], un servicio para comunicaciones por radio que permite estimar el nivel de conectividad entre dos puntos en el espacio, lo cual permite generar rutas progresivamente mejores con cada iteración. El proceso de generar rutas se repite hasta que las rutas calculadas por el algoritmo cumplan con los requisitos de la misión, los cuales tienen que ser definidos de alguna forma matemática. En el estudio de Gromli se probó el algoritmo implementado con simulaciones y mostró que produce rutas efectivas que se ajustan a una gran variedad de situaciones y obstáculos. Si bien el trabajo de Gromli no está enfocado en la exploración sino en cómo distribuir efectivamente los UAVs de una flota sobre un terreno, el enfoque de usar MILP resulta muy interesante

ya que puede ser adaptado a múltiples situaciones, incluyendo la de optimizar la exploración.

Shang [30] en su trabajo desarrolló un algoritmo híbrido para resolver el problema de misiones de vigilancia con flotas de UAV, planteado como un problema de optimización combinatoria. El algoritmo híbrido desarrollado por Shang busca la planificación de rutas para los UAVs de modo que sea posible poder obtener el máximo beneficio en la vigilancia satisfaciendo la limitante de energía de los UAV. Se propuso un algoritmo híbrido que combina dos tipos diferentes de métodos, los cuales son un algoritmo genético (Genetic Algorithm, GA) y un algoritmo de optimización de colonia de hormigas (Ant Colony Optimization, ACO). El problema involucró objetivos de vigilancia que deben ser visitados por la flota de UAV. Cada uno de los objetivos provee de un beneficio de vigilancia determinado según su importancia. Las dos métricas que se utilizaron para medir la eficiencia y eficacia del algoritmo fueron la suma total de beneficios de vigilancia obtenidos y el tiempo de la misión. El problema planteado fue modelado por el autor mediante un grafo conectado en el cual los nodos son los objetivos a visitar, cada arista representa un camino entre nodos y cada objetivo tiene asociado un beneficio de vigilancia particular.

El objetivo del algoritmo es encontrar m caminos empezando en uno origen y finalizando en un destino , visitando los objetivos de modo de maximizar el beneficio de vigilancia. Cada nodo se puede visitar como máximo una sola vez. La restricción aplicada es un tiempo máximo de duración de la misión que depende de la energía restante de los UAV de la flota.

La especificación del algoritmo híbrido implementado distingue responsabilidades distintas para cada componente: el GA es el encargado de intentar mantener y evolucionar a una población (en este caso de nodos objetivo a visitar) hacia una mejor solución mientras que el ACO permite iterativamente que un conjunto de agentes (en este caso los elementos de la flota de UAV) busquen feromonas para encontrar mejores soluciones. El algoritmo propuesto incorpora habilidades de ambos algoritmos, GA y ACO, donde la suplantación de malos individuos de la población por nuevos individuos es generada por el ACO.

Al comparar el algoritmo propuesto por el autor con otros algoritmos como son el de búsqueda local guiada (Guided Local Search, GLS), ACO sin el GA y el slow path relinking (SPR), se encontró que el algoritmo híbrido obtiene mejores resultados en función de las métricas preestablecidas.

El estudio realizado por Shang permite obtener otras medidas para comparar el desempeño de distintos algoritmos, como puede ser la forma de caracterizar la eficiencia y eficacia del territorio a recorrer más allá del cubrimiento.

2.3.2. Establecimiento de una red de comunicación entre la flota de drones

Schleich [28] et al. desarrollaron un modelo de cubrimiento de áreas para una flota de drones no pilotados con la restricción de mantener la mayor conectividad posible entre los drones y con una estación base.
El modelo se basa en la selección de las rutas a tomar por los drones mediante el seguimiento de feromonas. La estrategia planteada consiste en hacer que los drones dejen un rastro en su camino, análogo al rastro de feromonas que dejan las hormigas cuando exploran una ruta. Los caminos más exitosos se refuerzan con una mayor cantidad de feromonas para que los demás drones las detecten y las sigan, generando un círculo virtuoso en el que los mejores caminos se ven cada vez más reforzados con más feromonas. Schleich et al. definieron cuatro métricas para la evaluación de su modelo y analizaron los resultados obtenidos comparándolos con los de otros modelos tomados de otros trabajos, enfocándose en otros modelos basados en seguimiento de feromonas y en algoritmos greedy. El trabajo resalta la capacidad que tiene una flota de drones en misiones de reconocimiento, exploración y vigilancia para mantenerse por un lapso de tiempo indefinido ya que un subconjunto de la flota puede estar recargando sus baterías mientras el resto continúa con la misión.
El escenario propuesto en el trabajo de Schleich et al. implica el mantenimiento de una red ad-hoc entre los drones de la flota y la obtención de medidas en tiempo real del área a explorar en la misión. Las tres métricas introducidas son: la velocidad de cubrimiento del área, la exhaustividad del cubrimiento del áreay el mantenimiento de la conectividad. Para obtener valores para las métricas se divide el área a explorar en pequeños rectángulos de largo y ancho determinado, de modo que cada vez que un dron pasa por esa área se envía un mensaje a los drones conectados y a la base. El modelo de cubrimiento de área desarrollado está formado por tres etapas secuenciales: la selección del dron vecino al cual se intenta estar conectado, el cálculo de la ruta alternativa viable para dónde debe dirigirse y la aplicación del comportamiento basado en feromonas para seleccionar la mejor ruta basado en los cálculos anteriores.

Los resultados obtenidos para las métricas evaluadas fueron comparables con los de los otros modelos de referencia para una flota comprendida por 20 drones o más. En cuanto a conectividad entre la flota y porcentaje de nodos conectados a la base se obtuvieron mejores resultados que en los otros modelos, manteniéndose relativamente similar en cuanto al resto de las métricas.
Schleich et al. abordaron una de las limitantes principales de su proyecto, que es la conectividad entre los drones, evaluando la calidad de la conexión en relación a cuántos drones están conectados a una estación base. Las métricas planteadas para evaluar la calidad de la misión son aplicables a este proyecto, si bien se debería adaptar la medida de conectividad para que los drones no utilicen una base central.

Bekmezci et al. [3] propusieron una nueva denominación para la familia de redes ad hoc encargadas de la comunicación en sistemas multi UAV: "Flying ad hoc Network (FANET)". También presentaron una definición para este tipo de redes, una comparación con redes existentes como las MANETs (Mobile ad hoc Network) y VANETs (Vehicle ad hoc Networks), desafíos en el diseño de las FANET, escenarios donde aplique el uso de FANET, distintos protocolos utilizados en ellas e investigaciones abiertas, mediante la revisión de artículos de la literatura. Bekmezci et al. realizaron estudios focalizados en distintas áreas de las FANET como las características de diseño de las mismas, consideraciones al diseñarlas y distintos tipos de protocolos existentes en las FANET. Finalmente se mencionaron estudios donde se generen resultados a través de simulaciones.
Inicialmente se analizaron los problemas que surgen al utilizar un pequeño grupo de UAVs para una misión particular. Entre los problemas más importantes está el problema del diseño de la infraestructura necesaria para la comunicación entre los distintos UAVs. Los autores mencionaron dos enfoques utilizados en la literatura: comunicación entre los UAV a través de una infraestructura con una estación base y comunicación entre los distintos UAV directamente mediante una red ad hoc generada entre ellos. Las características principales de los elementos de una FANET según la literatura presentada son las siguientes: gran movilidad de los nodos y gran distancia entre ellos, topología de la red muy cambiante y nodos que deben soportar una comunicación peer-to-peer y también recolectar información del entorno mediante sensores. En el artículo se mencionan distintos escenarios en donde se puede aplicar el concepto de una

FANET. Entre ellos se destacan escenarios donde se desee extender la escalabilidad de una misión con multi UAVs en los que generalmente no existe o no se puede comunicar con una estación base (debido a las grandes distancias entre los nodos y la base por ejemplo), escenarios donde se necesite una comunicación multi UAV confiable, escenarios de enjambres de UAVs, y otros.
Dentro de las características del diseño de una FANET, se mencionaron los siguientes puntos: los nodos (UAV) tienen gran movilidad y velocidad, por lo cual las FANET deben recalcular continuamente las rutas de vuelo; la baja densidad de los nodos y la distancia entre ellos; cambios rápidos y continuos en la topología de la red (debido a la movilidad y/o fallas en los nodos); el poder computacional limitado por la escasez de espacio y energía para la instalación de procesadores y la posibilidad de geo-localizar los UAV mediante GPS.
Luego se hace referencia a estudios donde se realizan consideraciones a tener en cuenta para el diseño de una FANET, de las que se destacan: la adaptabilidad a entornos diversos y cambiantes; la escalabilidad para poder implementar una FANET con cualquier cantidad de UAVs; la latencia mínima en el envío de paquetes entre los UAV; las limitantes impuestas por el hardware de las UAV y un ancho de banda necesario para el envío de la información obtenida por los UAV.
Como último ítem importante se destacan los distintos protocolos de comunicación en una FANET presentados en la literatura. El análisis realizado observa que los protocolos se dividen de acuerdo a la capa de red que utilizan. Los protocolos que operan en la capa física se clasifican en dos modelos: el modelo de propagación de ondas de radio y el modelo de estructura de antena. En la capa de enlace el protocolo IEEE 802.11 es el más utilizado. En la capa de red las propuestas utilizan generalmente protocolos de enrutamiento provenientes de las MANET. Sin embargo, una estrategia de enrutamiento basada solamente en información de la locación de los nodos puede satisfacer las necesidades de una FANET. En la capa de transporte los estudios mencionan que la principal responsabilidad de los protocolos es la de la confiabilidad en el transporte de los paquetes, el control de flujo y el control de congestión de los mismos.
El artículo otorga una visión global del problema de diseño de una red ad hoc entre UAVs, el estado actual de investigación en el área y el relevamiento de algunos estudios realizados en áreas específicas del problema planteado. Se considera este artículo de gran valor para este proyecto por la recopilación y análisis de literatura disponible.

Kopeikin et al. [2]. analizaron las tecnologías existentes sobre control de comunicación por red en equipos de UAVs. Dado que los UAVs son móviles, las comunicaciones en estas redes deben ser inalámbricas y en general transmiten la información hacia una base central donde la información se procesa. También se debe permitir que los UAVs puedan comunicarse entre ellos, de forma que puedan actuar de forma coordinada. Existen varios mecanismos de control que permiten mejorar la calidad de una red de UAVs. La mayoría de los mecanismos consisten en implementar una o varias de las siguientes acciones: posicionar los UAVs de forma que se reduzca la distancia entre ellos; aumentar la potencia de los transmisores, o elegir ubicaciones especialmente favorables para la transmisión; elegir activamente las conexiones habilitadas en una red y determinar caminos óptimos dentro de la red; aumentar la cantidad de UAVs en la red.

Los principales desafíos que una red inalámbrica tiene que superar son la distancia entre los nodos y la interferencia provocada ya sea por obstáculos físicos o por el ruido de otras señales. Los UAVs aéreos en general funcionan en cielo abierto y eso les permite evitar la mayoría de estos problemas. Sin embargo pueden existir situaciones donde los UAVs deben volar en zonas urbanas o a baja altitud. Por lo tanto, el objetivo de la red y el tipo de vehículo que se utilizará en ella es fundamental para decidir su diseño. La potencia de los transmisores del UAV puede verse limitada por el consumo de energía que requieren, por el espacio que ocupan, o porque otros instrumentos del UAV (incluyendo la CPU) también consumen estos mismos recursos y tienen una mayor prioridad.

La arquitectura de la red también depende del tipo de información que se quiere enviar a través de ella. Se puede subdividir los tipos de datos a enviar en dos clases:

- Mensajes de control y comandos. Pueden dar órdenes a los UAVs, transmitir telemetría básica sobre el estado del UAV (como la posición o el estado de carga de la batería), entre otros. Estos mensajes requieren poco ancho de banda pero no son resistentes a fallos y pueden requerir una baja latencia.
- Información captada por los sensores del UAV. Pueden ser imágenes, video, audio, datos complejos, etc. Estos mensajes requieren un mayor ancho de banda y tienen los mismos requerimientos que tendrían si fueran

transmitidos por Internet.

2.3.3. Resumen

A partir de las investigaciones reseñadas se extraen conceptos que son de utilidad para la instrumentación de sistemas colaborativos basados en agentes. El artículo de Mufalli et al. [19] sirve como punto de partida para comprender el problema de vigilancia de objetivos preestablecidos, pero su enfoque principal es determinar con que tipo de sensor se debe equipar a cada dron de la flota de exploración para obtener un mayor beneficio de la vigilancia de cada objetivo, lo cual no tiene relación con el objetivo de este proyecto.
La investigación realizada por Grøtli and Johansen [13] se basa en resolver un problema de tipo MILP para obtener la mejor ruta a utilizar por una flota de drones. Lo que diferencia al trabajo realizado por Grøtli and Johansen [13] de este proyecto es que el cálculo de la ruta a seguir por los drones se realiza previo al comienzo de la exploración, para que cada dron siga la ruta precalculada como óptima.
El artículo de Shang [30] propone resolver el problema de coordinar los miembros de la flota de drones por medio de un algoritmo genético. Al igual que en la investigación de Grøtli and Johansen [13], el algoritmo propuesto por Shang [30] requiere que la ruta a seguir por los drones sea calculada previamente al inicio de la exploración.
Por otra parte, el trabajo realizado por Cesare [7] tiene metas compartidas con este proyecto, planteando como objetivo la coordinación de los miembros de la flota de exploración. Del artículo de Cesare [7] se toma como guía para la implementación del sistema de exploración, la estrategia de utilizar una máquina de estados que sea ejecutada por cada dron.
El artículo de Schleich [28] resuelve el problema de coordinar a los drones participantes de la exploración, por medio de un algoritmo que logra que los drones opten por una ruta compuesta por los caminos que mejores resultados hayan obtenido. El sistema propuesto por Schleich [28] requiere de la implementación de una red ad-hoc entre los miembros de la flota de exploración y un canal de comunicación entre cada dron y una base central, encargada de monitorizar la ejecución del sistema.
Dentro de los trabajos relacionados a resolver el problema de mantener la flota de exploración comunicada durante la ejecución del sistema, se encuentra la

investigación realizada por Bekmezci et al. [3], donde se define la nomenclatura FANET para denominar a las redes ad-hoc de vehículos autónomos. El artículo de Bekmezci et al. [3] considera limitantes comunes para los sistemas implementados sobre drones, como lo son la escasa duración de la batería y las condiciones cambiantes del ambiente sobre el que operan los drones.
La investigación realizada por Andrew Kopeikin and How [2] analiza las características de los canales de comunicación establecidos sobre vehículos autónomos. Andrew Kopeikin and How [2] detallan los problemas a tener en cuenta al momento de implementar la red inalámbrica entre los drones, como lo es controlar la distancia entre los drones para poder mantener una señal estable.

Capítulo 3

Arquitectura del sistema y funcionalidades básicas

En este capítulo se describen las características del modelo de drones utilizados, y las herramientas de software empleadas para el desarrollo del sistema implementado. También se analizan alternativas para establecer un canal de comunicación entre los miembros de la flota de exploración, considerando ventajas y desventajas de cada una. A su vez se detalla el problema de controlar el posicionamiento de cada dron en el espacio, destacando distintas alternativas para su resolución.

3.1. Características del hardware utilizado

Para la realización del proyecto se utilizaron dos drones de la marca Parrot, específicamente el modelo Bebop 2 [22]. Los drones Bebop 2 poseen un diseño resistente y simple, una cámara de buena calidad, sistema de posicionamiento vía GPS y sistemas de estabilización de vuelo. Todas estas características resultan de utilidad para el proyecto.
Sin embargo, el modelo Bebop 2 plantea algunas limitantes. En particular no es un modelo para ejecutar programas implementados por terceros directamente sobre su hardware. Existe un kit de desarrollo de software (SDK) provisto por Parrot que permite desarrollar software para controlar ciertas características del Bebop 2 y que se basa en controlar el dron a distancia, ya sea por medio de una computadora o un dispositivo móvil. Un ejemplo de esto es FreeFlight-

Pro [11], una aplicación para dispositivos móviles que le permite a los usuarios controlar el Bebop 2 de forma manual. Utilizar el SDK de Parrot de forma directa para ejecutar programas desde el hardware del dron resulta muy complejo, por lo tanto se optó por usar pyparrot [25], una biblioteca en Python que implementa una interfaz para el SDK original de Parrot. Más adelante se discute con mayor profundidad porqué es complejo usar el SDK de forma directa, el funcionamiento de pyparrot y como resuelve el problema de implementar programas que puedan ser ejecutados desde el dron.

Los drones Bebop 2 tienen instalado como sistema operativo embebido una distribución de Unix llamada Busybox [5], la cual impide la ejecución de ciertas tareas. La configuración de Busybox que viene de fábrica no permite que el dron se conecte a Internet, solo permite el acceso a una parte limitada de la memoria y no permite que se instalen o compilen programas nuevos directamente desde el dron. El no permitir la instalación de nuevos programas desde el dron fue particularmente problemático para el proyecto, ya que obligó a compilar los programas a utilizar en otros sistemas para luego ser instalados.

Los drones Bebop 2 presentan la arquitectura ARMv7[23]. ARMv7 es una arquitectura de tipo RISC, enfocada en implementar pocas instrucciones de forma eficiente. Esto es en contraste con las arquitecturas CISC, como las de los procesadores x86 que se encuentran en la gran mayoría de las computadoras de sobremesa, y que poseen un enfoque más generalista pero menos eficiente. Estas dos arquitecturas son incompatibles entre ellas y el hecho de que los drones tengan una arquitectura ARMv7 implica que los sistemas y dispositivos que se utilicen para compilar los programas a utilizarse en ellos deben tener la misma arquitectura.

En las secciones 3.3 y 3.4 se discuten distintas alternativas propuestas para superar las limitaciones presentadas tanto por el sistema operativo Busybox como por la arquitectura ARMv7. En la Tabla 3.1 se describen las principales especificaciones del dron. [9].

Tabla 3.1: Características del dron Parrot Bebop 2.

Característica	Descripción
Sistema de rotores	Cuatro rotores, tres hojas por cada hélice, cinco centímetros de diámetro, de plástico flexible, bastante resistentes y fácilmente reemplazables
Velocidad máxima horizontal	18m/s
Velocidad de ascenso máxima	6m/s
Alcance de señal	300 metros
Batería	2.700 mAh
Autonomía	25 minutos de vuelo
Cámara frontal	14 megapíxeles
Dimensiones	33 x 30 x 10 centímetros
Memoria accesible por el usuario	8GB
Peso	480 gramos
Sistema Operativo	Unix (Busybox)
Arquitectura de CPU	ARMv7
Sensores (accesibles vía el SDK)	Cámara digital (1080pi), sensor GPS, sensor de altura (ultrasonido).
Sensores (no accesibles vía SDK)	Acelerómetro, giroscopio, brújula.

3.2. Simulador de drones

Para evaluar y testear los programas desarrollados en este proyecto fue necesario utilizar un simulador, ya que no resulta conveniente probar código en desarrollo directamente en el hardware. Emplear un simulador permite proteger tanto al dron como a las personas que se encuentren en las cercanías al momento de realizar vuelos. Además, se utilizaron simuladores para realizar la evaluación experimental del sistema de control y vigilancia desarrollado ya que permiten realizar pruebas automatizadas de forma eficiente.
Para este proyecto se optó por utilizar Sphinx [31], el simulador de drones oficial de Parrot. Sphinx es un simulador basado en el software de robótica Gazebo [12] para ambientes Linux de 64 bits. Para las simulaciones se requiere una tarjeta gráfica con soporte de OpenGL 3.0 o superior. En caso de que

sea necesario utilizar la cámara frontal del dron, los desarrolladores de Sphinx recomiendan contar con una tarjeta gráfica GeForce 1060 Ti o superior, ya que simular la imagen producida por la cámara aumenta significativamente el consumo de recursos de GPU. Junto con Sphinx, Parrot provee firmwares que simulan modelos específicos de sus drones, incluyendo el Bebop 2, lo cuales son configurables. Los modelos se pueden cargar en el simulador y permiten emular todas las características físicas del dron.

Sphinx también permite crear mundos (entornos de simulación) que se pueden editar mediante archivos de configuración. Para agregar varios drones a la simulación es necesario editar estos archivos de configuración, donde además es posible definir las características de los drones, incluyendo los parámetros de su cámara y su batería.

Es posible configurar los drones simulados en Sphinx para que utilicen una interfaz Wi-Fi del equipo sobre el que se ejecuta la simulación. Otros dispositivos pueden enviar órdenes al dron simulado a través de esta interfaz Wi-Fi. Hay que tener en cuenta que las interfaces Wi-Fi quedan usadas con este propósito son completamente acaparadas por Sphinx, es decir que no pueden ser usadas con otros fines, como por ejemplo para acceder a Internet. Si no se desea utilizar una interfaz Wi-Fi real, cada dron en la simulación genera una interfaz virtual que permite el envío de órdenes al dron.

Sphinx facilita el desarrollo de aplicaciones para drones. Se puede optar entre distintas opciones para implementar programas que se comuniquen con el dron, ya sea de forma directa con el SDK de Parrot [29], o mediante otras bibliotecas desarrolladas por terceros como pyparrot [25].

La principal desventaja de Sphinx es que requiere una cantidad significativa de recursos de GPU para ejecutar las simulaciones; los desarrolladores de Sphinx recomiendan el uso de tarjetas gráficas NVIDIA de gama alta. Además, Sphinx presenta problemas para simular varios drones en la misma instancia del programa. En particular, los comandos que controlan el movimiento de los drones no funcionan de forma apropiada. Los problemas para simular múltiples drones de forma simultánea se se solucionan al ejecutar cada dron simulado en una instancia de Sphinx, lo que genera que el consumo de recursos de hardware incremente. Para distribuir el consumo de recursos generado por las múltiples instancias de Sphinx es posible emplear varias computadoras, lo que implica que se debe establecer un canal de comunicación entre las instancias de los drones simuladas por medio de la red en la que están conectadas

las computadoras.

3.3. Ambiente de desarrollo

En esta sección se detalla el análisis realizado para la elección del entorno de desarrollo en el cual se trabajó para implementar el sistema de control y vigilancia.

3.3.1. Herramientas de compilación

El primer desafío encontrado fue el de implementar una solución que pudiera ser ejecutada en la arquitectura ARMv7 que posee el dron Parrot Bebop 2, teniendo en cuenta que los equipos donde se desarrolla la solución están basados en la arquitectura x86 de los procesadores Intel.
Se plantearon tres alternativas para solucionar el problema de compilar un programa implementado para ARMv7. La primera alternativa fue realizar una compilación cruzada (en inglés, cross-compile) [24]. Se entiende por cross-compile la acción de compilar código en un sistema huésped que no tenga la misma arquitectura que el sistema objetivo en el cual se piensa ejecutar el programa. De esta forma se genera un ejecutable compatible con la arquitectura del sistema objetivo. En el contexto de este proyecto, el sistema huésped es la arquitectura x86 y el sistema objetivo es el dron con arquitectura ARMv7. Esta opción requiere la utilización de compiladores específicos para la arquitectura objetivo seleccionada.
La segunda alternativa fue utilizar el emulador Qemu [26], que permite la ejecución de sistemas operativos diseñados para arquitecturas diferentes a la arquitectura del sistema anfitrión. Emular la arquitectura del dron hace que no sea necesario hacer uso de la técnica de cross-compile dado que el ejecutable generado con Qemu es compatible con el dron.
Como tercera alternativa se plantea la de utilizar otra CPU que posea arquitectura ARM y compilar los programas directamente en la arquitectura del dron, de forma similar a como lo haría Qemu pero sin requerir de emuladores ni técnicas engorrosas como cross-compile. Una de las opciones de CPU más accesibles en el mercado con una arquitectura compatible a la ARMv7 son los dispositivos Raspberry Pi [27]. Estos son dispositivos de propósito general basados en arquitectura ARM.

En este proyecto se decidió utilizar la Raspberry Pi para compilar directamente en ella los programas necesarios, en lugar de realizar un cross-compile o utilizar el emulador Qemu. Cross-compile genera dificultades al momento de compilar bibliotecas con muchas dependencias (como por ejemplo el SDK del dron). Qemu, si bien otorga las mismas funcionalidades que la Raspberry Pi, tiene una complejidad mayor en cuanto a instalación y posterior configuración. De todas formas, compilar e instalar programas en el dron es una actividad que resulta compleja y consume mucho tiempo, por lo que en las siguientes etapas del proyecto se trabajó con la idea de instalar el mínimo número posible de programas en el dron.

3.3.2. Elección del lenguaje de programación

El siguiente desafío fue el de establecer un lenguaje de programación que permita comunicarse con los drones y que permita desarrollar programas con facilidad en dos arquitecturas a la vez: la arquitectura x86 en la cual funciona el simulador Sphinx y la arquitectura ARMv7 en la cual funciona el dron. El SDK oficial provisto por Parrot está disponible en tres lenguajes de programación distintos: C, Java y Objective C. Sin embargo, todos estos lenguajes son compilados, lo que implica que cada vez que se realiza una modificación en un programa, por más mínima que sea, hay que volver a compilar todo de nuevo en las dos arquitecturas. Si bien la Raspberry Pi permite realizar este proceso de la forma más simple posible, en la práctica sigue siendo un proceso engorroso que consume mucho tiempo y es preferible evitar.
La solución a este problema surgió de utilizar una biblioteca para el lenguaje de programación Python llamada pyparrot [25] la cual implementa una interfaz para el SDK oficial de Parrot. Python es un lenguaje de programación de scripting por lo que no es necesario compilar los programas desarrollados en él. Lo único que hay que compilar para ejecutar scripts de Python en ARMv7 es el intérprete de Python en sí mismo y aquellas bibliotecas que se encuentren escritas directamente en C. En el caso de este proyecto solo fue necesario compilar la biblioteca netifaces [10], usada por pyparrot para implementar sus funcionalidades de red. Este proceso de compilar el intérprete de Python y las bibliotecas en C se puede realizar una única vez y luego es posible desarrollar el resto de la aplicación Python sin necesidad de usar intermediarios para compilar y sin tener que desarrollar dos versiones de la misma aplicación en

dos arquitecturas.
Siguiendo este razonamiento se utilizó la Raspberry Pi para generar ejecutables del intérprete de Python en ARM7. Luego se instaló esta versión compilada de Python en cada uno de los drones. Finalmente se repitió el proceso con la biblioteca netifaces que se encuentra implementada en C. Los ejecutables se pueden generar una única vez y luego se pueden usar en cualquier cantidad de drones. Esto permite que se pueda desarrollar código en equipos que no cuentan con arquitectura ARM y ejecutarlo en el dron sin necesidad de realizar ninguna configuración extra.

3.3.3. Descripción de la biblioteca pyparrot

Pyparrot es una interfaz para el SDK oficial de Parrot que soporta a los modelos Mambo [20], Swing [21], Bebop 1 [8] y Bebop 2 [22]. Ofrece una amplia variedad de opciones de control y resulta muy simple de utilizar, aunque no está pensada para ser usada en una flota de drones y fue necesario realizar algunas modificaciones al método *connect* que establece la conexión entre pyparrot y el dron. La versión original de este método busca conectarse automáticamente al primer dron detectado y cuando hay varios drones en la cercanía pyparrot se termina conectando a uno de ellos al azar, lo cual no resulta conveniente. Para solucionar este problema se modificó el método *connect* para que reciba una dirección IP específica como parámetro y que pyparrot intente conectarse solo al dron con esa IP.
Algunas de las funciones incluidas por pyparrot para el Bebop 2 son:

- **Funciones de conexión y desconexión**: Se incluyen métodos que permiten establecer una conexión con el dron, desconectarse del dron y comenzar el flujo de datos de los sensores.
- **Funciones de despegue y aterrizaje**: Estos métodos tienen precedencia sobre los métodos de control de movimiento, esto resulta útil para implementar un mecanismo de control que permita aterrizar al dron automáticamente en caso de emergencia.
- **Control de movimiento**: Existen dos variantes de control de movimiento, el vuelo directo y el vuelo relativo. El vuelo directo consiste en controlar el movimiento alterando el pitch (cabeceo), yaw (guiñada) y roll (alabeo) del dron, así como el tiempo que dura la orden. Por otra parte, el vuelo relativo consiste en establecer un vector (x, y, z, θ) que

define respectivamente la cantidad de metros que el dron debe moverse sobre sus tres ejes y la rotación a realizar. En la sección 4.4.2 se discuten con más detalle las diferencias entre estos dos modos.

- **Control de la cámara**: Se puede controlar el ángulo de rotación de la cámara, comenzar y parar las grabaciones de video, tomar fotos y ajustar varios parámetros de configuración que afectan la calidad de imagen.
- **Lectura de sensores**: Se tiene acceso a todos los sensores que publica el SDK original de Parrot. La lista detallada de sensores se encuentra en la Tabla 3.1.
- **Métodos de control**: Se pueden establecer una alturas de vuelo y velocidades máximas y mínimas, además de otros parámetros.

Pyparrot consiste en un conjunto de clases, cada una especializada en controlar un modelo de dron específico. Para controlar un dron Bebop 2 se debe crear una instancia de la clase Bebop y luego utilizar el método *connect* de la clase para conectarse a un dron compatible. Una vez que una instancia de clase establece una conexión con un dron esta puede controlarlo mediante otros métodos de clase como por ejemplo *safe_takeoff* para despegar, *relative_move* para desplazar al dron en el modo relativo, *safe_land* para aterrizar, etc. La Tabla 3.2 presenta un ejemplo básico de uso de pyparrot.

Tabla 3.2: Ejemplo básico de uso de pyparrot

```
from pyparrot.Bebop import Bebop          ▷ Importar la clase Bebop
bebop = Bebop()                  ▷ Crear una instancia de la clase Bebop
success = bebop.connect(10)   ▷ Conexión al dron (máx. 10 intentos)
if success then
    bebop.smart_sleep(5)                        ▷ Esperar por 5 segundos
    bebop.ask_for_state_update()   ▷ Actualizar estado de los sensores
    bebop.safe_takeoff(10)                ▷ Despegue (máx. 10 intentos)
    bebop.relative_move(1, 0, 0, 0)   ▷ Mover un metro hacia adelante
    bebop.safe_land(10)                   ▷ Aterrizaje (máx. 10 intentos)
    bebop.disconnect()                                    ▷ Desconexión
```

3.4. Conectividad entre drones

Para crear una flota verdaderamente autónoma de drones se requiere que los miembros de la flota puedan establecer comunicaciones entre ellos de forma directa, sin necesidad de utilizar equipamiento externo. Por esta razón la primera opción para crear una red de comunicación entre los drones fue la de crear una red ad-hoc. En una red ad-hoc todos los nodos de la red se encuentran en igualdad de condiciones, es decir que no hay un nodo central que controle las acciones de los demás y los nodos pueden seguir operando sin necesidad de estar conectados a toda la red o incluso de forma completamente offline. En el caso particular de la exploración de una zona con una flota de drones esto plantea varias ventajas. En primer lugar las redes ad-hoc permiten que los drones individuales sean completamente autónomos entre sí, incluso pueden seguir trabajando habiendo perdido toda conexión con el resto de la red. Además, cada dron pueden servir de relé para otros drones, por ejemplo, puede suceder que dos drones se encuentren muy lejos entre sí como para establecer una comunicación directa pero si existe un tercer dron que se encuentra en un punto intermedio entre ellos este puede servir de relé entre los dos. Esto aumenta de forma considerable el alcance que puede tener la red, lo cual incrementa el área que se puede explorar de forma coordinada.
Debido a las ventajas descritas se analiza la posibilidad de implementar una red ad-hoc en los drones Bebop 2. Sin embargo la tarjeta de red de estos drones no viene con drivers para el modo ad-hoc incluidos.Los drones poseen una tarjeta de red integrada Broadcom BCM4360 [4] y que solo incluye de fábrica drivers para los modos Master y Managed. Al configurar el modo Master el dron es el encargado de hospedar la red Wi-Fi, lo que permite que otros dispositivos se conecten a la red creada; por otra parte el modo Managed permite al dron conectarse a una red Wi-Fi a la que tenga acceso. Los Bebop 2 por defecto tienen su interfaz de red configurada en modo Master, ya que de esta forma los usuarios controlan a los Bebop 2 por medio de aplicaciones externas que se conectan al dron y por esta razón es que la tarjeta de red que viene de fábrica no necesita otros modos de conexión.
Se puede intentar incorporar el modo Ad-Hoc a los drivers que traen los Bebop 2 pero su instalación no se puede realizar sin efectuar operaciones riesgosas sobre el sistema operativo BusyBox que pueden dañarlo. Debido a estos inconvenientes se determina descartar el uso directo de una red ad-hoc y se opta

por buscar soluciones que presenten ventajas similares y que puedan ser implementadas usando solamente los modos Master y Managed disponibles en la tarjeta de red.
La primera opción es la de crear una red de drones compuesta por un dron líder en modo Master y con los demás drones conectados a él en modo Managed. Sin embargo esta configuración anula todas las ventajas de una red ad-hoc porque toda la red pasaría a depender de un único dron y todos los drones deben mantener una conexión directa con el líder para seguir conectados a la red.
Finalmente se decide usar en su lugar una red Wi-Fi. Este enfoque tiene algunas desventajas con respecto a las redes ad-hoc, la más importante es que en una red Wi-Fi los drones dependen de una infraestructura externa a ellos que mantenga la red. Esta infraestructura puede estar compuesta por módems portátiles y routers ubicados en lugares estratégicos para cubrir la zona que se busca explorar. Se pueden usar incluso teléfonos celulares.
Por otra parte, las redes Wi-Fi poseen algunas ventajas. En primer lugar tienen un alcance mejor definido con respecto a las redes ad-hoc, por lo que resulta más fácil evitar las pérdidas de conexión con la red. Por otro lado un dron conectado a una red Wi-Fi puede conectarse a cualquier otro miembro de esa red, sin importar la distancia entre ellos. Esto permite establecer comunicaciones a gran distancia sin necesidad de recurrir al uso de otros drones como relés, que si bien es una técnica muy útil, también resulta más difícil de gestionar.
Para implementar la red Wi-Fi es necesario realizar modificaciones en la configuración por defecto de los drones. En primer lugar se debe cambiar el modo de la interfaz a Managed, para que todos los drones puedan estar conectados a la misma red. Luego se le configura a cada uno una dirección IP, el nombre del punto de acceso y su contraseña. Para que este cambio de configuración no interfiera con el funcionamiento de las aplicaciones que utilizan la configuración de fábrica se instala un script en el dron que permite cambiar el modo de la interfaz al presionar 3 veces el botón de apagado.
De esta forma se consigue que todos los drones pertenecientes a la flota de exploración formen parte de una misma red que permite establecer un canal de comunicación entre ellos.

3.5. Sistema de control de posicionamiento

Uno de los principales desafíos encontrados en el transcurso de este proyecto fue el de geo-localizar los drones. Se entiende geo-localizar a un dron como la capacidad de determinar sus coordenadas en el espacio en un momento dado y con un cierto margen de error. La geo-localización es fundamental para el correcto funcionamiento de varias funcionalidades del sistema, como el despegue y el aterrizaje del dron, la toma de datos, el mantenimiento del estado del mapa, la evasión de obstáculos, entre otras.

Algunas de las funcionalidades requieren un sistema de geo-localización de mayor precisión que otras, el aterrizaje en particular requiere un margen de error de menos de un metro, debido al riesgo que supone que el dron aterrice por fuera de la plataforma de aterrizaje. Para la toma de datos y el mantenimiento del mapa la precisión requerida puede variar en función de los objetivos que tenga el sistema y de las dimensiones que tenga la zona a explorar, por lo tanto el aterrizaje es la funcionalidad más restrictiva y se busca implementar un sistema que cumpla con sus requisitos.

Sin embargo, los Parrot Bebop 2 no cuentan con sensores de distancia, a excepción del sensor de altura. Tampoco es posible acceder a la información del acelerómetro ni a la del giroscopio ya que el SDK de desarrollo no permite acceder a sus datos. Sin estos sensores resulta difícil implementar un sistema de geo-localización propio.

Dadas las dificultades para geo-localizar a los drones se analizaron cuatro posibles soluciones. Estas fueron: el posicionamiento relativo, el posicionamiento por GPS, el posicionamiento mediante señales de Wi-Fi y el posicionamiento mediante el uso de imágenes de la cámara.

La primera opción analizada fue la del posicionamiento relativo, la cual consiste en determinar la posición del dron en función de la ubicación del punto de partida y un registro de los desplazamientos realizados. Esta medida se puede implementar usando la modalidad de vuelo relativo de pyparrot.

Al utilizar el posicionamiento relativo se consigue determinar la ubicación del dron sin necesidad de emplear sensores ni equipos externos, pero depende demasiado de la precisión con la que se realicen los movimientos. En la práctica este método termina acumulando imprecisiones.

La segunda opción fue la de usar el sistema de posicionamiento global (Global Positioning System, GPS) incorporado a los Bebop 2. La información de este

sensor es accesible vía pyparrot y permite obtener la latitud y longitud aproximadas del dron. Sin embargo, el uso de este sensor tiene varias desventajas. Entre otras desventajas se tiene que es necesario que el vuelo se realice en espacios abiertos sin interferencias de árboles o edificios, se depende del clima ya que si está nublado se debilita la señal y además le toma algunos minutos al dron conectarse a los satélites, lo cual es mucho cuando se considera que se tiene una autonomía de vuelo de solo 20 minutos. Además el GPS tiene un margen de error de aproximadamente 5 metros y la información no se actualiza en tiempo real sino que tiene un retardo que alcanza los 10 segundos.

La tercera opción consiste en determinar la posición mediante el uso de señales Wi-Fi. Este sistema de posicionamiento se denomina WPS [17] (Wi-Fi Positioning System) y se basa en determinar la ubicación por trigonometría, de forma similar a un GPS. Los sistemas WPS miden la intensidad de la señal recibida de los puntos de acceso de Wi-Fi cercanos. Luego, si se conoce la intensidad original de las señales y la tasa a la que esa intensidad decae en función a la distancia, se puede estimar la distancia a los puntos de acceso donde se originaron esas señales. Si se conoce la ubicación de los puntos de acceso, un dispositivo puede estimar su propia posición usando trigonometría.

En teoría solo se necesitan tres puntos de acceso para estimar una posición, pero en la práctica se recomienda usar por lo menos cinco ya que esto aumenta considerablemente la precisión de la estimación.

Existen varias formas de obtener la información requerida para crear un sistema WPS. En primer lugar, se pueden usar las bases de datos que poseen las grandes empresas tecnológicas y que almacenan la información referente a la intensidad, la tasa de decaimiento y la ubicación de miles de millones de puntos de acceso Wi-Fi en todo el mundo. Estas bases de datos son usadas comúnmente en aplicaciones comerciales de localización y se combina con la información del GPS para obtener mejores resultados.

Sin embargo existen limitantes con estas bases de datos: la mayoría no son públicas y las que sí lo son no brindan servicios que sean precisos ya que ofrecen una longitud y latitud con cuatro cifras significativas, equivalente a un área aproximada de $1km^2$ [18].

La otra opción es implementar un sistema de ubicación propio en base a puntos de acceso conocidos. Existen trabajos que demuestran que se pueden usar estos sistemas para obtener una precisión en la ubicación en el orden de los decímetros [17]. Sin embargo estos métodos requieren de un equipamiento del

que el Bebop 2 no dispone. En particular se requiere que el dron posea tres antenas para disminuir el efecto multipath, que sucede cuando una misma señal de Wi-Fi rebota en varios obstáculos y entonces llega al destino por diferentes caminos de distancia variable. Después de algunas pruebas preliminares que no mostraron ser muy precisas y debido a estas limitantes es que se opta por no seguir este enfoque.

La última opción analizada para geo-localizar a los drones fue la de procesar las imágenes obtenidas por la sus cámaras. Esto se puede hacer calculando la posición relativa con respecto a otros objetos de ubicación conocida.

Para esto se puede emplear una técnica basada en marcadores, en la cual se colocan los mismos en puntos estratégicos que puedan ser reconocidos por la cámara y determinar la ubicación del dron. Utilizar marcadores ha demostrado ser una estrategia precisa, pero tiene como desventaja que se necesita de la colocación de marcadores en el territorio a explorar. Otra problema a tener en cuenta es el alto costo en recursos computacionales que supone para el dron tener que hacer procesamiento de imágenes en vivo. De todas las técnicas analizadas hasta el momento esta es la que muestra mayor potencial porque los Bebop 2 cuentan con el equipamiento necesario para implementarla y su uso ya ha sido probado en otros trabajos.

En resumen, todas las técnicas de geo-localización presentan ventajas y desventajas pero la que presenta un mayor potencial es la del posicionamiento mediante el uso de la cámara. Sin embargo, dado que el procesamiento de imágenes en vivo en los drones es un tema que se sale del alcance de este proyecto y que ya existe otro grupo de investigación trabajando específicamente en implementar un sistema de este tipo, se plantea como trabajo a futuro integrar ese sistema con el desarrollado en este proyecto.

Para la implementación de este proyecto se opta por la utilización de un sistema de posicionamiento relativo ya que, sin considerar la técnica de procesamiento de imágenes, es la única alternativa que se puede implementar con efectividad en un Bebop 2.

Capítulo 4

Implementación

En este capítulo se detallan las características del sistema de exploración implementado, el cual está basado en el paradigma de la programación orientada a agentes para permitir la colaboración de los drones pertenecientes a la flota de exploración. En la sección 4.1 se describe el funcionamiento de la máquina de estados, mientras que en la sección 4.2 se presentan características adicionales que pueden ser activadas de manera opcional al iniciar la ejecución del sistema.

4.1. Máquina de Estados

En esta sección se describe la máquina de estados de la *Figura 4.1*. Cada dron de la flota ejecuta una instancia de la máquina de estados y en ella se contempla la estrategia de exploración a utilizar, la definición del protocolo de comunicación entre los drones y la monitorización del estado de la batería de cada dron.

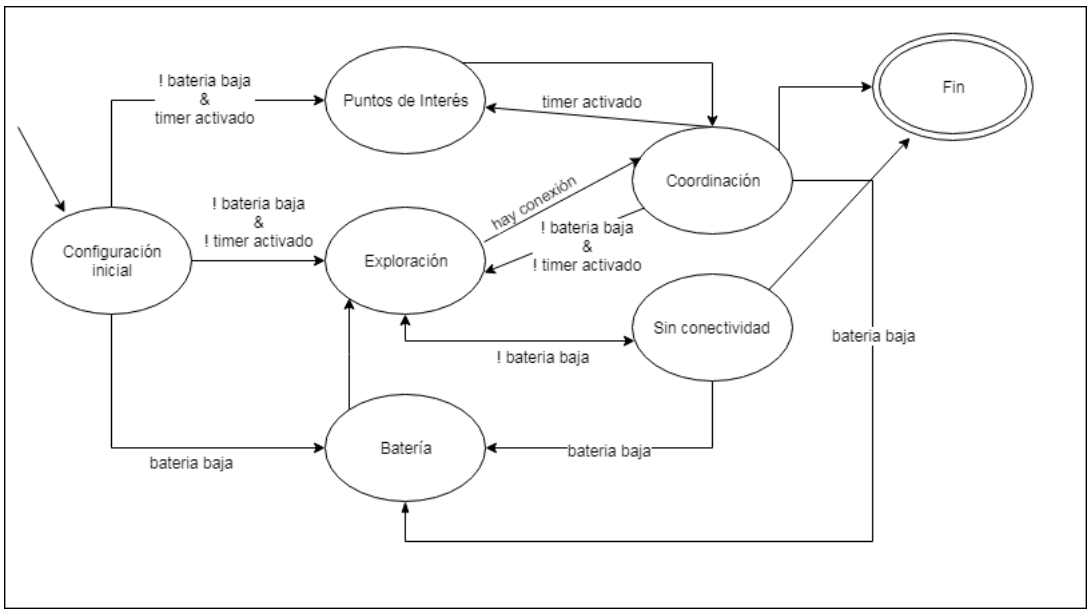

Figura 4.1: Máquina de estados general que modela los estados que ejecuta un dron y las transiciones entre los mismos.

4.1.1. Estado: Configuración inicial

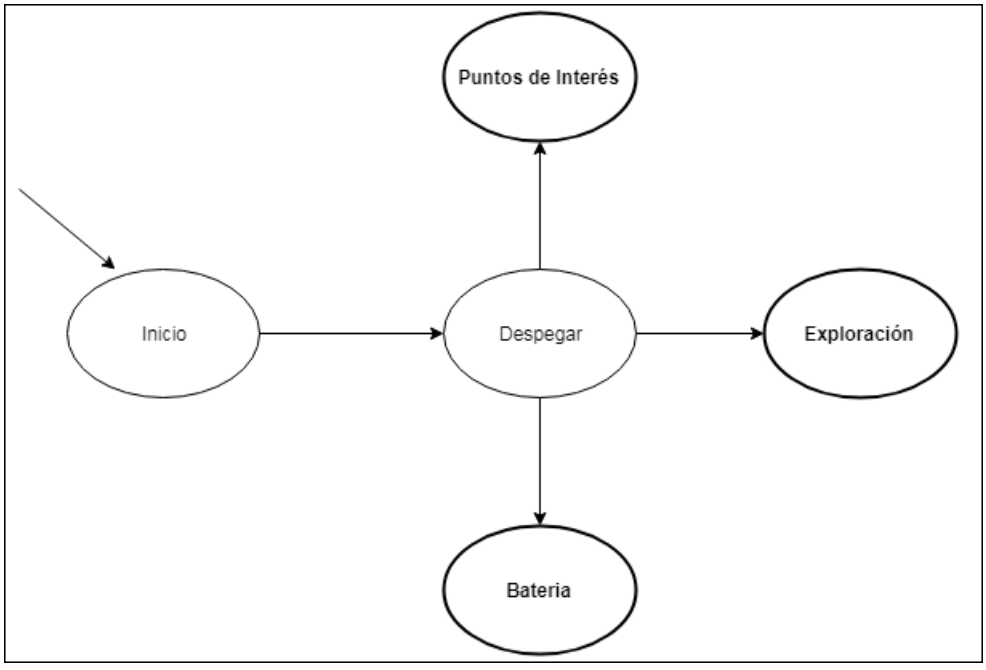

Figura 4.2: Especificación del estado Configuración inicial.

Al comenzar la ejecución del sistema se realiza la carga de los parámetros iniciales en el estado *Inicio*, donde se configuran las propiedades necesarias

para la ejecución del sistema en base a los valores ingresados en el archivo de configuración. El sistema cuenta con varias propiedades parametrizables para su ejecución entre las que se incluyen:

- las dimensiones del territorio a explorar
- las propiedades de la red Wi-Fi a la cual se conecta el dron
- el tiempo total de ejecución del sistema
- el algoritmo de exploración a utilizar
- los puntos de interés a explorar
- la altura a la que se mantiene el vuelo durante la exploración
- la posibilidad de activar funcionalidades complementarias tales como el streaming de video de la exploración

Además en el estado *Inicio*, se ejecuta un algoritmo de sincronización con el fin de coordinar los temporizadores internos de cada dron que forman parte del sistema. Esto permite que el tiempo de misión total de todos los drones sea el mismo, además de coordinar los tiempos en los que se deben visitar los puntos de interés.
Una vez concluida la etapa de inicio se procede a la ejecución del estado *Despegar*, donde se le envía a cada dron perteneciente a la flota de exploración la instrucción de despegue. Finalmente se procede a ejecutar el estado *Exploración* de la máquina de estados general.

4.1.2. Estado: Exploración

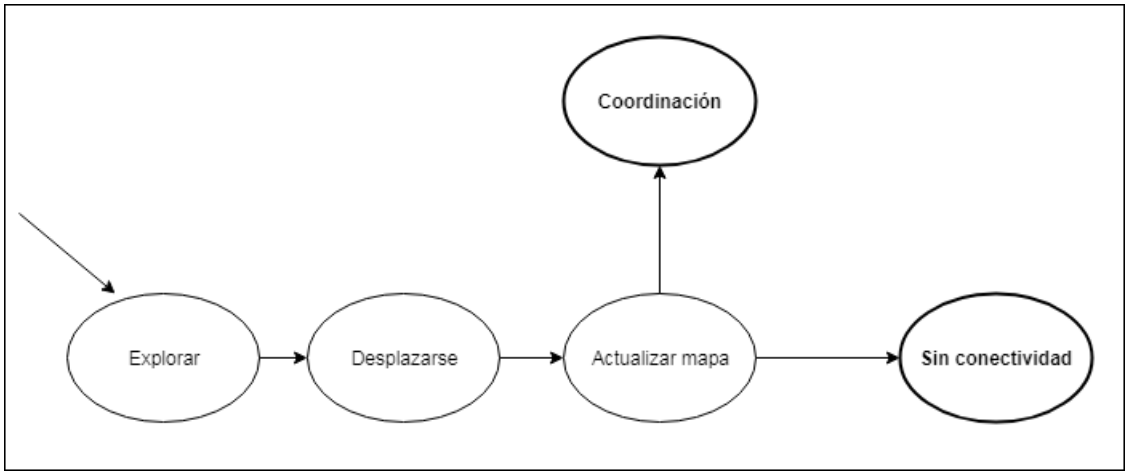

Figura 4.3: Especificación del estado Exploración.

La estrategia de exploración sigue el esquema presentado en la *Figura 4.3*, donde se determina cuál es la siguiente zona a visitar. El hecho de definir los movimientos de forma dinámica a medida que avanza la ejecución del sistema permite que el algoritmo de exploración pueda utilizar los datos recabados en tiempo real para determinar la trayectoria del recorrido del dron.
En el estado *Explorar* se determina cual es el siguiente movimiento a realizar. Para esto se implementaron dos alternativas.
Una estrategia está basada en un algoritmo de tipo greedy, *Algorithm 1*, que elige para visitar la zona vecina que haya sido visitada con mayor antelación o que nunca haya sido visitada.

Algorithm 1 Algoritmo greedy

```
1: procedure SELECCIONARDESTINO
2: para cada zona vecina Z:
3:     timer = obtenerTiempoUltimaVisita(Z)
4:     minimo = obtenerUmbralConfigurado()
5:     if timer < minimo then
6:         minimo = timer.
7:         zonaSeleccionada = Z.
8: return zonaSeleccionada
```

La otra estrategia es idéntica a la anterior pero contempla todas las zonas del territorio, como se puede ver en *Algorithm 2*. Para esto se divide la totalidad del territorio a explorar en 4 regiones y se realiza un control sobre el cubrimiento de todas las regiones del mapa, lo cual le permite al dron desplazarse de forma directa a una región diferente a la región que se encuentra al momento de evaluar su siguiente movimiento.
Una vez determinado el siguiente movimiento a realizar se ejecuta el estado *Desplazarse*, donde se le envía al dron la instrucción de moverse hacia la posición calculada en el estado *Explorar*.

Al finalizar el desplazamiento del dron se abandona el estado *Desplazarse* para ejecutar el estado *Actualizar mapa*, donde se actualiza el mapa interno del dron al registrar la posición actual como visitada. Una vez culminada esta etapa si el dron consigue establecer una conexión con otro miembro de la flota se ejecuta el estado *Coordinación* de la máquina de estados general; en caso contrario se ejecuta el estado *Sin conectividad* de la máquina de estados general.

Algorithm 2 Algoritmo por regiones

```
1: procedure SELECCIONARDESTINO
2: para cada región R:
3:     coverage = obtenerCubrimiento(R)
4:     if coverage < minimoCoverage then
5:         minimoCoverage = coverage
6:         regionSeleccionada = R
7: If minimoCoverage < minimoPermitido
8:     return regionSeleccionada
9: para cada zona vecina Z:
10:    timer = obtenerTiempoUltimaVisita(Z)
11:    if timer < minimo then
12:        minimo = timer
13:        zonaSeleccionada = Z
14: return zonaSeleccionada
```

4.1.3. Estado: Puntos de interés

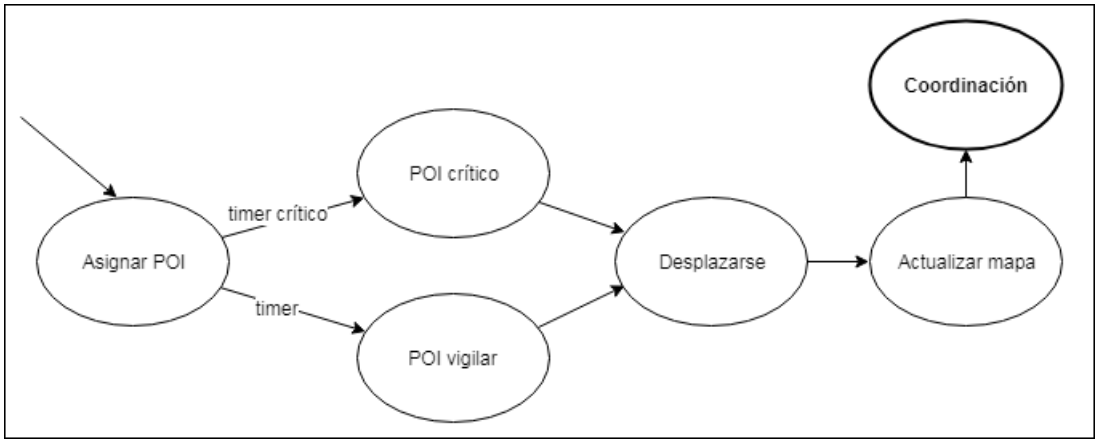

Figura 4.4: Especificación del estado Puntos de interés.

Cada zona del mapa marcada como punto de interés tiene un intervalo de tiempo asociado en el cual debe ser visitado por algún dron. En el caso de que no se cumpla con el requisito de tiempo los drones pertenecientes a la flota de exploración ejecutan el estado *Asignar POI*, en el cual los drones eligen de forma conjunta a uno de ellos como responsable de ir a visitar el punto de interés. Para tomar esta decisión los drones ejecutan un algoritmo de consenso basado en Paxos [16]. Paxos es un algoritmo de consenso basado en el envío

de mensajes que asegura un consenso entre N procesos participantes siempre y cuando se cumplan las siguientes condiciones:

- La cantidad de procesos que fallan en algún momento del algoritmo no sea mayor o igual a N/2.
- Los procesos pueden tener fallos pero no envían información incorrecta de forma intencional (no se producen fallos bizantinos).
- Los procesos se envían mensajes asíncronos que pueden tomar un tiempo indeterminado en llegar o directamente pueden perderse.
- Si un mensaje llega a destino este no es corrupto, o si lo es se puede detectar.
- Todos los procesos pueden enviarles mensajes a los demás de forma directa.
- Durante la ejecución del algoritmo no tiene porqué haber un proceso que actúe de líder, de hecho varios procesos pueden actuar como si fueran el líder, pero eventualmente se debe decidir un líder que dirima situaciones de conflicto.

En la implementación desarrollada del algoritmo todos los procesos actúan como líder hasta el final del proceso. Se define un líder solo en caso de que haya empates en la decisión final tomada y se selecciona como líder al dron que cuente con el número de IP más grande. Al no tener un dron como líder durante la mayor parte de la ejecución del algoritmo de consenso se minimiza la dependencia sobre los drones individuales y se disminuye la probabilidad de error en el algoritmo por problemas de conexión a la red Wi-Fi. Se lista etapas las etapas del algoritmo de consenso.

- **Paso 1:** Los drones que se encuentran disponibles, entendiéndose por disponible que tienen suficiente batería y no se encuentran realizando otras tareas prioritarias, intercambian mensajes anunciando que van a participar en la toma de decisión para el punto de interés correspondiente. Cada dron genera una lista con todos los drones participantes en la toma de decisión. Para evitar inconsistencias en la lista de participantes, la ventana de tiempo en la que los drones intercambian mensajes se mide desde el momento en que el primer dron envía sus mensajes de participación. Esta técnica para sincronizar las ventanas de tiempo en las que se pueden enviar mensajes se aplica también en las demás etapas del algoritmo.

- **Paso 2:** Los procesos participantes intercambian mensajes informando su distancia al punto de interés a atender.
- **Paso 3:** Cada proceso participante determina cual es el dron más cercano al punto de interés y envía un mensaje a los demás proponiendo a ese dron como candidato a vigilar el POI. En caso de existir dos drones que se encuentran a la misma distancia, se elige al que tenga mayor valor de IP.
- **Paso 4:** Cada proceso revisa las propuestas realizadas por los demás y aquella que fue propuesta más veces se convierte en su nueva decisión y la comunica al resto de los drones participantes. En caso de que existan diferentes candidatos se repite el *Paso 4* pero tomando como entrada los resultados del *Paso 4* anterior, aunque este caso es poco probable que suceda.
- **Paso 5:** Los procesos participantes le comunican a los procesos receptores la decisión tomada. Todos los drones pasan a monitorizar al dron elegido para asegurarse de que cumpla su misión.

Dado que las distancias son valores absolutos todos los procesos que siguen el algoritmo de forma correcta deberían elegir al mismo dron en el *paso 3*, esto sucede incluso en caso de empates en las distancias porque en estos casos se dirime en base a las direcciones IP que también son absolutas y además son únicas. Esto implica que para que se elija un dron erróneo en el *paso 4* más de la mitad de los procesos deben de haber fallado en alguna parte del algoritmo. Si bien la probabilidad de error en el algoritmo de consenso es muy baja, se establece como medida de control que todos los drones revisan de forma periódica que los puntos de interés asignados a algún dron efectivamente están siendo vigilados, por lo tanto si se llega a dar un error, los drones lo detectan y se vuelve a realizar el proceso de selección.
Los drones que no fueron seleccionados en el estado *Asignar POI* retornan al estado *Exploración*. El dron seleccionado, en cambio, debe priorizar la visita de la zona que le fue asignada, por lo que debe modificar la forma de seleccionar los desplazamientos a realizar. En caso de que no se haya superado el tiempo en que el punto de interés es considerado como crítico ejecuta el estado *POI Vigilar* donde se realiza un conjunto de pasos similar al del comportamiento descrito en el estado *Exploración*, con la principal diferencia que se consideran como zonas vecinas a visitar solo a aquellas que lo acerquen al punto de interés.

Si se sobrepasa el tiempo establecido para considerar la zona a visitar como crítica sin que haya sido visitada por algún miembro de la flota de exploración, se procede a la ejecución del estado *POI Crítico*, en el cual el dron designado a explorar la zona se dirige a ella directamente. Para determinar el camino a seguir para llegar al destino, se utiliza el *algoritmo A* (A aster)* [14], el cual permite identificar el camino más corto entre dos puntos de una red haciendo uso de la *distancia de Manhattan* ($d((a1, b1), (a2, b2)) = |a2 - a1| + |b2 - b1|$) como función de heurística.
Cuando se visita un punto de interés se reinicia su temporizador, para que la zona vuelva a ser visitada más adelante y se envía un mensaje a los demás drones para que estos también restablezcan su temporizador para ese punto de interés.

4.1.4. Estado: Batería

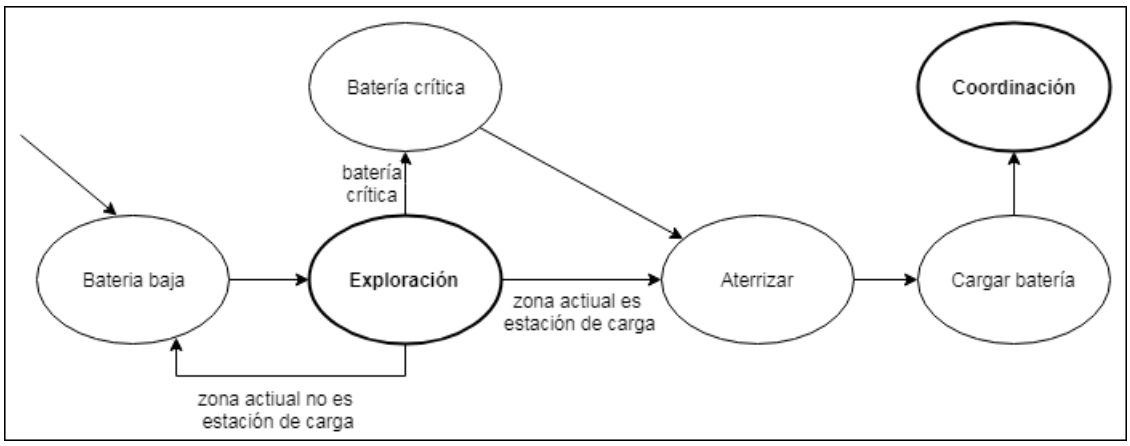

Figura 4.5: Especificación del estado Batería.

Esta sección de la máquina de estados se encarga de controlar el estado de la batería del dron, con el principal objetivo de evitar que el mismo se quede sin carga en la exploración y deba realizar un aterrizaje forzoso.
Se realiza un chequeo del porcentaje de carga de la batería y en caso de que se encuentre por debajo de un umbral configurable, se ejecuta el estado *Batería baja*. Una vez que el dron se sitúa en el estado *Batería baja*, el mismo realiza las tareas del estado *Exploración*, con la diferencia de que para determinar la siguiente zona a visitar, se consideran como válidas sólo las zonas que acerquen al dron a la base de carga.

A su vez, también se realiza un control sobre la distancia entre el dron y la estación de carga y se compara la misma con la cantidad de movimientos necesarios para volver a la estación de carga de forma directa. En caso de que la cantidad de movimientos para volver a la estación de carga sea igual a la cantidad de movimientos restantes, se ejecuta el estado *Batería crítica*, donde el dron abandona la exploración y se dirige a la estación de carga de forma directa.
Para calcular la distancia a la estación de carga y el camino a seguir se hace uso del *algoritmo A* (A aster)* [14]. Una vez que el dron llega a la estación de carga pasa al estado *Aterrizar*, donde se ejecutan las instrucciones para realizar el procedimiento de aterrizaje seguro. Más adelante se ejecuta el estado *Cargar batería*. Cuando se termina la carga de la batería el dron se reincorpora a la flota de exploración.

4.1.5. Coordinación

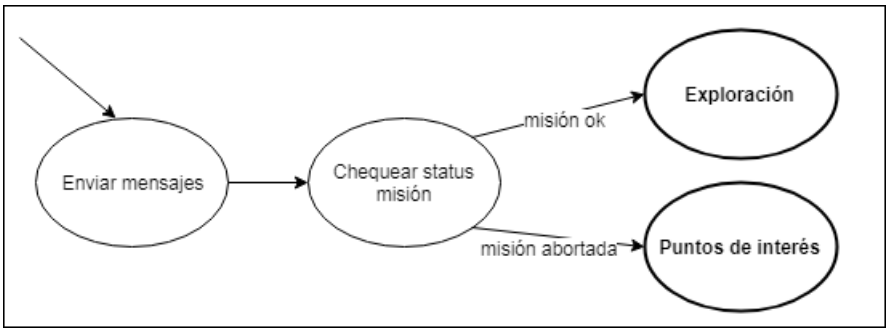

Figura 4.6: Especificación del estado Coordinación.

En esta etapa se realizan las actividades de comunicación de la flota de exploración para optimizar la estrategia a seguir. Cuando un dron se encuentra en el estado *Enviar Mensajes* se comunica por medio de la conexión establecida en la *Configuración inicial* con los demás miembros de la flota, para transmitirle la zona del mapa en la que se encuentra actualmente. De esta forma los demás drones marcan la zona recibida como visitada.
Por otra parte, se controla el estado de la vigilancia de los puntos de interés asignados a otros miembros de la flota al ejecutar el estado *Chequear status misión*, donde todos los drones disponibles buscan establecer una conexión con el dron que tiene la misión de vigilar la zona asignada. Cuando se recibe una respuesta satisfactoria del dron que se encuentra encargado de la misión, se les

envía un mensaje a los demás drones para avisarles que la misión está siendo ejecutada con normalidad. En caso de que ningún miembro de la flota pueda recibir una respuesta por parte del dron con la misión asignada se determina que la misma fue cancelada y se pasa al estado *Puntos de interés* de la máquina de estados general para reasignar el punto de interés a un nuevo dron disponible. En caso de que ninguna misión haya sido cancelada se ejecuta el estado *Exploración* de la máquina de estados general

4.1.6. Sin conectividad

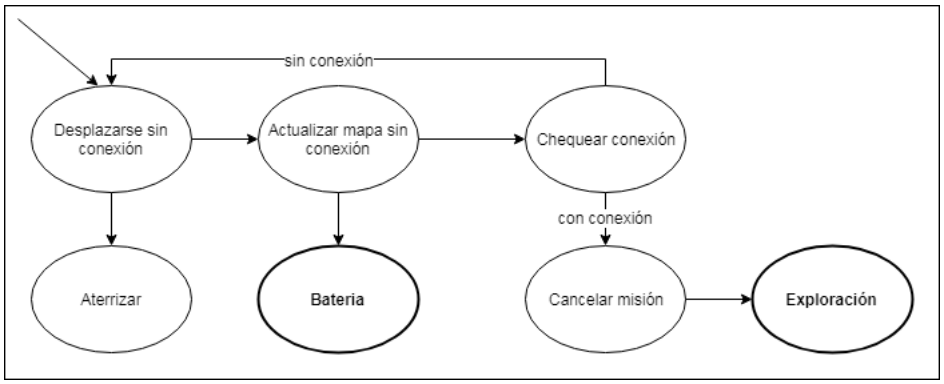

Figura 4.7: Especificación del estado Sin conectividad.

En caso de que el dron no pueda establecer un canal de comunicación con ningún otro miembro de la flota, se declara a sí mismo sin conectividad e inicia la ejecución de los estados que se muestran en la *Figura 4.7*.
El primer estado en ser ejecutado es *Desplazarse Sin Conexión*, donde se sigue una estrategia de movimientos similar a la del estado *Exploración*, con la particularidad de decidir las siguientes zonas a visitar buscando disminuir la distancia con la base, con la finalidad de poder recuperar la conexión con algún dron.
Una vez que el dron haya terminado su desplazamiento, se ejecuta el estado *Actualizar Mapa Sin Conexión*, donde se actualiza la zona como visitada y se guarda la posición para poder enviarla al resto de la flota más adelante, cuando se consiga restablecer la conexión.
En el estado *Chequear conexión* se verifica si se recuperó la conexión, buscando establecer una comunicación con algún otro dron. En caso de no tener éxito se retorna al estado *Desplazarse Sin Conexión*. En cambio, si se obtiene una respuesta de otro dron se abandona el modo *Sin Conectividad* para volver al

comportamiento descrito en la sección 4.1.2.
Si el dron tenía asignado un punto de interés previo a la pérdida de la conectividad se ejecuta el estado *Cancelar misión*, donde se envía un mensaje a los demás drones comunicando que se debe reasignar el punto de interés para posteriormente ejecutar las acciones de *Exploración*.
En caso de no recuperar la conectividad y llegar a la base se ejecuta el estado *Aterrizar* para finalizar la ejecución del sistema. Al igual que cuando se tiene conexión, se realiza un control sobre el porcentaje de carga de la batería y se ejecuta el estado *Batería* de la misma forma que se describe en la sección 4.1.4.

4.1.7. Fin

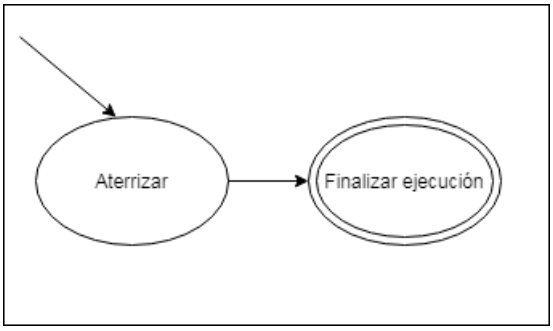

Figura 4.8: Especificación del estado Fin.

Al finalizar la ejecución del sistema se procede al aterrizaje del dron en la base para posteriormente ejecutar el estado *Finalizar ejecución* donde se cierra el canal de comunicación y se registra la información relevante de la ejecución en archivos con formato CSV.

4.1.8. Envío de mensajes

Para que los drones se comuniquen de forma efectiva se implementó un mecanismo de envío de mensajes sobre el protocolo TCP/IP. A continuación se detalla la estructura de los mensajes.

- **message_id**: Identificador del mensaje, de tipo numérico, que se usa para determinar si un mensaje ya fue procesado.
- **state**: Enumerado que determina a que estado el mensaje está dirigido, esto permite separar los mensajes según su funcionalidad.

- **message_type**: Enumerado que identifica el tipo de mensaje en que caso de que un mismo estado deba procesar varios tipos diferentes de mensajes (como por ejemplo el estado Asignar POI).
- **content**: Datos estructurados de tipo JSON, con la información que debe ser procesada.

Cuando se recibe un mensaje este se envía a una cola, donde permanece hasta que la máquina de estados pasa al estado definido en *state*. En ese momento el dron analiza el *message_id* para corroborar que no sea un mensaje ya leído, luego obtiene su contenido. Para los mensajes de alta prioridad se cuenta con un valor extra para el campo *state* que permite que el mensaje sea leído de forma inmediata por el dron, sin importar el estado en el que se encuentre. Se definieron varias rutinas de envío de mensajes entre los drones de la red, las cuales se listan en la *Tabla 4.1*.

Tabla 4.1: Listado de rutinas de envío de mensajes

Rutina	Estado	Descripción
Sincronización	Inicio	Sincronización de los timers y del despegue de los drones al inicio de la ejecución del algoritmo.
Algoritmo de consenso	Asignar POI	Mensajes enviados durante el algoritmo de consenso para asignar POIs mediante el algoritmo Paxos.
Actualización del mapa	Actualizar Mapa	Mensajes enviados luego de que un dron explora un punto del mapa para informarles a los demás drones.
Control de POIs asignados	Chequear Status Misión	Mensajes para controlar el estado de los drones que tienen asignados algún POI a vigilar.
Misión finalizada	Sin estado	Informa a los demás drones del final de una misión, ya sea la exploración de un POI o el final de la ejecución general del algoritmo. La rutina informa si la misión finalizó con éxito o no.

4.2. Funcionalidades complementarias

El sistema desarrollado cuenta con funcionalidades opcionales que se pueden incorporar en función de las necesidades que tenga el usuario final. La

mayoría de estas funcionalidades se encuentran habilitadas por defecto (como el control externo) o poseen una configuración por defecto, como el modo de navegación y la altura de vuelo, ya que mejoran el funcionamiento general del sistema. Las demás funcionalidades, como la transmisión de imágenes en directo y el sistema de auditoría, deben ser habilitadas por el usuario porque no son estrictamente necesarias para el funcionamiento del sistema de exploración y consumen recursos del dron.

4.2.1. Transmisión de imágenes en directo

En caso de querer hacer uso del sistema para la vigilancia de un terreno, puede ser de gran utilidad poder visualizar las imágenes de video generadas por la cámara de los drones desde un centro de control.
Por esta razón se implementa la funcionalidad de poder transmitir la grabación de la exploración en tiempo real. Para hacer uso de esta funcionalidad basta con estar conectado a la misma red inalámbrica que el dron y recibir los datos de imagen enviados mediante una conexión UDP en el puerto 4444. El formato y la calidad de imagen se pueden configurar mediante comandos de pyparrot.

4.2.2. Modo de navegación

Se analizan dos alternativas para la forma de navegación de los drones, una que consiste en realizar todos los movimientos mediante desplazamientos laterales, y otra en la que todos los desplazamientos se ejecutan realizando una rotación y luego una traslación hacia adelante. La elección del modo de navegación tiene grandes implicaciones sobre el posicionamiento de la cámara. La opción de usar solo desplazamientos laterales cuenta con la ventaja de que requiere un solo movimiento por desplazamiento y esto hace que sea la opción más rápida y precisa. La desventaja de este método es que limita el campo de visión de la cámara, ya que esta siempre va a estar apuntando en la misma dirección. La solución a este problema es la de apuntar la cámara directamente hacia abajo, lo cual permite obtener una visión bien definida de un área rectangular que tiene por centro la posición del dron. Esto permite ver a la vez en todas las direcciones pero el área de visión es menor a si la cámara apuntase de forma más tangencial al suelo. Con esta disposición de la cámara, el área de visión del dron queda determinada por los ángulos de visión horizontal θ_h y vertical θ_v de la cámara y la altura de vuelo h, de acuerdo a la fórmula

$$A = 4h^2 \times \tan(\frac{\theta_h}{2}) \times \tan(\frac{\theta_v}{2})$$

Las pruebas realizadas con el Bebop 2 determinan que posee un ángulo de visión de 82°x 53°, por lo que a una altura de vuelo estándar de 10 metros se puede observar de forma simultánea un área de $A = 18m \times 10m = 180m^2$

El otro modo de navegación, que realiza una rotación y luego una traslación hacia adelante en cada desplazamiento, permite tener la cámara en una posición más natural, cercana a los 45°con respecto a la horizontal. Esto da un área de visión mayor y que resulta más natural para una persona que esté viendo las grabaciones. Sin embargo, solo se puede ver un ángulo horizontal θ_h a la vez. Por otra parte, la rotación implica realizar un movimiento extra cada vez que se desplaza el dron, lo que duplica la cantidad de movimientos requeridos. Además, es necesario contemplar un mecanismo que permita determinar con precisión el valor del ángulo de giro, hecho que resulta igual de costoso que determinar con precisión la posición del dron. Estas dificultades hacen que la opción de utilizar giros en los movimientos resulte lenta e imprecisa. En la *Figura 4.9* se ilustran las diferencias entre los dos modos de navegación.

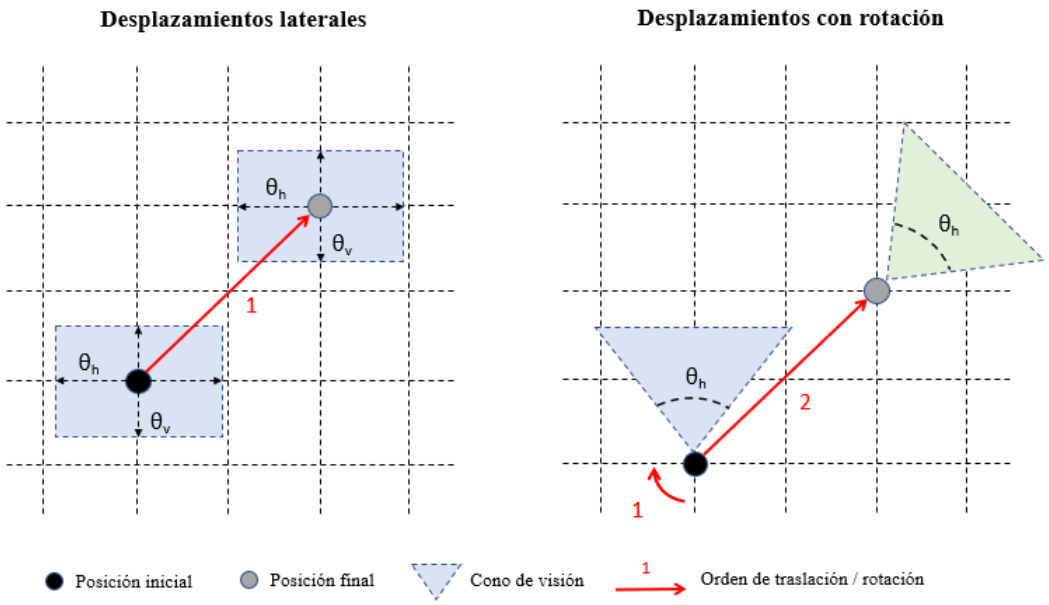

Figura 4.9: Realización de un desplazamiento con el modo de navegación con desplazamiento lateral y el modo de navegación con rotación.

Finalmente se opta por implementar ambas opciones y dejar que los usuarios configuren cual desean utilizar. Sin embargo, durante el resto de este trabajo, se decide usar la opción sin rotación ya que produce resultados más

precisos.

4.2.3. Auditoría de la exploración

Para determinar si una ejecución del sistema cumple con los objetivos establecidos, se incorpora la posibilidad de almacenar en cada dron de la flota un archivo que contiene estadísticas detallando el porcentaje de mapa cubierto, la cantidad de puntos de interés visitados y los tiempos de respuesta a las alertas de los puntos de interés. Esta funcionalidad es particularmente útil para la evaluación experimental detallada en el capítulo 5.

4.2.4. Altura de vuelo

Dependiendo de las características del terreno a explorar puede ser necesario configurar a qué altura se debe realizar la exploración. Configurar la altura de vuelo consiste en asignar el rango de altura permitido en metros.
Para determinar la altura a la que se encuentra el dron se utiliza la información del sensor de altura publicada por el SDK para alturas menores a 5 metros. Para alturas mayores se usa la información del GPS ya que el sensor de altura deja de ser preciso.

4.2.5. Control externo

Si bien uno de los principales objetivos del trabajo realizado es el de generar un sistema de control autónomo, en ocasiones resulta de gran utilidad contar con un sistema de control externo que permita dirigir al dron de forma remota desde una computadora, en particular en las etapas de configuración y pruebas. Debido a esto se implementa un mecanismo que permite controlar el dron de forma remota, permitiendo indicar los movimientos y giros que el dron debe realizar.
A su vez, también se incorporan dos métodos de aterrizaje para casos de emergencia, uno de ellos que realiza un aterrizaje seguro del dron disminuyendo gradualmente la altura de vuelo hasta encontrarse en el suelo y otro mecanismo que consiste en frenar el giro de las hélices apagando el dron. Este último mecanismo solo debe usarse en casos de urgencia extrema. Cabe destacar que los mecanismos de aterrizaje se ejecutan de forma independiente al sistema principal, de forma que si se produce un error en el sistema aún es posible

indicarle al dron que aterrice. Estos mecanismos de emergencia son medidas de seguridad de gran utilidad al momento de verificar el funcionamiento del sistema.

Capítulo 5

Experimentación

En este capítulo se presentan las pruebas que se realizaron sobre el sistema para validar la calidad de los resultados obtenidos, comparando los mismos con otros algoritmos. En la *sección 5.1* se describe la metodología de evaluación utilizada. La *sección 5.2* explica en detalle los casos de prueba y se describen sus escenarios, mientras que en la *sección 5.3* se exponen los resultados obtenidos y el correspondiente análisis de los mismos.

5.1. Metodología de evaluación

A continuación se describe la metodología utilizada para la evaluación experimental de las diferentes variantes de los algoritmos de exploración. Las pruebas se realizaron de forma automatizada con el simulador Sphinx con una flota compuesta por dos drones. La decisión de utilizar una flota de dos drones está basada en que las computadoras de desarrollo utilizadas al momento de ejecutar las simulaciones, solamente tenían capacidad de procesamiento gráfico para ejecutar dos instancias del simulador Sphinx en simultáneo.
En total se cuenta con tres estrategias de exploración diferentes. Las estrategias greedy y por regiones se describen en el capítulo 4.1.2. La tercera estrategia es una caminata aleatoria, es decir, un algoritmo que determina los movimientos del dron de forma completamente aleatoria. Esta última estrategia sirve principalmente como prueba de control para los otros dos algoritmos.
El primer paso para la evaluación es la de definir las métricas a obtener en las simulaciones. Para cada uno de los algoritmos se reportan varias métricas. Las flotas de drones simuladas cuentan con dos objetivos principales: deben

cubrir la zona a explorar de la forma más homogénea y rápida posible y deben atender los puntos de interés con la menor demora posible. Todas las métricas reportadas en las pruebas apuntan a medir la eficacia de los algoritmos en alguno de estos dos objetivos.

Las métricas se pueden dividir en dos categorías, las que refieren al cubrimiento del mapa y las que refieren a la vigilancia de los POI.

Para medir el cubrimiento del mapa se reportan dos métricas. La primera es el porcentaje de cubrimiento del mapa que refiere al porcentaje de territorio explorado al finalizar un experimento dado en función del tiempo. La segunda métrica es el porcentaje del cubrimiento del mapa en un período de tiempo determinado que refiere al porcentaje de territorio explorado hasta el momento en que se registra una medición. Las mediciones para esta segunda métrica se registran cada 10 segundos. Adicionalmente, se registra la cantidad de veces que los drones pasaron por cada ubicación del territorio en cada una de las métricas.

Para medir el nivel de efectividad para atender los POI se reporta la cantidad de veces que se desencadenó una interrupción para atender un POI, si esa interrupción es de tipo normal o crítica, la cantidad de interrupciones que los drones efectivamente atendieron y cuánto tiempo demoraron en atenderla. A todos los registros de esta categoría de métricas se les asocia el POI al que corresponden. Tanto para los POIs normales como para los POI críticos se reportan las siguientes métricas: Cantidad total de POIs atendidos, tiempo de respuesta promedio, peor tiempo de respuesta y mejor tiempo de respuesta.

Tabla 5.1: Métricas relacionadas al cubrimiento del mapa.

Métrica	**Descripción**
Cubrimiento total	Porcentaje de cubrimiento del mapa al finalizar un experimento dado. Una posición del mapa se considera “cubierta”si fue visitada por algún dron hace menos de 10 minutos.
Cubrimiento en función del tiempo	Porcentaje de cubrimiento del mapa para un experimento dado en función del tiempo. Se toma una medición cada 10 segundos. Mismo criterio para considerar si una posición está “cubierta”.

Tabla 5.2: Métricas relacionadas a la vigilancia de los POIs.

Métrica	Descripción
PoiVigilar totales	Cantidad de alertas desencadenadas por algún POI y respondidas por algún dron.
PoiCritico totales	Subconjunto de Triggered PoiVigilar que también desencadenó una alerta de POI Crítico.
Tiempo de respuesta PoiVigilar promedio	Tiempo promedio de respuesta a una alerta de POI.
Tiempo de respuesta PoiCritico promedio	Tiempo promedio de respuesta a una alerta de POI Crítico.
Peor tiempo de respuesta PoiVigilar	Mayor tiempo de respuesta a una alerta de POI.
Peor tiempo de respuesta PoiCritico	Mayor tiempo de respuesta a una alerta de POI Crítico.
Mejor tiempo de respuesta PoiVigilar	Mejor tiempo de respuesta a una alerta de POI
Mejor tiempo de respuesta PoiCritico	Mejor tiempo de respuesta a una alerta de POI Crítico.

5.2. Escenarios de prueba

En esta sección se describen las instancias de prueba y los respectivos escenarios de prueba de cada una de las instancias sobre las que se ejecutan los algoritmos. Se opta por crear escenarios que sean realistas y que representen realidades diferentes para poder evaluar el desempeño de los algoritmos en situaciones variadas. Se diseñan tres escenarios de diferente tamaño y con obstáculos de diferente tipo y distribución. Para cada escenario se crean diez instancias con variaciones aleatorias para evaluar los escenarios con menos sesgo por lo que tenemos en total treinta instancias donde se ejecutarán los algoritmos. Las instancias difieren entre sí en la ubicación de sus puntos de interés y en el tiempo en que deben ser atendidos. De esta forma se puede evaluar para cada escenario si existen puntos que resulten más difíciles de controlar.

Consideraremos una misión como la ejecución de una instancia. Se establece un tiempo de misión de veinte minutos, un poco menor a los 25 minutos de duración de la batería de un dron. Este tiempo de misión permite obtener mediciones de cuánto se puede explorar con una carga de batería dejando un

margen de seguridad de cinco minutos para imprevistos. Todos los drones despegan desde la misma ubicación, la cual representa la base a la que tienen que retornar cuando acabe la misión.
En total cada algoritmo se evalúa cuatro veces en cada misión. Considerando que se tienen 30 instancias a ejecutar para cada algoritmo y ejecutando cuatro veces cada instancia para cada algoritmo se tiene un total de 120 ejecuciones por algoritmo. Para los tres algoritmos a evaluar se tiene un total de 360 ejecuciones que reportarían un tiempo total de 7200 minutos o 120 horas. La decisión de ejecutar cuatro veces cada algoritmo en cada instancia se debe a que la cantidad de tiempo necesaria para realizar cinco o más ejecuciones de cada instancia para los tres algoritmos está fuera de las posibilidades de los equipos informáticos que con los que se dispone.
El primer escenario, que llamaremos *Escenario 1* representado en la *Figura 5.1*, consiste en un galpón de aproximadamente $5000m^2$. En este escenario los principales obstáculos vienen dados por paredes y racks que también forman estructuras en forma de pared. El galpón tiene una única entrada que se encuentra en la periferia del escenario y que representa un punto de interés a vigilar. Su ubicación y la frecuencia con la que debe ser vigilado varía en cada instancia del escenario, ambos valores de forma aleatoria. La frecuencia de atención de los puntos de interés se encuentra en el rango de 1 a 10 minutos.

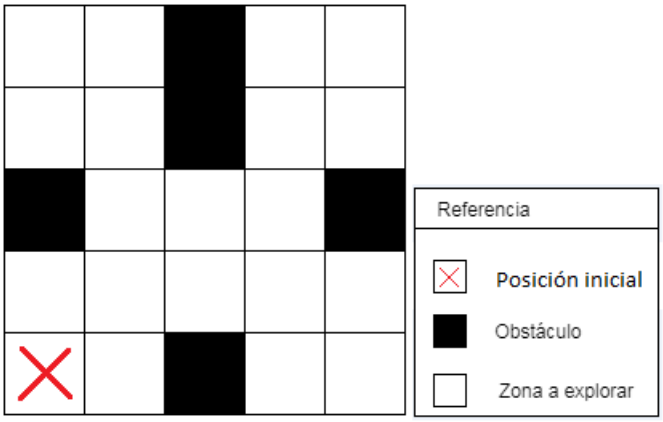

Figura 5.1: Versión base del escenario 1.

El segundo escenario, que llamaremos *Escenario 2* representado en la *Figura 5.2*, consiste en un establecimiento que ocupa una cuadra entera (aproximadamente $10000m^2$) compuesto por tres edificaciones principales y donde el resto del mapa es una zona de patio despejada. Los edificios definen zonas

prohibidas en las que los drones no pueden entrar. En este escenario se cuenta con tres puntos de interés que representan diferentes zonas de ingreso al predio y varían tanto en ubicación como en frecuencia con la que debe ser vigilado de forma aleatoria. La frecuencia de atención de los puntos de interés se encuentra en el rango de 1 a 10 minutos.

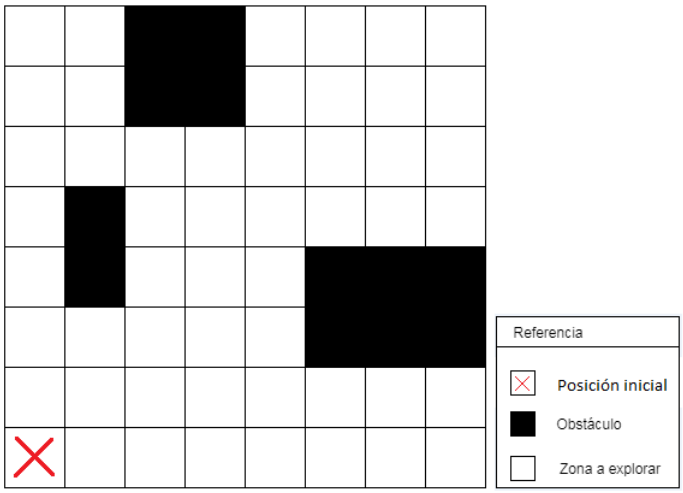

Figura 5.2: Versión base del escenario 2.

El último escenario, que llamaremos *Escenario 3* representado en la *Figura 5.3*, consiste en un campo de 2 hectáreas (aproximadamente $20000m^2$) en el cual solo hay algunos obstáculos aislados en forma de árboles y casas. En este caso se cuenta con cinco puntos de interés que representan corrales, puntos de entrada o la casa del dueño. Estos puntos de interés se ubican de forma aleatoria, con la restricción de que no pueden estar dos o más puntos de interés en una misma ubicación del escenario. La frecuencia de atención de los puntos de interés se encuentra en el rango de 1 a 10 minutos.

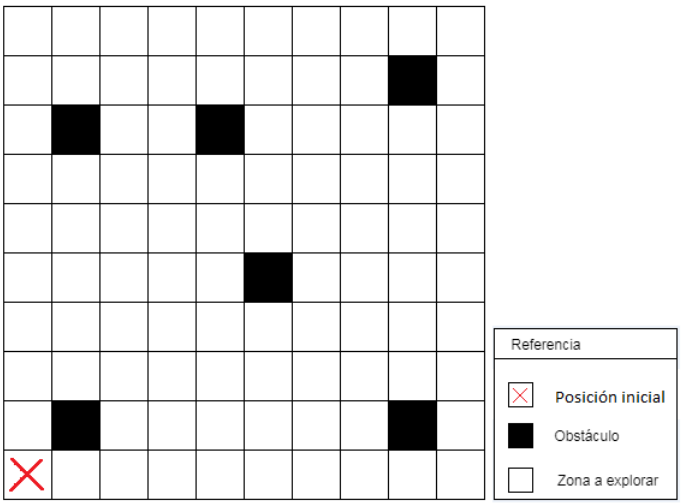

Figura 5.3: Versión base del escenario 3.

Por último se definen los parámetros de configuración con los que se ejecutan los algoritmos. La altura de vuelo se define en 10 metros ya que a esta altura se pueden esquivar la mayoría de los obstáculos (solo hay que preocuparse por obstáculos mayores y estáticos como paredes, árboles o edificios) pero a su vez es lo suficientemente cercano al suelo para no perder calidad de imagen de la cámara y poder vigilar la zona de forma efectiva. Además, se usa el modo de vuelo sin rotación como se describe en el capítulo 4.2.2. Para los drones Bebop 2 una altura de vuelo de 10 metros y con la posición de la cámara que implica el modo de vuelo sin rotación se tiene visión simultánea sobre un área de 10x18 metros. Para maximizar el área recorrida por los drones en cada movimiento, cada uno de los escenarios se representa como una matriz compuesta por rectángulos de 10x18 metros. Dada esta unidad de medición y las dimensiones de los escenarios 1, 2 y 3, cada escenario queda representado como una matriz de 5x5 unidades, 8x8 unidades y 10x10 unidades, respectivamente.

En las tablas 5.3 a 5.7 se presentan las características mencionadas de la evaluación experimental.

Tabla 5.3: Algoritmos evaluados.

Algoritmo	Descripción
Caminata aleatoria	Este algoritmo determina los movimientos del dron de forma completamente aleatoria. Sirve principalmente como prueba de control.
Greedy	Un algoritmo greedy que determina el siguiente movimiento del dron en base al tiempo que pasó desde la última vez que un dron visitó los puntos adyacentes a la posición actual en el mapa.
Por regiones	Un algoritmo que divide el mapa en regiones. Un dron explora una misma región hasta que esta se encuentre mayormente explorada o hasta que se detecta que hay otras regiones por las que la flota no pasa desde hace mucho tiempo. Luego cambia de región y repite el proceso.

Tabla 5.4: Descripción de los escenarios

Atributo	Escenario 1	Escenario 2	Escenario 3
Descripción	Galpón de tamaño pequeño	Predio de tamaño mediano que ocupa una cuadra entera	Terreno de tamaño grande
Dimensiones físicas	$5000m^2$	$10000m^2$	$20000m^2$
Representación	Grilla de 5x5 zonas	Grilla de 8x8 zonas	Grilla de 10x10 zonas
Puntos de interés	1	3	5
Obstáculos	En forma de paredes	En forma de edificios	En forma de árboles, casas, etc

Tabla 5.5: Descripción de instancias.

Atributo	Descripción
Instancias por escenario	10
Cantidad de ejecuciones por instancia y por algoritmo	4
Posición de los POI	Aleatoria
Puntos de interés	1 a 5
Frecuencia de atención de los POI	Aleatoria entre 1 y 10 minutos

Tabla 5.6: Datos paramétricos.

Atributo	Descripción
Cantidad de drones	2
Altura de vuelo	10 metros
Tamaño de la unidad de vuelo	10x18 metros
Tiempo de misión	20 minutos
Zona de despegue	Una única base en la posición (0,0) de cada mapa

Tabla 5.7: Especificaciones técnicas.

Atributo	Descripción
Tiempo total de simulación	120 horas
Sistema operativo	Ubuntu 16.04 64 bits
Procesador	Intel® Core™ i7-4510U CPU @ 2.00GHz
Memoria RAM	8GB
Tarjeta gráfica	Intel® Haswell Mobile
Simulador	Sphinx 0.29.1, Gazebo 7.0.1

Una vez establecidos todos los parámetros de ejecución, se definen las métricas a tener en cuenta durante la evaluación experimental y se implementan los programas necesarios para registrarlas.

5.3. Resultados

Al aplicar el *Test de Friedman* sobre los conjuntos de datos obtenidos se refleja la superioridad del algoritmo *greedy* sobre los algoritmos *por regiones* y la *caminata aleatoria*. Por otra parte el algoritmo por regiones obtiene mejores resultados para cubrir el territorio que la caminata aleatoria pero no para responder a los POIs. La *tabla 5.8* muestra los resultados del test de Friedman para todos los escenarios.

Tabla 5.8: Resultados del Test de Friedman sobre todos los escenarios.

Métrica	Algoritmos		
	Greedy	**Por regiones**	**Caminata aleatoria**
Cubrimiento de territorio explorado	1.09	2.30	2.62
Tiempo de respuesta PoiVigilar promedio	1.03	2.8	2.16
Tiempo de respuesta PoiCritico promedio	1.03	2.8	2.16

5.3.1. Exploración

En cuanto al cubrimiento del mapa en función del tiempo, todos los algoritmos mostraron la misma tendencia: el porcentaje de cubrimiento crece progresivamente hasta alcanzar un cierto límite y luego se mantiene relativamente estable hasta el final de la ejecución. Este límite varía notoriamente en cada escenario con el tipo de algoritmo utilizado. En las figuras 5.4, 5.5 y 5.6 se puede observar el cubrimiento promedio del mapa en función del tiempo para todas las instancias de todos los escenarios y para los algoritmos greedy, por regiones y la caminata aleatoria respectivamente. En la figura 5.7 se muestran el cubrimiento promedio por escenario y por algoritmo en función del tiempo.

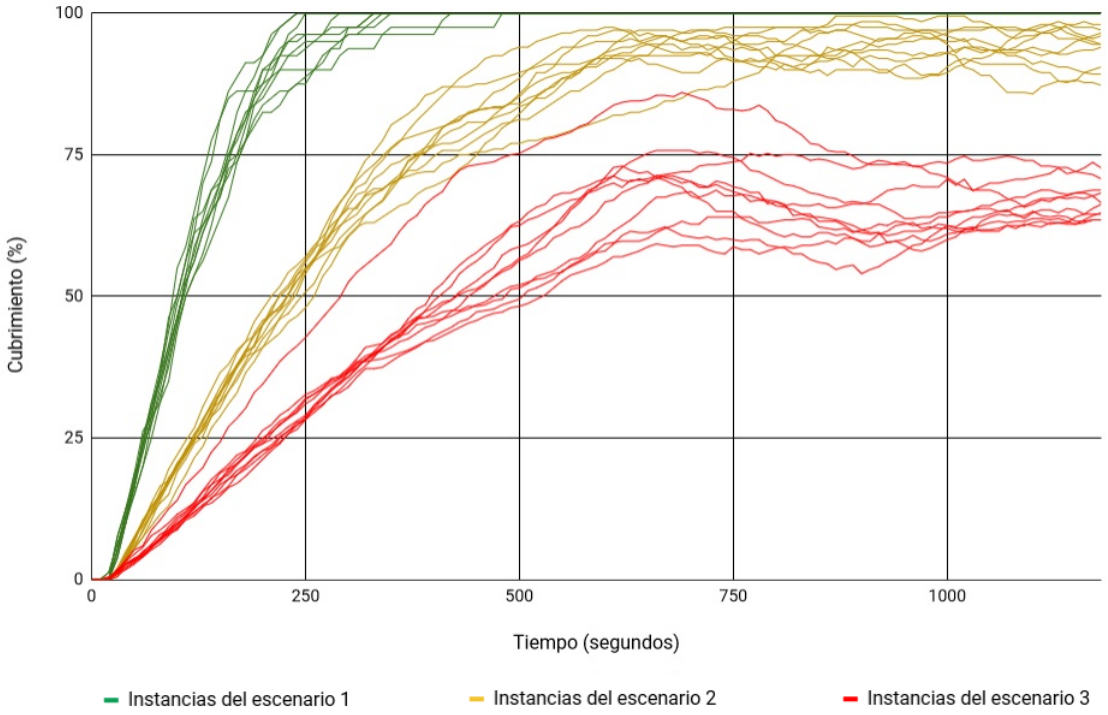

Figura 5.4: Cubrimiento del mapa promedio por instancia en función del tiempo para el algoritmo Greedy

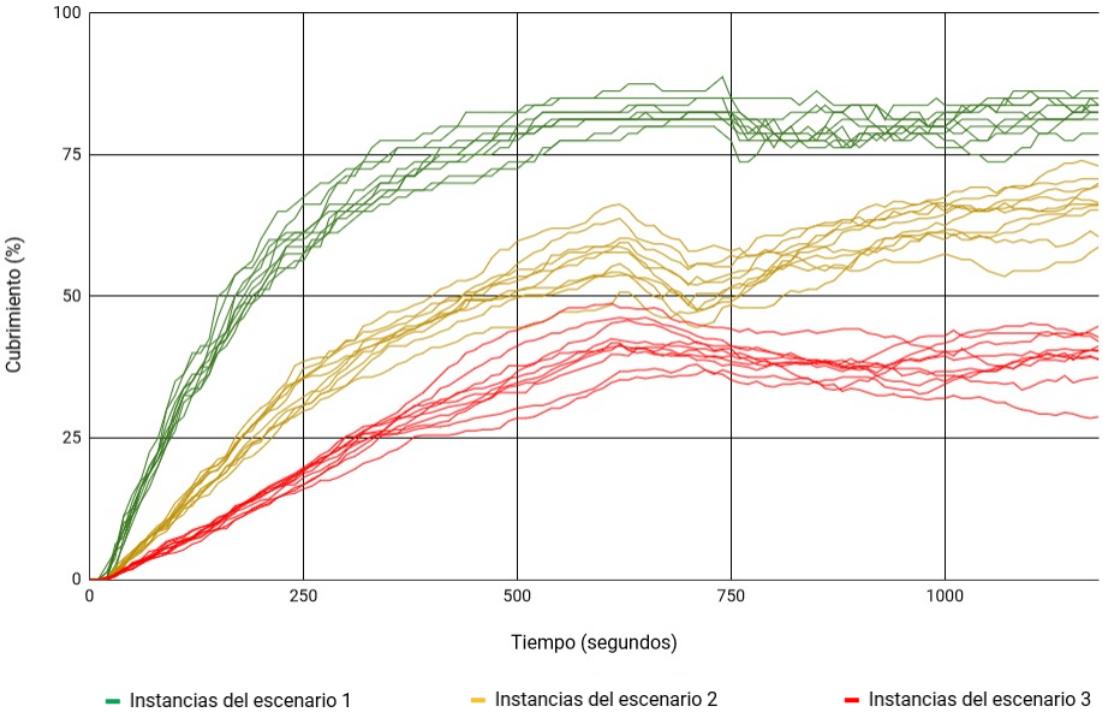

Figura 5.5: Cubrimiento del mapa promedio por instancia en función del tiempo para el algoritmo por regiones

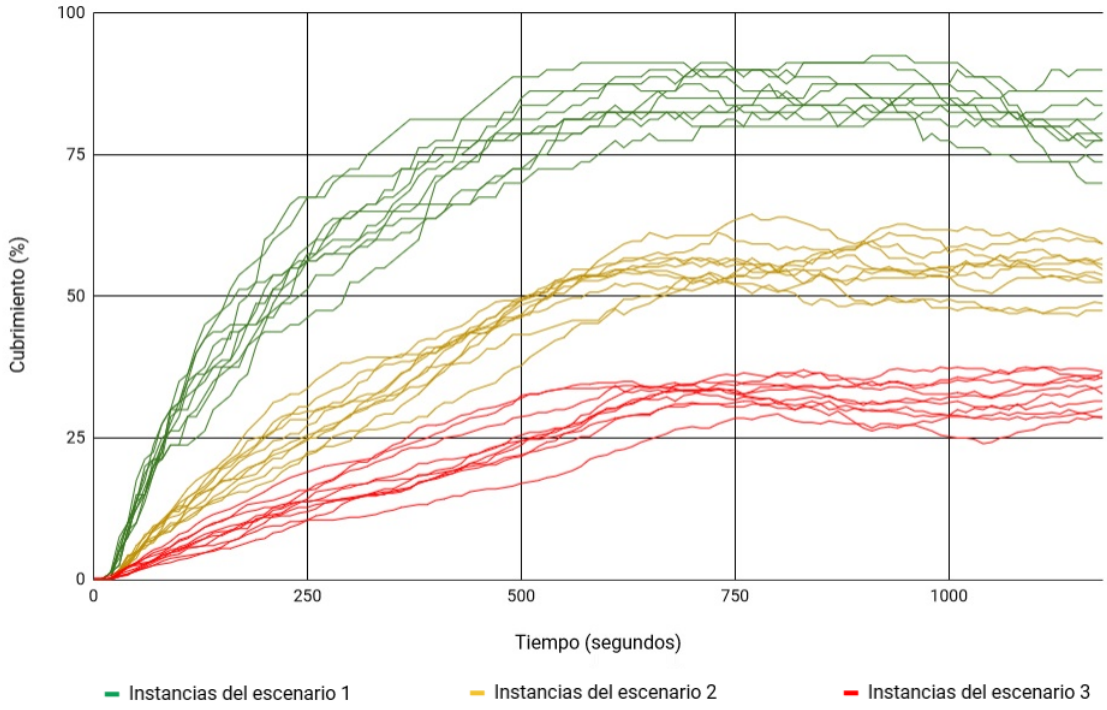

Figura 5.6: Cubrimiento del mapa promedio por instancia en función del tiempo para la caminata aleatoria

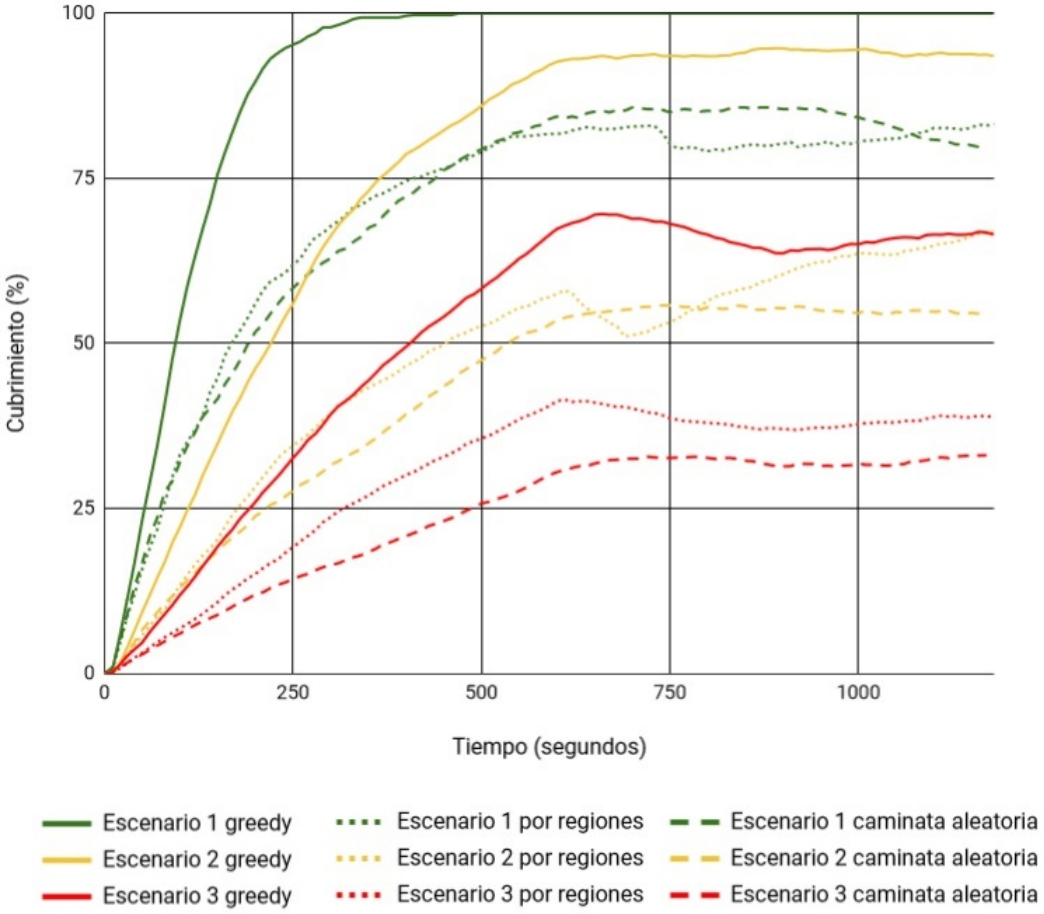

Figura 5.7: Cubrimiento del mapa promedio por escenario y por algoritmo en función del tiempo

No se observan grandes diferencias entre las instancias pertenecientes a un mismo escenario, por lo que la ubicación de los POIs no parece afectar el cubrimiento general de la zona. También se destaca que en todos los escenarios el algoritmo greedy tiene claramente un rendimiento superior: en el escenario 1 directamente consigue un cubrimiento perfecto del 100.00 % en todas las instancias y en el escenario 2 obtiene un cubrimiento casi perfecto del 93.50 % en promedio. El algoritmo por regiones consigue resultados claramente inferiores pero aún así son levemente mejores que los de la caminata aleatoria. Las tablas 5.9, 5.10 y 5.11 resumen los resultados para las instancias de los escenarios 1, 2 y 3 respectivamente.

Tabla 5.9: Resultados de cubrimiento finales para el escenario 1 sobre 40 ejecuciones.

Algoritmo	Cubrimiento promedio	Desviación Estándar	Coeficiente de variación
Greedy	100.00 %	0.00 %	0.00
Por regiones	83.25 %	5.00 %	0.06
Caminata aleatoria	79.75 %	12.40 %	0.16

Tabla 5.10: Resultados de cubrimiento finales para el escenario 2 sobre 40 ejecuciones.

Algoritmo	Cubrimiento promedio	Desviación Estándar	Coeficiente de variación
Greedy	93.50 %	8.64 %	0.09
Por regiones	66.90 %	16.48 %	0.25
Caminata aleatoria	54.12 %	14.06 %	0.26

Tabla 5.11: Resultados de cubrimiento finales para el escenario 3 sobre 40 ejecuciones.

Algoritmo	Cubrimiento promedio	Desviación Estándar	Coeficiente de variación
Greedy	66.46 %	8.07 %	0.12
Por regiones	41.23 %	15.39 %	0.37
Caminata aleatoria	32.31 %	11.49 %	0.34

5.3.2. Puntos de interés

Cuando se analizan los tiempos de respuesta de los POIs se obtienen conclusiones similares, aunque con algunas diferencias destacables. El algoritmo greedy sigue siendo el que obtiene los mejores resultados, pero por otra parte ya no parece haber muchas diferencias entre el algoritmo por regiones y la caminata aleatoria. Además, no se obtienen resultados tan consistentes entre las instancias de un mismo escenario como sucede en el caso de la cobertura. Al comparar los resultados obtenidos para cada escenario se destaca que en función de la distribución de los POIs para cada instancia las métricas de tiempo de respuesta se ven afectadas. Sin embargo al analizar la distribución

de los POIs en cada una de las instancias no es posible determinar cuál tipo de disposición es el que genera mejores resultados. En las tablas 5.12, 5.13 y 5.14 se reportan los resultados obtenidos para la métrica tiempo de respuesta promedio para la atención de un POI.

Tabla 5.12: Tiempo de respuesta promedio de atención a un POI para el escenario 1 sobre 40 ejecuciones.

Algoritmo	**Tiempo promedio**	**Desviación estándar**	**Coeficiente de variación**
Greedy	65.74	15.29	0.23
Por regiones	104.63	26.34	0.25
Caminata aleatoria	78.35	19.72	0.25

Tabla 5.13: Tiempo de respuesta promedio de atención a un POI para el escenario 2 sobre 40 ejecuciones.

Algoritmo	**Tiempo promedio**	**Desviación estándar**	**Coeficiente de variación**
Greedy	93.89	13.90	0.15
Por regiones	132.92	24.67	0.19
Caminata aleatoria	112.85	17.23	0.15

Tabla 5.14: Tiempo de respuesta promedio de atención a un POI para el escenario 3 sobre 40 ejecuciones.

Algoritmo	**Tiempo promedio**	**Desviación estándar**	**Coeficiente de variación**
Greedy	114.18	13.83	0.12
Por regiones	179.19	23.56	0.13
Caminata aleatoria	158.01	16.88	0.11

Los resultados reflejan que el algoritmo greedy es, en la gran mayoría de los casos, el que consigue mejores tiempos de respuesta promedio. La diferencia en el tiempo de respuesta entre el algoritmo greedy y el resto se incrementa con el tamaño del mapa, además de que greedy obtiene resultados más consistentes en las instancias grandes, lo que sugiere que ante situaciones más complejas este algoritmo es más adaptable y escalable.

Por otra parte resulta interesante comparar los desempeños de los algoritmos por regiones y la caminata aleatoria, ya que en gran parte de los casos la caminata aleatoria es el que obtiene un mejor desempeño para atender POIs. Una posible explicación a este fenómeno es que el algoritmo por regiones tiende a dirigir a todos los drones de la flota a una misma zona y entonces, cuando surge una alerta de POI en una zona que no sea en la que se concentra la flota, se deben recorrer distancias mayores para llegar al POI. Estos resultados indican que el algoritmo por regiones no es una buena opción para la vigilancia de POIs. En las tablas 5.15, 5.16 y 5.17 se reportan los resultados obtenidos para la métrica tiempo de respuesta promedio para la atención de un POI crítico.

Tabla 5.15: Tiempo de respuesta promedio de atención a un POI crítico para el escenario 1 sobre 40 ejecuciones.

Algoritmo	**Tiempo promedio**	**Desviación estándar**	**Coeficiente de variación**
Greedy	27.68	17.13	0.62
Por regiones	52.49	16.94	0.32
Caminata aleatoria	36.76	14.67	0.40

Tabla 5.16: Tiempo de respuesta promedio de atención a un POI crítico para el escenario 2 sobre 40 ejecuciones.

Algoritmo	**Tiempo promedio**	**Desviación estándar**	**Coeficiente de variación**
Greedy	45.29	17.14	0.38
Por regiones	75.32	14.98	0.20
Caminata aleatoria	67.56	14.55	0.22

Tabla 5.17: Tiempo de respuesta promedio de atención a un POI crítico para el escenario 3 sobre 40 ejecuciones.

Algoritmo	**Tiempo promedio**	**Desviación estándar**	**Coeficiente de variación**
Greedy	58.80	13.40	0.23
Por regiones	121.94	14.47	0.12
Caminata aleatoria	104.66	14.46	0.14

Al comparar los tiempos de respuesta promedio de los algoritmos ante un POI crítico se obtienen resultados análogos a los de los POI Vigilar. El algoritmo greedy se mantiene como el que ofrece mejores resultados. Esta situación también se reitera si en lugar de analizar el tiempo de respuesta promedio de una ejecución se analiza el peor tiempo de respuesta de una ejecución. En la *Tabla 5.18* se reportan los resultados obtenidos para la métrica tiempo de respuesta en el peor caso de atención a un POI, mientras que en la *Tabla 5.19* se reportan los resultados para los POIs críticos.

Tabla 5.18: Tiempo de respuesta en el peor caso de atención a un POI.

Algoritmo	**Escenario 1**	**Escenario 2**	**Escenario 3**
Greedy	120.56 s	170.06 s	179.80 s
Por regiones	156.83 s	196.43 s	319.37 s
Caminata aleatoria	147.08 s	253.23 s	310.89 s

Tabla 5.19: Tiempo de respuesta en el peor caso de atención a un POI Crítico.

Algoritmo	**Escenario 1**	**Escenario 2**	**Escenario 3**
Greedy	65.02 s	120.48 s	119.23 s
Por regiones	96.83 s	136.43 s	266.03 s
Caminata aleatoria	87.08 s	201.03 s	258.17 s

Capítulo 6

Conclusiones y trabajo futuro

En esta sección se presentan las conclusiones extraídas del trabajo realizado. En la sección 6.1 se mencionan los objetivos cumplidos y los desafíos encontrados y en la sección 6.2 se presentan líneas de trabajo a futuro.

6.1. Conclusiones

Durante el transcurso de este trabajo se buscó desarrollar un sistema de navegación autónomo, multiobjetivo, enfocado en una flota de drones. Este sistema se debía enfocar en dos objetivos principales: la exploración de un territorio y la vigilancia de puntos de interés (o POIs) ubicados dentro del territorio a explorar. Para cumplir con los objetivos planteados se implementó un sistema que sigue el paradigma de agentes. El sistema de navegación se implementó en base a una máquina de estados. Esta permite distintas configuraciones que ofrecen la posibilidad de utilizar diversos algoritmos de exploración para poder evaluar el rendimiento individual de cada uno.
El sistema fue desarrollado para ejecutarse en una flota de drones modelo Parrot Bebop 2. El código fuente del sistema de navegación y la máquina de estados en la que se basa se implementó utilizando el lenguaje de programación Python, el cual permite reutilizar código en dispositivos cuyos sistemas operativos presenten distintas arquitecturas. Para controlar los drones Parrot Bebop 2 se utilizó la librería pyparrot (también escrita en Python) la cual implementa una interfaz que permite la comunicación del sistema de navegación con las funcionalidades disponibles de los drones Bebop 2. Aunque este trabajo haya sido desarrollado para un modelo de dron en particular, tanto el sistema

de navegación como la máquina de estados están implementados de tal forma que se pueden adaptar con facilidad a otros modelos de drones. Ésta facilidad viene dada porque la interacción entre el sistema de navegación y los drones se encuentra centralizada en una única interfaz que puede adaptarse para que funcione con otros modelos. Las pruebas de simulación se realizaron utilizando el simulador Sphinx para drones de la marca Parrot.

Una de las principales contribuciones de este proyecto fue la implementación del sistema de navegación , el cual incluye varias funcionalidades extra que pueden aprovecharse para diferentes casos de uso. Dentro de las funcionalidades extra se incluyen transmisión de video mediante streaming, un sistema para manejar el dron a control remoto por medio del teclado de una computadora, control de altura y sistemas de aterrizaje de emergencia. Otro aporte importante fue la creación de un ambiente de desarrollo para el lenguaje de programación Python que permite la implementación de programas que se ejecuten directamente en un Parrot Bebop 2. En particular, este ambiente permite desarrollar programas desde dispositivos con cualquier tipo de arquitectura sin necesidad de realizar tareas de cross-compile para instalar esos programas en los drones. El hecho de tener este ambiente de desarrollo puede resultar de gran utilidad para posteriores proyectos implementados en Bebop 2.

Durante el desarrollo de este trabajo se encontraron dificultades asociadas a los Bebop 2 que obligaron a realizar modificaciones al sistema que se tenía pensado implementar originalmente. La primera dificultad fue la de desarrollar programas que pudieran ejecutarse directamente en el dron, ya que los Bebop 2 están originalmente diseñados para ser controlados por algún dispositivo externo. Este problema pudo solucionarse con la implementación del ambiente de desarrollo en Python. Otro problema encontrado fue el de crear una red adhoc entre los drones como se tenía planeado originalmente ya que los Bebop 2 no cuentan con los firmwares necesarios para soportar este modo de red. Para solucionar este problema se optó por usar una red de tipo Wi-Fi en su lugar, la cual sí cuenta con soporte en los Bebop 2. Por último, los drones Bebop 2 no cuentan con sensores que permitan crear un sistema de geo-localización y esto afecta la precisión de sus movimientos. Después de analizar diferentes alternativas se concluyó que la mejor opción es la de geo-localizar a los drones por medio de las imágenes de su cámara. Sin embargo, se optó por no implementar este sistema ya que ya hay otro grupo de investigación trabajando sobre este mismo tema. A modo de respetar el alcance del presente trabajo

se decidió implementar un sistema de geo-localización simple por movimiento relativo.
La evaluación experimental realizada sobre el sistema de navegación desarrollado comparó los resultados de tres algoritmos de exploración diferentes. Estos fueron: un algoritmo greedy, un algoritmo que divide el territorio a explorar en regiones y una caminata aleatoria que sirve a modo de prueba de control. Los resultados de la evaluación muestran que el algoritmo greedy es el que obtiene los mejores resultados en todos los escenarios planteados, tanto para cubrir el territorio a explorar como para vigilar a los POI. En cuanto a los otros dos algoritmos, el algoritmo por regiones mostró ser mejor que la caminata aleatoria para cubrir el territorio pero no así para vigilar a los POIs.

6.2. Trabajo futuro

El proyecto realizado genera varias líneas de investigación que pueden ser abordadas en futuros trabajos.
En primera instancia y como se describió en la sección *Conclusiones*, una de las principales contribuciones del presente trabajo es el de crear un ambiente de desarrollo para el lenguaje de programación Python que permite la implementación de programas que se ejecuten directamente en un Parrot Bebop 2. Se considera que la creación del ambiente mencionado puede permitir un desarrollo a mayor velocidad de nuevas aplicaciones en este modelo de drones e impulsar nuevas investigaciones utilizando flotas de drones.
Centrando el análisis de posibles trabajos a realizar relacionados a extender o mejorar el presente trabajo, se destaca la necesidad de buscar alternativas para mejorar la precisión de la ubicación del dron en el territorio a explorar. Se considera que este punto es el principal obstáculo a resolver para conseguir una solución factible de ser implementada sin inconvenientes sobre una flota de drones Parrot Bebop 2. Las limitaciones de hardware de los drones hacen que la mejor alternativa sea la posibilidad de emplear su cámara para medir distancias por medio de procesamiento de imágenes. Al existir actualmente otro proyecto de grado que investiga la posibilidad de realizar la geo-localización mediante el procesamiento de imágenes se plantea combinar los resultados de ese otro proyecto con los del presente trabajo para obtener un sistema más completo. Otra vía de investigación es la de incorporar sensores adicionales al dron Parrot Bebop 2 con el fin de mejorar sus capacidades en general y en

particular para mejorar la precisión de la geo-localización.
Por otra parte, existe la posibilidad de implementar la solución desarrollada en otra clase de drones, si bien la misma fue implementada considerando las características de los drones Parrot Bebop 2, el diseño de la arquitectura del sistema permitió centralizar las operaciones que dependen del modelo específico del dron, manteniéndose la máquina de estados independiente.
Finalmente, se considera que otra tarea que requiere investigación es la respuesta del dron frente a factores externos, como pueden ser obstáculos dinámicos y la evasión de los mismos, factores climáticos como el viento, y analizar como estos factores afectan al movimiento del dron.

Referencias bibliográficas

[1] Amarjot Singh, Devendra Patil, S. O. Eye in the sky: Real-time drone surveillance system (dss) for violent individuals identification using scattternet hybrid deep learning network. *IEEE Computer Vision and Pattern Recognition (CVPR) Workshops*, 2018.

[2] Andrew Kopeikin, S. S. P. and How, J. P. Handbook of unmanned aerial vehicles. *Springer Science Business Media Dordrecht*, 2015.

[3] Bekmezci, I., Sahingoz, O. K., and Temel, S. Flying ad-hoc networks (fanets): A survey. *SciVerse ScienceDirect*, 2013.

[4] Broadcom. Especificación de tarjeta de red bcm4360. https://www.broadcom.com/products/wireless/wireless-lan-infrastructure/bcm4360#overview. [Online]. Accedido Noviembre 2018.

[5] Busybox. Busybox. https://busybox.net/. [Online]. Accedido Agosto 2018.

[6] Cacace J, Finzi A, L. V. Multimodal interaction with multiple co-located drones in search and rescue missions. *Computers and Operations Research*, 2016.

[7] Cesare, K. Multi-uav exploration with limited communication and battery. *Computers and Operations Research*, 2015.

[8] CNET. Especificación técnica del parrot bebop 1. https://www.cnet.com/products/parrot-bebop-drone/specs/. [Online]. Accedido Abril 2019.

[9] Drones, P. Análisis de parrot bebop 2. https://planetadrones.es/bebop-2/. [Online]. Accedido Agosto 2018.

[10] Foundation, P. S. netifaces. https://pypi.org/project/netifaces/. [Online]. Accedido Abril 2019.

[11] FreeFlightPro. Freeflight pro. https://www.parrot.com/es/freeflight-pro#pilota-tu-dron-parrot-con-freeflight-pro. [Online]. Accedido Julio 2018.

[12] Gazebo. Gazebo. http://gazebosim.org/. [Online]. Accedido Abril 2019.

[13] Grøtli, E. I. and Johansen, T. A. Path planning for uavs under communication constraints using splat! and milp. *Journal of Intelligent and Robotic Systems*, 2012.

[14] Idelab. Algoritmo a*. http://idelab.uva.es/algoritmo. [Online]. Accedido Octubre 2018.

[15] Ju C, Park S, P. S. S. H. A haptic teleoperation of agricultural multi-uav. *International Conference on Intelligent Robots and Systems*, 2017.

[16] Lamport, L. Paxos made simple. *SIGACT News*, 2001.

[17] Manikanta Kotaru, Kiran Joshi, D. B. S. K. Spotfi: Decimeter level localization using wifi. *Stanford University*, 2015.

[18] Mozilla. Servicio de localización por wifi de mozilla. https://location.services.mozilla.com/. [Online]. Accedido Octubre 2018.

[19] Mufalli, Batta, and Nagi. Simultaneous sensor selection and routing of unmanned aerial vehicles for complex mission plans. *Computers and Operations Research*, 2012.

[20] Parrot. Especificación técnica del parrot mambo fpv. https://www.parrot.com/global/drones/parrot-mambo-fpv#parrot-mambo-fpv-details. [Online]. Accedido Abril 2019a.

[21] Parrot. Especificación técnica del parrot swing. https://www.parrot.com/global/minidrones/parrot-swing#parrot-swing. [Online]. Accedido Abril 2019b.

[22] Parrot. Especificación técnica del parrot bebop 2. https://www.parrot.com/es/drones/parrot-bebop-2#parrot-bebop-2-details. [Online]. Accedido Julio 2018.

[23] plc, A. H. Arm. https://www.arm.com/products/silicon-ip-cpu. [Online]. Accedido Agosto 2018.

[24] Project, G. Cross-compilation. https://www.gnu.org/software/automake/manual/html_node/Cross_002dCompilation.html. [Online]. Accedido Abril 2019.

[25] PyParrot. Documentación oficial de pyparrot. https://pyparrot.readthedocs.io/en/latest/. [Online]. Accedido Agosto 2018.

[26] Qemu. Página oficial de qemu. https://www.qemu.org/. [Online]. Accedido Agosto 2018.

[27] RaspberryPi. Arquitectura raspberry pi. https://www.macs.hw.ac.uk/~hwloidl/Courses/F28HS/slides_RPi_arch.pdf. [Online]. Accedido Agosto 2018.

[28] Schleich, J. Uav fleet area coverage with network connectivity constraint. *MobiWac*, 2013.

[29] SDKParrot. Sdk par drones parrot bebop 2. https://developer.parrot.com/docs/SDK3/. [Online]. Accedido Agosto 2018.

[30] Shang, K. A ga-aco hybrid algorithm for the multi-uav mission planning problem. *International Symposium on Communications and Information Technologies (ISCIT)*, 2014.

[31] Sphinx. Simulador sphinx. https://developer.parrot.com/docs/sphinx/whatissphinx.html. [Online]. Accedido Julio 2018.

[32] SPLAT! Splat. https://www.qsl.net/kd2bd/splat.html. [Online]. Accedido Agosto 2018.

[33] XLSemanal. Aplicaciones militares de drones. https://www.xlsemanal.com/conocer/tecnologia/20180814/drones-militares-en-espana.html. [Online]. Accedido Mayo 2018.