

UNIVERSIDAD NACIONAL DE ASUNCIÓN

Facultad de Ingeniería Ingeniería Mecatrónica



Sistema de Guía de Múltiples Drones de Superficie Acuática utilizando Información Dinámica

Federico Daniel Peralta Samaniego

San Lorenzo - Paraguay

2019

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

Miembros del Consejo Directivo

Consejeros Titulares

Prof. Dr. Ing. Rubén Alcides López Santacruz (Decano)
Prof. Ing. Primo Antonio Cano Coscia (Vicedecano)
Prof. Ing. Cirilo Jorge Hernáez Medina (Docente)
Prof. Ing. Rubén Darío Zárate Rojas (Docente)
Prof. Ing. Marcial Almada Ibáñez (Docente)
Prof. Ing. Miki Saito (Docente)
Prof. Ing. Sergio Elías Gavilán Martínez (Docente)
Prof. Ing. Marco Aníbal Cáceres Arce (Docente)
Ing. Tania Anahí Vargas Agüero (Graduada No Docente)
Ing. Fausto Ariel Alfonso Duarte (Graduado No Docente)
Ing. José Luis González Osorio (Graduado No Docente)
Est. Sebastián Daniel Olivella Noguera (Estudiante)
Est. María Paula Oviedo Velázquez (Estudiante)
Est. Ulises Manuel Leónido González Galeano (Estudiante)

Consejeros Suplentes

Prof. Ing. Verónica Alexandra Blanco Bogado (Docente)
Prof. Ing. Francisco Manuel Arrom Irún (Docente)
Prof. Ing. Luis Vidal Poisson Spessot (Docente)
Prof. Ing. Luis Sosa Almirón (Docente)
Prof. Ing. Abilio Mancuello Petters (Docente)
Prof. Ing. Silvana Rocío Anahí Ibáñez Candia (Docente)
Est. Federico Daniel Peralta Samaniego (Estudiante)
Est. Antonic Josué Wuyk Borja (Estudiante)
Est. Sofía Vierci Cotas (Estudiante)

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

*A quienes piensen
no como individuos,
sino como especie.*

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

Agradecimientos

A quienes me han dado la vida y las herramientas para comprenderla en mi corto entendimiento de la misma: mamá, papá y Dios.

A mis hermanos Victor y Ali, y a toda mi familia extensa por el apoyo recibido.

A mis amigos y compañeros de estudio de la Facultad, que no solo me acompañaron en el proceso, sino que actuaron como fuente de motivación para seguir adelante y no rendirse. A los mejores amigos y amigas que me dió la Facultad, por ser pilares de apoyo emocional y cognitivo.

A mis profesores, por tan incansable labor en el día a día, por sus enseñanzas académicas y de vida. Por dedicarse a generar que la humanidad entera conozca sobre la naturaleza de las cosas.

A los amigos y compañeros que conseguí en la facultad, quienes demostraron ser parte del grupo de personas que quieren hacer un Paraguay mejor para todos.

A todos los integrantes del Laboratorio de Sistemas Distribuidos, en especial a los Dres. Derlis Gregor y Mario Arzamendia, a las MSc. Maira Santacruz y Regina León y al Ing. Kevin Cikel, por la guía, educación y formación recibida. Por hacer y generar espacios de investigación tan necesarios en el Paraguay.

A los Doctores Sergio Toral y Daniel Gutiérrez Reina por sentar las bases del TFG y por cooperar con la Universidad Nacional de Asunción.

Federico Peralta

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

Índice general

1. Marco Introductorio	7
1.1. Introducción	7
1.2. Antecedentes	8
1.2.1. Sistemas Multi-Robot	8
1.2.2. El Lago Ypacaraí y problemáticas	11
1.3. Justificación	14
1.4. Planteamiento del problema	16
1.5. Objetivo General	18
1.6. Objetivos Específicos	18
1.7. Alcance y limitaciones	18
2. Marco Conceptual	21
2.1. Vehículos Autónomos de Superficie	21
2.1.1. Tipos o grupos de ASVs	22
2.1.2. Sistemas del ASV	23
2.2. Arquitectura de toma de decisiones del Sistema de Guía	25
2.2.1. Sistema GNC	25
2.2.2. Subsistema de Guía	27
2.2.3. Arquitectura de toma de decisiones	28
2.2.4. Información Dinámica	49
2.3. Subsistema de Comunicación para ASVs	50
2.3.1. Comunicación Inalámbrica	51
2.3.2. Protocolos de Comunicación para ASV	53
2.4. Estrategia de Cooperación Distribuida para Drones de Superficie .	56
2.4.1. Cooperación	56
2.4.2. Estrategias de Cooperación Distribuida	60

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

2.4.3. Algoritmos de estrategias de cooperación para ASV	63
2.5. Propuesta de Sistema de Guía y Cooperación de Múltiples Drones de Superficie	64
2.6. Control en Sistema Embebido	66
2.6.1. Sistema Embebido	66
3. Marco Procedimental	69
3.1. Ruta de Diseño del proyecto	69
3.2. Descripción general	70
3.2.1. Modelación del Entorno	71
3.2.2. Planificación de Ruta y Trayectoria	72
3.3. Diseños de implementación	93
3.3.1. Algoritmos del subsistema de guía	93
3.3.2. Implementación de software y control	101
3.4. Implementación.	106
3.4.1. Sistemas de comunicación inalámbrica.	106
3.4.2. Sistema de Guía propuesto	107
3.5. Pruebas del sistema.	108
4. Marco Analítico	115
4.1. Discusión y análisis de resultados particulares	115
4.1.1. Planificador Local de Ruta	115
4.1.2. PODMP y MCTS como solucionador de PODMP	116
4.1.3. Sistema Embebido	120
4.1.4. Software en Sistema Embebido	120
4.2. Discusión de Resultados y Conclusión Final	121
4.3. Evaluación y seguimiento del impacto	122

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

Índice de figuras

1.1. Simulación de Boids	9
1.2. Lago Ypacaraí	12
1.3. Algas en el Lago Ypacaraí	13
1.4. Cobertura del Lago Ypacaraí usando balizas	14
1.5. Sistemas del ASV	17
2.1. Modelo del ASV	24
2.2. Diagrama de bloques.	28
2.3. Diagrama de bloques del subsistema de guía.	28
2.4. Diagrama de secuencias de Toma de Decisiones de un Vehículo	30
2.5. Diagrama de secuencias para calcular patrón de cobertura	33
2.6. Ejemplo MCTS	37
2.7. Diagrama de secuencias para A^*	41
2.8. Diagrama de secuencias para <i>Potential Fields</i>	43
2.9. Diagrama de secuencias para RRT^*	45
2.10. Diagrama de secuencias para la matriz TOA	47
2.11. Diagrama de secuencias para calcular FMS	48
2.12. Topologías de Conexión	54
2.13. Diagrama Voronoi	60
2.14. Diagrama Voronoi Distribuido	63
2.15. Sistema de guía y cooperación de los ASVs	65
2.16. Kits de Desarrollo	68
3.1. Modelo del Lago Ypacaraí	71
3.2. Diagrama de secuencias para actualizar matrices TOA	85
3.3. Diagrama de secuencias para calcular $uFMS$	86
3.4. Comparación FMS - uFMS	87

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

3.5. Visualización de planificadores locales de ruta	90
3.6. Visualización de rendimientos de planificadores locales de ruta. .	92
3.7. Ejemplo del AutoPilot Pixhawk	94
3.8. Diagrama de secuencias de la arquitectura	95
3.9. Diagrama de secuencias del subsistema de guía	96
3.10. Pruebas del sistema	102
3.11. Visualización del sistema	104
3.12. Pruebas en Sistemas Embebidos	105
3.13. Cobertura 3G	107
3.14. Tabla de Procesos	108
3.15. Pruebas de sistema propuesto	109
3.16. Pruebas de patrones de cobertura por barrido	111
3.17. Pruebas para comparación	112
3.18. Porcentaje de cobertura por hora	114
4.1. Porcentaje de cobertura por hora por cant. de drones	119

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

Índice de tablas

2.1. Características de comunicación	53
2.2. Características de algunos sistemas embebidos	67
3.1. Comparación de planificadores de ruta	92
3.2. Síntesis de pruebas finales realizadas varios sistemas	113

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

Índice de anexos

Anexo A: Tecnologías que utilizan Zigbee disponibles	133
Anexo B: Propiedad del Voronoi	134
Anexo C: Programas en lenguaje Python	135
Anexo D: Programa de Estación Base	151

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

CAPÍTULO 1

Marco Introductorio

1.1. Introducción

Con el aumento de nuevas tecnologías robóticas, en los años más recientes, y observando las tareas que se realizan usando un solo robot, se ha visto la factibilidad de implementar tareas distribuidas en varias más pequeñas, asimismo, una misma actividad puede ser realizada con menor intensidad de trabajo, utilizando varios robots, preferiblemente de complejidad y costo menor, creando lo que se conoce como un Sistema Multi-Robot (SMR). El SMR fue introducido por J. Ferber y otros [1] como una parte de los Sistemas Distribuidos Multi-Agente (SDMA). En el trabajo mencionado se define lo que se conoce como Agentes Autónomos a los entes capaces de interactuar con el entorno y con otros agentes. El trabajo mencionado propone la idea básica de que si en un entorno multi-agente, cada agente posee habilidades, motivaciones y formas de comunicación; él puede trabajar cooperativamente con sus vecinos (generalmente otros agentes iguales) para que el “enjambre” pueda cumplir el objetivo que poseen en común. El comportamiento de este sistema puede ser comparado con lo que se observa en los agrupamientos vivos que se encuentran en la naturaleza, como son los movimientos de bandadas de aves, el trabajo de las colonias de hormigas, la generación de miel por enjambres de abejas, y otros resultados que generan otros colectivos de animales, insectos e incluso células, según J. A. Shapiro [2].

En el caso de los drones de superficie, M. Duarte y otros [3] presentan una

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

de las aplicaciones más recientes de SMR de superficie (a efectos de trabajo siempre se referirá a superficie acuática), que consiste en un grupo de botes motorizados que, organizados en forma autónoma, pueden cumplir diferentes tareas cooperando entre sí, a partir de controladores entrenados, a costa de que exista una plataforma que los controle, incluyendo una robusta y confiable comunicación, coordinación y cooperación entre ellos. Esta plataforma es objeto de estudio en diferentes centros de investigación y todavía no es explotada en forma industrial por la complejidad y costo del sistema, por lo que diseñar un sistema que minimice estas características es también un objetivo de investigación.

El presente trabajo diseña el sistema de guía de una flota de drones de superficie acuática que base sus decisiones en información cambiante con el transcurso del tiempo, desarrollándose con técnicas y métodos propios de sistemas multi-agente o multi-robot, no dejando de lado el desarrollo interno de un robot parte del sistema multi-robot, en cuanto a guía o planificación de movimiento, comunicación y cooperación.

1.2. Antecedentes

1.2.1. Sistemas Multi-Robot

El trabajo hecho por C. R. Kube y otros [4], realizado en 1993, corresponde a una de las primeras definiciones formales de la robótica multi-agente, en esos tiempos, la tecnología era limitada, de tal forma que varios centros de investigación proponían crear poblaciones de robots con niveles bajos de inteligencia (J.-L. Deneubourg y otros [5], L. Steels [6]), sentando las bases de la línea de investigación de robótica que se concentraba en desarrollar sistemas flexibles pero simples.

Esta línea permitió desarrollar sistemas de cooperación generalizados, modelados matemáticos de conjuntos de agentes con objetivos en común y sistemas

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

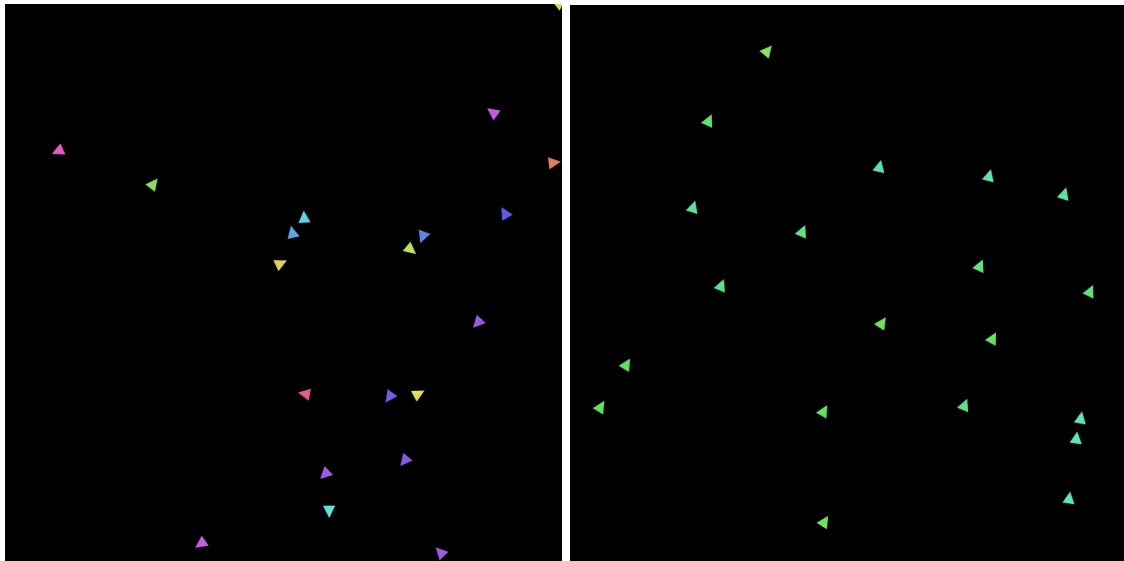


Figura 1.1: Agentes al inicio y final de la simulación. **Fuente:** Elaboración propia.

de enjambres robóticos, estos últimos más conocidos por sus siglas en inglés *swarm robotics*. Un trabajo realizado por C. W. Reynolds [7] consiste en un modelo muy simple de una bandada de pájaros, donde dichos agentes siguen 3 sencillas instrucciones:

- Separación: Conducir para evitar estar muy cerca de otros agentes.
- Alineación: Conducir hacia el promedio de direcciones de agentes cercanos.
- Cohesión: Conducir para moverse hacia el centro de masa de agentes cercanos.

En la Figura 1.1 se pueden observar un estado inicial y un estado luego de un tiempo considerable en el cual los agentes siguen las instrucciones definidas anteriormente sin cambio alguno, obteniendo un sistema multi-agente que posee la característica principal de ordenamiento y cohesión en una dirección dada. Este es el resultado de un sistema que en primera instancia aparenta ser controlado por un supervisor general, pero realmente es el generado por un sistema sencillo de reglas que, aunque irrelevantes a un nivel de agentes aislados, realiza

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

una tarea conjunta cuando los niveles de interacción se dan entre 2 o más agentes. Para una mejor visualización de la idea, el color de los *boids* corresponde al ángulo de inclinación con respecto a la línea horizontal en el plano donde se encuentran.

A comienzos del siglo XXI, los centros de investigación de robótica ya se encontraban desarrollando tecnologías que no solo mejoraban los resultados que un solo robot podía entregar, sino también los de los sistemas cooperativos más simples. El trabajo realizado por C. Şahin [8] consolida la teoría de la robótica de enjambre, definiendo los dominios de aplicación de ésta a: tareas peligrosas, cobertura de regiones y las que necesitan de redundancia de información.

En el entorno actual, existen varias aplicaciones de sistemas de robots que pertenecen a uno de los dominios de aplicación mencionados, aunque el sistema no corresponda al de un enjambre sino a un conjunto numeroso de robots. Y. Diaz-Mercado y otros [9] presentan un sistema de robots móviles terrestres que consiguen cubrir regiones con densidades dinámicas para interacciones humano-robot. S. Hauri y otros [10] desarrollan un sistema multi-robot para control de formación en tiempo real.

Recalcando el resultado de la investigación presentado por M. Duarte y otros [3] (2016), se debe mencionar específicamente que en él se realiza una investigación e implementación de robótica de enjambre, enfocándose en tareas simples como visita de puntos, dispersión, *clusterización* y monitoreo de área, mediante el uso de síntesis de procesos evolutivos, en donde los sistemas de guía y planificación de trayectoria son entrenados bajo ciertas características en simuladores, para luego realizar pruebas con una flota de botes autónomos reales. Es importante destacar la idea que el trabajo presenta, cuando menciona al monitoreo del área, mencionando que los enjambres de robots son ideales para tareas donde áreas muy grandes necesitan ser cubiertas para propósitos de monitoreo o vigilancia permanente. La tarea específica ha sido estudiada y explorada

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

con diferentes técnicas como traza de feromonas simuladas y redes neuronales artificiales evolucionadas.

Es importante notar las motivaciones para aplicaciones de sistemas multi-robot que define C. Şahin [8]: robustez, flexibilidad y escalabilidad. Cuando estas 3 características son necesarias, la solución óptima consiste en el desarrollo de sistemas de múltiples robots. Ejemplos cotidianos de escalabilidad consisten en la recolección de información mediante un grupo de agentes capaces de obtener los datos deseados, en vez de depender de la recolección necesaria por medio de un solo agente. Este dato va en concordancia con lo mencionado anteriormente por M. Duarte y otros [3].

El Laboratorio de Sistemas Distribuidos (LSD) de la Facultad de Ingeniería se encuentra desarrollando actualmente el proyecto **PINV15-177 "Vehículo Autónomo de Superficie (ASV) para el Estudio de Calidad del Agua en Lagos y Lagunas"** financiado por el *CONACYT*. Dicho proyecto se centra en diseñar e implementar el primer vehículo autónomo de superficie para el monitoreo de parámetros de la calidad del agua en lagos y lagunas de Paraguay. El proyecto mencionado contribuyó en una definición de método para recorrer la superficie total de un lago según los trabajos de M. Arzamendia y otros [11, 12], a ser discutidos más adelante. Numerosos estudios se han realizado con vistas a solucionar el problema de floración de algas del Lago Ypacaraí.

1.2.2. El Lago Ypacaraí y problemáticas

Se muestra en la Figura 1.2 una imagen satelital del Lago Ypacaraí tomada a finales de año 2018. Este lago, ubicado en el departamento central, presenta una notable degradación de la calidad del agua desde hace varios años según G. López y otros [13], ya que, durante los meses más calurosos, se genera una alta concentración de cianobacterias que cambia el color del agua a un tono verde debido a un incremento de nutrientes, que puede resultar perjudicial para

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**



Figura 1.2: Imagen Satelital del Lago Ypacaraí. **Fuente:** Maxar Technologies.

la calidad del agua. Estas cianobacterias conocidas como *blue-algae* son tóxicas para el ser humano y para los animales por lo que se debe buscar siempre la forma de mitigarlas y hacerlas desaparecer del Lago, ya que el lugar representa un punto turístico importante, tanto a nivel nacional como internacional.

Los trabajos de monitoreo se están realizando continuamente, como el proyecto realizado por *Hydroinformatics International Center* (CHI) y la Central Hidroeléctrica de Itaipú [14], pero éstas son estaciones fijas que toman manualmente muestras del agua en puntos específicos no habiendo una libertad de movimiento para seleccionar el estudio de calidad del agua en posiciones genéricas (no establecidas para casos determinados). El vehículo autónomo de superficie desarrollado en la FIUNA sí tiene esta capacidad, haciendo más interesante su desarrollo y optimización. Imágenes de estas algas pueden observarse en la Figura 1.3, que corresponden a las últimas observaciones de floraciones del Lago Ypacaraí (febrero, 2019).

El monitoreo propuesto en el proyecto **PINV15-177** incluye obtener valores del potencial de oxidación-reducción, oxígeno disuelto, potencial del Hidrógeno (pH),

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA



Figura 1.3: Algas en el Lago Ypacaraí (Feb-2019) **Fuente:** ABC color [15]

conductividad eléctrica y temperatura. Estos 5 parámetros permiten obtener una observación sobre la calidad del agua e inferir la presencia de contaminantes, por lo que el control de estos parámetros en la extensión del lago, brinda una herramienta para solucionar la problemática.

El trabajo realizado por M. Arzamendia y otros [12] define una ruta global con *waypoints*, o puntos a visitar, ubicados a una distancia considerable de la costa del Lago, de forma que al visitar cada una de estas balizas se consigue obtener mayor cantidad de información del lago, cubriendo la mayor cantidad de área posible. Esto lo realiza el vehículo mediante la utilización de un algoritmo genético evolucionado que busca optimizar el área cubierta por el vehículo. Uno de los resultados del trabajo propuesto puede ser observado en la Figura 1.4, en la cual se presentan 60 puntos de visita sobre una representación del Lago Ypacaraí y la secuencia de viaje, de tal forma a cubrir la máxima superficie posible. El monitoreo constante de la calidad del agua es una buena forma de determinar las causantes y elaborar posibles soluciones, por lo que es de suma importancia que existan trabajos como los mencionados, aunque se encuentra interesante elaborar un sistema multi-robot para obtener de forma más rápida y eficiente una lectura de diferentes parámetros de la calidad del agua. A pesar de que un vehículo podría viajar directamente entre baliza y baliza, se necesita alguna for-

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

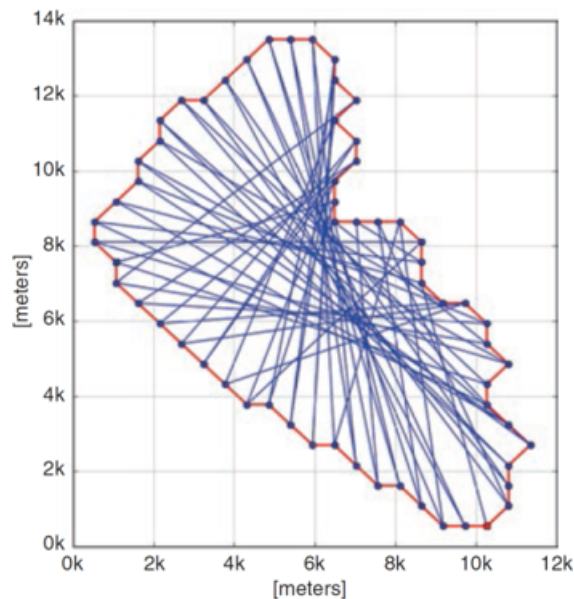


Figura 1.4: Cobertura del Lago Ypacaraí usando balizas **Fuente:** M. Arzamendia y otros [12]

ma de evitar posibles obstáculos no previstos o modificaciones de la ruta traza que tenga en cuenta las limitaciones del propio vehículo tales como: velocidades angulares y lineales permitidas, función de movimiento y limitaciones del entorno como facilidad de tránsito y otros datos. Teniendo en cuenta lo mencionado, antes de tomar una ruta, se puede obtener un resultado esperado o estimado que podría ser considerado como información útil para tomar decisiones pero se deben tomar las debidas precauciones, argumentando que la naturaleza siempre responde acorde a modelos no deterministas. Además de todo lo mencionado anteriormente, en mapas de dimensiones grandes, es imposible predecir la presencia de obstáculos dinámicos en entornos naturales.

1.3. Justificación

La idea de este TFG se concentra en expandir el trabajo propuesto por M. Arzamendia y otros [12] a un Sistema Multi Robot para poder encontrar mayor cantidad de información en menor tiempo. Los SMR aplicados a monitoreo, control y prevención se destacan en la rama de la investigación de la robótica debido

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

a que agilizan, de una manera exponencial, la obtención de datos y generación de resultados. Un vehículo autónomo, en la gran mayoría de los casos, posee sistemas de comunicación externos, por lo que la adaptación de tecnologías para lograr una cooperación entre agentes se centraría en desarrollar solamente el software para generar comunicación entre múltiples drones de superficie. Esto no aumenta de manera significativa el costo monetario del proyecto, sino que aprovecha aún más las capacidades de hardware de los drones. La dificultad de manejar un sistema así se concentra en generar la cooperación que beneficie tanto al trabajo del vehículo como a cumplir el objetivo de la misión, y, desarrollar la estrategia de cooperación distribuida genera valor científico tecnológico debido a la limitada cantidad de conocimiento e implementación en el mundo sobre este tema específico. En general, diseñar una arquitectura de toma de decisiones basada en información dinámica aplicada a un conjunto de agentes, permite expandir el conocimiento de la robótica de enjambre y, en menor medida, permite poner a prueba modelos de cooperación que pueden ser aplicados en cualquier ámbito científico.

Por otro lado, obtener datos químicos de la calidad del agua de varios puntos de un gran volumen de agua, permite tener una visión más realista de los parámetros en cada zona del cuerpo acuático y, a partir de esto, encontrar focos de contaminación, elaborar hojas de ruta para solucionar la problemática, modelar las relaciones entre parámetros, definir acciones concretas para eliminar o mitigar agentes nocivos para la fauna y flora del cuerpo de agua en particular, así como también de las zonas aledañas (en general ciudades urbanas, bosques, esteros, lagunas, ríos, humedales y montes). En la implementación de estos temas que quedan fuera del alcance del Trabajo Final de Grado (TFG), se podrá observar un aumento del desarrollo de la economía de los entes intervinientes y, una vez finalizada la “recuperación” del Lago, el índice de turismo mejorará, al ser saludable para las personas, aparte de ser más atractivo y seguro.

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

1.4. Planteamiento del problema

Para desarrollar de forma óptima y eficiente un sistema multi robot se puede comenzar definiendo el problema específico y descomponerlo en problemas resolubles de forma genérica, para no perder de vista la visión, así también, encontrar soluciones que funcionen en forma genérica para todos, o por lo menos la mayoría, de los casos. El problema principal específico se concentra en obtener una cantidad de información de parámetros de calidad en un gran cuerpo de agua, el cual se puede poder transformar en la propuesta de desarrollo de un sistema multi-robot capaz de recorrer un lago aplicando técnicas y desarrollando tecnologías que permitan una cooperación eficiente que busque minimizar el trabajo de un vehículo. Para ello se deben desarrollar sistemas de cooperación distribuida de forma abstracta, es decir, que no dependan del diseño de cada uno de estos vehículos. Así también se deben diseñar drones de superficie, que puedan tomar decisiones con respecto a toda la información útil recibida tanto internamente como externamente, incluyendo información que otros vehículos puedan ofrecer. Este intercambio de información debe ser lo suficientemente amplio como para propiciar un beneficio al agente receptor, pero reduciendo el costo de la acción realizada por el emisor, al mínimo posible. En la Figura 1.5 se puede visualizar el diagrama de bloques del sistema a desarrollar, donde el sistema general comprende a todos los robots o vehículos cumpliendo la misión de recorrer el lago. Este sistema, a su vez, se encuentra subdividido en otros dos subsistemas principales, que corresponden a 1. el Sistema de Cooperación Distribuida, que es un sistema abstracto que se implementa en cada uno de los vehículos, y 2. el Sistema de Drones de Superficie, que corresponde a la implementación de los vehículos, el cual, primero debe ser diseñado. Para un mejor desarrollo durante el diseño, estos sistemas se subdividen nuevamente en sistemas internos. En el diagrama se muestran los sistemas más importantes para este TFG.

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

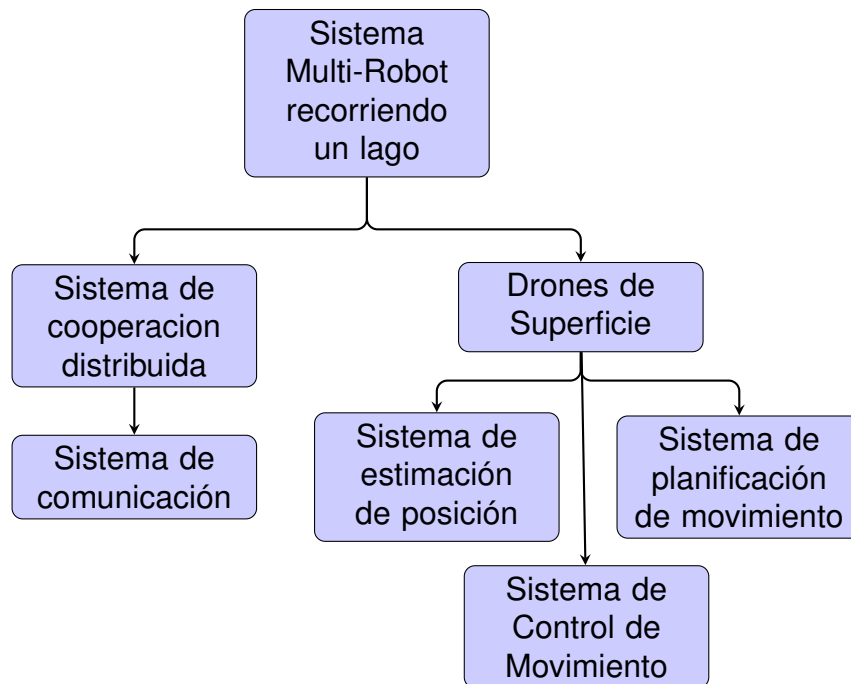


Figura 1.5: Definición de sistemas del ASV. **Fuente:** Elaboración propia.

Debido a la complejidad de desarrollar cada uno de estos sistemas, el presente TFG se centra en desarrollar algunos subsistemas específicos, de forma tal a cumplir el objetivo general, que es desarrollar el sistema general, concentrando esfuerzos en generar la cooperación distribuida para que cada vehículo pueda planificar su movimiento con algún tipo de optimización. En específico, se busca desarrollar el sistema de toma de decisiones, que corresponde al de planificación de movimiento con base en los datos obtenidos y una información dinámica, como puede ser el estado de lecturas actual y la posición del vehículo y de los demás; dicho esto, es necesario diseñar una estrategia de cooperación distribuida. Ambos objetivos deben ser validados a través de implementaciones de tecnologías disponibles actualmente y utilizables en drones acuáticos, con las limitaciones y ventajas que estos drones y el cuerpo de agua ofrezcan.

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

1.5. Objetivo General

Diseñar un sistema de guía de múltiples drones de superficie, que descentralice eficazmente la obtención de datos de un entorno, utilizando información dinámica.

1.6. Objetivos Específicos

1. Desarrollar la arquitectura de toma de decisiones del sistema de guía.
2. Simular un vehículo autónomo de superficie que utilice el sistema de guía desarrollado en entornos modelados, a través de algoritmos propios.
3. Analizar rendimiento de algoritmos aplicados para toma de decisiones del sistema de guía.
4. Diseñar una estrategia de cooperación distribuida entre drones de superficie y el mecanismo de comunicación.
5. Seleccionar sistemas embebidos para implementación, con base en métricas y requerimientos propios de un sistema de guía de múltiples drones de superficie.

1.7. Alcance y limitaciones

Como se mencionó previamente, el lago Ypacaraí (entorno de estudio), tiene dimensiones generales de 24 x 5 km de costa y un área de 60 km², según G. López y otros [13], por lo que se tomarán los datos de diferentes puntos del Lago en un tiempo muy elevado para lecturas completas de la superficie del lago con un solo vehículo. Para acelerar el proceso de toma de muestras de diferentes puntos, un enjambre de drones es una opción indicada. Este enjambre consiste en una flota de drones de superficie, donde dichos drones realizarán

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

la tarea de obtener datos de la calidad del agua, con la salvedad de que estos drones deben cooperar activamente en el trabajo. Para ello, es necesaria una plataforma de control y comunicación robusta, de forma a dividir los sectores de manera eficiente teniendo en cuenta la información que cambia de manera continua tanto interna (del mismo dron), como externa (entorno), logrando eficiente y eficazmente cubrir todo el lago y obtener la información deseada. El alcance de este TFG consiste en el desarrollo de la plataforma mencionada, con la correspondiente investigación, es decir, el diseño del sistema de planificación de movimiento del enjambre propuesto, así como inclusión de métodos y funciones de interacción no circunscritos a un sistema específico de drones en tiempo real, y generar comportamientos emergentes propios de enjambres. Otra característica de este trabajo incluye un enfoque a la futura implementación en sistemas embebidos para cada robot. Este sistema deberá ser genérico, para que el algoritmo pueda ser implementado en cualquier sistema multi-robot, en donde exista una información dinámica de interés. Particularmente, las características individuales del dron a usar para realizar la misión asignada deben incluir: un sistema autónomo de navegación y un sistema de comunicación.

En la actualidad, existen numerosos drones de superficie que pueden considerarse aptos para el trabajo, como son los trabajos realizados por A. L. Christensen y otros [16] y R. Song y otros [17]. Los vehículos utilizados en los trabajos mencionados son drones acuáticos de bajo costo que tienen características de comunicación y movimientos bastante robustos. Los drones a desarrollar en este proyecto deberán compartir información solamente entre ellos para el trabajo cooperativo, y con un servidor, para la adquisición de datos fuera del sistema multi-robot. Es también posible que, en caso de que los robots no puedan comunicarse directamente por la distancia entre ellos, el servidor estaría formando parte del sistema de comunicación entre los drones que pertenecen a la flota de trabajo, como un puente transparente. La implementación del control distribuido

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

dentro de un sistema embebido formará parte del desarrollo, por lo que la selección de un sistema adecuado, así como la elección y el control del hardware, serán objeto de estudio e investigación durante el desarrollo del TFG.

CAPÍTULO 2

Marco Conceptual

En este capítulo se definirán todos los conceptos utilizados en el TFG, además de incluir los métodos y técnicas estudiados para su aplicación en Sistemas de Múltiples Drones. El capítulo define y brinda ejemplos sobre los Vehículos Autónomos de Superficie, los sistemas propios de ellos, el sistema de guía y sus componentes, la comunicación inalámbrica en cuanto a las tecnologías, topologías y protocolos, luego se definen y estudian métodos de cooperación, y por último se estudian sobre sistemas embebidos, sus características y algunos ejemplos de implementación.

2.1. Vehículos Autónomos de Superficie

Los Drones de Superficie o Vehículos Autónomos de Superficie (ASV por sus siglas en inglés *Autonomous Surface Vehicles*) son un grupo de robots móviles capaces de transitar en la superficie de cuerpos de agua, usando diferentes técnicas de propulsión y diseñados para cumplir ciertos tipos de objetivos. Se diferencian principalmente de otros tipos de drones en que los ASV deben cumplir obligatoriamente algún factor de impermeabilidad, característica que comparte con los vehículos autónomos submarinos. Es importante mencionar que los ASV poseen autonomía y son, en su gran mayoría, no tripulados, a excepción de los vehículos diseñados para rescate, según lo expuesto en el estudio hecho por V. Bertram [18]; no obstante, la autonomía debe poder ser controlada en situaciones de emergencia. Los vehículos de rescate se pueden encontrar dentro de uno

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

u otro grupo según la especificación del propósito o según su diseño. Las divisiones en grupos permiten estudiar a profundidad un problema específico, por lo que es importante conocer estas divisiones y algunos ejemplos de ellas.

2.1.1. Tipos o grupos de ASVs

Según el propósito:

- **Vigilancia:** Son todos los ASVs cuya misión se enfoca en proporcionar información civil o de inteligencia, y son utilizados principalmente por fuerzas militares de algunos países.
- **Rescate:** Vehículos Autónomos capaces de realizar tareas de rescate de personas o animales. Consta de espacio dentro de la cabina del vehículo o en la cubierta. Un ejemplo es el proyecto realizado por G. Clauss y otros [19].
- **Control o Monitoreo:** ASVs compuestos con sensores específicos para controlar u obtener los valores de algún parámetro interesante para un servicio, estudio o investigación. Algunos ejemplos incluyen: batimetría, calidad del agua y/o nivel de contaminación, este último propuesto por M. Dunbbabin y otros [20].
- **Propósitos Específicos:** Se encargan de realizar una tarea definida, principalmente tratan de modificar un parámetro dentro del cuerpo de agua donde se encuentran. Un ejemplo reciente es el *WasteShark*, propiedad de RanMarine Technology [21], dron acuático que recoge la basura en la superficie del cuerpo de agua.

Según el diseño del casco:

- **Casco simple:** Consiste en el modelo más conocido de los vehículos de superficie acuática. Generalmente es propulsado por un solo motor en la

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

parte trasera del mismo.

- **Doble casco o Catamarán:** Posee dos cascos, uno al lado del otro, conectados por la cubierta. Puede poseer motores diferenciales en cada parte trasera de los cascos, permitiendo un mejor control sobre la dirección de la propulsión. Presentan una mayor estabilidad que los vehículos de un solo casco.
- **Triple casco o Trimarán:** Es un vehículo que posee un casco principal central y dos cascos pequeños a los costados. Generalmente, son propulsados por un motor ubicado en el casco principal, y dirigido con velas.
- **Híbrido AAV:** Actualmente, se encuentran diseños de drones aéreos (cuadricópteros) impermeables, que pueden modificar las disposiciones de sus motores para moverse en la superficie del agua.

2.1.2. Sistemas de Drones de Superficie

El trabajo realizado por Z. Liu y otros [22] define que los sistemas principales de un ASV incluyen: sistemas de Guía, de Navegación y de Control. Además de esto, los ASVs incluyen sistemas de energía, propulsión, comunicación, y por supuesto sistemas específicos, ya que lo principal en un ASV no debería centrarse en su autonomía e inteligencia, sino en la aplicación de este vehículo y su fin último, que en el caso de este TFG consiste en la medición de valores de los parámetros que indiquen la calidad del agua en lagos y lagunas.

El vehículo que será parte del sistema multirobot (Figura 2.1) deberá incluir, además de los sistemas principales para asegurar su autonomía, un sistema de comunicación y cooperación. Para el primer subsistema mencionado, en este TFG se desarrolla un algoritmo que permita comunicación entre todos los drones, para el segundo, que esta comunicación permita realizar cooperación como sistema emergente. Otros sistemas incluyen el sistema de adquisición de pará-

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**



Figura 2.1: Modelo del Vehículo Autónomo de Superficie a ser adquirido por el LSD. **Fuente:** J. Balbuena y otros [23]

metros de calidad del agua, que es parte crucial para el proyecto general, pero no para el sistema de guía, aunque se debe mencionar que es de vital importancia conocer los valores de estos parámetros en el Lago Ypacaraí, por más que los parámetros de calidad del agua a obtener no midan directamente la presencia de las cianobacterias, con estos parámetros se puede inferir la capacidad de supervivencia del contaminante mediante una correlación según el trabajo de G. López y otros [13]. Con este monitoreo se puede controlar apropiadamente la calidad del agua y tener una sólida evidencia para solicitar intervenciones, planes de acción y de recuperación que permitan mejorar la calidad del agua.

En la siguiente sección, se definen en detalle los sistemas principales, es decir, los de Guía, Navegación y Control, ejemplificando los principales componentes del Sistema de Guía. Además, se definen los Sistemas de Comunicación a ser desarrollados y utilizados en el presente trabajo.

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

2.2. Arquitectura de toma de decisiones del Sistema de Guía

Vince Lombardi expresó en una entrevista que “los logros de una organización son el resultado del esfuerzo combinado de cada individuo” queriendo significar la importancia del trabajo de cada componente existente dentro de un grupo o equipo. Los sistemas de múltiples agentes, tanto en ámbitos sociales como en los tecnológicos, no se deberían diseñar poniendo énfasis en las direcciones de alto nivel, sino más bien, en la calidad de trabajo de cada individuo del sistema así como en la cooperación y combinación de los esfuerzos que éstos pueden ofrecer. El diseño y desarrollo de arquitecturas de múltiples agentes se deben definir desde los niveles más bajos hacia los superiores, como así lo indica el paradigma de diseño *bottom-up* según M. C. McFarland [24], el cual busca la generación de sistemas complejos a partir del desarrollo de sistemas más simples que cooperarán de alguna forma a que se consiga cumplir el objetivo general del sistema complejo. En esta sección se definirá la arquitectura de toma de decisiones del Sistema de Guía (sus componentes, unión de sistemas, las tomas de decisiones de estos sistemas y por último la arquitectura completa).

2.2.1. Sistema Guía, Navegación y Control

En concordancia con el mismo paradigma, según M. S. Grewal y otros [25], el sistema general para brindar autonomía a un vehículo se puede separar en 3 subsistemas principales.

1. Guía: Se refiere a la determinación de las trayectorias y rutas a seguir para cumplir un determinado objetivo, incluye además los cambios de velocidad, rotación y aceleración deseados, para lograr el objetivo.
2. Navegación: es la determinación, en un tiempo establecido, de la posición, velocidad, aceleración, además de su ángulo con respecto a alguna recta predefinida.

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

3. Control: La manipulación de fuerzas en los actuadores para ejecutar las decisiones que los subsistemas de Guía y Navegación decidieron, manteniendo sobre todo la estabilidad del vehículo. Estos actuadores generalmente son los motores, alerones, aletas, entre otros.

El conjunto de sistemas citados anteriormente se denomina GNC por sus siglas y es implementado en todos los sistemas autónomos de alguna u otra forma, debido a que la definición de un vehículo autónomo se puede descomponer en estos 3 subsistemas. Adicionalmente, el trabajo hecho por Z. Liu [22] menciona el mismo sistema general para los vehículos autónomos de superficie, por lo que el diseño para estos, también debe seguir la misma dirección. Considerando las definiciones de cada una de las partes del GNC, cada subsistema puede subdividirse, a su vez, en componentes que obtienen y utilizan cierta información. En el caso del Subsistema de Guía, se tienen los siguientes componentes:

- Planificación de misión: Tiene en cuenta los objetivos, y en conjunto con informaciones del entorno, la energía y las tareas posibles, direcciona a las siguientes tareas a realizar como son las planificaciones de movimiento, la comunicación u otras.
- Planificación de movimiento: Con información del Sistema de Navegación, procesa la información de la misión actual y genera una ruta a seguir o una velocidad deseada.
- Generación de Trayectoria: obtiene información de la velocidad deseada, información de seguridad, etc. para modificar la velocidad deseada según las limitaciones y los requerimientos del sistema físico. A continuación, se envía esta información al Sistema de Control.

Toda información que ingresa al Sistema de Guía, se procesa durante la Planificación de Misión, y se entrega la respuesta para planificar el movimiento, por

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

último, la información de salida de la Planificación de Movimiento se utiliza como entrada al componente de Generación de Trayectoria, para que este genere la velocidad deseada.

A partir de esta separación de sistemas, se puede estudiar cada uno de ellos de forma específica, y existen módulos genéricos de cada subsistema de modo a que la implementación pueda generalizarse sin mayores inconvenientes. Aunque los subsistemas se muestren separados, los mismos no necesariamente deben estar en distintos hardware. Un ejemplo de esto fue realizado por A. West y otros [26], en donde los subsistemas de guía y navegación procesan información en una computadora, pero la obtención de datos está físicamente en el robot. El trabajo de J. Balbuena y otros [23] define al control de los 3 subsistemas dentro de dos hardware diferentes que trabajan en conjunto. El factor común, de nuevo, es la separación de los subsistemas, por lo menos a un nivel de software. El presente TFG diseña e implementa el subsistema de guía para Vehículos Autónomos de Superficie.

2.2.2. Subsistema de Guía

Para un entendimiento más profundo de este subsistema, primero se puede definir un sistema general y luego definir específicamente el sistema de Guía. El libro *Ingeniería de Control Moderna* hecho por K. Ogata [27] define a un sistema como “una combinación de componentes que actúan juntos y realizan un objetivo determinado.” Este sistema, en su forma más simple consiste en una *planta*, la cual posee una señal de entrada *in* que alimenta a la planta y ésta produce una señal de salida *out*. La *planta* es una de las partes del sistema que se encarga de ejecutar una operación. Una visualización del sistema en cuestión puede observarse en la Figura 2.2. Esta figura representa al sistema mediante un Diagrama de Bloques, el cual consolida toda la información relevante del sistema mediante gráficos fáciles de comprender.

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**



Figura 2.2: Diagrama de bloques de un sistema simple. **Fuente:** Elaboración Propia.

Para el caso específico de este TFG, se puede inferir por lo anteriormente presentado que la salida del Sistema de Guía (SG) comprende las velocidades deseadas para lograr el objetivo, ya que el Sistema de Control (SC) solo necesita información sobre cómo modificar los estados de los actuadores. El SG no necesita información sobre los estados de los motores siempre y cuando pueda obtener informaciones confiables del Sistema de Navegación (SN), que, en conjunto con informaciones de entrada a la planificación de misión son las entradas del sistema, los cuales generan la ruta a viajar y, como salida, las velocidades deseadas. Con esto se puede definir por medio de diagrama de bloques, al diagrama del subsistema de guía, conjunto con los componentes definidos en la subsección anterior. En la Figura 2.3 se puede observar el diagrama y a partir de él se empiezan a definir con mayor detalle cada componente y algunos algoritmos o métodos existentes para cada uno de ellos, cabe destacar que $\vec{p}$ corresponde a la posición actual y proviene del sistema de navegación, mientras que $\vec{V}$ corresponde a la velocidad deseada.

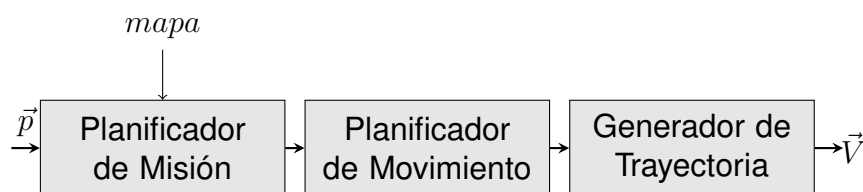


Figura 2.3: Diagrama de bloques del subsistema de guía. **Fuente:** Elaboración Propia.

2.2.3. Arquitectura de toma de decisiones

El diagrama de secuencias mostrado en la Figura 2.4 contiene cada uno de los componentes del SG y se demuestra el flujo de información que permite que

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

el SG pueda cumplir con su objetivo en particular. Este algoritmo debe obtener las informaciones del Sistema de Navegación (su posición y orientación y el entorno compuesto de camino libre y obstáculos) y utilizarlas para determinar la velocidad deseada del vehículo. Al seguir esta secuencia se logra utilizar toda la información disponible en el sistema de guía, se hace uso del planificador de misión que obtiene las posiciones de las metas a visitar. Luego cada una de las metas son consideradas como metas globales y se generan rutas con el planificador de ruta seleccionado. El componente de generación de trayectoria calcula siempre y cuando sea posible, la diferencia entre el final de la trayectoria, es decir la siguiente de las metas según el planificador de ruta, y la posición actual del vehículo y si esta diferencia no es cero, se calcula la velocidad deseada según el vector diferencia, luego se modifica esta velocidad según las restricciones del entorno y del vehículo y por último se determina ésta como la velocidad que el sistema de control recibe como entrada. Por otro lado si es que la diferencia es cero o algún valor cercano a este, se procede a obtener la siguiente trayectoria según un planificador de ruta o la siguiente lista de las metas según el planificador de misión. Cuando la trayectoria es nula, es decir, no existe una siguiente meta, finaliza el algoritmo. Las variables pueden sufrir modificaciones sin previo aviso, que permiten que el vehículo no varíe su trabajo, pero se adapte a las condiciones cambiantes, con esto y otros algoritmos a diseñar más adelante se consigue efectivamente trabajar con información dinámica a un nivel individual de vehículos. A continuación se definen propiamente las entradas, los componentes y salidas del subsistema de guía.

Modelado del entorno

La primera parte del diagrama de bloques del subsistema de guía mostrado en la Figura, define como entrada a el mapa modelado, es decir, la representación del entorno en donde se encuentra trabajando un robot. El estudio realizado

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

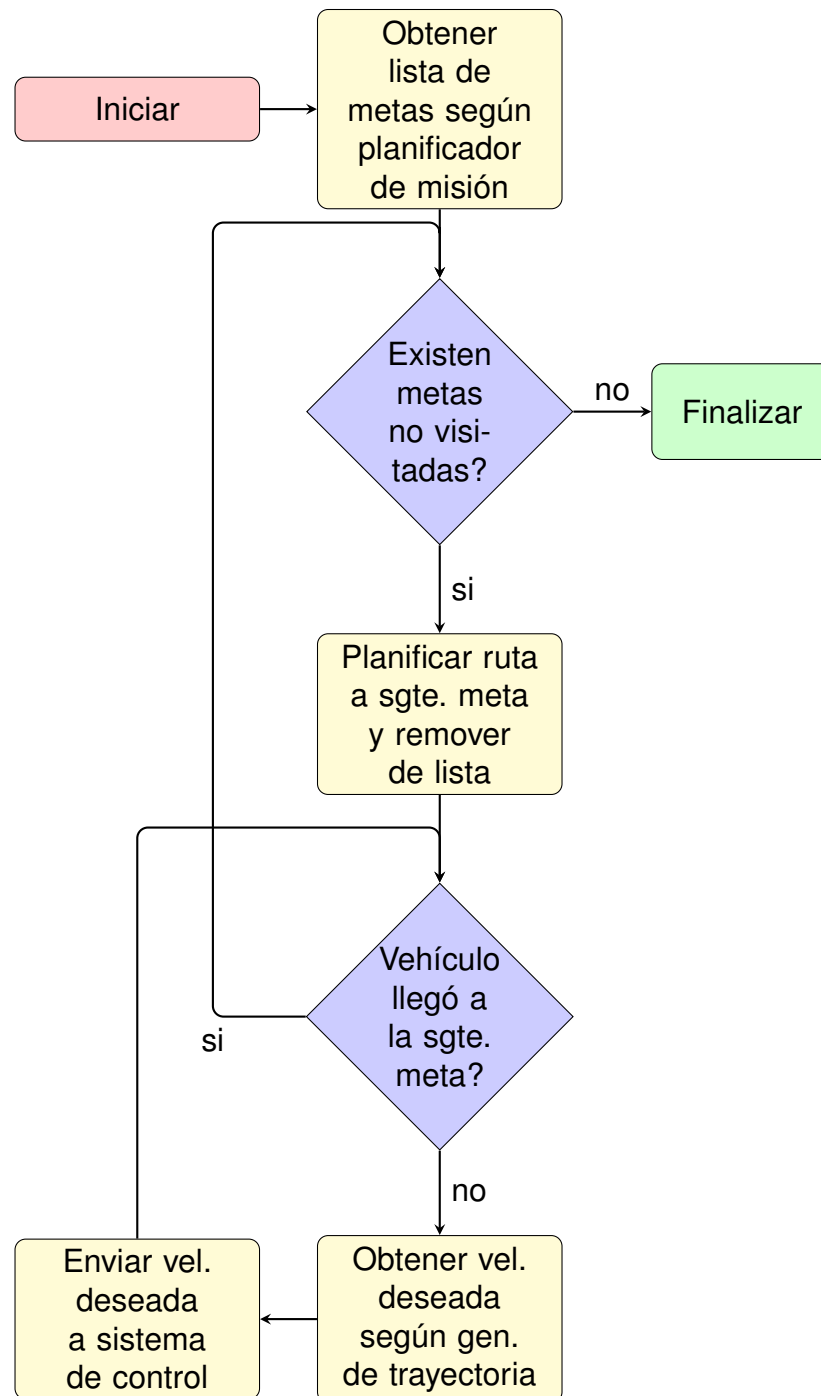


Figura 2.4: Diagrama de secuencias base de la arquitectura de toma de decisiones de un vehículo. **Fuente:**Elaboración Propia.

por O. Souissi y otros [28] menciona que en la planificación de movimiento se debe tener un modelado del entorno, como un primer paso indispensable para ejecutar la gran mayoría de algoritmos de planificación de ruta (componente del SG). En ese sentido para este TFG, se procede a definir formalmente el mapa

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

según un modelado del espacio de interés.

Información del espacio de interés: Para modelar de una forma apropiada el entorno se debe definir al espacio de interés en las diferentes dimensiones de estudio, como son informaciones de zonas transitables y no transitables, además de informaciones concretas sobre parámetros a ser utilizados durante las planificaciones a ser implementadas en el subsistema de Guía. Algunos métodos de modelado de entorno, según O. Souissi y otros [28], son los grafos de visibilidad, las mallas de navegación y las grillas de ocupación. El último modelado presentado es el modelado de mapa seleccionado para este TFG de entre las variantes debido a la facilidad de implementación y, principalmente, a que permite encontrar rutas que seas navegables fácilmente para cualquier tipo de vehículo, en especial para ASVs. A continuación, se definen con detalle las informaciones mencionadas:

1. Grilla de Ocupación (*map_data*): Corresponde una matriz que contiene valores de ocupación que poseen cada uno de los puntos x pertenecientes al entorno de estudio. Si en un punto se lee un valor $> 0,5$ significa que corresponde a un punto ocupado por cualquier objeto en la superficie de agua que no sea el vehículo, tales como los obstáculos y otros, y también que corresponde a un punto en la tierra, lugar donde no puede transitar el vehículo.
2. Mapas de parámetros (*map_param_i*): Informaciones de la misma dimensión que Ω' con datos de cierto parámetro definido en la implementación.

Una vez definido el modelado del entorno, ya se puede continuar y estudiar sobre la planificación de misión, componente que recibe como entrada a la posición del vehículo y el modelado del entorno.

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

Planificación de Misión

La planificación de la misión es la parte más importante en el sistema de guía, ya que se podría definir como el puente entre la abstracción del objetivo y la realización de las acciones. Es sobre este componente donde recae el nivel de escalabilidad y robustez del sistema general. Una buena planificación de misión provee de herramientas, algoritmos y criterios de decisión que permiten que un robot móvil busque encontrar caminos o rutas para cumplir un objetivo principal. Si este componente del sistema se realiza lo suficientemente robusto como para detectar y/o corregir errores provenientes de los otros sistemas, el sistema general encontrará las formas de cumplir las misiones.

En términos de cumplimiento de objetivos, es necesario recordar el diagrama en la Figura 1.5, donde en la parte superior se encuentra el objetivo general, el cual es recorrer la superficie de un cuerpo de agua con múltiples vehículos autónomos de superficie. Para cumplir esto se debe implementar un método o técnica que cumpla con él y para seleccionar uno apropiado se estudian métodos que maximizan la operación de robots teniendo en cuenta ciertos parámetros.

Implementaciones de cobertura de área en la literatura: C. Blum y otros [35] se encargaron de estudiar la inteligencia de enjambre en optimización y robótica, donde citan, por ejemplo, el método de *Ant Colony Optimization (ACO)* u Optimización de Colonia de Hormigas, el cual modela el comportamiento de las colonias de hormigas. Como son insectos sociales el gobierno de la colonia está enfocado a la supervivencia de ella por encima de la supervivencia de los individuos. La meta de una colonia es encontrar comida y llevarla al nido, para lo cual, las hormigas transitan por el entorno guiados por los rastros que otras hormigas dejaron al pasar, siendo más propenso seguir los caminos donde ese rastro es más fuerte. Este rastro se denomina feromona, y es más fuerte cuando el rastro es nuevo y la comida que transportaba la hormiga era de mejor calidad. Con

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

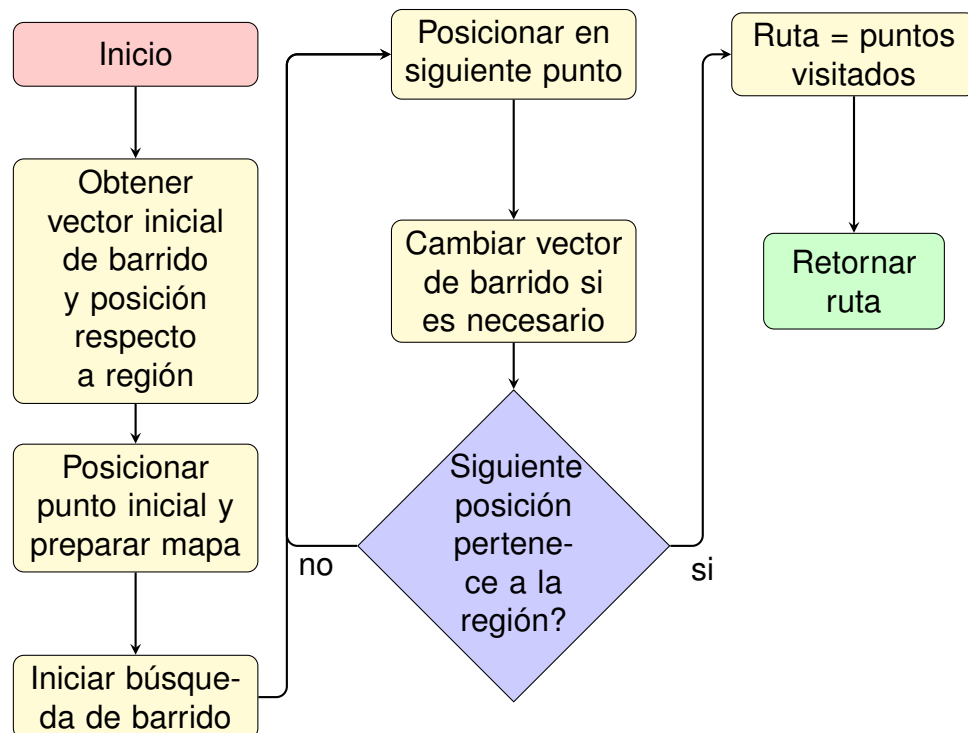


Figura 2.5: Diagrama de secuencias para calcular ruta a seguir según un patrón de cobertura por barrido. **Fuente:**Elaboración Propia.

bajos niveles de feromonas, las hormigas pasan a transitar en una forma aparentemente aleatoria, pero son guiadas por la mínima información que pueden obtener de sus “sensores” que cambian continuamente debido a la falta de una certeza de la información recibida y el ruido del entorno. Una forma de recorrer y cubrir el entorno con un vehículo autónomo se podría dar cuando se modele inversamente al método, es decir, que mientras menos explorado o menos reciente sea el valor obtenido de la “feromona virtual” de un punto en particular, sea más probable seguir en esa dirección. En cualquier caso, con este método no se busca realizar acciones que sean más beneficiosas en el futuro a largo plazo, sino solo el futuro inmediato, por lo que no es apropiado para el objetivo de este TFG.

Entre los métodos disponibles de cobertura de área existe un método que encuentra patrones de cobertura por barrido en polígonos o regiones, usando parámetros que definen una serie de puntos a visitar, de tal forma a poder cubrir

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

áreas específicas. La implementación propuesta por A. Sakai y otros [36] implica seguir la secuencia mostrada en la Figura 2.5. Como se puede observar, y teniendo en cuenta algunos parámetros definidos, el resultado del método presentado en el diagrama consiste en que, a partir de un punto inicial y un vector, se consiga recorrer una serie de segmentos cortos de la misma distancia colocados uno a continuación del otro en la misma dirección. Para lograr visitar toda una región se cambia la dirección del vector de barrido, cuando un próximo punto se encuentre cruzando los límites perpendiculares a la dirección del vector. Cuando la siguiente posición cruce los límites paralelos significa que los puntos visitados cubren totalmente la región en forma de barrido. Este método es realmente útil para tareas de rastreo y vigilancia, debido a que se considera que se necesita visitar la mayor cantidad de área en el menor tiempo. Por otro lado, cualquier vehículo que utilice este método debe poder comenzar y terminar la misión sin que existan errores, cambios de región o eventos inesperados debido a que perjudicarían el resultado de una forma directa. Otra desventaja percibida corresponde a que los vehículos se encontrarán planificando muchas rutas, debido a que se generan muchos puntos a visitar si es que se lo quiere usar exclusivamente como planificador global. Pero así también, el sistema se verá comprometido si se adiciona una planificación local entre punto y punto debido a que las rutas se verían modificadas según cada tipo o técnica de planificación. Por último en cada inicio de la planificación siempre se busca empezar la cobertura desde el lado opuesto al lado más largo de la región, por lo que idealmente el vehículo, siempre debe estar en esta posición, y si existen reinicios imprevistos a mitad de camino, el vehículo comenzará de nuevo la misión. Por estas razones, el método de patrones en polígonos no se considera óptimo ni suficiente para el caso específico de este TFG no obstante, se puede utilizar como punto de comparación.

Actualmente, en la rama de ciencias de datos, en trabajos de búsqueda se usan técnicas o métodos que optimizan la búsqueda de nuevas informaciones.

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

Se encuentra muy común realizar procesos que optimicen ciertos valores con respecto a nuevas informaciones obtenidas al interactuar con el entorno de búsqueda, consiguiendo realizar procesos que son óptimos para entornos no deterministas. Los dominios o entornos, en su gran mayoría, pueden ser controlados mediante Procesos de Decisión de Markov que definen una serie de parámetros fácilmente deducibles o asignables. Cuando se conoce y se puede obtener la totalidad de los valores de los parámetros y a su vez realizar acciones que con certeza maximicen cierto parámetro se conoce al proceso de control como Proceso de Decisión de Markov Totalmente Observable; el caso más generalizado, denominado Proceso de Decisión de Markov Parcialmente Observable (PODMP), se basa en observaciones, y probabilidades para definir la relación entre el agente y el entorno. En el caso real siempre se tendrá un POMDP ya que es imposible conocer a la perfección todos los parámetros o medirlos, por eso en este TFG se utiliza el POMDP para obtener misiones para los vehículos. El libro de C. White [37] define los siguientes parámetros para el POMDP:

- **S** (Lista de estados $s_1, s_2, \dots, s_n$): Son todos los estados posibles dentro del entorno.
- **A** (Lista de acciones $a_1, a_2, \dots, a_m$): Son todas las acciones posibles que se pueden tomar.
- **T** ($S \times A \times S \rightarrow [0, 1]$): Es una función de transición de un estado a otro, y se interpreta como una probabilidad de encontrarse en el estado s' cuando se ejecuta la acción a en el estado s , ejemplo: $T(s, a, s') = p(s'|s, a)$.
- **R** ($S \times A \rightarrow R$): es la recompensa obtenida al ejecutar la acción a en el estado s .
- ω : Lista de observaciones $o_1, o_2, \dots, o_l$.

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

- $O (S \times A \times \omega \rightarrow [0, 1])$: es una función de observación que representa la probabilidad de observar o cuando la acción a fue ejecutada.
- $\gamma \in [0, 1]$: es un factor de descuento usado para variar la obtención de la mejor acción a ser ejecutada.

Con estos parámetros, se puede modelar cualquier situación si se definen bien los mismos, por ejemplo, se pueden definir los estados como conjuntos de posiciones del vehículo con los mapas de parámetros, acciones como decidir viajar a nuevos *waypoints*, las recompensas definibles según la calidad de una información recibida, las observaciones que pueden definirse como los parámetros de calidad del agua, etc. Resolver los POMDPs corresponde a encontrar la política óptima que maximice la recompensa esperada al tomar una serie de acciones. Encontrar esta serie de acciones podría tomar un tiempo infinito de cálculo, si los entornos puede variar sin previo aviso, por lo que en el caso real se deben definir límites temporales o de cantidad de acciones a tomar debido a que es imposible recuperar toda la información de tomar infinitas acciones posibles en entornos como éste. Para esto se puede utilizar el método de Árbol de Búsqueda Monte-Carlo (MCTS) para obtener la mayor recompensa posible teniendo en cuenta los parámetros del POMDP. MCTS, o *Monte Carlo Tree Search*, es un método que puede ser utilizado para planificar misiones bajo incertidumbre, según D. Silver y otros [38], esto se debe a que el algoritmo define un árbol con estados como nodos u hojas, y las une mediante las ramas, que se refieren a realizar ciertas acciones, luego se determina cual camino entre nodos es el más apropiado según una recompensa esperada. Cuando la idea se concentre en obtener una lectura de un entorno genérico utilizar, MCTS para resolver una versión simplificada del POMDP es suficiente, pero se puede también buscar cubrir un área con respecto a la cantidad observada de cierto parámetro, ya que, en situaciones reales, no es posible obtener el valor correcto de la totalidad de los

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

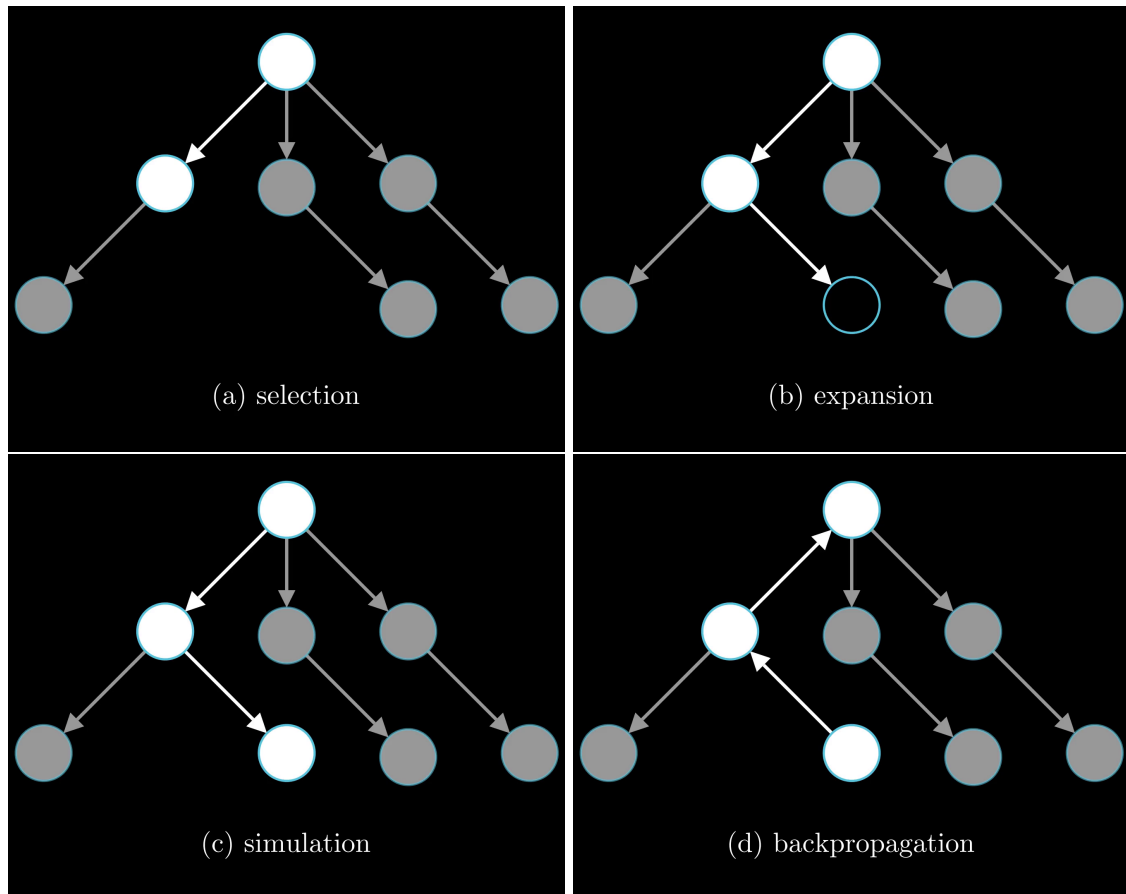


Figura 2.6: Ejemplo del procedimiento del Monte Carlo Tree Search **Fuente:** Elaboración Propia

puntos que conforman una superficie a cubrir.

Retomando la idea de crear vehículos con capacidad autónoma de traslado, se tiene en cuenta todo lo mencionado anteriormente en cuanto a la situación interna y externa de un vehículo, con esto se procede a obtener una misión usando POMDP resuelto mediante MCTS. Según la literatura disponible, para aplicar el algoritmo MCTS, primero se debe crear una hoja inicial, a partir de esto, comienza un proceso iterativo que selecciona una hoja para convertirla en nodo, se expande el árbol creando una hoja nueva y se la conecta al nodo mencionado, para formalizar la creación de la hoja se debe simular la transición entre el nodo y la hoja, es decir, simular tomar una acción. Una vez finalizada la creación de estas hojas, se realiza una propagación hacia atrás para determinar el mejor conjunto

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

de acciones para el caso según la recompensa esperada en cada caso. El MCTS debe poder simular una acción desde el estado anterior para que se encuentre propiamente en el estado de la hoja nueva, y, por último, se debe propagar la recompensa hacia atrás para informar a la raíz sobre el rendimiento de la nueva hoja. De acuerdo con lo establecido, la Figura 2.6 muestra un ejemplo del proceso de la búsqueda por árbol de Monte Carlo. Se puede observar y definir cuál es el set de acciones que provee de mejores resultados para lograr el objetivo. Por razones previamente descritas, la búsqueda se realiza hasta una profundidad definida d que permita encontrar resultados en tiempos relativamente cortos.

Planificación de Movimiento

La planificación de movimiento es una rama de la robótica que busca optimizar los movimientos que un robot realizará. Se usa principalmente porque un planificador busca una secuencia válida de movimientos que son posibles en un robot y permiten llegar a la posición deseada. En cuanto al grupo de robots móviles, existen 2 grandes conjuntos de planificaciones: La planificación global de ruta y la local de ruta.

Planificación Global: Planificación que toma en cuenta informaciones que permitan a un vehículo encontrar una ruta teniendo en cuenta factores de posición inicial y final, además de un mapa que corresponde a un modelado del entorno donde se encuentra el vehículo. En casi la totalidad de los casos se usa al comienzo de un viaje entre dos puntos, debido a que el fin de una ruta planificada globalmente comprende las rutas que el vehículo debe seguir y las posiciones que debe alcanzar para llegar a la meta o posición final deseada.

Planificación Local: Como la información del entorno va actualizándose continuamente, es preferible que además de planificar la ruta de una forma global, se tenga también una planificación que busque realizar el mismo movimiento que el planificador global pero además, de ser necesario, replanifique las rutas

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

deseadas para evitar colisionar con cualquier obstáculo presente o se adecuen a variaciones imprevistas del entorno.

Una buena planificación de rutas tiene en cuenta los factores de optimización, las restricciones, como por ejemplo la movilidad del vehículo, las velocidades posibles, los obstáculos estáticos o dinámicos y las regulaciones. A su vez, el costo de cómputo no debe ser demasiado grande ni consumir demasiados recursos, de tal forma a favorecer replanificaciones de ruta globales, o planificaciones locales óptimas. De igual manera, la planificación global se enfoca precisamente en entender y obtener rutas libres de obstáculos *a priori*, y la planificación local en mantener las direcciones dadas por la planificación global, adecuándose al entorno dinámico. Por estas razones es importante separar ambas planificaciones en la medida que sea posible, aprovechando las características que posee cada una de ellas.

En el 2013, O. Souissi y otros [28] resumieron las técnicas de planificación de ruta global para vehículos aéreos autónomos. La comparación de los métodos se enfoca en 3 de los más conocidos: A^* , *RRT* y *Potential Fields*. Cada uno de estos pueden ser definidos como planificadores globales, pero su rápida entrega de resultados y la dinámica de adaptación a cambios, permite que con pequeñas modificaciones, puedan ser utilizados como planificadores locales. Un vehículo autónomo puede usar un planificador de ruta, siempre y cuando este se adecue a las ventajas y limitaciones que presenta el mismo, y para ello en este TFG se investiga a profundidad sobre estos métodos para hacer una elección científica del planificador de ruta más adecuado. Una comparación inicial de estos métodos realizó F. Peralta [29], dicho trabajo tenía como objetivo comparar los modelos actuales de planificadores de ruta en ambientes acuáticos. De forma tal a seleccionar el método de planificación de rutas con el mejor rendimiento posible es necesario entender a profundidad cada uno de ellos y definir métricas que permitan validar la elección de una técnica por sobre otras. A continuación, se

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

presenta parte del trabajo, actualizado y revisado para los objetivos y el alcance de este TFG.

A* [30]: Estudiado por S. -G Cui y otros, el A^* es un planificador de ruta que agrega una función heurística al algoritmo propuesto por E. Dijkstra [31], el cual busca una ruta entre dos puntos, examinando los vecinos inmediatos de una posición inicial, luego estos vecinos son considerados como iniciales y se examinan sus vecinos que todavía no han sido examinados, hasta encontrar la posición de la meta, uniendo la posición inicial y la final con un conjunto de secuencias. En lugar de examinar en todas las direcciones, es decir, a todos los vecinos de un punto, A^* utiliza una fila, que prioriza la evaluación de nodos más cercanos a la meta, o que poseen unos menores costos de movimiento. En este caso, el costo es definido por las restricciones de movimiento del robot. El programador puede, por ejemplo, asignar mayores costos a nodos que estén en las cercanías de los límites u obstáculos. Estos costos son sumados y asignados como la prioridad de los nodos, obteniendo nodos de menores costos con mayores prioridades (ya que a mayor costo, menor prioridad). De esta manera existe una probabilidad mayor de llegar en menor tiempo a la meta, manteniendo los niveles de seguridad y convergencia que el Dijkstra ofrece. Como se observa en la Figura 2.7., el algoritmo sigue una secuencia fácil de comprender, el cual contiene 3 bloques de decisión, dos utilizados durante la creación del mapa, y uno durante la creación de la ruta.

El diagrama sigue una secuencia en donde se crea una fila con prioridad, examinar un nodo en la frontera de búsqueda y luego se agregan en la fila con prioridad los próximos nodos a ser examinados. Se busca encontrar el nodo que corresponde a la meta, y al encontrarlo una ruta es generada desde este nodo final hasta el inicial.

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

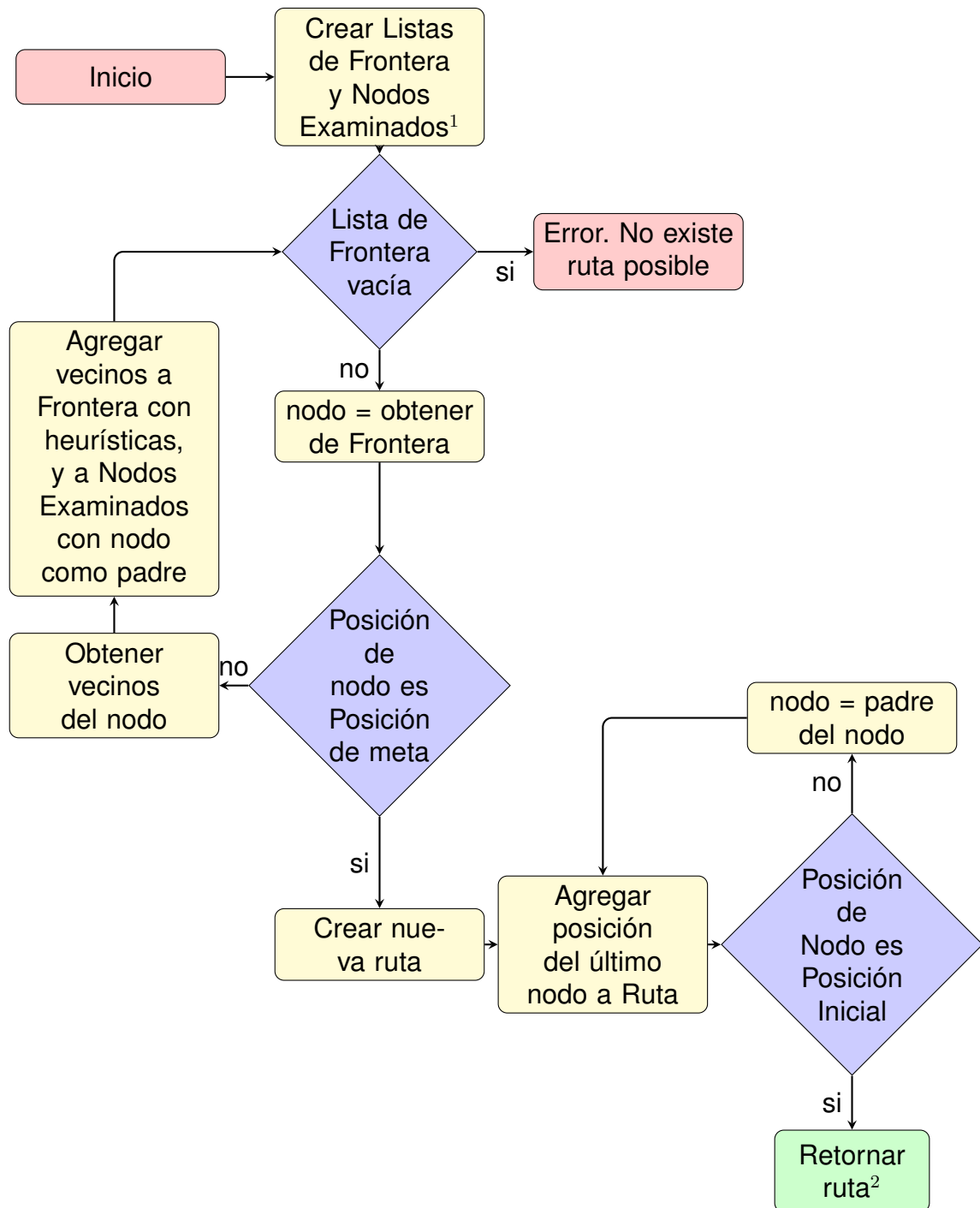


Figura 2.7: Diagrama de secuencias para A*. **Fuente:**Elaboración Propia.

1: Ambas listas incluyen inicialmente la posición del robot.

2: Se deben voltear las posiciones para ir de nodo inicial a final.

Campos Potenciales: Planificadores de ruta con campos potenciales, o, en inglés, *Potential Fields (PF)*, son utilizados por los beneficios y las mejoras que ofrecen por la posibilidad de entregar trayectorias de funciones suavizadas, en

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

vez de cambios discretos de dirección. La idea del *PF* consiste en obtener valores potenciales para cada punto dentro de la región de manejo o mapa, teniendo en cuenta dos tipos de campos: los repulsivos, y los atractivos. J. Barraquand y otros [32] fueron los primeros en investigar sobre este método y definieron 2 principales funciones: Ecuación 2.1, y Ecuación 2.2, como las ecuaciones de atracción y repulsión que posee un punto, por lo tanto, se deben sumar y asignar como valor potencial del punto p .

$$U_{atr} = \frac{1}{2} * k_a * pdist(pose, goal) \quad (2.1)$$

$$U_{rep} = \frac{k_r}{pdist(pose, obst_i)} \quad (2.2)$$

En estas ecuaciones, k_a y k_r son constantes de atracción y repulsión, y $pdist$ es la distancia euclidiana entre la posición examinada y la meta actual, o el obstáculo $obst_i$, según sea el caso. Una ventaja del *PF* es que, modificando las constantes definidas, se controla específicamente zonas de peligro (obstáculos, zonas no visitables, etc.) con la ecuación 2.2. Un mapa potencial es calculado para cada meta y usado para determinar en qué dirección debe ir el ASV, la cual es una trayectoria generada mediante el uso del gradiente descendente, esto es generar una secuencia de puntos en los cuales el valor del potencial es cada vez menor hasta alcanzar la posición con potencial 0 ($U_{atr} = 0$). En este algoritmo, no se generan puntos de visita intermedios que quedan notablemente fuera de la recta que une la meta y la posición como sucedía en A^* , pero si una trayectoria a seguir que puede llevar al vehículo a encontrarse en mínimos locales que no corresponden a la meta establecida.

El algoritmo se puede observar en la Figura 2.8., el cual muestra el procedimiento simple del método de Campos Potenciales. Los bloques de acción que requieren de mucho procesamiento computacional son: el de Generar el mapa

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

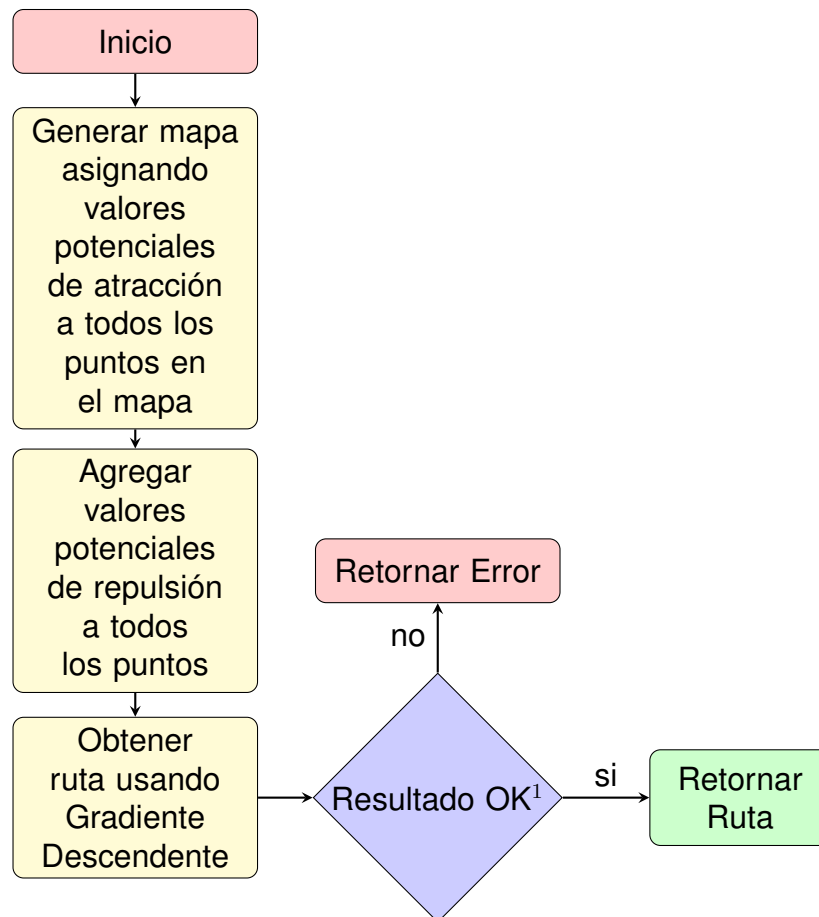


Figura 2.8: Diagrama de secuencias para *Potential Fields*. **Fuente:**Elaboración Propia. 1: OK: ruta completamente definida.

con campos atractivos, y el que agrega los campos repulsivos. Se puede realizar el gradiente descendente para obtener una ruta suavizada, o mover en tiempo real al vehículo a la zona de menor potencial.

Arbol Aleatorio de Exploración Rápida: Uno de los planificadores basados en muestras más populares es el *Rapidly-Exploring Random Tree (RRT)*, es una técnica propuesta por O. Adiyatov [33], este método no solo provee una ruta rápidamente, sino que también exige muy poco poder de computación computacional en comparación con otros planificadores. *RRT* hace crecer un árbol definiendo como raíz a la posición inicial, y generando nodos que se conectan usando “ramas”. Estos nodos son creados aleatoriamente y agregados al árbol hasta que un nodo aleatorio se conecte con la posición final después de cierto número de

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

iteraciones que ubique al nodo en las proximidades de la posición final o meta. De forma a mejorar el tiempo de cálculo, se utiliza el modelo o técnica que incluye heurísticas denominado *RRT**, que es una modificación del planificador original que agrega funciones de elecciones de conexiones a nodos padre en función a la distancia que del punto a la meta final. El diagrama de secuencias para este método es presentado en la Figura 2.9. La técnica se separa en tres algoritmos diferenciados, los cuales son accedidos dependiendo de la decisión que se tome respecto a los puntos, o nodos, estudiados a medida que el árbol crece. El flujo empieza creando la raíz con la información brindada, luego se crean nodos que cumplen una serie de procesos para conectarlos a través de ramas con nodos usando una función que minimiza el costo de esta conexión, hasta encontrar el nodo de la meta, para luego calcular la ruta se inicia con el nodo final y se recorre con propagación hacia atrás hasta alcanzar la raíz, la inversa de esta ruta es la entregada como salida del componente del planificador de movimiento.

En *RRT**, un árbol es inicializado, conteniendo como raíz al nodo inicial, luego una posición aleatoria es obtenida dentro de la espacio posible, posteriormente, se adapta esta posición por medio una función, para ubicar este nodo nuevo lo más cerca posible de un nodo del árbol seguidamente O. Adiyatov y otros [33] sugieren que el nuevo nodo adaptado obtenga un costo y se conecte al mejor nodo padre posible que se encuentre en la vecindad de este nodo por último este nodo es conectado y luego se comienza de nuevo la iteración. Para facilitar la obtención de rutas, se puede generar nodos en las cercanías de la meta, favoreciendo a la conexión de la meta con el árbol. *RRT** es uno de los planificadores que puede calcular una ruta en direcciones de cualquier ángulo, significando que el robot es instruido de viajar en cualquier dirección al contrario del planificador *A**. Además cabe destacar que *RRT** agrega muchos nodos innecesarios, pero provee una ruta más corta y directa entre los puntos de cálculo.

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

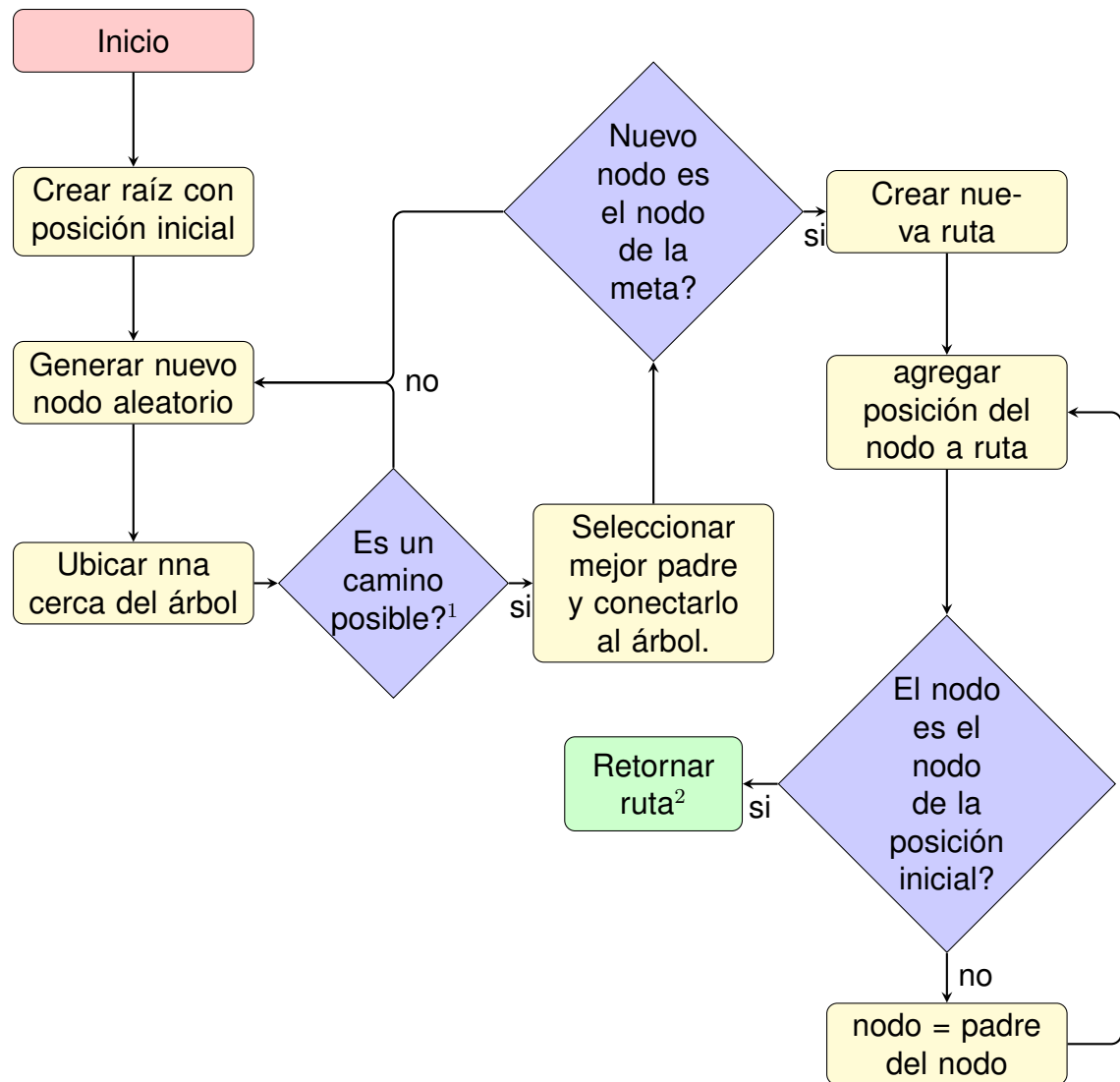


Figura 2.9: Diagrama de secuencias para *RRT**. **Fuente:**Elaboración Propia.

1: Camino posible entre nuevo nodo y el nodo más cercano.

2: Se deben voltear las posiciones para ir de inicio a fin.

Método de la Marcha Rápida y variantes: El Método de la Marcha Rápida, o *Fast Marching Method (FMM)*, fue desarrollado para resolver problemas de condiciones de frontera de la ecuación eikonal, ecuación no-lineal en derivadas parciales encontradas en problemas de propagación de onda. Esta ecuación fue utilizada para planificar rutas en un ASV por Y. Liu y otros [34], y tiene la forma de:

$$|\nabla T(p)|V(p) = 1 \quad (2.3)$$

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

donde $T(p)$ representa el tiempo de arribo al punto p y $V(p)$ la velocidad de propagación en el punto p . De forma a discretizar la ecuación, para calcular $T(p)$ $\forall p(x, y)$, se utiliza una mapa M que contiene todos los puntos $p \in \mathbb{R}^2$ que pertenecen a la región de interés, y un espacio de movimiento Δx , y Δy , para describir los mínimos movimientos x e y posibles. Por lo tanto, los vecinos *Von Neumann* de un punto p son $(x + \Delta x, y)$, $(x - \Delta x, y)$, $(x, y + \Delta y)$ y $(x, y - \Delta y)$. Luego, el valor de $T(p)$ puede encontrarse mediante la discretización:

$$T_1 = \min(T_{(x+\Delta x, y)}, T_{(x-\Delta x, y)}) \quad (2.4)$$

$$T_2 = \min(T_{(x, y+\Delta y)}, T_{(x, y-\Delta y)}) \quad (2.5)$$

$$\left(\frac{T_{(x, y)} - T_1}{\Delta x}\right)^2 + \left(\frac{T_{(x, y)} - T_2}{\Delta y}\right)^2 = \frac{1}{(V_{(x, y)})^2} \quad (2.6)$$

Finalmente, la solución de estas ecuaciones está dada por:

$$T_{(x, y)} = \begin{cases} T_1 + \frac{1}{V_{(x, y)}}, & \text{if } T_2 \geq T \geq T_1. \\ T_2 + \frac{1}{V_{(x, y)}}, & \text{if } T_1 \geq T \geq T_2. \\ \text{Sol. of Eq. 2,6,} & \text{if } T > \max(T_1, T_2). \end{cases} \quad (2.7)$$

La matriz de velocidades $V_{(x, y)}$ puede, por ejemplo, tomar valores binarios 0 o 1, implicando la relación directa entre esta matriz y la de representación del mapa M . El paso Δ se define como único para ambas direcciones x e y , y se toma como unidad para facilitar la resolución de las ecuaciones y estar en concordancia con los valores posibles de M . Para determinar los valores correctos de forma iterativa, el orden de la obtención de estos valores debe coincidir con la propagación de esta hiper-superficie, es decir, los valores deben ser calculados desde un punto inicial hacia afuera de él, por lo tanto se puede decir que la resolución del *FMM* es parecida a la del Dijkstra pero de forma continua.

Se presenta en la Figura 2.10 el diagrama de secuencias para obtener la ma-

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

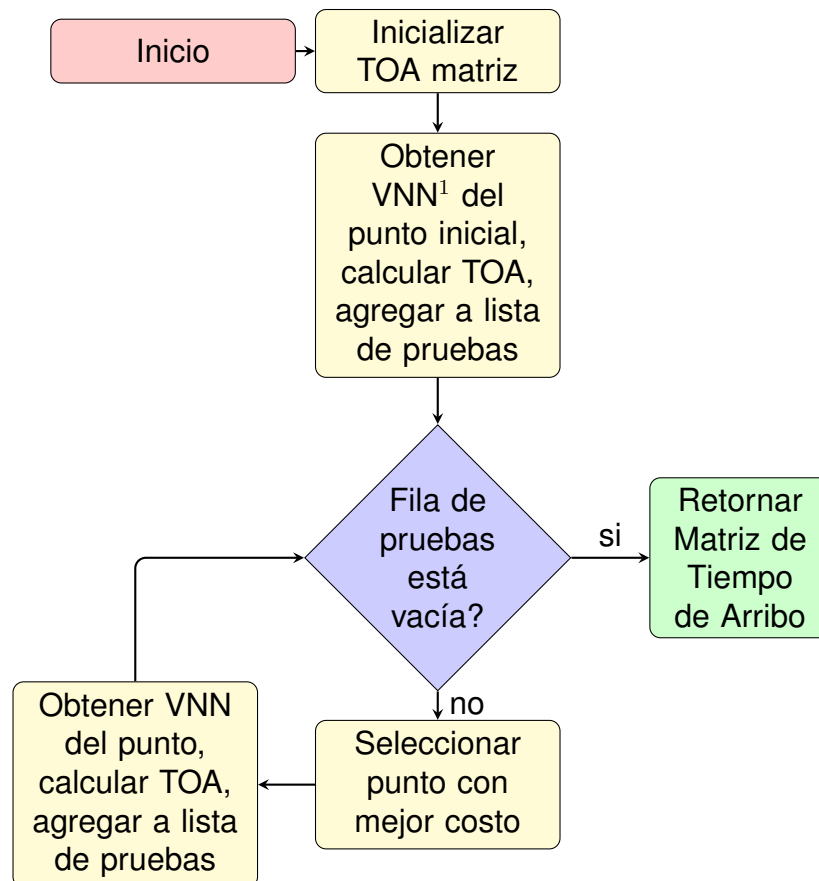


Figura 2.10: Diagrama de secuencias para calcular la Matriz de Tiempo de Arribo. **Fuente:**Elaboración Propia.
1: Von Neumann Neighbors.

triz de tiempo de arribo, este algoritmo requiere de un punto (o puntos) inicial(es) y un mapa de velocidades, y retorna la matriz de Tiempo de Arribo, o TOA ,por sus siglas en inglés *Time of Arrival*, que puede ser considerada como la matriz final para calcular la ruta usando gradiente descendente.

Se debe tener en cuenta que, en lugar de calcular por medio de división en puntos lejanos, en juicio y conocidos, que fue lo establecido por Y. Liu y otros [34], se realiza mediante una propagación desde un punto inicial conocido, y de igual forma, el tiempo de arribo T_{p_0} al punto inicial es siempre cero. Como se puede visualizar en el diagrama, la función retorna una matriz de tiempos de arribo al punto genérico p . La matriz de velocidades v se genera dando valores de uno a los lugares en donde es posible alcanzar la velocidad deseada, y cero en donde

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

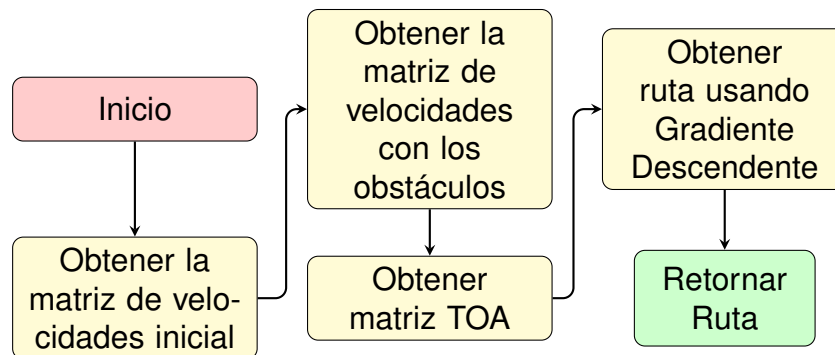


Figura 2.11: Diagrama de secuencias para calcular una ruta utilizando *FMS*
Fuente:Elaboración Propia.

existan obstáculos. Una función de costo retorna el valor de la Ecuación 2.7, siempre y cuando se verifique que el valor de la velocidad sea mayor a cero en el punto para evitar costos indefinidos ($cost \rightarrow \infty$).

Fast Marching Square Method (FMS) o el método de marcha rápida al cuadrado realiza dos veces el *FMM*, con la diferencia de los puntos de inicio de propagación de la interfaz. Los autores Liu y Otros proponen el método ya que el nivel de seguridad de las rutas calculadas con *FMM* no son tan óptimos por lo que se realiza una primera obtención de la matriz TOA para obtener los mapas de velocidades desde los obstáculos, es decir obtener tiempos virtuales de arribo de los obstáculos a todos los puntos del mapa, y luego se realiza una segunda vez la obtención de la matriz TOA para obtener la verdadera matriz pero usando como mapa de velocidades a la matriz entregada por el cálculo de la matriz TOA de los obstáculos. El Algoritmo de *FMS* que fue propuesto por Y. Liu y otros [34], muestra que se utiliza el mismo método dos veces, la primera vez utiliza el *FMM* para calcular el mapa de velocidades teniendo en cuenta los obstáculos presentes, por lo que el programa debe estar preparado para recibir como estado inicial a más de un punto, y a partir de esos puntos generar el mapa de velocidades, y la segunda vez es igual a la forma usual del *FMM*. El diagrama se encuentra en la Figura 2.11.

Con la presentación de los métodos actuales y más utilizados de planifica-

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

ción de ruta, finaliza la subsección de Planificación de movimiento. Debido a la existencia de varios métodos aplicables para una situación general de planificación de movimiento es necesario seleccionar un planificador apropiado para las circunstancias del proyecto y de la misión. Para el efecto pueden definirse puntuaciones y métricas a la hora de implementar el proyecto, dicho proceso se realiza en el siguiente capítulo. Antes de eso, es necesario definir otros datos importantes para la planificación correcta de ruta.

2.2.4. Información Dinámica

Es de suma importancia reconocer y comprender la información del entorno de estudio y su implicancia dentro del diseño del subsistema de guía. La modelación del entorno es una parte de la definición completa del entorno, pero también se tienen informaciones que no son del tipo deterministas, es decir, que no dependen del tiempo. Un ejemplo de este tipo de información es la disponibilidad de un vehículo durante la misión, cualquier vehículo puede sufrir un desperfecto antes o durante el proceso, que imposibilite su presencia para terminar la misión. Otra información que puede cambiar de forma no prevista es la inclusión de un nuevo vehículo una vez comenzada la misión. Si el sistema de guía de múltiples drones puede superar este tipo de situaciones el sistema es más robusto. Obstáculos temporales o no previstos modifican las rutas planificadas de modo a que la información del mapa tampoco puede considerarse estática. Otras causas de replanificación incluyen reinicios de sistemas imprevistos, niveles de batería proyectados como insuficientes, entre otros. Todas estas informaciones, al igual que otras no especificadas, obligan a que cualquier vehículo pueda considerar que la decisión tomada no puede ser la óptima a muy largo plazo, pero aun así debe tomar la decisión que cree que es la más óptima posible, y debe estar preparado para modificar la ejecución de la decisión o realizar otra que mejore el rendimiento del vehículo, siempre, a largo plazo. Como estos datos pueden ser recibidos

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

en cualquier momento, se los clasifica como información asincrónica, pero como el flujo de datos no necesariamente debe seguir un orden preestablecido o el tipo de información puede no estar estructurado, aparte de ser asincrónica, la información también se define como dinámica. La información dinámica es objeto de estudio actual en economía e ingeniería de ciencia de datos, y según lo expuesto por P. H. Au [39] se puede utilizar para persuadir a que un receptor parcialmente informado tome decisiones y realice acciones al respecto. El mismo trabajo concuerda con la teoría de Markov mencionada anteriormente y su parte principal se basa en la emisión y recepción de señales, ya que una señal es un transportador de datos ideal para intercambiar información dinámica entre procesos concurrentes. En este TFG, la toma de decisiones y el rendimiento de la misión son totalmente dependientes de estas señales y varían de acuerdo a la estructura generada en torno a las mismas. En definitiva, la arquitectura de toma de decisiones se debe basar en las señales recibidas en cualquier momento, de parte del entorno, de otros vehículos y del sistema interno de cada vehículo. Siendo la misión principal recorrer la máxima cantidad de área posible con los vehículos, una de las informaciones que puede ayudar a cumplir con la misión es el tiempo que ha pasado desde la última vez que se ha visitado cierto punto del Lago. Luego, si se considera encontrar una serie de puntos se puede definir el mejor camino posible para lograr el resultado.

2.3. Subsistema de Comunicación para ASVs

La comunicación es fundamental en todos los procesos que involucren una cooperación, ya que permite que todos los agentes intervinientes modifiquen alguno de sus parámetros para optimizar su propia tarea para cumplir la misión asignada. Mediante tecnologías de comunicación inalámbrica, los drones de superficie pueden compartir datos sin la necesidad de estar conectados físicamente, cosa que sería imposible para un enjambre de drones, en entornos acuáticos

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

de dimensiones de entorno grandes. Implementar apropiadamente una tecnología de comunicación inalámbrica consiste en aprovechar los sistemas de comunicación disponibles en el mercado y combinarlos con los otros sistemas del ASV de tal forma a usarlos en casos de necesidad, con el menor gasto de energía posible.

2.3.1. Comunicación Inalámbrica

Para lograr una comunicación inalámbrica eficiente se propone que el consumo de energía debe ser mínimo, los enlaces deben estar siempre disponibles, la señal de comunicación no debe ser notoriamente degradada por interferencias presentes y la comunicación debe ser segura. Otras características encontradas en la literatura incluyen el rango, el área, la longitud de los mensajes y el tiempo de propagación. La literatura de enjambre de robots siempre apunta a usar módulos que estén diseñados para los enjambres de robot, es decir, módulos de interconexión nodal entre dispositivos. De entre ellos se destaca transceptor Xbee ofrecido por Digi Internacional Ing. [40]. Este dispositivo usa el protocolo ZigBee que interconecta todos los dispositivos según un algoritmo que combina bajo-consumo y fiabilidad de interconexión, generando lo que denomina como redes de sensores inalámbricos. La versión PRO del más reciente de los módulos XBee disponibles tiene una característica de transmisión de hasta 3200 metros, apropiado para generar conexiones entre dispositivos de corto alcance. En su libro, R. Faludi [41], afirma que existe una cantidad considerable de opciones parecidas al XBee, pero el XBee es preferido ante las demás opciones debido al esfuerzo reducido en desarrollo, un costo más económico y un ecosistema tecnológico de amplio espectro. En el Anexo A se encuentran otros módulos que usan ZigBee.

Por otro lado, se puede hacer uso de módulos que usan la tecnología WiFi (IEEE 802.11n), para lo cual se estudia y considera el módulo más común uti-

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

lizado en el mercado. El *ESP8266*, desarrollado por Espressif Systems [42], es un módulo de bajo costo que permite conexión WiFi entre dispositivos. Según diferentes estudios ([43–45]), en condiciones normales, los módulos pueden conectarse a una red WiFi hasta a 150 metros de distancia con antena PCB en el módulo, y una antena normal en el *router* WiFi. Las pruebas con antenas normales en el módulo arrojaron resultados de conexión de hasta 300 metros con poco porcentaje de pérdida de paquetes. Con antenas especiales en el *router*, en el último trabajo mencionado se consiguieron enviar y recibir paquetes a distancias de 1.42 km. Utilizar módulos WiFi para interconexión entre dispositivos es posible, pero muy difícil, debido a la interconexión necesaria entre dispositivos.

También es posible utilizar módulos que posean conexión a antenas de telefonía móvil, aprovechando la disponibilidad de la red 3G/4G en Paraguay, y que existen diversos dispositivos embebidos capaces de conectarse a estas redes para hacer uso de ellos. La compañía Telit ofrece módulos 4G [46] diseñados para Internet de las Cosas capaces de conectarse a antenas de redes de telefonía móvil, con versiones para cada continente. Aunque la tecnología 3G/4G no está propiamente diseñada para enjambres, si permite una conexión indirecta entre dispositivos separados por cualquier distancia siempre y cuando estos puedan conectarse a antenas de comunicación de telefonía móvil usando Internet. Es importante conocer que para este tipo de tecnologías, según *JPL's Wireless Communication Reference Website* [47, 48], el tamaño de la “celda” es de entre 5 a 30 kms. La “celda” se refiere a un área geográfica cubierta por un transmisor. Para el caso de este TFG, se puede considerar que los vehículos están interconectados siempre y cuando exista conexión de cada vehículo a alguna antena cercana, y que exista un software que permita comunicación entre los vehículos de forma segura y eficiente.

Se muestra en la Tabla 2.1 un resumen de las tecnologías de comunicación inalámbrica encontradas en la literatura, conjunto con dispositivos de ejemplo y

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

su nivel de adaptación a vehículos autónomos de superficie para una comprensión de lo definido en esta sección.

Tabla 2.1: Características de algunos sistemas de comunicación

Especificaciones	Xbee	Módulos WiFi	M. de Telefonía Móvil
Rango [metros]	3200	~300 (1420 ¹)	< 30000 ²
Protocolo	ZigBee	IEEE 802.11n	3G/4G
Facilidad de Implementación en ASVs	Múy fácil	Difícil	Fácil

¹ Con antenas especiales.

² Distancia máxima permitida de un dispositivo a una antena. Rango máximo de conexión a antenas de 3G/4G según [47, 48].

Fuente: Elaboración propia.

2.3.2. Protocolos de Comunicación para Drones de Superficie

Sobre el protocolo de comunicación, o más explícitamente sobre el software de comunicación, numerosos trabajos concluyen que la coordinación en una robótica de enjambre puede verse afectada si la comunicación no es la ideal. Dicha afirmación toman en cuenta M. Li y otros [49], quienes estudian sobre las arquitecturas y aplicaciones disponibles en el año 2008 para las redes de comunicación de enjambres de robot. Los protocolos a utilizar deben seguir la idea de que la coordinación de los vehículos sea distribuida, robusta y que la calidad de servicio o *Quality of Service* (QoS) sea suficiente para asegurar la veracidad y la integridad de los datos. En la literatura, es usual encontrar topologías de conexión del tipo malla para enjambres, debido a que permite una robusta y fiable conexión, donde existen conexiones redundantes y en casos más extremos, topologías de tipo totalmente conectada que son las topologías en donde todos los nodos o dispositivos están interconectados. Estas dos topologías comparten la característica de poder conectar a dos vehículos por más de un camino o ruta. Para compartir mensajes, estas topologías necesitan de formatos de mensaje tales que la carga útil sea leída únicamente por el receptor, así como los identificadores deben ser únicos para evitar que el mensaje sea leído errónea-

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

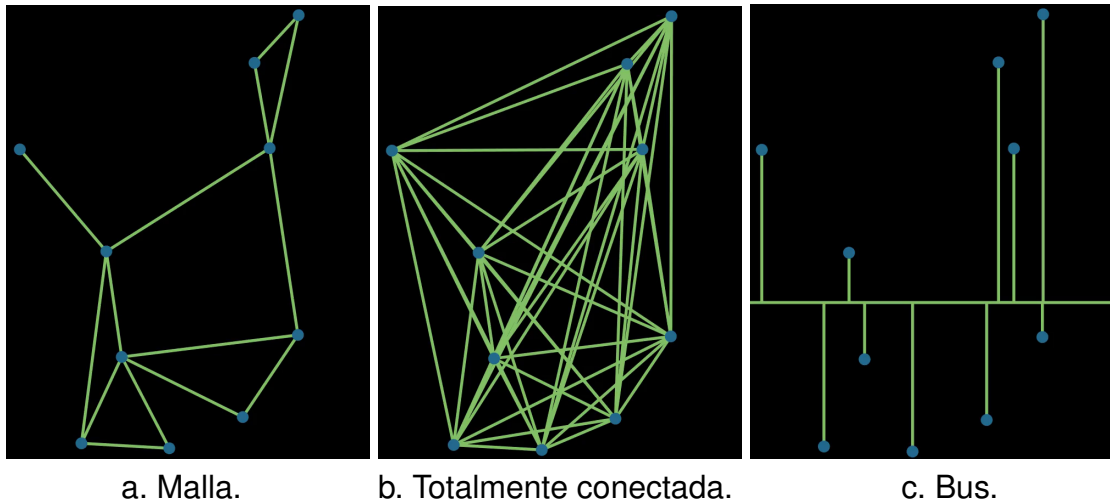


Figura 2.12: Distintas topologías de conexión **Fuente:** Elaboración Propia

mente por más de un receptor. La complejidad de conexión de esta topología se dificulta con la distancia, ya que, a mayor distancia entre dispositivos, menor la cantidad de conexiones posibles para compartir o enviar/recibir mensajes entre ellos. Una forma de solucionar esta problemática es instruir a los vehículos a conectarse a servidores o corredores de información, los cuales funcionarían como puentes transparentes entre dispositivos, manteniendo la QoS, disminuyendo, en parte, la distribución de trabajo, pero aumentando la velocidad de comunicación, la fiabilidad de datos entre otros. Esta definición puede ser considerada como la topología de bus, en donde los dispositivos se conectan a través de buses que los interconectan. En la Figura 2.12 se pueden observar las topologías mencionadas.

Para casos prácticos, la topología de bus permite distintos tipos de mensajes, de entre los cuales se destaca la mensajería basada en publicación/suscripción. Esta política de intercambio de mensajes se basa en la creación de tópicos donde los dispositivos pueden publicar en algún tópico, informaciones que puedan ser útiles para otro dispositivo (acción definida como altruista, ya que no aporta provecho para el emisor), y también pueden suscribirse a tópicos para recibir informaciones externas a él. Una de las desarrolladoras de software robótico más

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

prestigiosas y extendidas en el mundo también utiliza éste sistema para la comunicación interna y entre robots. Open Robotics utiliza la mensajería pub/sub en el middle-ware ROS (*Robotic Operating System*), presentado por M. Quigley y otros [50], debido a que los mensajes compartidos entre sistemas generalmente son cortos y requieren un bajo nivel de computación adicionalmente este tipo de mensajería es el patrón de comunicación más común y útil para sistemas robóticos. Así también, la mensajería pub/sub requiere una trama de datos que se hace realmente eficiente a la hora de comunicar una cantidad considerable de sistemas debido a que, en caso contrario, un vehículo deberá tener una larga lista de todos los dispositivos conectados, tener un sistema de actualización de esta lista en tiempo real y sistemas para asegurar la comunicación que harán que el sistema de comunicación sea mucho más lento. El sistema de comunicación de ROS actúa en la capa de aplicación, como lo hacen distintos protocolos de comunicación, de entre los cuales se destaca el standard realizado por A. Banks y otros [51]: MQTT (*Message Queuing Telemetry Transport*), el cual es un protocolo de mensajería extremadamente simple y liviano. Este protocolo fue diseñado para minimizar el uso de ancho de banda de la red y los recursos de los dispositivos. El protocolo se adapta perfectamente a la idea de conectar todos los dispositivos a Internet (conocido como *Internet of Things*). Además, el protocolo está diseñado para ofrecer seguridad de la comunicación entre publicadores y suscriptores. Se puede modelar al protocolo de mensajería pub/sub para trabajar en una topología de bus, en donde los mensajes se filtran en el mismo bus y se destinan a todos los dispositivos que se hayan suscrito a un tópico en particular, de esto se encarga el *broker*, que recibe todos los mensajes, descifra los tópicos de publicación, pero no los mensajes y se encarga de enviarlos a destino, es decir, a todos los dispositivos suscritos al tópico. El protocolo MQTT no trabaja descentralizadamente, el broker se encarga de recibir y retransmitir todos los mensajes a los destinos, pero se puede trabajar y diseñar el sistema como uno

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

descentralizado.

2.4. Estrategia de Cooperación Distribuida para Drones de Superficie

A lo largo de este trabajo, se han presentado importantes algoritmos, métodos, arquitecturas de decisión y subsistemas que hacen que un sistema de guía de un vehículo sea el adecuado, pero todavía no se ha determinado ni definido sistemas para unir el trabajo de más de un vehículo, creando los sistemas multi-robot. En ese sentido, a continuación, se presenta la teoría y el diseño de cooperación distribuida, según definiciones que optimicen el trabajo de los vehículos autónomos. Usando sistemas abstractos de cooperación pueden generarse comportamientos deseados en grupos de vehículos autónomos, consiguiendo que el sistema de guía sea para múltiples drones de superficie. La investigación se enfoca en los sistemas cooperativos y en la cooperación distribuida, por lo que se definen estos ítemes, conjunto con diseños y algoritmos encontrados en la literatura actual.

2.4.1. Cooperación

En el libro titulado “La Evolución de la Cooperación”, R. Axelrod y otros [52] expresan que la teoría de la cooperación como tal nace en un grupo de individuos que persiguen intereses personales, sin la existencia de una autoridad central que obligue a cada cual a cooperar con los demás. Esta definición paradójica, conjunto con las definiciones de acciones altruistas, beneficios y costos presentes en el libro mencionado, se hace clara al observar que H. Ohtsuki y otros [53] mencionan que es importante realizar una cooperación cuando la razón beneficio-costos es mayor a la cantidad de cooperantes existentes. Considerando sistemas de drones, sin cooperación, es decir sin comunicación positiva e intencional entre vehículos, no se optimizará el despliegue de una flota de drones y todos los vehículos intentarán cubrir la mayor cantidad de área de toda la

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

superficie a su manera, por lo que, se debe distribuir el espacio a cubrir en forma equitativa y distribuida, siempre y cuando los vehículos estén preparados para compartir informaciones.

Distribución del espacio a cubrir

Para obtener la forma más óptima de distribuir o separar un espacio en regiones asignadas a cada uno de los vehículos que transitarán en el espacio de interés, lo primero es tener cuenta lo definido en la sección 2.2.3 sobre la planificación de misión y la cobertura de área. Luego, se debe tener en cuenta que la partición en regiones no solo permite intervalos de comunicación mayores, que favorecen al rendimiento de la energía disponible, sino que también favorece a un control más riguroso sobre el rendimiento de un vehículo en particular, ya que los datos obtenidos de una región corresponderán a la cuál un vehículo está, por decir, encargado. Esta separación se debe realizar de forma autónoma, y debe de poder brindar a cada uno de los vehículos la suficiente libertad para maximizar la obtención de datos por cada uno de ellos. Además, debe ser genérica, actualizable en el tiempo sin generar trabajo adicional de movimiento no productivo y ser capaz de tener en cuenta conjuntos de puntos con diferentes tipos de interés en visitar, esto es, encontrar o definir puntos que tendrá información de mayor utilidad.

Algunas técnicas incluyen partición en polígonos regulares y *packing*. El primer método no obtiene facilidad de actualización debido a que adicionar o remover un polígono después de iniciado el procedimiento cambia totalmente la distribución inicial; mientras que el segundo método no cubre enteramente el terreno, produciendo una falta de información. También existen trabajos que buscan cubrir un área de interés particionándola en pequeñas áreas para cada uno de los agentes intervinientes. Los investigadores Y. Liu y otros [54] decidieron trabajar con particiones de áreas e instruir a cada agente el viajar por una de

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

esas particiones cumpliendo con el algoritmo modificado de optimización de alimentación de bacteria. El algoritmo original toma el mapa y asigna valores de “alimento” a cada punto, luego el agente viaja por cada punto para “alimentarse” y así consigue recorrer todo el mapa. La modificación incluye actualizar el área a cubrir de cada agente, consiguiendo una menor cantidad total de pasos y un menor porcentaje de recobertura del área de interés. Si bien este trabajo se adecua a las necesidades que la implementación requerirá, el cálculo de las particiones en cada paso requiere de demasiado tiempo de cómputo y de tiempo total, que será contraproducente para el cumplimiento de los objetivos.

Se puede también tener en cuenta la continuación del trabajo que implementó el *FMM* en vehículos autónomos de superficie, ya que Y. Liu y otros [55] desarrollan algoritmos para la asignación de puntos o *waypoints* a seguir cumpliendo asignaciones inteligentes multitarea. En su trabajo, asignaron *waypoints* preestablecidos a una cantidad definida de vehículos, optimizando el tiempo y la potencia utilizada en el proceso, para visitar cada uno de estos *waypoints*. El mayor inconveniente de este método es que no está diseñado para la inclusión o el retiro no previsto de vehículos, dificultando obtener el mayor provecho con la energía disponible de los vehículos presentes.

Los autores D. Brandtner y otros [56] proponen la técnica de *swarming* con requerimientos mínimos de computación y comunicación mediante un conjunto de reglas parecidas al sistema de Boids, el cual fue mencionado en el primer capítulo en la sección 1.2.1. Los vehículos autónomos (aéreos en ese caso) viajan en la dirección en la que apuntan si se cumple la condición de que tenga una cantidad definida de vecinos que apunten a la misma dirección, en caso contrario buscan alinearse mediante chequeos de coherencia, siempre y cuando no estén demasiado cerca los unos a los otros. Los resultados indican que es posible controlar un conjunto de vehículos compartiendo mínima información para realizar la tarea de cubrir espacios, pero de forma aparentemente aleatoria y con la pre-

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

misa que no se tiene en cuenta la optimización de esta búsqueda para obtener parámetros o datos de todo el entorno.

Varias investigaciones recientes, como el trabajo de F. Aurenhammer y otros [57], indican que un buen método para particionar regiones consiste en usar la Partición de Voronoi, este método se adapta a cualquier número de regiones deseadas, facilita la actualización de regiones y es genérica, obteniendo siempre una distribución eficiente y uniforme. La partición de Voronoi consiste en encontrar las N regiones V_i que pertenecen a N puntos p_i de interés (en adelante generadores) en Ω , en donde cada región V_i está conformada por todos los puntos x que se encuentran a una distancia d_i de un generador p_i menor a la distancia d_j de cualquier otro generador p_j . Esta definición se puede visualizar en la Ecuación 2.8.

$$V_i = \{x \in \Omega \mid \|x - p_i\| < \|x - p_j\| \quad \forall j = 1, 2, \dots, N; j \neq i\} \quad (2.8)$$

El valor de $\|x - p_i\|$ es la norma de la diferencia de posición entre el punto y un generador, definido como la distancia euclidiana ($\sqrt{x \cdot x}$) entre ellos. Un ejemplo puede observarse en la Figura 2.13 en donde los generadores se representan como triángulos azules, para los límites de las regiones se trazan las aristas con líneas verdes y los vertices de cada región en rojo. Los diagramas de Voronoi poseen algunas propiedades, como por ejemplo que las aristas corresponden a un segmento del lugar geométrico entre dos generadores p_i y p_j , si estos son adyacentes. Otra propiedad corresponde a que si Ω no está limitado, las particiones de los generadores que se encuentran más alejados del “centro” de ellos poseen regiones teóricamente infinitas. Esto se puede solucionar definiendo límites como es el caso de este TFG. Se observa un ejemplo en la Figura 2.13. Con el hecho de particionar al área de estudio en regiones se define la primera acción de cooperación entre los vehículos si cada uno de ellos se encarga de cubrir ca-

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

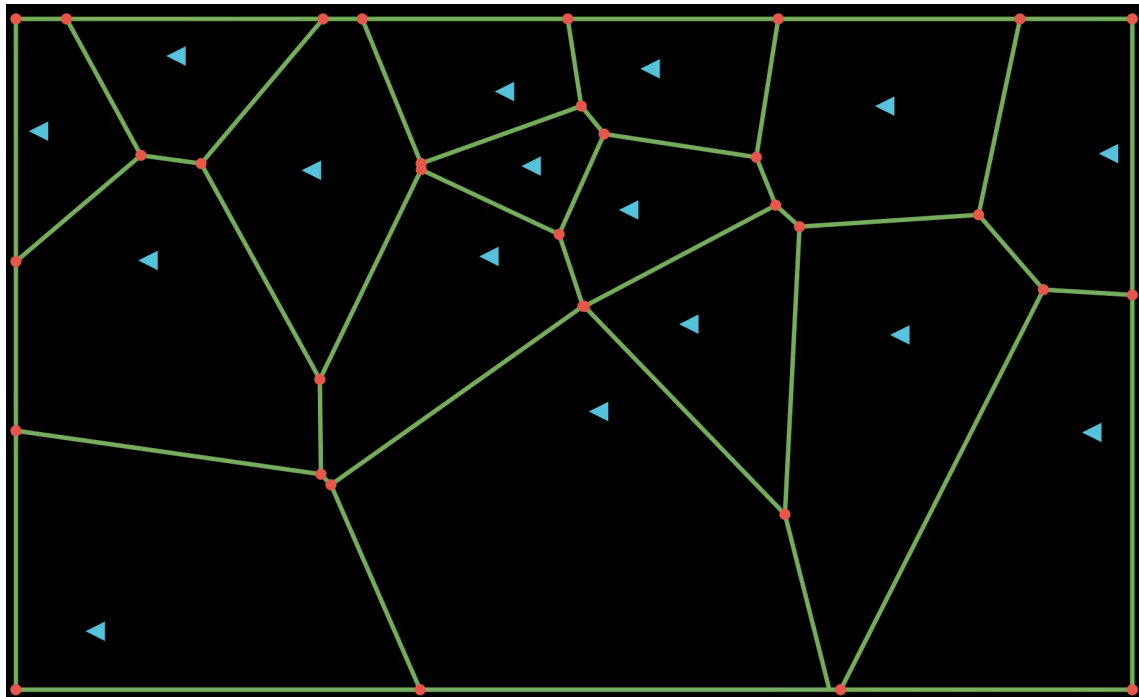


Figura 2.13: Partición o Diagrama de Voronoi para 16 generadores ubicados aleatoriamente. **Fuente:** Elaboración propia.

da una de las regiones, debido a que pueden compartir información entre ellos de posición y de región designada. El hecho de compartir información se considera como una acción altruista ya que transmitir la información de posición de un vehículo a otro vehículo es beneficioso para el otro vehículo, únicamente, pero tiene un costo (cálculo computacional y envío de datos por medios inalámbricos) para el vehículo que realiza la transmisión.

2.4.2. Estrategias de Cooperación Distribuida

Este TFG pretende ser la base de todos los trabajos cooperación distribuida de sistemas de múltiples robots, por lo que para hablar de sistemas distribuidos se deben desarrollar algoritmos que permitan que todas, o la mayoría de, las decisiones se tomen en conjunto, así como el cálculo de estas se realicen sin la necesidad de un servidor central o un algoritmo maestro que controle el trabajo y las decisiones de movimiento de cada uno de los vehículos. Esto no solo facilita la generación de comportamientos emergentes, sino que también hace robusto

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

y escalable al sistema para un número no definido de agentes intervinientes.

Voronoi Descentralizado

La separación de regiones para cada uno de los vehículos se definió bajo el método del Diagrama de Voronoi, pero para realizar este cálculo se debe tener información de las posiciones de los generadores de cada uno de los N vehículos. Dicho esto, un vehículo puede solicitar y obtener las posiciones de los N generadores cada vez que quiera calcular un diagrama de Voronoi, pero solo necesita una cantidad mínima M de las posiciones de otros vehículos, ya que la construcción de una región en particular queda limitada por los vértices y aristas que son, como ya se había mencionado, los lugares geométricos de los generadores adyacentes. Cabe mencionar que si observamos nuevamente la Figura 2.13 encontramos que todos los generadores poseen 2 o más vecinos. Esto se debe a razones matemáticas según Anexo B, por lo que el valor de M debe ser mayor o igual a 2 cuanto menos e igual a N en el caso de tener una cantidad mínima de vecinos. J. Cortes y otros autores [58] propusieron encontrar una región específica V_i para un vehículo usando una variación del algoritmo para obtener las regiones, enfocándose en el cómputo distribuido, haciendo énfasis en que un vehículo, o robot, solo necesita la información que le concierne. Por lo que para este TFG, en concordancia con el trabajo mencionado, cada vehículo define un círculo, con radio de sensibilidad r , con centro en la posición de un vehículo y solicita que todos los otros vehículos que se encuentren dentro de esta zona respondan al mensaje con sus respectivas posiciones. Teniendo en cuenta la cantidad de vehículos que responden y el tamaño del radio, se decide computar una región de Voronoi W_i del vehículo y de sus vecinos o agrandar el radio de sensibilidad para buscar más vecinos. El trabajo de J. Cortes y otros [58], además, menciona que para un generador en particular la región W_i es igual a la región V_i (la calculada con todos los datos) si el radio de sensibilidad es mayor o

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

igual al doble de la mayor distancia entre la posición del generador y los vértices de la región calculada W_i , es decir que el radio mínimo adecuado cumple $r_{i,min} = 2\max_{q \in W_i} ||p_i - q||$, siendo q uno de los vértices de la región calculada W_i . La definición propuesta se encuentra en forma de código en el Algoritmo 1 y un ejemplo es observable en la Figura 2.14. En la primera subfigura se observa que la región de Voronoi calculada de forma distribuida (W_i) es diferente a la región V_i , notando que existe una diferencia de los vértices, sus posiciones y la cantidad de generadores que crean las regiones de Voronoi. Esto se debe a que el radio de sensibilidad es menor al doble de la máxima distancia del generador i a los vértices de W_i , pero al hacer que el radio sea el doble que el anterior, se observa en la subfigura de la derecha que $W_i = V_i$, y que $r_i > 2\max_{q \in W_i} ||p_i - q||$. Notando que el radio es mayor al mínimo se puede recalcular el W_i , pero esta computación no se considera porque el costo de un cálculo es demasiado alto en comparación con el beneficio posible que este brindaría. Si un vehículo utiliza este procedimiento para calcular la región de la cual está encargado, se puede encontrar, sin inconvenientes, una región de trabajo, calculada por un algoritmo de baja complejidad computacional comparada con la complejidad de tener que tener información de todos los vehículos que además provee de una superficie donde todos los puntos cumplen con una función minimizada. No se puede dejar de mencionar el hecho de que este cálculo, al ser hecho en forma distribuida, puede ser realizado en todo momento y por todos los vehículos, siempre y cuando el vehículo pueda intercambiar mensajes con sus vecinos dentro del radio de sensibilidad.

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

Algoritmo 1: Determinación del diagrama de Voronoi descentralizado.

```

1 requiere: posición del generador( $p_i$ );
2  $r = r_{min}$ ;
3  $gen = generadores\_dentro\_de(radio=r, centro=p_i)$ ;
4 while  $gen < 3$  do
5    $r* = 2$ ;
6    $gen = generadores\_dentro\_de(radio=r, centro=p_i)$ ;
7 end
8  $W_i = obtener\_diagrama\_voronoi(gen)$ ;
9 while  $r < 2max_{q \in W_i} ||p_i - q||$  do
10   $r* = 2$ ;
11   $gen = generadores\_dentro\_de(radio=r, centro=p_i)$ ;
12   $W_i = obtener\_diagrama\_voronoi(gen)$ ;
13 end
14 retornar:  $W_i$ 

```

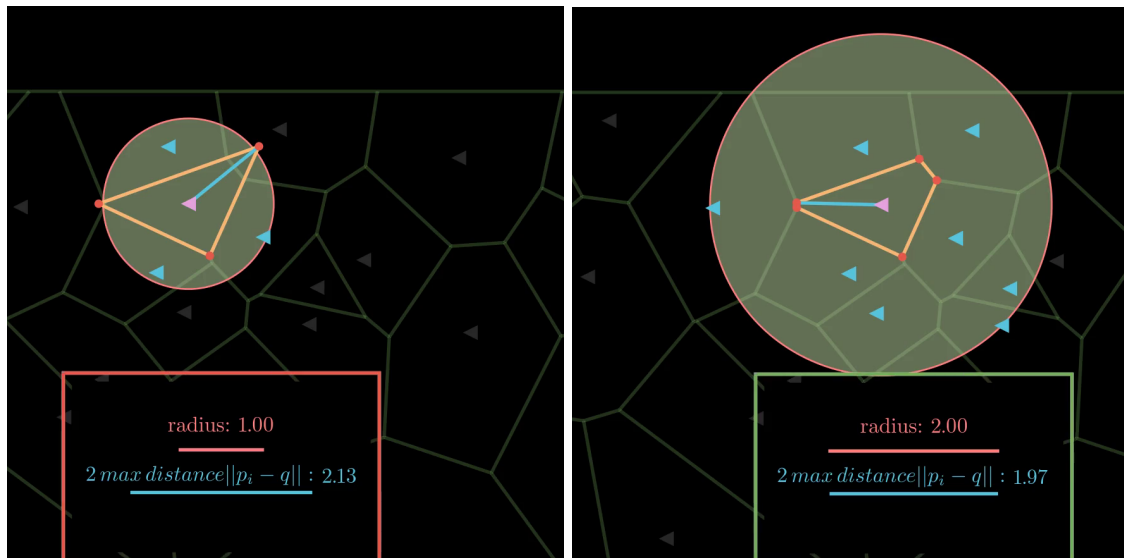


Figura 2.14: Diagrama de Voronoi calculado en forma distribuida para un generador. **Fuente:** Elaboración propia.

2.4.3. Algoritmos de estrategias de Cooperación Distribuida para Drones de Superficie

Si bien el Algoritmo 1 funciona de forma correcta para las pruebas aisladas, es imposible usar el mismo algoritmo cuando se implemente el Sistema de Guía, debido a que el algoritmo está ocupándose en todo momento de obtener la región de Voronoi de cada agente, en cada agente y no puede ocuparse de realizar

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

otras operaciones al mismo tiempo. Además de este trabajo, el vehículo debe estar ejecutando algoritmos de planificación de ruta, o de misión, siempre y cuando pueda establecer la comunicación tanto interna, con otros subsistemas, y externa, para obtener información de otros vehículos. Conjuntamente todos los componentes definidos en este capítulo deben trabajar y permitir que el vehículo, de forma autónoma, cumpla su misión. Para esto, se vuelve a tomar la definición del subsistema de guía y de los componentes que lo componen, y se los agrupa de tal forma a obtener las velocidades deseadas con respecto a las informaciones obtenidas y la misión a cumplir.

2.5. Propuesta de Sistema de Guía y Cooperación de Múltiples Drones de Superficie

Se presenta a continuación la propuesta a diseñar e implementar de un sistema de guía de múltiples drones de superficie acuática utilizando información dinámica en la Figura 2.15. Se puede observar a cada uno de los componentes necesarios para la creación de este sistema de guía de múltiples drones. La similitud con la Figura 1.5 permite comprender la importancia de un buen diseño con base en las técnicas y métodos encontrados actualmente en la literatura, siempre y cuando exista un programa o un algoritmo principal en cada vehículo que pueda lograr el cometido de que el mismo pertenezca a un sistema de múltiples drones que busca cumplir un objetivo en común.

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

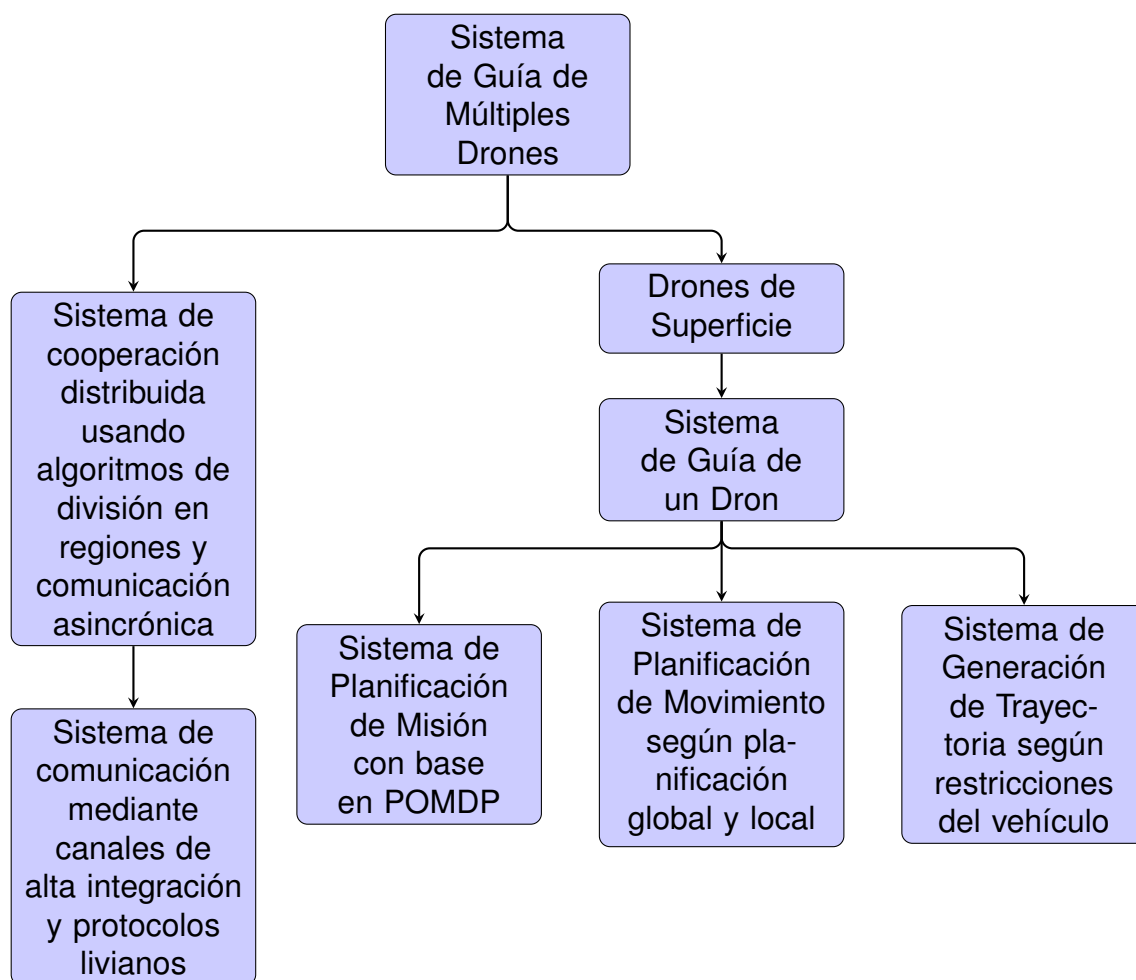


Figura 2.15: Definición y componentes del sistema de guía y cooperación de múltiples ASV. **Fuente:** Elaboración propia.

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

2.6. Control en Sistema Embebido

En el proceso de implementación de todo proyecto es importante seleccionar una buena plataforma para la implementación, por tanto, se realiza en este trabajo, una investigación para definir el uso de una placa de control seleccionada, con base en características apropiadas para el manejo de un Vehículo Autónomo de Superficie (ASV).

2.6.1. Sistema Embebido

Los sistemas embebidos referidos al ámbito electrónico y mecatrónico forman una parte fundamental dentro del diseño y la implementación de soluciones debido a que, según la definición encontrada en el libro de P. Marwedel [59], estos son sistemas de procesamiento de información embebidos en productos encerrados, y es sabido que toda implementación debe, en algún momento, procesar información de una índole preestablecido, de forma tal a decidir y realizar una acción dentro del entorno en el cual está operando. P. Marwedel también define en el mismo libro [59] que no se debe confundir al sistema embebido con el ciber-físico, el cual, en realidad consiste en la integración del procesamiento de información de un sistema embebido con procesos del mundo físico. Numerosas son las opciones durante el desarrollo de un sistema de control, y la elección del sistema embebido adecuado debe reunir las mejores características posibles teniendo en cuenta, por lo menos, los siguientes factores:

1. precio.
2. consumo máximo de potencia.
3. CPU.
4. tamaño.
5. alimentación.

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

6. entradas/salidas
7. velocidad de procesamiento de datos.
8. software de control.

Los primeros 6 factores son datos brindados por los fabricantes, considerados como características intrínsecas o propias de los sistemas embebidos, mientras que los siguientes son datos que dependen de la eficiencia y el rendimiento de los programas que son ejecutados dentro de estos sistemas, tanto de alto nivel, como son las interfaces usuario-maquina o programas desarrollados en lenguajes de alto nivel (Python, C++, Java, etc.), como de bajo nivel (assembler, programas del kernel o núcleo de los microprocesadores, etc.). La gran mayoría de los fabricantes de estos sistemas ofrecen kits de desarrollo que son sistemas embebidos genéricos capaces de ser adaptados a un sinnúmero de aplicaciones, ofreciendo capacidades para convertirse en sistemas ciber-físicos como GPIO (Entradas/salidas de propósito general), conexiones de sensores mediante I2C, UART, entre otros. En la Figura 2.16 se pueden observar ejemplos de placas de desarrollo disponibles y en la Tabla 2.2 se presenta un resumen de las diferentes características de los kits de desarrollo más comunes encontrados en el mercado actual [60–62]. Estos kits de desarrollo corresponden a la Nvidia Jetson TX2, Qualcomm Dragonboard 410c y Raspberry Pi.

Tabla 2.2: Características de Algunos Sistemas Embebidos Disponibles.

Especificaciones	NVIDIA Jetson TX2	Qualcomm Dragonboard 410c	Raspberry Pi 3B
Potencia	7.5W / 15W	1.3W / 4W	1.9W / 5.1W
CPU	Dual-Core NVIDIA Denver 2 2GHz 64-Bit & Quad-Core ARM Cortex-A57 2GHz MPCore	Quad-core ARM Cortex A53 1.2 GHz 32/64-bit support	Quad Core 1.2GHz Broadcom BCM2837 64bit
Alimentación	12V-19V	8V-18V	5V
Tamaño	222.25mm x 209.55mm x 120.65mm	85mm x 54mm	85.6mm x 56.5mm x 17mm
E/S	USB, PCI-E, GPIO, I2C, SPI, ISP	USB, I2C, SPI, PWM, GPIO	USB, I2C, CSI, PWM, GPIO
RAM	8 GB	1 GB	1 GB
Precio	399.00 \$	75.00 \$	35.00 \$

Fuente: Elaboración propia.

Otras características muy importantes, que los diferentes fabricantes mencio-

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**



Figura 2.16: Ejemplos de kits de desarrollo disponibles en el mercado.

Fuentes: Elaboración propia, con información de Raspberry Pi Foundation [60], Qualcomm Technologies Inc. [61] y Nvidia Corporation [62], respectivamente.

nan, incluyen conectividad (Bluetooth, WIFI, etc.) y los sistemas operativos recomendados, como L4T(Distribución basada en Ubuntu) para el NVIDIA Jetson TX2, Android 5.1 o Debian 8.0 para el Qualcomm Dragonboard 410c y Raspbian para el Raspberry Pi 3 B. A partir de la consideración de estos Sistemas Operativos (OS) en las diferentes placas de desarrollo se pueden determinar características de software más específicas. Todos estos OS poseen la capacidad de actualizar paquetes a través de Internet, y además una forma de aprovechar la compatibilidad entre distintas versiones de paquetes publicados y usados. Como los 3 OS soportan versiones de GNU/Linux, propiedad de Free Software Foundation [63], el desarrollo de programas, por ejemplo en el lenguaje de Python [64], permite una comparación entre dispositivos más estandarizada, ya que se ejecutaría sobre una misma versión de Python programas que determinen las velocidades de procesamiento de los datos considerados como útiles para la realización del trabajo. Otra de las ventajas que ofrece el Jetson TX2, con respecto a otros sistemas embebidos, consiste en la integración de librerías y interfaces de programación de aplicaciones (API) de Inteligencia Artificial y Computación Visual, a través del Kit de Desarrollo de Software *JetPack*, ideales para su uso en proyectos de robótica móvil.

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

CAPÍTULO 3

Marco Procedimental

La elección de los componentes del subsistema de guía se debe realizar con base en métricas y rendimiento, y, también, en los requerimientos a ser definidos por la aplicación en particular del vehículo autónomo. En ese sentido, se debe diseñar una ruta para probar y determinar los componentes más óptimos para la implementación. Esta ruta de diseño comprende etapas de pruebas aisladas, con sus respectivas validaciones, las pruebas de interconexión o intercomunicación entre los componentes a ser seleccionados. Luego se realizan pruebas del sistema propuesto y se los compara con otros posibles sistemas que cumplirían la misión, además se realiza un estudio detallado acerca de el sistema propuesto.

3.1. Ruta de Diseño del proyecto

Es importante reconocer primero cuales son los segmentos o etapas a validar, antes de realizar las pruebas, por lo que se puede recordar cuales son los componentes de un sistema de guía, según lo expuesto en 2.2.1. La Figura 2.15 muestra los componentes de este subsistema, y basados en métricas y puntuaciones específicas para cada segmento, se procede a validar la elección de algoritmos y técnicas para cada uno de ellos. La ruta de diseño iniciaría con la elección del componente de más bajo nivel, el cual debe acomodarse a generar trayectorias que sean posibles de seguir con Vehículos Autónomos de Superficie. Continuando, se debe validar un planificador local de ruta que tenga en cuenta lo seleccionado como componente de generación de ruta. Para finalizar el diseño

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

óptimo de la Planificación de Movimiento se deben realizar pruebas de planificadores globales, que se validará en forma conjunta con la planificación locales. Durante las pruebas, también es necesario evaluar el nivel del cumplimiento de la misión, a través de la evaluación a la Planificación de Misión. Con esto concluirá la evaluación de un vehículo autónomo, y, seguidamente, se procede a evaluar el subsistema de guía, agregando los componentes de cooperación y distribución de tareas, el cual representa una evaluación nueva en esta rama de la investigación.

3.2. Descripción general

Es de vital importancia validar la elección de las técnicas o métodos de planificación de cada componente del sistema de guía, teniendo en cuenta los objetivos de este TFG, y que existe una gran cantidad de posibles métodos que pueden ser utilizados. Para ello se presenta, en este trabajo, un método generalizado para obtener el rendimiento los subsistemas de planificación. Se debe destacar que, a la hora de implementar el método, se debe hacerlo específico para el caso, teniendo en cuenta el vehículo y el entorno. En primera instancia, en la búsqueda de un método o algoritmo apropiado para una implementación, se utilizan simuladores para determinar las características, ventajas y desventajas *a priori* de los métodos y algoritmos en la literatura. Y a pesar de que los simuladores no pueden representar a la perfección el entorno real, ellos proveen de una manera más segura, económica y rápida de poner a prueba las soluciones planteadas. Para determinar los rendimientos de los algoritmos se usa el simulador desarrollado en el LSD por F. Peralta y otros [65]. El simulador provee de una respuesta rápida y confiable de la posible interacción del vehículo con el entorno, entregando mediciones de tiempo, distancia y otros parámetros importantes para la determinación del rendimiento de los algoritmos de planificación de ruta.

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**



Figura 3.1: Modelo del Lago Ypacaraí **Fuente:** Elaboración Propia

3.2.1. Modelación del Entorno

El espacio de interés consiste en la superficie superior de un cuerpo de agua dulce, como ejemplo, puede visualizarse, en la Figura 3.1, una representación del Lago Ypacaraí, obtenida mediante la utilización de una herramienta online hecha por A. Krogh [66] que provee de mapas actualizados (2019) de la tierra con una precisión ideal para su uso lejano de los polos. Todos los puntos de esta superficie se encuentran aproximadamente en el mismo plano horizontal $\mathbb{R}^2$. Si encerramos al espacio de interés dentro de un rectángulo, se puede modelar esta información como una matriz rectangular en $\mathbb{R}^2$, donde la zona que no corresponde a agua, corresponde a terreno o a un espacio no transitable. A esta matriz se la denomina Ω en este TFG. El espacio Ω es un espacio continuo excepto en los límites, que corresponden al conjunto de puntos donde más allá de ellos no existe información de ningún tipo, por lo que el vehículo no debería encontrarse en cualquiera de estos puntos. En el resto de los puntos, el vehículo puede encontrarse sin mayores dificultades, siempre y cuando se cumpla el requisito de ser un punto alcanzable y no ocupado. El espacio Ω no puede variar, pero la información dentro de él si y no necesariamente con una función continua

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

de cambio. Sin embargo, tanto Ω como el conjunto de las informaciones de él, no pueden ser almacenados enteramente debido a la necesidad de una memoria infinita, por lo que cuando el vehículo utilizara alguna información, ésta se podrá acceder o almacenar en una forma discreta, es decir que el espacio Ω será reemplazado por $\Omega' \subseteq \mathbb{R}^2$ donde existe un $\delta > 0$ tal que para cada par $\mathbf{x}_1, \mathbf{x}_2 \in \Omega'$, la distancia euclidiana d entre estos puntos siempre cumple $d \geq \delta$, siendo $\mathbf{x}$ un punto de valores discretos $x, y \in \mathbb{R}^2$. En adelante se mencionará al espacio como Ω , indistintamente si corresponde o no al espacio continuo.

A continuación se implementará y validará la elección de un planificador de ruta antes de implementar el planificador de misión. Esto se debe a que el planificador de misión debe usar el mismo planificador de ruta a utilizar para poder simular una acción en el MCTS para resolver el POMDP.

3.2.2. Planificación de Ruta y Trayectoria

Entorno de Simulación

Siguiendo con lo estipulado en la subsección anterior y en la sección 2.2.3, se toma en cuenta la definición del espacio Ω y el modelo obtenido según la herramienta online, con esto se obtiene una imagen de dimensiones 1000 x 1500 pixeles, y mediante cálculos de traslación y escalamiento, se obtiene que la distancia entre centros de pixeles contiguos es equivalente a una distancia aproximada de 10.33 metros en el entorno real. De nuevo, es importante señalar que el vehículo puede encontrarse en una posición $\mathbf{x}$ cualquiera siempre que $0 < \mathbf{x}_x < 1000$ y $0 < \mathbf{x}_y < 1500$, pero la posición para la planificación siempre será un par de números naturales entre los límites mencionados. Esto se da de esta forma, debido a que los vehículos a ser adquiridos serán de dimensiones mucho menores a 10 x 10 metros, y el movimiento de éstos no será restringido a un mínimo de 10.33 metros. En el entorno de simulación, el vehículo simulado es capaz de moverse de acuerdo a un conjunto de configuraciones y restricciones que modelan

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

de forma aproximada al comportamiento de los vehículos a adquirir. El vehículo simulado también tiene en cuenta que los vehículos reales no podrán alcanzar las velocidades deseadas instantáneamente o de forma perfecta, para esto, el simulador asigna como velocidad “real” al resultado de una función que contiene la velocidad deseada. Como los vehículos a adquirir son del tipo catamarán, se puede inferir que el vehículo tendrá propulsiones diferenciales, una en cada casco, concluyendo que es posible que vehículo pueda girar sobre si mismo, dependiendo del centro de fuerzas que generen los propulsores. Esta configuración de movimiento corresponde a la dinámica de unicycle, propiamente definido por B. A. Francis y otros [67]. El modelo en cuestión utiliza la información de la velocidad deseada, así como la orientación θ (ángulo de dirección o *yaw*) y las Ecuaciones 3.1-3.3 para obtener un par de velocidades, la velocidad lineal v y la velocidad angular ω . En las ecuaciones, los términos k_v y k_ω , corresponden a las máximas velocidades alcanzables, por lo que el resto del cada lado derecho de las ecuaciones están normalizados.

$$A = (\cos \theta, \quad \sin \theta) \quad (3.1)$$

$$v = k_v (A_1 \vec{V}_x + A_2 \vec{V}_y) \quad (3.2)$$

$$\omega = k_\omega \frac{\arctan(-A_2 \vec{V}_x + A_1 \vec{V}_y, v)}{\frac{\pi}{2}} \quad (3.3)$$

Con estas ecuaciones, es posible determinar la velocidad lineal y angular que aproxima apropiadamente el movimiento de un catamarán, aunque no se tienen en cuenta factores como oleaje, viento, etc. De igual manera es suficiente para el caso en cuestión, y este par de velocidades es el par que utiliza el simulador y permite determinar la posición final del ASV simulado después de un tiempo Δt . El simulador a ser utilizado provee de métodos y funciones para medir los rendimientos, con esto se consigue determinar qué tan apto es el uso de cierto algoritmo para el caso en cuestión.

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

Procedimiento de simulación

Para la prueba de planificadores locales de ruta se usa la planificación global encontrada por M. Arzamendia y otros [12], encontrando 60 puntos en la costa del Lago y un orden de visita según lo establecido en el mismo trabajo. Estos puntos son considerados como posiciones de meta y el vehículo usa un planificador para encontrar la ruta a cada una de las 60 posiciones. La misión se considera como completada cuando el vehículo visita cada uno de estos 60 puntos. El algoritmo principal para estas pruebas puede observarse en el Algoritmo 2, en donde después de la inicialización, se obtiene la ruta planificada con destino a un *waypoint* según un planificador, luego se recorre la ruta, dirigiendo al ASV rumbo al *waypoint*. Una vez alcanzado el punto de visita, se determina la ruta al siguiente punto, iterando este algoritmo hasta lograr la visita a los 60 puntos.

Algoritmo 2: Algoritmo principal

```
1 requiere: way-points: wp, vehicle: ASV, map_data: map;  
2 pose = ASV.get_pose();  
3 goal = 1;  
4 path = p_planner(map, pose, wp(goal));  
5 sub_goal = 1;  
6 while goal ≤ t_goals do  
7   pose = ASV.get_pose();  
8   if pose - path(sub_goal) < error_allowed then  
9     sub_goal++;  
10    if pose - wp(goal) < error_allowed then  
11      goal++;  
12      path = p_planner(map, pose, wp(goal));  
13      sub_goal = 1;  
14    end  
15  end  
16  dx = calculate_velocity(path(sub_goal), pose, k);  
17  ASV.set_velocities(dx);  
18 end
```

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

La única variación entre métodos se da en la forma en que se calcula la ruta, luego es el mismo algoritmo como puede ser apreciado en el algoritmo mostrado anteriormente. Esto permite una comparación estandarizada y genérica para determinar el rendimiento de cualquier planificador local de ruta. Los algoritmos de cada técnica (A^* , PF , RRT^* , FMS) son llamados en las líneas 4 y 12, en diferentes simulaciones. Para compararlos, se proponen las siguientes puntuaciones igualmente proporcionales, las cuales se suman y brindan una puntuación general que es tomada como el rendimiento que posee un planificador local.

1. Tiempo de promedio de planificación de ruta: medido a partir de que se llama a la función que determina o planifica la ruta hasta que esta ruta es entregada al algoritmo principal. Idealmente se planifican rutas en el menor tiempo posible, con lo que, a menor tiempo de planificación, mejor puntuación. La mejor puntuación por lo tanto, se da cuando el tiempo de planificación es nulo. Cuando el planificador retorna sencillamente una trayectoria recta el tiempo de planificación de ruta es cero. El criterio mencionado depende inequívocamente de la distancia entre puntos, por lo que existirá siempre una diferencia de tiempo cálculo entre las rutas a planificar, debido a que los puntos a visitar no siempre están igualmente distanciados entre sí. La mejor forma de encontrar el valor correcto para comparación, entonces, consiste en encontrar el tiempo promedio de planificación de ruta. Para obtener un valor normalizado, la Ecuación 3.4 es la propuesta, en donde t_{pp} se refiere al tiempo promedio de cálculo de planificación de ruta, w a una constante normal y t_{ppmax} al valor máximo permitido como tiempo promedio de planificación de ruta. Cualquier tiempo mayor a este es considerado nulo y la puntuación será de 0. Se define a t_{ppmax} según el tiempo máximo que pueda tomarse para calcular una ruta, con esto se puede definir cuales planificadores son útiles en este sentido.

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

$$t_{wpp} = \frac{w}{t_{ppmax}} * (t_{ppmax} - t_{pp}) \quad (3.4)$$

2. Tiempo total de simulación: Como cada planificador de ruta genera un camino diferente del resto, generalmente el tiempo total de simulación t_{st} también es un factor a medir y comparar. Este tiempo incluye el tiempo en que se tarda en planificar la ruta y el tiempo en que se tarda en viajar de un punto a otro, viajando por la ruta planificada. Esto permite también puntuar a las rutas más directas entre puntos con un mejor rendimiento, debido a que el tiempo de cálculo de ruta no retorna información sobre la calidad de la ruta en cuanto al tiempo a tardar entre puntos. La función para obtener este valor se puede observar en la Ecuación 3.5, la cual varía un poco de la Ecuación anteriormente presentada, porque en este caso si existe un tiempo mínimo t_{stmin} posible considerablemente mayor a cero. El valor de t_{stmax} se selecciona de acuerdo al tiempo máximo que se puede tomar para completar la misión. El tiempo mínimo en este caso es el tiempo mínimo posible de simulación.

$$t_{wst} = \frac{w}{t_{stmax} - t_{stmin}} (t_{stmax} - t_{st}) \quad (3.5)$$

3. Diferencia de distancia manejada: Debido a que no todas las rutas calculadas por los planificadores de ruta son exactamente iguales a la línea que une los puntos de inicio y fin, es necesario calcular el nivel de concordancia de la ruta planificada y el segmento de recta entre los puntos, esto se realiza en este trabajo debido a que, en el mejor de los casos, siempre la ruta más corta entre los puntos es la ruta deseada de planificación, aunque, por razones matemáticas y algorítmicas, no siempre son así. El modelado del entorno contribuye en un gran porcentaje en la generación de la ruta. Se

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

define éste parámetro teniendo en cuenta las definiciones mencionadas y se utiliza la función propuesta en la Ecuación 3.6, como primera medida para obtener el valor se mide la distancia $dist_i$ entre los puntos de inicio y meta de cada ruta planificada, y luego durante la ejecución se mide la distancia viajada por el ASV $dist_{travel_i}$ entre los puntos. Esta medición también incluye las consideraciones del modelado del ASV, muy importantes a la hora de evaluar planificadores de ruta. La ecuación citada da como resultado al promedio todas las diferencias normalizadas entre las rutas planificadas y ejecutadas, con respecto a la distancia entre los puntos que se utilizan para encontrar rutas. Una forma de normalizar el valor obtenido de la media de n rutas se da utilizando la Ecuación 3.7, en donde para cada simulación, se utiliza el resultado del promedio, y con definiciones de normalización y con una definición de ddf_{wmax} se obtiene el valor del rendimiento normalizado. La máxima diferencia admisible ddf_{wmax} se determina con la máxima diferencia entre la ruta planificada y la ruta ejecutada que se puede dar, siempre relacionada al método que determinó las metas globales.

$$m_{dif} = \frac{\sum_{i=1}^n \frac{dist_{travel_i} - dist_i}{dist_i}}{n} \quad (3.6)$$

$$ddf_w = \frac{w}{ddf_{wmax}} * (ddf_{wmax} - m_{dif}) \quad (3.7)$$

4. Nivel de Seguridad de Ruta: Los planificadores de ruta no toman en cuenta a la dinámica del vehículo, porque solo reciben información del mapa en cuestión, el punto inicial y el punto final o meta. Esto impide calificar correctamente a un planificador en términos de adaptación a las necesidades específicas de un proyecto. Para solucionar este problema, el nivel de seguridad de ruta se define como una métrica genérica para cualquier tipo de

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

vehículo autónomo. Las rutas generadas, así como también el movimiento de los vehículos, son observados y dependiendo del comportamiento, se le asignan puntuaciones basadas las siguientes métricas:

- Nivel 5: La ruta no contiene curvas o giros que al vehículo le cueste seguir, y en la totalidad de los casos el vehículo puede superar con facilidad cualquier obstáculo que encuentre en el camino, sin colisionar.
- Nivel 4: La ruta planificada contiene generalmente maniobras que el vehículo puede realizar y todos los obstáculos son evitados a toda costa, incluso si el camino lleva a mínimos locales o zonas atrapadas sin colisionar con ningún obstáculo.
- Nivel 3: La ruta planificada contiene algunos giros drásticos y maniobras que el ASV puede ejecutar pero con algunos inconvenientes. Los obstáculos son evitados, pero no se toma una distancia prudencial de ellos.
- Nivel 2: La ruta contiene maniobras que son muy difícilmente ejecutadas por el vehículo, aunque los obstáculos sean evitados. La dinámica del vehículo no es la apropiada para la planificación de ruta en cuestión.
- Nivel 1: La ruta contiene maniobras o giros que no son posibles de seguir o ejecutar con el ASV.
- Nivel 0: La ruta cruza un obstáculo o falla a la hora de reprogramar rutas para evitarlos, colisionando con los obstáculos.

El nivel de seguridad depende mucho de la dinámica del vehículo, por lo que la aplicación de estas métricas depende de cada vehículo y aplicación y varía de acuerdo con ellos.

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

Con estas definiciones de criterios a tener en cuenta, y los procedimientos ya definidos, se procede a implementar cada uno de los algoritmos de planificación de ruta expuestos en este TFG y luego, se realizan las simulaciones para obtener el rendimiento de cada planificador. Estas simulaciones se realizaron usando Python 3.6.5 en una computadora con sistema operativo Windows 10 x64, procesador CPU Intel Core i7-7700HQ @ 2.8 GHz y 8Gb de memoria RAM DDR5. El simulador y las implementaciones de los planificadores locales de ruta puede ser encontrado el repositorio en línea provisto por F. Peralta y otros [65]. En todas las simulaciones se usa el mismo *set* de puntos a visitar, y se los visita en el mismo orden. Todos los parámetros definidos en esta subsección son medidos y anotados para su posterior discusión. A continuación, se procede a implementar los algoritmos de planificación de ruta:

A*: El algoritmo correspondiente a la planificación de ruta A^* se observa en 3. El cual sigue los pasos propuestos en la definición mostrada en el capítulo anterior. Cuando los requerimientos son entregados, el programa comienza creando

Algoritmo 3: Algoritmo de planificación de ruta A^*

```
1 requiere: vehicle: ASV, goal_pose: goal, map_data: map;  
2 vehicle_node = Node(ASV.get_pose(), parent_node=None);  
3 frontier = PriorityQueue(vehicle_node);  
4 examinedNodes = List(vehicle_node);  
5 while frontier.not_empty() do  
6     currentNode = frontier.pop();  
7     if currentNode.position == goal.position then  
8         examinedNodes.add(currentNode);  
9         break;  
10    end  
11    foreach aNeighborNode in neighborNodesOf(currentNode, map) do  
12        priority = get_cost_of(aNeighborNode);  
13        frontier.add_with_priority(aNeighborNode, priority);  
14        examinedNodes.add(aNeighborNode);  
15    end  
16 end  
17 path_to_follow = calc_path_to_follow(examinedNodes);  
18 return path_to_follow
```

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

las listas de frontera y nodos examinados, para luego expandir la frontera desde la posición inicial hacia afuera siempre priorizando las posiciones más cercanas a la posición de la meta que aún no han sido examinadas.

Campos Potenciales: Como se había expuesto, la implementación de la planificación de los campos potenciales es sumamente fácil, comprende la etapa de asignar puntos de atracción y luego la de modificar el campo sumando la interacción de los obstáculos, es decir, la repulsión. Esto se observa en el Algoritmo 4. Por último, se construye la ruta usando la función de gradiente descendiente. Las funciones que obtienen los valores de atracción y repulsión corresponden a las Ecuaciones 2.1-2.2.

Algoritmo 4: Algoritmo de planificación de ruta por Campos potenciales

```
1 requiere: Map:  $M$ , Starting Point:  $p_{start}$ , Goal Point:  $p_{goal}$   
   PF=new_map( $M$ .shape);  
2 foreach point in  $M$  do  
3   | PF[point] = attractive_function(  $p_{goal}$ );  
4 end  
5 max_attr = max(PF);  
6 foreach point in  $M$  do  
7   | PF[point] += repulsive_function(  $M$ );  
8   | if PF[point] > max_attr then  
9     | PF[point] = max_attr;  
10  | end  
11 end  
12 path = gradientDescent (PF,  $p_{start}$ ,  $p_{goal}$ );  
13 return path
```

Árbol Aleatorio de Exploración Rápida: La complejidad de esta planificación de ruta es considerablemente mayor que las anteriores, por lo que es necesario separar en varios algoritmos secundarios que permiten generalizar la búsqueda de una ruta libre de obstáculos creando el árbol aleatorio de exploración rápida. El Algoritmo 5 es el principal, el cual recibe los datos de posición inicial, meta y el mapa. En este algoritmo se genera el árbol con la raíz y luego se procede a crear y adaptar nuevos nodos para luego unirlos al árbol. Al seleccionar

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

un nuevo nodo aleatorio, se llama a la función *steer()*, la cual se puede observar en el Algoritmo 6, con esto el nuevo nodo se encuentra en condiciones de ser agregado o “unido” al árbol. Para esto, se selecciona el mejor nodo padre cuando se ejecuta el Algoritmo 7. El nuevo nodo unido al árbol se considera como futuro posible padre para nuevos nodos, y se repite el proceso hasta encontrar un nodo que conecte a la posición de la meta con el árbol. El camino a seguir, entonces, se toma como inicio en el nodo de la meta y recorre hasta el padre de ese nodo, así sucesivamente hasta llegar al nodo inicial, el cual es la raíz, obteniéndose el camino más rápido encontrado por el método de *RRT**.

Algoritmo 5: Algoritmo principal de *RRT**

```
1 requiere: vehicle: ASV, goal_position, max_iterations: m_i;  
2  $\tau$  = initializeTree();  
3 insertNode(ASV.get_pose(),  $\tau$ );  
4 for i=1: i<=m_i do  
5    $p_{rand}$  = randomize_sample();  
6    $p_{nearest}$  = select_nearest_node();  
7    $p_{rand}$  = steer( $\tau$ ,  $p_{rand}$ ,  $p_{nearest}$ );  
8   if isFeasiblePath( $p_{rand}$ ,  $p_{nearest}$ ) then  
9      $p_{nearNodes}$  = selectNearNodes( $p_{rand}$ ,  $\tau$ );  
10     $p_{min}$  = chooseParent( $p_{rand}$ ,  $p_{nearNodes}$ ,  $p_{nearest}$ );  
11    insertNode( $p_{rand}$ ,  $p_{min}$ ,  $\tau$ );  
12    if  $p_{rand}$  == goal_position then  
13      break;  
14    end  
15  end  
16 end  
17 path_to_follow = calc_path_to_follow( $\tau$ , goal_position);  
18 return path_to_follow
```

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

Algoritmo 6: Función de *steer* de *RRT**.

```
1 requiere: Tree:  $\tau$ , new_point:  $p_{new}$ , nearest_point:  $p_{nearest}$ ;  
2 pos =  $p_{new}$ .position;  
3 if  $dist(p_{nearest}, p_{new}) > maxStepDist$  then  
4    $\theta = atan(p_{new}.position, p_{nearest}.position)$ ;  
5   pos =  $p_{nearest}.position +$   
      $sign(p_{new}.position - p_{nearest}.position) * max\_step * angles(\theta)$ ;  
6 end  
7 return pos
```

Algoritmo 7: Función de seleccionar mejor nodo padre para *RRT**

```
1 requiere: new_point:  $p_{new}$ , near_nodes:  $p_{nearNodes}$ , nearest_node:  $p_{nearest}$ ;  
2 best_pt =  $p_{nearest}$ ;  
3 min_cost =  $p_{nearest}.cost + dist(p_{nearest}.position - p_{new}.position)$ ;  
4 foreach  $p_{near}$  in  $p_{nearNodes}$  do  
5    $t\_cost = p_{near}.cost + dist(p_{near}.position - p_{new}.position)$ ;  
6   if  $t\_cost < min\_cost$  then  
7      $[min\_cost, best\_pt] = [t\_cost, p_{near}]$ ;  
8   end  
9 end  
10 return best_pt
```

Método de la Marcha Rápida: En este TFG, se decidió probar únicamente el método de la marcha rápida al cuadrado, y su variación de actualización. Pero es importante recordar que el *FMS* utiliza el método de cálculo de la Matriz de Tiempo de Arribo dos veces, siendo esa la vital diferencia entre el *FMM* y el *FMS*. El algoritmo principal de *FMS* (Algoritmo 8) calcula una matriz de velocidades iniciales donde se asigna el valor de 1 a todos los lugares o pixeles, en donde se puede alcanzar una velocidad deseada, y 0 donde no. Con esto se calcula la matriz de Tiempo de Arribo a los obstáculos, es decir, la Matriz de Velocidades para calcular la matriz de Tiempo de Arribo a la meta. Estos dos últimos cálculos mencionados se realizan mediante la ejecución del Algoritmo 9, que realiza el mismo proceso mencionado en el capítulo anterior. El algoritmo principal finaliza cuando se tiene la Matriz de Tiempo de Arribo a la meta y, con ella, se calcula un camino mediante el uso del gradiente descendente.

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

Algoritmo 8: Algoritmo principal del Método de la Marcha Rápida.

```
1 requiere: Initial Map:  $M$ , Starting Point:  $p_{start}$ , Goal Point:  $p_{goal}$ ;  
2  $V = \text{calculate\_speed\_matrix}(M)$ ;  
3  $OB = \text{perform\_fmm}(p_{obstacle_i}, V)$ ;  
4  $TOA = \text{perform\_fmm}(p_{start}, OB)$ ;  
5  $path = \text{gradientDescent}(TOA, p_{start}, p_{goal})$ ;  
6 return path
```

Como siempre hay que calcular los mapas para planificar rutas usando FMM o FMS, se propone en esta sección una modificación, que permita una actualización simple, para caso en que los mapas se actualicen en la información de un obstáculo solamente.

Updating Fast Marching Square Method (uFMS): o actualización del Método de Marcha Rápida al Cuadrado es el nombre propuesto para mejorar la eficacia y eficiencia que el *FMS* presenta cuando se detectan cambios imprevistos en los mapas generados, utilizando métodos locales de actualización en los mapas. Se divide en dos partes fundamentales, así como el método original, pero disminuyendo drásticamente el tiempo total para generar una ruta, si se conoce de antemano un mapa realmente parecido a la nueva situación del entorno. El problema principal a ser mitigado es que, al utilizar *FMM* o *FMS*, se calcula nuevamente la totalidad del mapa de tiempos de arribo y de velocidades en el caso de *FMS*, igual si la diferencia entre el mapa anterior y el nuevo es pequeña. Dicho esto, la solución propuesta guarda el mapa generado y actualiza ambas matrices (Velocidad y Tiempo de Arribo), exclusivamente en una región definida que contiene la modificación m del mapa. Como primera instancia, se define propiamente la región a modificar mediante un parámetro de posición a actualizar p_{new} y un radio ρ de actualización. Para esto, se generan dos vectores que contengan la distancia horizontal d_x , y la vertical d_y de p_{new} a todo punto p

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

Algoritmo 9: Algoritmo para obtener la matriz de tiempo de arribo (observaciones y final)

```

1 requiere: Starting Point:  $p_{start}$ , Velocity Map:  $v_{map}$ ;
2  $toa = \text{new\_matrix}(v_{map}.\text{size}());$ 
3  $toa(p_{start}) = 0;$ 
4  $\text{trial\_queue} = \text{PriorityQueue}();$ 
5 foreach  $neighbor$  in  $vn\_neighbors(p_{start})$  do
6    $\text{time\_to\_p} = \text{cost\_update}(neighbor, toa, v_{map});$ 
7    $\text{trial\_queue.add\_with\_priority}(neighbor, \text{time\_to\_p});$ 
8    $toa(neighbor) = \text{time\_to\_p};$ 
9 end
10 while  $\text{trial\_queue.not\_empty}()$  do
11    $\text{next\_point} = \text{get\_lowest}(\text{trial\_queue}, toa);$ 
12    $\text{trial\_queue.remove}(\text{next\_point});$ 
13   if  $\text{next\_point.cost} == \text{inf}$  then
14      $\text{break};$ 
15   end
16   foreach  $n$  in  $vn\_neighbors(\text{next\_point})$  do
17     if  $toa(n) == \text{undefined}$  then
18        $t\_to\_p = \text{cost\_update}(n, toa, v_{map});$ 
19        $\text{trial\_queue.add\_with\_priority}(n, \text{time\_to\_p});$ 
20        $toa(n) = t\_to\_p;$ 
21     end
22   end
23 end
24 return  $toa$ 

```

de M . Luego se genera una máscara radial mediante la ecuación 3.8

$$mask = \begin{cases} 1, & \forall d_x^2 + d_y^2 \leq \rho^2. \\ 0, & \text{else.} \end{cases} \quad (3.8)$$

Esta ecuación provee de los valores que se encuentren a una distancia menor o igual que un radio, formándose una máscara radial. Esta máscara radial es utilizada como base para determinar que valores se debe recalcular, por lo que en la matriz de Tiempo de Arribo y en la matriz de Velocidades se altera exclusivamente esta zona, definiendo que los valores de Velocidad y Tiempo de Arribo de los puntos dentro de esta zona aún no han sido calculados. El diagrama de flujo

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

en la Figura 3.2 muestra el procedimiento para encontrar los nuevos valores de las matrices a modificar, notándose inmediatamente la igualdad con el diagrama de *FMM* (Figura 2.10), pero en este flujo se utiliza las siguientes informaciones: las matrices TOA y de Velocidad generadas en un mapa anterior, la posición de alteración del mapa y el radio para esta modificación m a realizar.

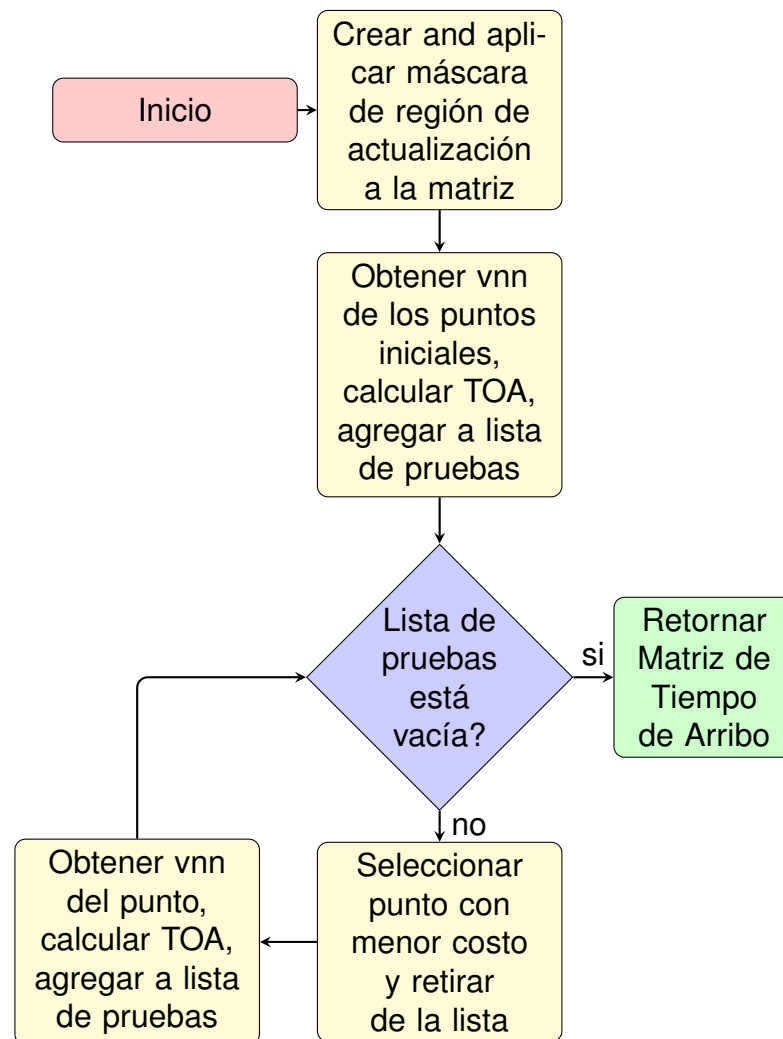


Figura 3.2: Diagrama de secuencias para actualizar matrices de Tiempo de Arribo. **Fuente:** Elaboración Propia.

La nueva secuencia para encontrar una ruta se encuentra plasmada en la Figura 3.3. Debido a la modificación de solo cierta parte de las matrices de Velocidad y de Tiempo de Arribo, la continuidad entre el límite “por dentro”(de adentro hacia afuera) de la región a modificar y el límite de afuera hacia adentro se ve

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

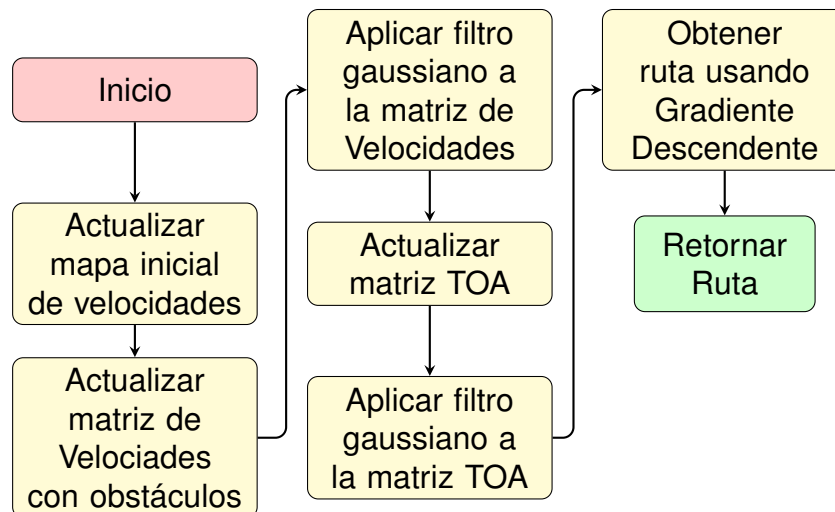
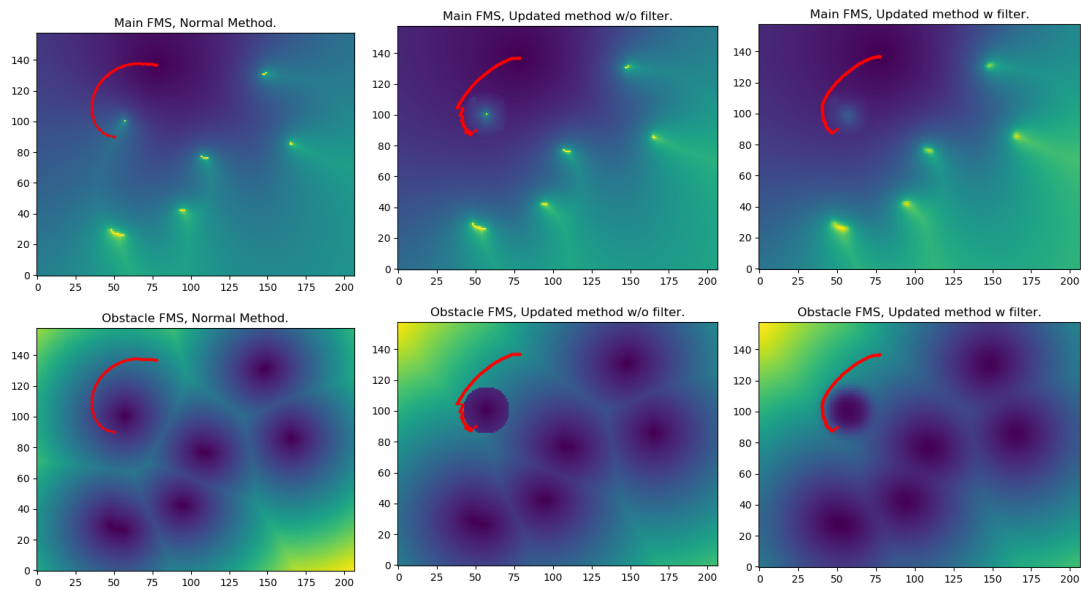


Figura 3.3: Diagrama de secuencias para calcular una ruta utilizando *uFMS*.
Fuente: Elaboración Propia.

afectada. Esto es debido a que el cálculo de tiempo de arribo no tiene en cuenta el mapa en macro, sino exclusivamente dentro de la zona. Para solucionar esta problemática se propone aplicar un filtro gaussiano de dos dimensiones, utilizado ampliamente en la literatura desde que G. Deng y otros [68] presentaron el filtro adaptivo, para generar una suavidad mejor, es decir, menor ruido en informaciones de dos dimensiones como las imágenes y matrices de representación. Concluyendo, se agrega una función para aplicar un filtro gaussiano, para suavizar el resultado, necesario para mejorar los costos calculados para Tiempos de Arribo, debido a que las informaciones de los puntos vecinos se utilizan para calcular estos costos, y también para mejorar la respuesta del gradiente descendente, al obtener un mapa más suavizado. Por otro lado, se agrega una función de actualización para actualizar las matrices de Velocidad y de Tiempo de Arribo, que agrega la funcionalidad de modificar estrictamente mediante la utilización de la Ecuación 3.8. En este algoritmo, se genera que un mapa deba obtener los valores nuevos en la zona deseada cuando se utiliza la ecuación mencionada. El resto del procedimiento es idéntico a *FMS*. A modo de ejemplo se pueden observar en la Figura 3.4 los resultados de aplicar este método, la primera subfigura

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**



a.

FMS recalculado. b. uFMS s/ filtro gaussiano. c. uFMS c/ filtro.

Figura 3.4: Comparación de las matrices de Velocidad con Obstáculos y de Tiempo de Arribo para FMS y uFMS. **Fuente:** Elaboración propia.

corresponde al método FMS común, donde se observa en la parte superior sobre el Mapa de Tiempo de Arribo a la ruta planificada, y en la parte inferior, la misma ruta planificada sobre el Mapa de Velocidades con respecto a los Obstáculos. El mismo orden se respeta en las demás subfiguras, donde el segundo par corresponde al uFMS sin el filtro gaussiano, con rutas con muchos cambios de dirección y para nada óptimas, debido a la no suavidad existente en el mapa. En el uFMS con filtro gaussiano (último par), se puede observar el drástico cambio comparándolo en el Mapa de Velocidades. La ruta planificada cumple con las mismas expectativas que el FMS, con la diferencia de que el cálculo se concentró en la zona del cambio de información y se realizó en un tiempo considerablemente menor.

Definido el método propuesto, *uFMS*, se procede a implementarlo. Cabe destacar que es el único algoritmo o método que varía ligeramente en los requerimientos, debido a que este método explícitamente necesita de informaciones de

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

Algoritmo 10: Updating Fast Marching Square Method.

```
1 requiere: Starting Point:  $p_{start}$ , Goal Point:  $p_{goal}$ , Information from last  
   map: (TOA, OB, V), Modification Point:  $p_{new}$ , Modification Radius:  $\rho$  ;  
2 update_velocity_map(V,  $p_{new}$ );  
3 OB = update_fmm(OB, V,  $p_{new}$ ,  $\rho$ );  
4 OB = apply_gaussian_filter(OB);  
5 TOA = update_fmm(TOA, OB,  $p_{new}$ ,  $\rho$ ,  $p_{start}$ );  
6 TOA = apply_gaussian_filter(TOA);  
7 return path, TOA, OB, V
```

mapas de cálculo obtenidos anteriormente por el *FMS*. Si es posible conservar la información obtenida al realizar el algoritmo de *FMS*, aplicar el *uFMS* consiste sencillamente en realimentar estos datos y datos de cambios conocidos que ocurrieron en el mapa, es decir lugares en donde se encontró un obstáculo nuevo o donde ya no existe un obstáculo en la situación actual. El Algoritmo 10 denota los pasos a seguir para obtener la nueva ruta, el cual no difiere demasiado del Algoritmo 8 (algoritmo principal del *FMS*). La obtención de las matrices de Velocidades y de Tiempo de Arribo a la meta son también bastante parecidos al método *FMS*, con la diferencia de que las matrices actualizan la información según los parámetros establecidos. Esta actualización se da cuando se ejecuta el Algoritmo 11.

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

Algoritmo 11: Algoritmo para actualizar mapas usando *uFMS*.

```
1 requiere: Current TOA: toa, Velocity map: v, Modification Point:  $p_{new}$ ,  
   Modification Radius:  $\rho$ , Starting Point:  $p_{start}$ ;  
2 trial_queue = PriorityQueue();  
3  $d_x, d_y$  = distance_from_point(v,  $p_{new}$ );  
4 mask =  $d_x^2 + d_y^2 \leq \rho^2$ ;  
5 toa(mask) = undefined;  
6 if  $p_{start}$  is null then  
7   toa( $p_{new}$ ) = 0;  
8    $p_{start} = p_{new}$ ;  
9 end  
10 foreach neighbor in  $vn\_neighbors(p_{start})$  do  
11   time_to_p = cost_update(neighbor, toa, v);  
12   trial_queue.add_with_priority(neighbor, time_to_p);  
13   toa(neighbor) = time_to_p;  
14 end  
15 while length_of(trial_queue) > 0 do  
16   next_point = get_lowest(trial_queue, toa);  
17   trial_queue.remove(next_point);  
18   if next_point.cost == inf then  
19     break;  
20   end  
21   foreach n in  $vn\_neighbors(next\_point)$  do  
22     if toa(n) == undefined then  
23       t_to_p = cost_update(n, toa, v);  
24       trial_queue.add_with_priority(neighbor, t_to_p);  
25       toa(n) = t_to_p;  
26     end  
27   end  
28 end  
29 return toa
```

Resultados de simulación

El simulador utilizado permite la visualización durante la ejecución. Con esto primero se generan algunas imágenes para observar el comportamiento del vehículo, y determinarse el nivel de satisfacción de Seguridad de Ruta. Para el resto de las puntuaciones se usa directamente los sistemas de medición que tiene el simulador, estas últimas simulaciones se realizan sin el gráfico correspondiente, pero simulando al vehículo usando el planificador de ruta seleccionado.

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

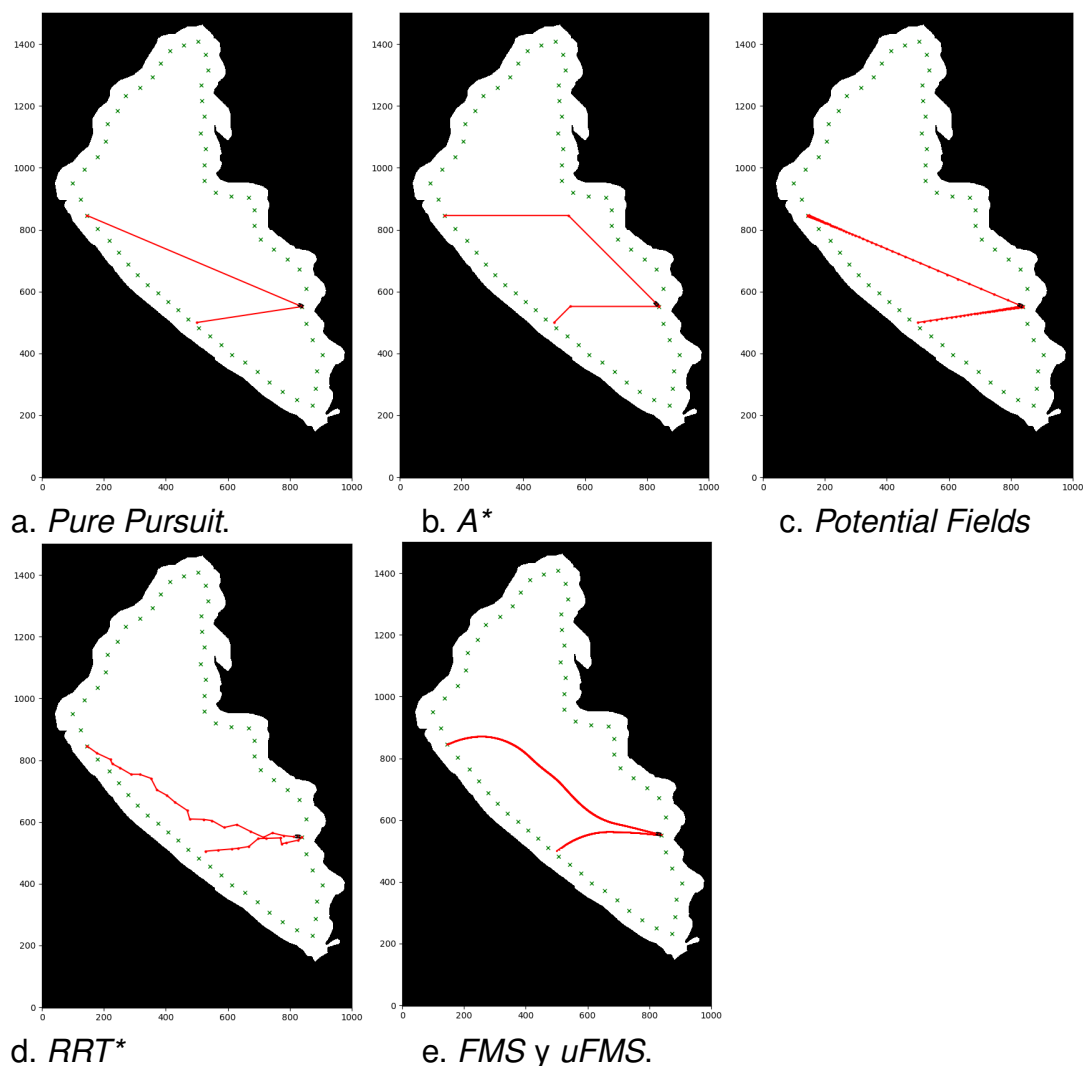


Figura 3.5: Visualización de las primeras rutas planificadas por los algoritmos de planificación local de ruta **Fuente:** Elaboración Propia

Se observa en la Figura 3.5.a. el caso más simple, cuando no se usa un planificador de ruta, y la ruta a seguir es el segmento de línea recta entre la posición inicial y final. En la Figura 3.5.b. se observa el resultado de encontrar rutas utilizando el Algoritmo A^* . Se nota inmediatamente que este método genera puntos intermedios adicionales, y es debido a la naturaleza de la búsqueda en grillas regulares. Todos los caminos encontrados por A^* consisten en segmentos de recta en una de las 8 direcciones posibles (cada 45° empezando de cero).

Continuando con los siguientes algoritmos de planificación de ruta, se encuentra en la Figura 3.5.c las rutas obtenidas usando el método de los Campos

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

Potenciales. Con los parámetros correctos, se consigue obtener direcciones rectas, siempre que sea necesario, evitando obstáculos. En algunos caminos es posible encontrarse atrapado en mínimos locales, debido a que *PF* no contempla la recuperación de encontrarse en esta situación. En la Figura 3.5.d se observan las primeras rutas generadas usando *RRT**. Aquí es muy claro que se agregan muchos nodos, al parecer, innecesarios, que son producto de la aleatoriedad de la creación de ellos, existen numerosos giros o maniobras muy cerradas que son muy difíciles de seguir por el ASV. Por último, se presenta en la Figura 3.5.e la trayectoria generada por *FMS*, bastante suave y con curvas generadas por la matriz de velocidades que toma en cuenta todos los obstáculos que en este caso consiste en los lugares donde no hay agua. Este resultado es el mismo cuando no existen obstáculos nuevos en el *uFMS*, por lo que no se agrega una imagen, ya que sería la misma.

Es de suma importancia señalar que el programa utilizado, simula en forma acelerada, es decir que el tiempo total de simulación se basa en el tiempo tardado en calcular y posicionar al vehículo en cada paso de simulación, simulando un tiempo transcurrido que no refleja al tiempo tardado para tal efecto. En ese sentido, el tiempo total de simulación refleja exactamente el tiempo total que tarda el programa en obtener para cada paso de simulación, las posiciones y calcular los estados correspondientes, además de las funciones de planificación de ruta. En todas las simulaciones realizadas se tomaron los datos establecidos y se obtienen los resultados numéricos de los tiempos totales de simulación, los tiempos promedio de cálculo de ruta y la diferencia de distancia viajada. Con esto, primero se realiza la tabla de comparación entre los métodos, que puede ser observada en la Tabla 3.1.

Pure Pursuit (Persecución Pura) no es una técnica de planificación local de ruta, pero se incluye en la tabla para tener un punto de comparación ideal (El menor tiempo de cálculo de ruta, tiempo total de simulación y diferencia de dis-

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

Tabla 3.1: Comparación de planificadores de ruta.

Planificador de ruta	Tiempo prom. de calc. [s]	Tiempo total de simu.[s]	Diferencia de dist. v. [%]	Nivel de Seguridad
A*	0.0771	140.99	5.797	✓✓✓✓✓
PF	13.5500	933.61	0.304	✓✓✓✓✓
RRT*	0.0036	137.48	13.795	✓✓✓
FMS	1,957.2800	116,741.00	14.700	✓✓✓✓✓
uFMS	3.1700	305.65	14.700	✓✓✓✓✓
Pure Pursuit	0	110.16	0	✓

Fuente: Elaboración propia.

tancia viajada.) Pero lleva una puntuación de cero en el Nivel de Seguridad, ya que cae dentro del Nivel 0.

Con estos resultados preliminares, ya es posible seleccionar un planificador local de rutas, pero se debe validar la selección usando las puntuaciones definidas al comienzo de esta sección. Este análisis más exhaustivo permite determinar el rendimiento de los planificadores locales de ruta, y, en este caso, se utiliza una base de $w = 25$ puntos de tal manera a alcanzar un máximo de 100 puntos sumando los 4 parámetros evaluados. Usando como valores mínimos a los valores mostrados en la tabla y como máximos a los máximos admisibles según especificaciones, se genera el gráfico de barras en la Figura 3.6. Visualizando

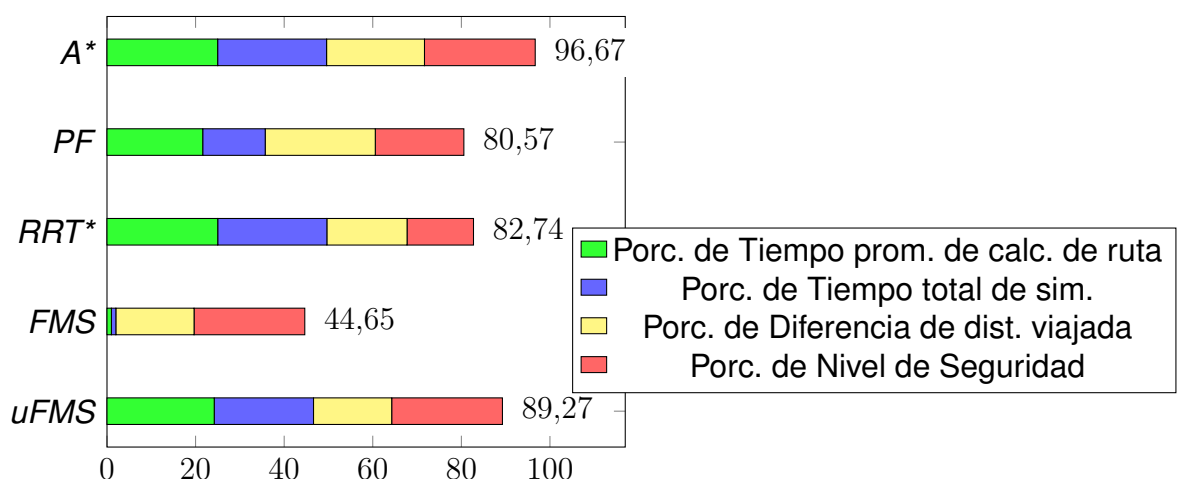


Figura 3.6: Visualización de los rendimientos de planificadores locales de ruta.

Fuente: Elaboración Propia.

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

los diferentes rendimientos de cada uno de los planificadores de ruta, se permite seleccionar al A^* o al $uFMS$ como métodos apropiados para la planificación local de ruta en Vehículos Autónomos de Superficie.

3.3. Diseños de implementación

Como bien se definió, el alcance de este TFG no comprende el diseño e implementación de los sistemas de navegación y control, por lo que se tiene como dados estos sistemas en embebidos. Ellos deben entregar la información necesaria al subsistema de guía y deben poder recibir la información por parte del subsistema de guía. El LSD adquirió el controlador de vehículos autónomos llamado Pixhawk, el cual posee es un sistema embebido libre con firmware modificable a tal punto que todos los sistemas con el que cuenta están expuestos mediante APIs. El Pixhawk, Figura 3.7, comprende el sistema GNC, pero es posible utilizar solamente los subsistemas de Control y Navegación, y hacer uso de paquetes o librerías disponibles para hacer el control del sistema de guía. El desarrollo de este TFG aprovecha esta característica para implementar el subsistema de guía desarrollado y poner a prueba las soluciones planteadas. Mediante el uso del firmware ArduRover, propiedad de ArduPilot Dev Team [69], y la librería de Dronekit para Python, realizada por 3D Robotics Inc. [70], se desarrollan e implementan algoritmos que, usando técnicas de planificación de misión y de ruta, entreguen velocidades deseadas al subsistema de guía.

3.3.1. Algoritmos del subsistema de guía

La implementación más óptima del sistema de guía para múltiples drones de superficie, incluye, en la fase de diseño, las consideraciones de descentralización de información, en este caso, la obtención del diagrama de Voronoi calculado de forma distribuida. Para esto, se crea un algoritmo principal que controlará la autonomía del vehículo y un algoritmo secundario que se encargará de planificar

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA



Figura 3.7: Ejemplo del AutoPilot Pixhawk **Fuente:** L. Meier [71]

la misión, las rutas y trayectorias de acuerdo a las informaciones recibidas, y el estado actual del vehículo.

Siguiendo con el paradigma de diseño *bottom-up*, y la visión de desarrollar un vehículo colaborativo, se desarrolla el diagrama de flujo mostrado en la Figura 3.8, en el cual se puede observar el algoritmo a implementar para lograr cumplir el objetivo de este TFG usando vehículos autónomos. En la figura mencionada, se observa de forma muy abstracta todo el proceso que se realizará. Empezando con inicializar el sistema toma de decisiones del subsistema de guía, y definiendo las siguientes metas según PODMP usando MCTS. Luego cuando las metas están definidas, se planifica la ruta desde la posición inicial a la meta y se procede a viajar para llegar a la meta. Como el vehículo será destinado al monitoreo de parámetros, se solicita al subsistema de medición que proceda a realizar las mediciones correspondientes y se espera a que el subsistema finalice su trabajo. Luego se procede a viajar hasta la siguiente meta, y posteriormente a buscar nuevas metas según PODMP hasta que finalice el proceso. Este algoritmo debe trabajar en segundo plano, ya que todavía faltan definir las comunicaciones y la cooperación distribuida entre agentes. Este proceso corresponde a la parte más crucial de este TFG, por lo que se lo considera como el principal.

Para el programa principal, se considera todo lo propuesto en esta sección y lo expuesto sobre diseños de cooperación distribuida en el capítulo anterior. El

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

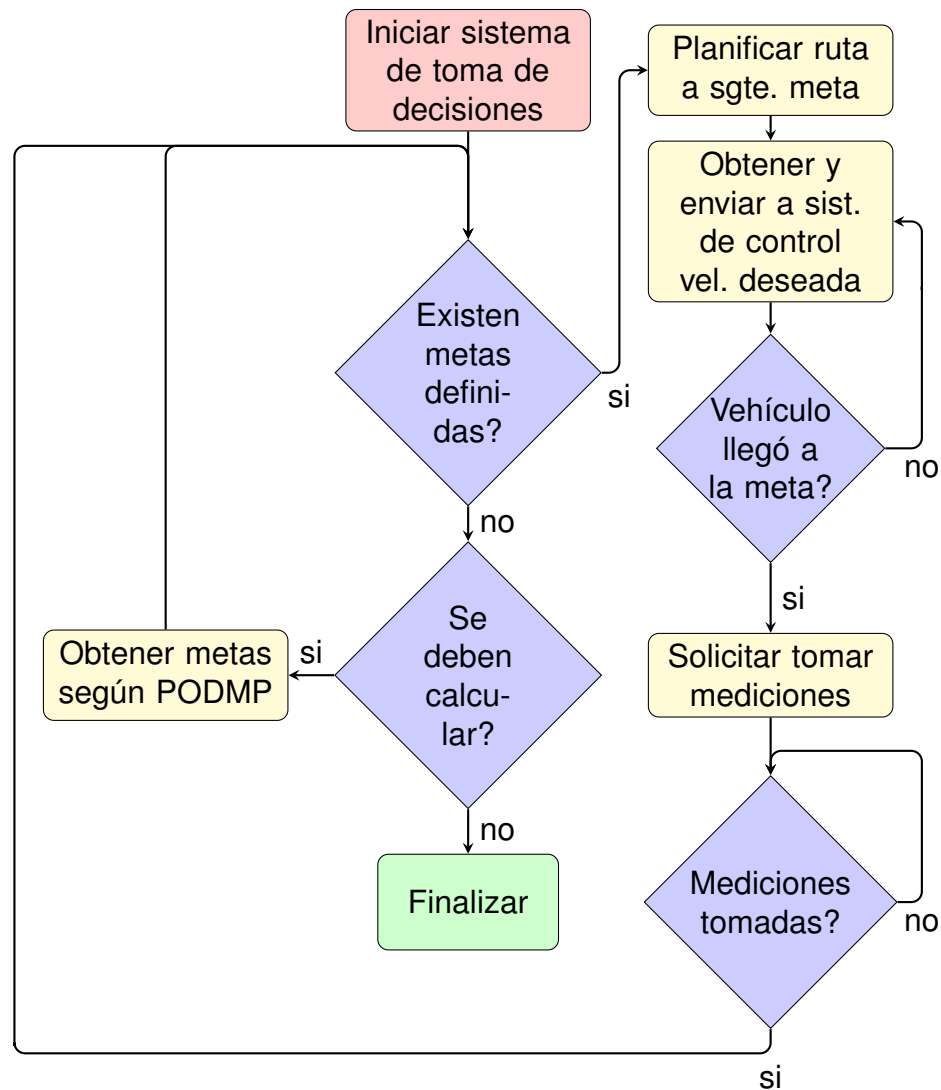


Figura 3.8: Diagrama de secuencias de la arquitectura de toma de decisiones del subsistema de guía interno. **Fuente:** Elaboración Propia.

programa principal, entonces, debe comunicarse con el programa secundario y con los subsistemas de control y navegación siempre que sea necesario. Con las definiciones y el diseño establecido, se desarrolla el diagrama de secuencias del subsistema de guía, mostrado en la Figura 3.9. Con esta configuración de secuencias, el sistema de guía es capaz de planificar y encontrar las velocidades deseadas y al mismo tiempo comunicarse con los demás robots en caso de ser necesario. También si las acciones de responder mensajes, realizar mediciones y calcular las regiones a cargo se hacen con tareas en hilos separados, se pue-

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

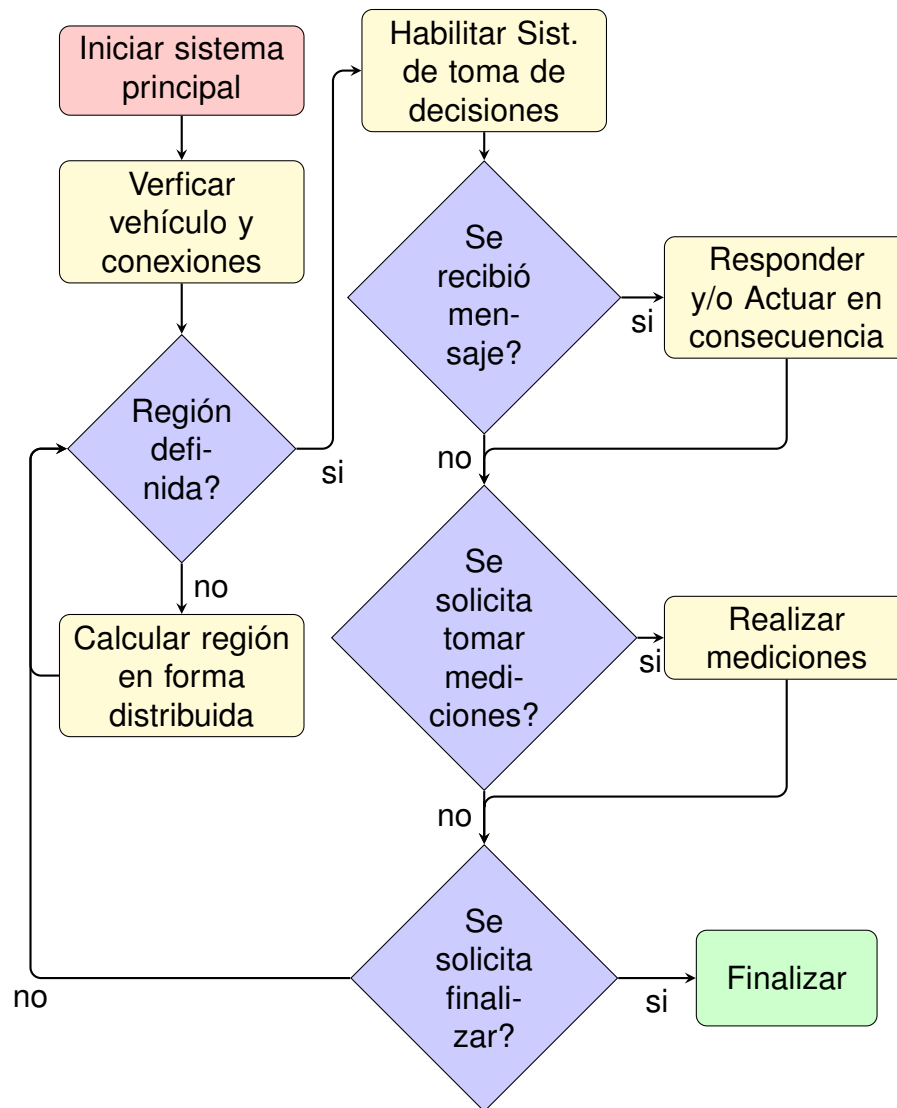


Figura 3.9: Diagrama de secuencias principal del subsistema de guía. **Fuente:** Elaboración Propia.

den realizar todas estas acciones de forma conjunta, mejorando el *throughput* o la cantidad de ejecución de instrucciones importantes en el tiempo. Los dos algoritmos hacen uso de muchas funciones encontradas en librerías disponibles para Python, el detalle de la utilización de las librerías se muestra a continuación:

- **Dronekit:** Se utiliza para obtener los valores de los subsistemas de control y navegación del vehículo. Además, posee funciones que permiten recibir y enviar mensajes para modificar los estados internos del vehículo. A través de la misma librería se puede obtener información acerca del nivel de ener-

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

gía restante y otros datos que no son propios del sistema GNC, pero son importantes a la hora de planificar rutas y definir velocidades deseadas.

- *MAVLink*: Librería que funciona como puente entre el sistema de comunicación interna del programa y el sistema embebido del vehículo. Con esta librería, se envían y reciben mensajes sobre informaciones como por ejemplo la velocidad y posición actual.
- *Cormoran-Simulator*: Es la librería diseñada en el LSD que contiene todas las implementaciones de los planificadores de ruta, además de funciones de conversión de píxeles a coordenadas. Esta es la librería que fue utilizada para determinar los rendimientos de los planificadores de ruta.
- *Numpy*: Librería que contiene funciones matemáticas.
- *Serial*: Librería que permite conexión para realizar una comunicación serial a través de puertos. En este caso, se utiliza para enviar y recibir mensajes del sistema de medición a bordo de un vehículo. Esta comunicación permite la obtención segura y confiable de datos de la calidad del agua cuando éste se encuentre disponible.
- *Threading*: Es una librería propia de Python que permite la ejecución simultánea de *scripts* o programas. Se utiliza para poder realizar al mismo tiempo las tareas de planificación de misión, planificación de rutas, comunicación externa a otros vehículos o a la estación base y comunicación con el sistema de medición a bordo.
- *Logging*: Librería de Python utilizada para crear archivos del tipo *log*. Estos archivos documentan los eventos, las acciones tomadas y las informaciones pertinentes en todo momento, permitiendo realizar análisis o usar la información según se crea necesario. Se documentan todas las informaciones posibles, ya sea de estados del sistema de guía, de navegación,

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

sobre las decisiones del sistema, y cualquier error que pueda ocurrir, como desconexiones imprevistas, etc.

- *Yaml*: La librería YAML es una interface de programación de aplicación utilizada para determinar las condiciones iniciales de los archivos a ser provistos al programa, como son: los mapas, los estados actuales, las definiciones para inicializar, y otros.
- *Json*: Todos los mensajes que no sean transmitidos al Pixhawk, es decir, los mensajes que no necesiten usar el protocolo provisto por MAVLink, se codifican como mensajes del tipo JSON. Estos *strings* o cadenas de caracteres se codifican con un sistema orientado a objetos, permitiendo el intercambio de, por ejemplo, diccionarios, listas, objetos, vectores, etc. Los mensajes tipo JSON son ampliamente utilizados en la literatura por la facilidad de integración a casi cualquier tipo de lenguaje de programación y la facilidad de creación de datos de este tipo. En esta implementación, la comunicación serial, la comunicación entre scripts y los datos transmitidos por MQTT utilizan JSON.
- *Scipy*: Librería que se encarga de obtener las regiones de Voronoi calculadas internamente en cada vehículo. Permite obtener los vértices, y aristas de cada región. Define las regiones de cada vehículo.
- *Shapely*: Librería que facilita el manejo de datos geométricos, en este caso, de datos de posiciones de vehículos, centros de regiones, vertices de cada región.
- *Paho-mqtt*: Librería liviana (mejor definida como API(Interface de programación de aplicación)) de mensajería del tipo pub/sub que es utilizada para la comunicación entre vehículos y con la estación base. Se encarga de comunicar los estados internos del vehículo a sus pares y recibir la misma

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

información de ellos. Es la única librería que se utiliza solo en el programa principal.

El código fuente de los programas principal y secundario pueden ser observados en el Anexo C, ambos *scripts* fueron escritos en Python. Los códigos fuente corresponden a la implementación de los diagramas de secuencia mostrados en esta sección, y utilizan las librerías mencionadas para facilitar y minimizar el tiempo de desarrollo de la implementación. Principalmente, el programa secundario incluye funciones de transformaciones, de estructuración de datos, implementaciones de generadores de trayectoria y secuencias para control y seguridad. El diagrama Voronoi se calcula en el programa secundario, pero a partir de una señal generada en el programa principal. También se desarrolla e implementa la librería MCTS según la sección de planificación de misión.

MCTS: Para el caso en que se desea cubrir un terreno en específico, es decir, la implementación planificada, sin necesidad de agilizar la búsqueda en lugares que tengan cierto nivel de valores de los parámetros, sino que se agilice la búsqueda en lugares que todavía no han sido visitados, se implementa el MCTS para resolver un PODMP que tiene las siguientes informaciones: Un modelo del vehículo, un set de acciones posibles, el modelo del entorno y un mapa de visita. El modelo del vehículo debe ser lo suficientemente simple como para poder simular su comportamiento, pero en un intervalo de tiempo realmente pequeño para facilitar la prueba de diferentes acciones. Esto es una función simple que tiene como entrada la velocidad deseada del vehículo y como salida la posición después de un intervalo de tiempo relativamente corto. El set de acciones acompaña el modelado del vehículo, de tal forma a limitar a un número finito de posibles acciones que puedan cumplir el objetivo o encaminar al vehículo a lograrlo. El modelo del entorno es la grilla de ocupación definida anteriormente y el mapa de visita es el primero de los mapas de parámetros, donde se define el tiempo transcurrido desde un tiempo inicial t_0 . Se proceden a definir los parámetros del

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

POMDP para el caso de este TFG cuando se busca cubrir toda una superficie:

- S: Conjunto de posición y la grilla de ocupación.
- A: Set de posibles direcciones a las que dirigir. En este caso, se definen direcciones que no dificulten el manejo ni la generación de ruta según un planificador: recto, y giros suaves a los lados.
- T: La trayectoria generada al moverse según un planificador de ruta seleccionado.
- R: Obtenida mediante la asignación de un puntaje por transitar cruzando un punto o llegar a uno en el mapa. Se usa como base la expresión $r_o = e^{(t_u - t_a)/k}$, donde t_a es el tiempo actual y t_u corresponde al tiempo de la última vez que el vehículo se encontró en la misma posición que la actual, k corresponde a una constante que permite variar el resultado de la expresión para determinar tiempos mínimos requeridos entre cada visita a un mismo punto. Es importante notar que la diferencia siempre será ≤ 0 , que brindará un valor $r_o \rightarrow (0, 1]$.
- ω : Son todas las observaciones de recompensa que se obtuvieron al realizar la acción a .
- O: es un valor que corresponde a la unidad en este caso, ya que, existiendo una velocidad en el vehículo, siempre se puede llegar al punto deseado al realizar la acción a .
- γ : El factor de descuento se toma directamente igual a uno porque así se maximiza exclusivamente la suma de las todas recompensas a recibir, en vez de la inmediata.

En cuanto a las acciones se toman 5 posibles destinos, un destino o punto de visita en la misma dirección actual, y los demás son puntos que provocarían

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

cambios de orientación suaves o duros a los costados. En caso de no poder tomar la acción, primero se busca acortar la distancia al posible destino, y, en última instancia, elegir puntos de visita que queden detrás del vehículo que no han sido visitados aún. Con el diseño del planificador de misión y planificador global detallado, en el Anexo C también puede observarse el programa auxiliar que implementa todo lo establecido, la implementación corresponde a funciones de MCTS que resuelvan el PODMP en lenguaje Python. Además de lo establecido, también se incluye en la implementación de este diseño, el simulador del cuál se habló en la subsección 3.2.2. De esta forma queda definido todo el MCTS, y se puede utilizar el algoritmo para encontrar posiciones de búsqueda de información que maximicen la obtención de nuevas informaciones del mapa.

3.3.2. Implementación de software y control

Pruebas de software realizadas en Python

Una prueba preliminar se realizó simulando vehículos autónomos en la misma computadora que la que realizó las pruebas en la subsección 29. Con la diferencia que los vehículos estaban simulados, no por el *cormoran-simulator* de F. Peralta y otros [65], sino por el APM Rover 2 simulator, provisto, a través de Mission Planner, por ArduPilot. Este simulador es capaz de conectarse a un controlador que utilice dronekit y MAVLink, las diferencias principales entre el simulador **APMRover2** y el vehículo real consisten en: 1. Conexión por *TCP*, en vez de serial. 2. Manejo del tipo Dirección + Acelerador, en vez de Acelerador Izquierdo + Acelerador Derecho. 3. GPS + RTK, en vez de solamente GPS. Las demás diferencias, como aceleraciones y uso de batería no se consideran importantes para la realización de pruebas, pero se tendrán en cuenta a la hora de elaborar conclusiones. El simulador del **APMRover2** incluye ruidos y otros efectos indeseados que ayudan a mejorar la igualdad entre el simulador y el vehículo real. Simulando un conjunto de 4 vehículos se puede observar, por medio del estudio

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

posterior de los archivos log de cada uno, que los vehículos se comunicaron para poder dividirse en regiones y empezar a visitar la región en búsqueda de información nueva. Una vez que algún vehículo obtenía suficiente información, calculaba los *waypoints* o planificaba globalmente, con respecto a lo que el planificador de misión establecía, luego, visitaba estos puntos siguiendo una ruta planificada localmente y se detenía en cada punto para obtener valores específicos de parámetros en cada uno de ellos. En la Figura 3.10 se observan las cuatro ventanas de simulación del *command window*, donde se encuentran los vehículos simulados del **APMRover2**. En el conjunto de pestañas de la ventana de *PowerShell* se observan a los subsistemas de guía implementados en Python funcionando, en donde se imprimen en pantalla todos los mensajes transmitidos usando MQTT en este caso se observan los mensajes enviados a un servidor central que actúa como observador únicamente. Los vehículos trabajaron de forma correcta consiguiendo alcanzar las velocidades deseadas según sus posiciones actuales y cada una de sus metas establecidas.

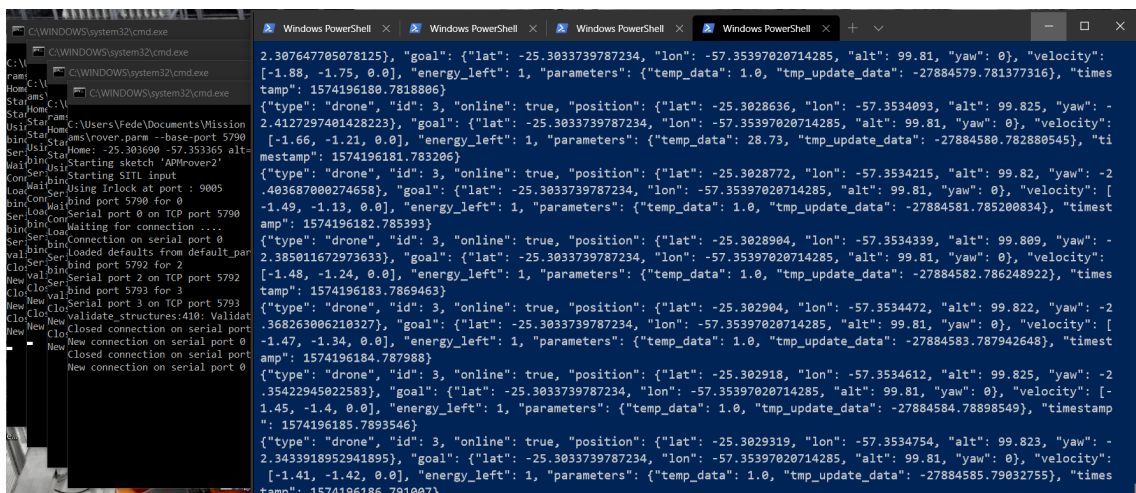


Figura 3.10: Pruebas del sistema en varios drones simulados. **Fuente:** Elaboración propia.

Aunque el Mission Planner está diseñado para visualizar en menor medida a un conjunto de vehículos, no está del todo preparado para poder observar todos los vehículos de forma individual u obtener los valores de los parámetros del

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

mismo. Dicha razón dificulta la observación correcta y fácil de la posición actual de cada uno de los vehículos, sus velocidades y estados correspondientes. Por eso se diseña un observador que servirá exclusivamente para poder facilitar a la observación y verificación de los estados de todos los vehículos en una misma ventana.

Estación Base de Observación: Como Trabajo Anexo a este TFG, se diseña, desarrolla e implementa un script de Python que recibe información, a través de MQTT, de los estados de los vehículos y los transforma en datos útiles para proyectarlos y usarlos para graficar los estados actuales de cada vehículo. Específicamente, se toman los datos de posición y los ubica en un espacio donde es posible graficar. También se toma la inclinación actual del vehículo con respecto al Este para poder dar una mayor facilidad de comprender la intención del vehículo. También se toma la información del estado de trabajo del vehículo, para el caso de que se encuentre funcionando, se lo grafica en un color verde, y rojo en caso contrario. Por último, se incluye en la imagen la posición de cada una de las metas actuales de cada vehículo.

En la Figura 3.11 se observa la ventana del *Base Station* de los 4 vehículos mencionados anteriormente como ejemplo. Se agregan la función para centrar en la imagen, la posición actual de alguno de los drones conectados, y también se agregan las informaciones recibidas separadas y ordenadas para una mejor visualización. Para realizar la ventana gráfica se utilizó Qt Creator 4.9.2 (Community), software propietario de The Qt Company [72], aprovechando la disponibilidad para realizar ventanas del tipo *.ui* (usables en todos los OS de Microsoft desde Windows Vista en adelante). El control de la ventana se realizó usando la librería *PySide2* disponible para Python. En el Anexo D, se encuentra la implementación completa del programa de control de la Estación Base, escrita en Python.

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

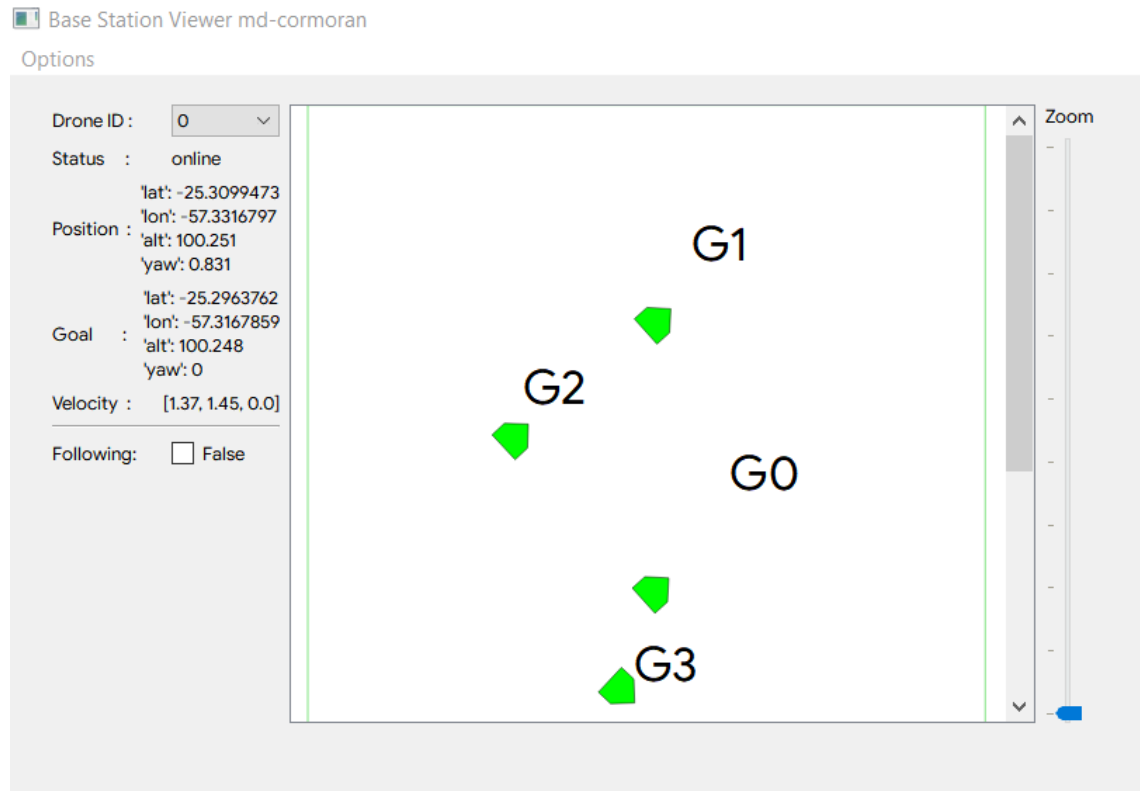


Figura 3.11: Visualización del sistema en el *Base Station*. **Fuente:** Elaboración propia.

Pruebas de software realizadas en los Sistemas Embebidos

En robots móviles, es necesaria una respuesta rápida cuando se planifican movimientos, ya que el ambiente puede estar variando de forma continua, forzando al robot a cambiar trayectorias en poco tiempo para salvaguardar su propio estado y el del ambiente. Esta es la razón principal del porqué se busca una velocidad rápida, o por lo menos adecuada, de procesamiento de datos, más específicamente, de respuesta de un planificador de rutas. Para realizar esta prueba, se utiliza un modelo básico y funcional del planificador de rutas que permita obtener la respuesta. Mientras tanto, se toma en cuenta la cantidad de tiempo transcurrido de cálculo y se lo compara con respecto a la distancia entre los puntos donde se busca una ruta. Las simulaciones usan el mismo set de puntos, y el objetivo del robot es el de visitar cada punto una vez, mientras se mi-

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

den variables de tiempo tardado de cálculo de rutas y la distancia existente entre los puntos. Las pruebas se realizaron en los sistemas de NVIDIA Jetson TX2 y Raspberry Pi 3 B debido a la disponibilidad de los equipos en los laboratorios; teniendo en cuenta las características de estos sistemas se puede concluir que se tendrán resultados que indiquen el mayor y el menor valor posible entre los sistemas embebidos estudiados. En la Figura 3.12, se observan los resultados del tiempo tardado para calcular una ruta entre 2 puntos, con respecto a la distancia entre estos. Se puede observar fácilmente que el tiempo de cálculo en la NVIDIA Jetson TX2 es menor, acentuándose esta diferencia más a medida que la distancia entre puntos es mayor.

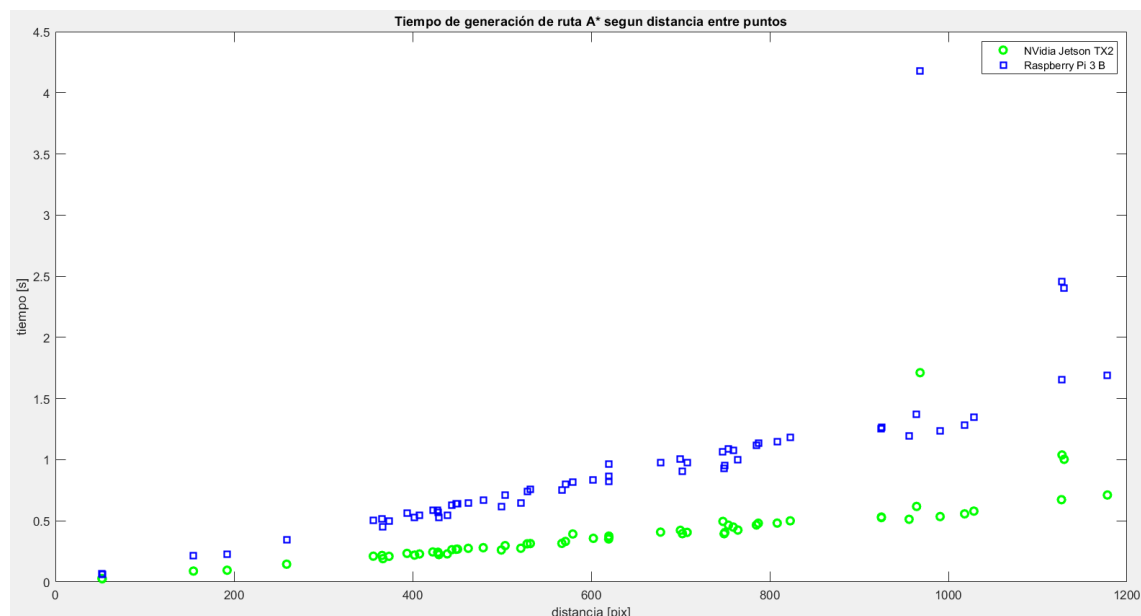


Figura 3.12: Resultado de las pruebas realizadas en sistemas embebidos.

Fuente: Elaboración propia.

Mediante las informaciones obtenidas, se puede concluir que usando la Raspberry Pi 3B se tiene aproximadamente un factor de 1.49 mS/pix y para la Jetson TX2 0.63 mS/pix, es decir que el primer sistema embebido puede planificar rutas a una velocidad de 671 pixeles por segundo, mientras que la Jetson lo hace a una velocidad de 1587 pixeles por segundo. Con respecto a esta información y a la Tabla 2.2 se puede inferir que para mapas poco detallados o mapas pequeños

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

es interesante usar la Raspberry Pi 3B, debido a que no se necesita una velocidad muy grande, mientras que para mapas detallados o mapas grandes se torna realmente necesario contar con la placa de desarrollo de Nvidia. A esto se suma el proceso de poder controlar el subsistema de guía sin mayores complicaciones, por lo que para la problemática y para la autonomía esperada de los vehículos es casi necesario el uso de la NVidia Jetson TX2. No está demás dejar en claro que el mapa para las pruebas, si bien era un mapa grande (1000 x 1500 pixeles), no era un mapa detallado, ni complicado, es decir que no se encontraban demasiadas dificultades a la hora de elaborar las rutas. A medida que existen más complicaciones de caminos, rutas y/o obstáculos, mayor será el tiempo en que tarda un planificador en encontrar una ruta óptima.

3.4. Implementación.

3.4.1. Sistemas de comunicación inalámbrica.

De forma a poder compartir los datos a través de MQTT y poder entregar una autonomía de manejo a cada uno de los vehículos, se opta utilizar la tecnología móvil 3G/4G. Esto es debido, principalmente a que al rango de utilización en campos abiertos, y que los vehículos puedan separarse libremente el uno del otro sin la necesidad de conectarlos. Además, según datos extraídos por investigadores del LSD en el mes de febrero del 2018 (Figura 3.13), existe una buena recepción de señal 3G en el Lago Ypacaraí. El nivel de señal se grafica desde rojo a verde según la calidad de la señal recibida, los lugares graficados en escala de grises corresponden a lugares en donde no se han tomado datos. De igual manera, usando herramientas en línea, como la desarrollada por nPerf [73], se puede determinar la calidad de la señal en las ciudades que rodean al Lago, que concuerdan con la información mostrada en la Figura mencionada.

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

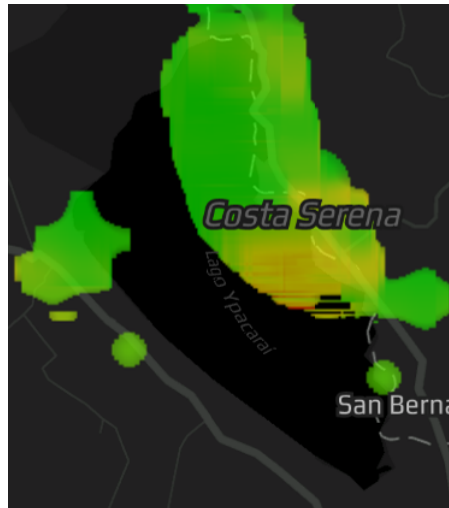


Figura 3.13: Visualización de la Cobertura 3G. **Fuente:** Rojas y otros (LSD).

3.4.2. Sistema de Guía propuesto

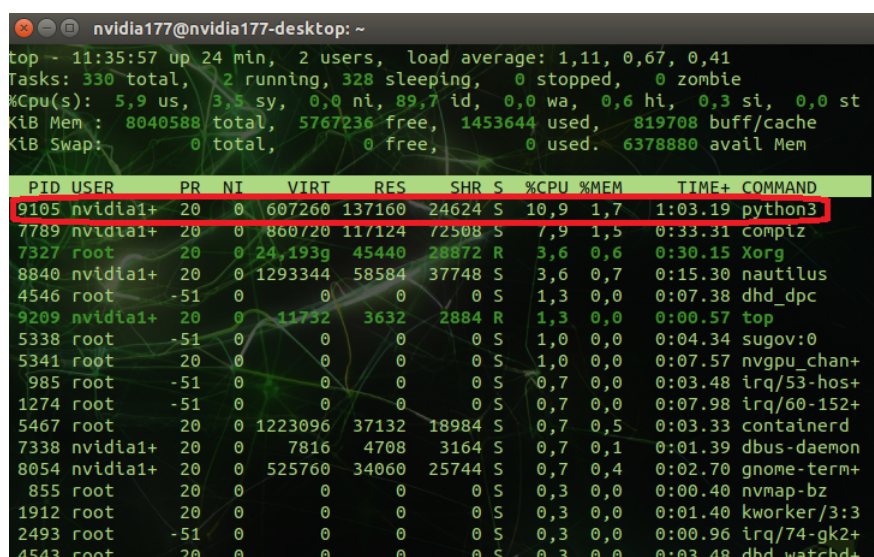
Con todo el diseño listo, se procede a instalar todas las librerías necesarias y a preparar la placa de desarrollo Nvidia Jetson TX2 para realizar una prueba de funcionamiento del sistema de guía desarrollado en este TFG. Finalmente, el sistema autónomo consta de:

1. Nvidia Jetson TX2, con sistema operativo Linux 4 Tegra en su versión más reciente, instalado usando JetPack. Adicionalmente, se instala Python 3.6.5 y las librerías anteriormente mencionadas.
2. Pixhawk con el firmware APMRover conectado a través de puertos seriales a la Jetson.
3. Conexión a Internet usando módem 4G.

En la Figura 3.14 se puede observar la tabla de procesos (*top*) mientras se está ejecutando el sistema de guía en la placa de Nvidia Jetson TX2. Ocupa aproximadamente un 11 % de los procesos y solo 1.7 % en la memoria RAM. Habiendo suficiente espacio para otros procesos como transmisiones de vídeo,

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

reconocimiento de imágenes, entre otros procesos comunes en vehículos autónomos de superficie.



```

top - 11:35:57 up 24 min,  2 users,  load average: 1,11, 0,67, 0,41
Tasks: 330 total,   2 running, 328 sleeping,   0 stopped,   0 zombie
%Cpu(s):  5,9 us,  3,8 sy,   0,0 ni, 89,7 id,   0,0 wa,   0,6 hi,   0,3 si,   0,0 st
KiB Mem : 8040588 total, 5767236 free, 1453644 used, 819708 buff/cache
KiB Swap:   0 total,   0 free,   0 used. 6378880 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM    TIME+  COMMAND
 9105 nvidia1+  20   0 607260 137160 24624 S   10,9   1,7   1:03.19 python3
 7789 nvidia1+  20   0 860720 117124 72508 S    7,9   1,5   0:33.31 complitz
 7327 root      20   0 24,193g 45440 28872 R    3,6   0,6   0:30.15 Xorg
 8840 nvidia1+  20   0 1293344 58584 37748 S    3,6   0,7   0:15.30 nautilus
 4546 root     -51   0      0      0      0 S    1,3   0,0   0:07.38 dhd_dpc
 9209 nvidia1+  20   0 11732 3632 2884 R    1,3   0,0   0:00.57 top
 5338 root     -51   0      0      0      0 S    1,0   0,0   0:04.34 sugov:0
 5341 root      20   0      0      0      0 S    1,0   0,0   0:07.57 nvgpu_chan+
 985 root     -51   0      0      0      0 S    0,7   0,0   0:03.48 irq/53-hos+
1274 root     -51   0      0      0      0 S    0,7   0,0   0:07.98 irq/60-152+
 5467 root      20   0 1223096 37132 18984 S    0,7   0,5   0:03.33 containerd
 7338 nvidia1+  20   0 7816 4708 3164 S    0,7   0,1   0:01.39 dbus-daemon
 8054 nvidia1+  20   0 525760 34060 25744 S    0,7   0,4   0:02.70 gnome-term+
 855 root      20   0      0      0      0 S    0,3   0,0   0:00.40 nvmap-bz
1912 root      20   0      0      0      0 S    0,3   0,0   0:01.40 kworker/3:3
2493 root     -51   0      0      0      0 S    0,3   0,0   0:00.96 irq/74-gk2+
 4543 root      20   0      0      0      0 S    0,3   0,0   0:03.48 dhd_watchd+

```

Figura 3.14: Captura de pantalla de la tabla de procesos de la Jetson TX2.
Fuente: Elaboración Propia.

De todas formas, para las pruebas se utilizará el sistema de guía en un vehículo simulado en una computadora, conectado por TCP/IP a la Jetson TX2, sirviendo ésta como sistema de guía y comunicación, mientras que el sistema de control y navegación será simulado al igual que el vehículo en el simulador **APMRover2**.

3.5. Pruebas del sistema.

Debido a la necesidad de validar y verificar en un sistema de múltiples drones, se introducirán al entorno de pruebas, una cantidad predefinida de drones simulados en computadoras con sus respectivos sistemas de guía en la misma computadora. Todos los sistemas se conectarán a un *broker* (corredor de información) para poder hacer uso del protocolo MQTT. Con la utilización de demás terminales para poder tener un sistema de múltiples drones se pueden realizar las pruebas finales del sistema de guía de múltiples drones de superficie. Se

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

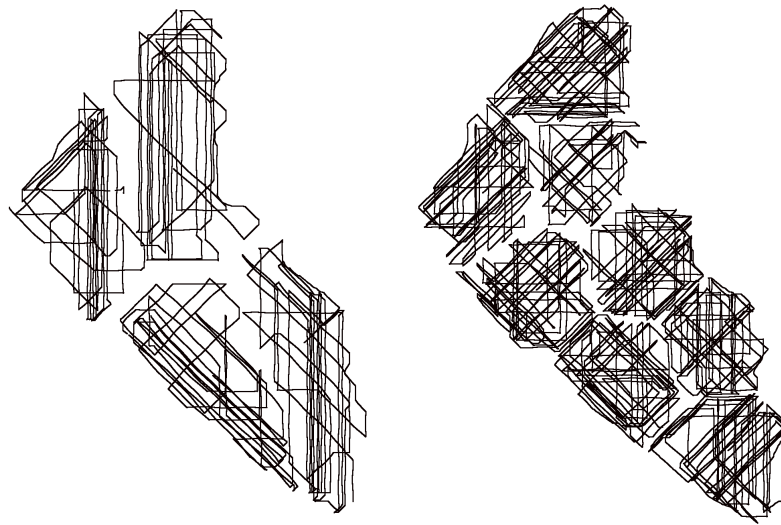


Figura 3.15: Resultados de la cobertura de área de las pruebas realizadas usando el sistema propuesto de POMDP con 4 y 8 drones.

Fuente: Elaboración propia.

realizan a continuación 2 pruebas con cuatro y ocho drones respectivamente, implementando el sistema diseñado, y se toman en cuenta los factores mencionados para la validación del mismo. El resultado de utilizar 4 drones para realizar y buscar cumplir la misión de cubrir la máxima cantidad de área posible como se observa en la Figura 3.15. Se realiza además el mismo proceso pero utilizando 8 drones simulados y se puede observar en la misma Figura 3.15 el resultado. Es de esperar que los vehículos finalicen y cumplan la misión en menor tiempo, por lo que es necesario definir un método para la elección de la cantidad óptima de vehículos.

De la Figura 3.15 presentada se puede concluir que el sistema propuesto distribuye las regiones a visitar, aunque existen ciertos solapamientos de áreas debido a la descentralización del cálculo de las regiones. Aunque esto no se haya considerado se consiguen cubrir efectivamente los límites de las regiones, suponiendo una mejor obtención de parámetros. También se observa que los vehículos buscan generalmente no cambiar demasiado sus direcciones, favore-

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

ciendo a una obtención más *macro* de los parámetros. Este proceso se repite, consiguiendo obtener información en segmentos de gran longitud. Es importante tener informaciones en segmentos de gran longitud para poder detectar con mayor facilidad las plumas de contaminación, ya que los tiempos de propagación de nutrientes depende del grado de solubilidad en agua de los mismos, con lo que pueden obtenerse distintos conjuntos de contaminantes esparcidos según sus propiedades químicas. De todas formas, la misión debe centrarse en obtener información casi continua del Lago, es decir, obtener informaciones que no se encuentren demasiado alejadas unas de otras. Esto ayuda a obtener una mejor representatividad del entorno. Cabe recordar que en el modelado utilizado cada punto representa 10.33 metros aproximadamente, y con lo definido como cobertura en el tiempo, se consigue obtener durante la misión, información útil cada vez que el vehículo se encuentra en un punto del mapa diferente, es decir, por lo menos cada 10.33 metros. También se pueden observar en la Figura 3.15 que las rutas trazadas casi no dejan espacios libres de monitoreo luego de finalizar la misión, que ayuda a obtener informaciones claras y concisas. El criterio de parada consideraba la cantidad total de metas según POMDP calculadas o si alguna de las metas retornaba una recompensa negativa. En todos los casos, los vehículos finalizaron su trabajo debido al primer criterio, visitando un total de 75 puntos cada uno dentro del lago.

Para formalizar la elección de una cantidad apropiada de drones, se debe tener en cuenta los factores de cooperación, costo económico, tiempo total de ejecución, porcentaje de cobertura de área, entre otros detalles que hacen que el uso de un sistema de múltiples drones sea indicado y apropiado para una misión o un objetivo. Para una comparación con el método de patrones en regiones presentado en 2.2.3, se utiliza la librería de Python desarrollada por A. Sakai y otros [36] para hacer uso de la función que genera los patrones. Exclusivamente se modifica el Algoritmo Secundario, en la forma de obtención de los puntos a

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

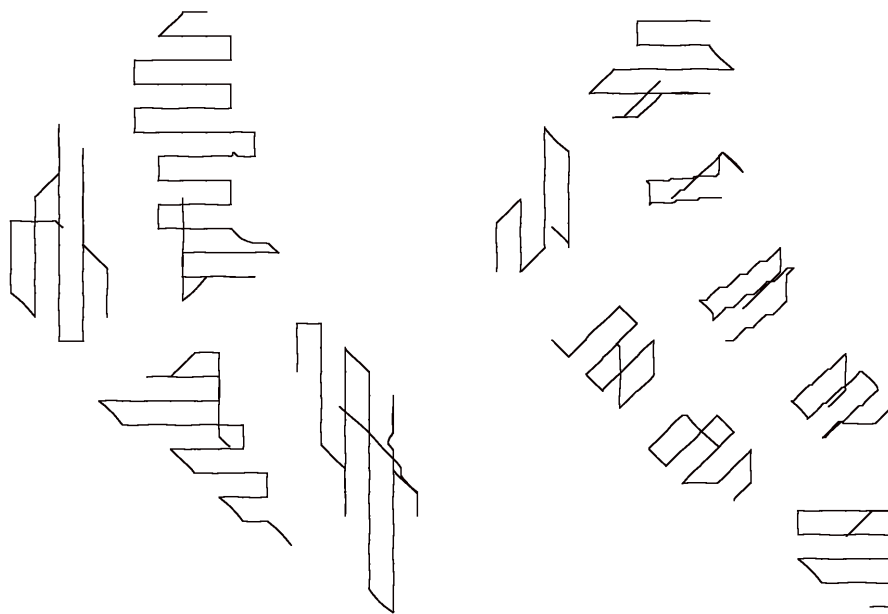


Figura 3.16: Resultados de la cobertura de área de las pruebas realizadas usando patrones de cobertura por barrido con 4 y 8 drones.

Fuente: Elaboración propia, con base en la librería de A. Sakai y otros [36].

visitar, el resto del subsistema de guía se mantiene, incluyendo el sistema de comunicación, y los componentes de planificación local de ruta con A^* y planificación de trayectoria, así como el método de cálculo de región de Voronoi en forma distribuida. El mismo método se usa para 4 y 8 drones y los resultados se observan en la Figura 3.16. Es claro que existe una cobertura de área que permite obtener información de toda el área o región a cargo de cada uno de los vehículos, a grandes rasgos, pero se obtiene muy poca información específica debido a los parámetros establecidos. De todas formas, se consigue seguir las rutas planificadas, aún con modificaciones por los planificadores locales de ruta y de trayectoria.

Por otro lado, para una comparación con un sistema de monitoreo más simple, en una prueba adicional, se utilizó una parte del sistema de guía, pero usando solo un vehículo, instruyéndolo de realizar el recorrido propuesto por M. Arzamendia y otros [12], usando planificador local de A^* , pero el planificador global

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

evolucionado por Algoritmos Genéticos (GA) mostrado en el trabajo mencionado. El resultado se observa en la Figura 3.17.

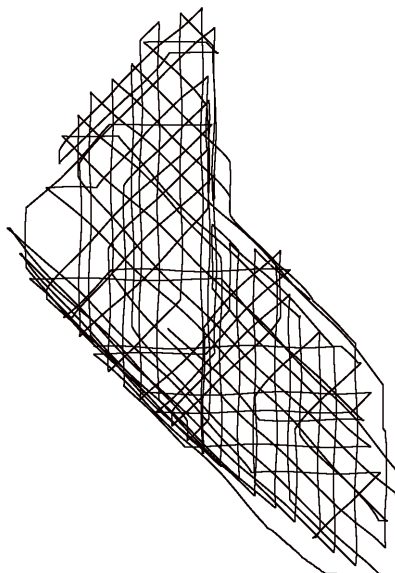


Figura 3.17: Resultado de la cobertura de área de la prueba realizada en un solo vehículo que visitar 60 puntos.

Fuente: Elaboración propia, con base en la propuesta de M. Arzamendia y otros [12].

Los datos recogidos durante el desarrollo de todas estas pruebas se consolidan en la Tabla 3.2. Para el caso de un solo vehículo se definió una velocidad máxima admisible de 5 metros por segundo, mientras que para los sistemas de múltiples drones una velocidad de solo 2 metros por segundo. El resto de los parámetros de simulación para cada vehículo se mantuvieron iguales.

El método de GA finalizó al recorrer las 60 rutas, consiguiendo cubrir aproximadamente 0.95 % del terreno por hora, al finalizar, se observan rutas propias de A^* y una buena cobertura en la mayor parte del espacio disponible. Si el sistema se reinicia por imprevistos, se provoca la repetición de rutas, si por el contrario se diseñara una función para ser más robusto con respecto a reinicios, el vehículo seguirá la ruta planificada en todos los casos.

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

Tabla 3.2: Síntesis de pruebas finales realizadas con varios sistemas

	Sistema de Guía	Cant. de Drones	Tiempo tot. de Ejec.	Porcentaje de cobertura	Velocidad max.
1	GA [12]	1	23.09 hs.	21.9	5 [m/s]
2	P. de barrido [36]	4	4.60 hs.	5.1	2 [m/s]
3	P. de barrido [36]	8	2.119 hs.	4.73	2 [m/s]
4	POMDP	4	18.73 hs.	24.23	2 [m/s]
5	POMDP	8	16.48 hs.	33.54	2 [m/s]

Fuente: Elaboración propia.

Sobre el método de patrones de cobertura por barrido en regiones, se obtienen coberturas rápidas para recorrer mucha área en poco tiempo, pero con poco nivel de detalle. Utilizando un parámetro de 50 pixeles (517 m. aprox.) de distancia, se consigue el resultado mostrado. Un reinicio imprevisto del sistema o de cualquier dron provoca la repetición de la misma ruta, y con funciones que prevengan este error, se conseguirá continuar las mismas rutas planificadas. El porcentaje de cobertura usando 4 drones a solo 2 [m/s] es de 1.11 [% /hora] y usando 8 drones 2.232 [% /hora].

Para el caso del sistema propuesto en este TFG, se consigue cubrir el terreno a una razón de 1.3 % por hora para 4 drones y 2.04 % cuando se utilizan 8 drones. *A priori*, para el caso de 4 drones, el sistema es un poco mejor en términos de porcentaje por hora con respecto a los otros dos presentados. Si ocurren imprevistos, se vuelven a calcular rutas que entreguen el mejor resultado para el punto en donde actualmente se encuentre el vehículo que debió reiniciar su sistema. Esto mejoraría a corto plazo el porcentaje de cobertura, ya que, en el mejor de los casos, el POMDP se calcula para cada acción a realizar, es decir cada velocidad deseada a enviar al subsistema de control. Haciendo un estudio más detallado, se observa el porcentaje de cobertura por hora según los métodos presentados. La Figura 3.18 presenta varias curvas que representan al porcentaje por hora. Cada curva fue realizada al obtener el porcentaje cubierto en 25 %, 50 %, 75 % y 100 % del tiempo total utilizado por los drones, luego se adaptaron

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

estos valores para obtener la curva mediante interpolación segmentaria cúbica (*spline*). En definitiva, se consigue un porcentaje de cobertura ideal, en donde se utilizan vehículos totalmente autónomos que rápidamente consiguen información sobre el entorno en el cual están trabajando.

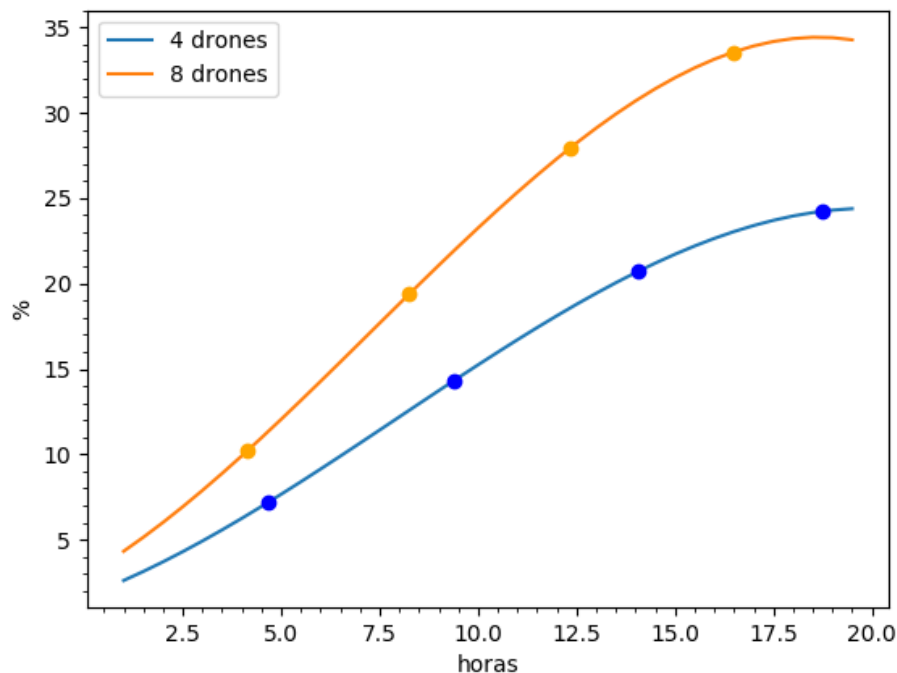


Figura 3.18: Curvas de porcentaje de cobertura de área por hora para pruebas del sistema propuesto con 4 y 8 drones.

Fuente: Elaboración propia.

Sobre el factor de cooperación se puede considerar que las acciones altruistas consisten en responder a mensajes de posición actual o centro de región correspondiente a cada vehículo. Para realizar la acción se debe tener capacidades de comunicación por 3G/4G, es decir, poseer un módulo 3G/4G, estar cerca de una antena de comunicación y poseer datos para navegación en Internet, aparte del tiempo consumido para realizar la comunicación. El beneficio es el de poder limitar su navegación en una zona de acuerdo a la cantidad de drones disponibles.

CAPÍTULO 4

Marco Analítico

4.1. Discusión y análisis de resultados particulares

Como ya se ha presentado los sistemas y realizado pruebas con cada uno de los componentes de subsistema diseñado, se procede a discutir sobre cada elección hecha y su valor académico, social y tecnológico.

4.1.1. Planificador Local de Ruta

Las tablas y figuras expuestas en la subsección 29 detallan claramente el rendimiento de los algoritmos o métodos de planificación local altamente utilizados en el mundo actual. Si bien el método propuesto *uFMS* obtuvo un rendimiento menor al método de A^* , su uso no queda desechado debido a que el nivel fue satisfactorio, contrario a la ruta encontrada usando *FMS*. La ventaja clara de este método con respecto a los otros se da en la reutilización de informaciones ya producidas con anterioridad que bien pueden ser realizadas en el laboratorio para su uso posterior. Del estudio detallado del tiempo de cálculo realizado, se pudo observar que el programa toma la mayor parte del tiempo para obtener la información de mapas anteriores alojada en un archivo, pero la replanificación propiamente toma un tiempo menor que el tiempo total de cálculo usando *RRT**, por ejemplo. *RRT** por otro lado provee rápidamente de rutas planificadas sin la necesidad de usar informaciones calculadas anteriormente, aunque el nivel de seguridad de las rutas se ve ampliamente degradado, debido a que genera co-

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

nexiones entre nodos distanciados y ubicados aleatoriamente. Además de esto, la diferencia de distancia viajada entre los puntos supera el 13 %, y aunque este factor no dificulta cumplir con el objetivo de la misión, este factor afecta a la calidad de la misma. RRT^* y A^* son los métodos que tardan el menor tiempo en producir u obtener rutas planificadas libres de obstáculos. El tiempo total de simulación son similares debido a ese motivo. PF calcula la ruta más directa posible, aunque su tiempo de cálculo y su seguridad no son los mejores. FMS y $uFMS$ producen rutas muy seguras aunque la diferencia de distancia viajada es realmente grande. $uFMS$ reduce dramáticamente el tiempo de cálculo de ruta, haciéndolo realmente competitivo por su nivel de seguridad. Sobre el nivel de seguridad de las rutas calculadas, solo A^* FMS y $uFMS$ cumplen todas las necesidades y los requerimientos predefinidos. Los demás planificadores fallan en alguna especificación que reducen el nivel de seguridad.

En definitiva, la primera opción para planificadores locales de ruta es la técnica A^* , la cual, según los parámetros previamente establecidos, es la que mejor rendimiento tiene para realizar coberturas de área en entornos acuáticos, combinando buenos niveles de tiempo de cálculo o planificación, diferencia de distancia viajada y niveles de seguridad.

4.1.2. PODMP y MCTS como solucionador de PODMP

La implementación realizada del POMDP consigue simular posibles acciones en el simulador realizado en el LSD, estas acciones consisten en viajar una distancia d en línea recta o cambiar la dirección 45° o 90° a uno de los lados, o en última instancia a 135° a uno de los lados. El MCTS simula 3 niveles de profundidad, es decir, crea rutas a seguir con 3 diferentes metas, con lo que, en total, se encuentran 125 posibles metas y se simulan viajar hasta dichas metas observando el mapa de parámetros y obteniendo la recompensa al pasar por uno punto. A partir de esto es posible encontrar el mejor conjunto de puntos a visitar

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

de tal forma a maximizar la recompensa obtenida al visitar cada uno de ellos, en el orden establecido. La recompensa fue seleccionada de tal forma a favorecer la selección de puntos que permitan conseguir la información más completa posible, es decir, que los puntos seleccionados de visita no lleven a los vehículos a transitar por caminos anteriormente visitados. Utilizando la función de recompensa presentada, se consigue encontrar valores positivos de recompensa al visitar puntos que no han sido visitados en un lapso de tiempo de 3 meses aproximadamente, tiempo considerado prudencial debido a factores físico-químicos del agua del Lago, como son la casi nula corriente del lago, el tiempo de eutrofización, etc. También se utiliza este límite para favorecer a la economía y reducir esfuerzos para conseguir parámetros, como son el transporte de los drones hasta el lago, la puesta en marcha, el control y otros gastos indirectos como recargas de batería, etc. Las acciones posibles predefinen de alguna forma los mejores caminos a tomar, pero como se toman puntos a visitar en todas las direcciones que no representen giros o maniobras peligrosas, aunque cuando estas maniobras no pueden ser ejecutadas, se busca realizar giros duros siempre y cuando los vehículos no colisionen con ningún obstáculo. Debido a que se toma el tiempo para generar una recompensa, el resultado de realizar alguna acción siempre será observable, lo que facilita enormemente la resolución hacia el mejor valor. Esta elección puede determinar teóricamente que el proceso es realmente observable, convirtiéndolo en un MDP o FOMDP (*Fully Observable Markov Decision Process*), pero no es así, ya que no se incluyen posibles obstáculos estáticos o dinámicos que puedan encontrarse, ni se simula exactamente al vehículo debido a las limitaciones de software desarrollado. Con esto la resolución se confirma para un entorno parcialmente observable y se reconfirma la definición como un POMDP. Si de igual forma no se considera suficiente razón como para considerar como un POMDP, un FOMDP puede ser reformulado como un POMDP, al definir que a cada observación corresponde un solo estado y que las observa-

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

ciones realizadas son certeras y determinadas. Por lo que, el diseño presentado en este TFG corresponde a un POMDP. Además de esto seleccionar la diferencia de tiempo con respecto a visitas anteriores para generar la recompensa permite disipar ambigüedades con respecto a los puntos elegidos, pero en caso que se elija cualquier otro tipo de recompensa, es preferible determinar el mejor punto a visitar según procesos que incluyan el reconocimiento de que las lecturas puedan incluir ruido o no ser idénticas a las lecturas simuladas. De igual forma, el sistema está preparado para este tipo de requerimientos y solo bastaría con modificar la formulación de la recompensa posible a obtener.

La idea de que la información puede cambiar en el tiempo, además de la necesidad de la mayor información posible, hacen que la selección del sistema que utiliza POMDP sea la indicada. Esta afirmación está respaldada por la tabla de comparación (Tabla 3.2) mostrada en el capítulo anterior. Para el caso de estudio de este TFG, es realmente necesario este sistema de guía de múltiples drones conformado por los componentes diseñados y desarrollados. Una opción aceptable es la de usar la modificación para utilizar el método de patrón de cobertura por barrido, pero este método pierde mucho rendimiento al incluir los planificadores locales de ruta, y tampoco está preparado para resolver casos no especificados, pero sirve como base para realizar eficientemente la cobertura de área. En comparación con el método propuesto que utiliza 60 *waypoints* y un orden de visita, pero usa un solo dron, se puede observar que el porcentaje de cobertura es parecida pero el tiempo total tardado y la cantidad de energía utilizada, que representa una cantidad de pausas y reinicios de misión, hacen que el método propuesto en este TFG, utilizando múltiples drones y POMDP para determinar los *waypoints*, sea la opción indicada.

Para determinar un número óptimo de drones, es importante que el porcentaje de cobertura por hora sea adecuada para cumplir con los objetivos de misión, y para ello, se realizar pruebas adicionales del sistema propuesto con 5 y 11

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

drones. Se vuelven a obtener las mismas curvas de porcentaje de cobertura por hora de cada prueba y, en este caso, se agrega una dimensión de cantidad de drones y se obtiene la superficie mediante una aproximación por Función de Base Radial (*RBF*). Con esto, cada punto representa el porcentaje de cobertura por hora por cantidad de drones, permitiendo obtener un valor ideal según sea cada caso. La Figura 4.1 presenta esta superficie, en donde se grafican las curvas para 4, 5, 8 y 11 drones y la superficie generada con estas curvas y el método de aproximación *RBF*.

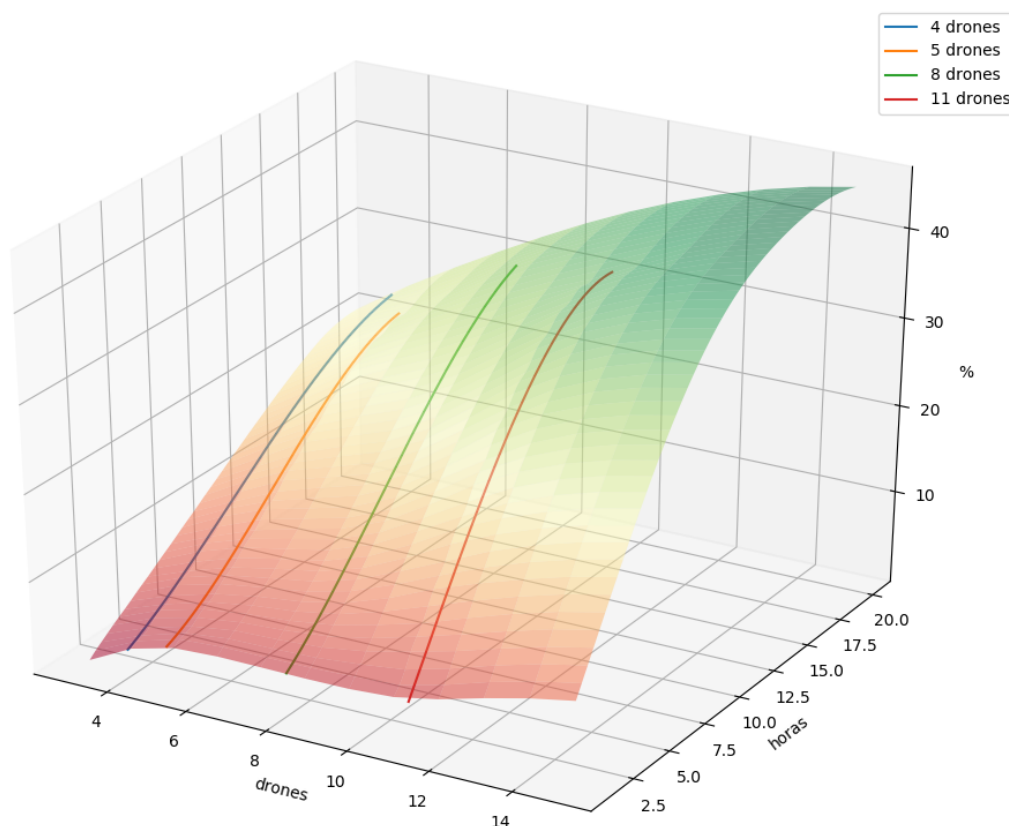


Figura 4.1: Curvas de porcentaje de cobertura de área por hora según la cantidad de drones del sistema propuesto.

Fuente: Elaboración propia.

Teniendo en cuenta esta Figura se puede determinar la cantidad apropiada de drones para realizar la misión según se desee cubrir un cierto porcentaje de área o según las horas de trabajo que se busca cumplir. Para una elección con respecto al factor económico completo, se debe tener el sistema autónomo com-

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

pleto, que queda fuera del alcance de este TFG, pero en términos exclusivamente técnicos, la decisión puede basarse en el gráfico presentado.

4.1.3. Sistema Embebido

La elección correcta de un sistema embebido encargado del subsistema de guía está ligado a factores intrínsecos del fin o el objetivo de cada robot. En este caso, es necesario desarrollar un sistema de vehículos autónomos que esté preparado recorrer toda la extensión del Lago Ypacaraí, por lo que cada vehículo debe poder reconocer y tener en cuenta todo el Lago para que, en conjunto, se pueda concretar la misión. Como es sabido, el Lago Ypacaraí es un Lago de grandes extensiones y debe de poder tenerse la mayor cantidad de información en el modelado, dicho de otra forma, mapas de grandes dimensiones y bien detallados. Por lo que, para que el sistema general de cada vehículo no se vea dificultado para planificar rutas locales o globales, para determinar velocidades deseadas en tiempos ajustados y para trabajar con el modelado de grandes dimensiones sin dificultades, se debe seleccionar la opción que menos dificulte el trabajo, en este caso usar la Jetson TX2 como sistema embebido. Existen también modelos más económicos de estas placas de desarrollo de la misma manufacturera, por lo que con un estudio más específico se podría elegir una placa parecida. Pero el rendimiento que ofrece la Nvidia Jetson TX2 es suficiente para con la misión definida.

4.1.4. Software en Sistema Embebido

La prueba de consumo de recursos en la Jetson TX2 demuestra que el sistema es apropiado y ofrece un rendimiento óptimo y eficiente como fue mostrado en la Figura 3.14. Hasta es posible usar parte de los recursos sobrantes para tareas de reconocimiento de imágenes, de transmisiones o *streaming* de los vehículos, y hasta de realizar tareas o algoritmos de simulación de mayor reque-

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

rimiento computacional para mejorar la salida del POMDP.

Además, es posible usar este sistema propuesto, con algunas modificaciones, en otro grupo de drones que busquen realizar procesos diferentes como coberturas de terreno por aire con vehículos aéreos, para rastreos, monitoreos de incendio, entre otros.

4.2. Discusión de Resultados y Conclusión Final

Definiendo propiamente todos los componentes del subsistema de guía y diseñando con el paradigma *top-down*, se consigue un sistema de guía de múltiples drones que permite encontrar en cada uno de los vehículos los puntos a visitar que maximicen la obtención de parámetros, sin la necesidad de una central de procesamiento, es decir de forma descentralizada. Además, el sistema de comunicación descentralizado permite usar tecnologías de comunicación destinadas a “Internet de las Cosas”, que es un concepto propio de los sistemas de enjambre.

La arquitectura de toma de decisiones basa sus elecciones en formulas y ecuaciones matemáticas que tienen en cuenta el dinamismo de la información, por más que la lectura de estas informaciones dinámicas esté dada mediante la recepción de señales. Cuando el estado es parcialmente observable, se usa la MCTS para seleccionar la mejor acción de entre algunas posibles y/o predefinidas. Una vez que se elijan estos puntos, la planificación local se encarga de encontrar las rutas libres de obstáculos en línea.

El uso del software de **APMRover2** permitió simular un vehículo autónomo con ecuaciones de movimiento que corresponden al movimiento de vehículos de superficie reales, por lo que las pruebas realizadas usando vehículos de **APMRover2** simulados sirven para determinar si el sistema y los componentes propuestos son los indicados respecto a varios factores y métricas que se definen según la necesidad de la misión específica.

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

Por supuesto que la elección de los componentes se debe basar en la necesidad de la misión, por lo que se desarrollaron y propusieron funciones, factores, definiciones y ecuaciones que determinen rendimientos de los planificadores locales de ruta utilizado como componente para la toma de decisiones, y luego como componente para determinar la velocidad deseada.

Por último, se debió de elegir el sistema embebido adecuado, que permita la planificación local y global de ruta, definición de la misión a grandes rasgos, la comunicación y la cooperación descentralizada. Para esto, los esfuerzos se concentraron en obtener informaciones sobre algunos sistemas y realizar pruebas de rendimiento en ellos. Los requerimientos definidos permiten seleccionar, según el entorno y las necesidades, el mejor sistema embebido.

4.3. Evaluación y seguimiento del impacto

Este TFG presenta un diseño de sistemas de guía para múltiples drones, el cuál es adaptable a muchas situaciones particulares, siempre y cuando la misión sea la de observar o monitorear un entorno. Se desarrolló el sistema con la idea de implementarlos en drones acuáticos, pero la generalización del problema, permite utilizar el sistema en cualquier tipo de dron, siempre y cuando los rendimientos de los componentes entreguen aproximadamente los mismos resultados. Asimismo, se propone un nuevo modelo o técnica de planificación de rutas para mapas preestablecidos, el *uFMS*, el cual presenta una ventaja con respecto a muchos planificadores locales de ruta actuales, a excepción del A^* , para esta aplicación en particular.

Con la implementación del sistema de guía propuesto, se puede lograr la misión de cubrir un gran porcentaje de área en muy poco tiempo, consiguiendo informaciones que pueden ser consideradas dinámicas. Además, el sistema descentralizado de cálculo de regiones aprovecha las herramientas disponibles para mejorar la productividad y optimizar los tiempos de cálculo, especialmente

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

a medida que se introducen numerosos agentes o vehículos en el sistema.

El trabajo aporta un sistema de determinación de rendimientos de planificadores de ruta. Además aporta diseños y resultados a la línea de investigación sistemas multirobot, a través de implementaciones de POMDP utilizando como solucionador al MCTS. Para modificar la implementación para trabajar con incertidumbre respecto a las informaciones recibidas, se puede expandir el POMDP a BO-PODMP, o POMDP optimizado de forma bayesiana. Esto permitiría, trabajar con datos capturados en tiempo real, como temperatura o parámetros de la calidad del agua del Lago.

Este sistema de guía propuesto puede ser implementado en un despliegue real de drones en el marco del Proyecto 2018/ACDE/000773 “Sistema de monitorización de agentes contaminantes en el lago Ypacaraí mediante el uso de Vehículos Acuáticos no Tripulados de Superficie” financiada por la Agencia Española de Cooperación Internacional para el Desarrollo (AECID).

De implementarse el método en sistemas reales, el resultado de la lectura y obtención de parámetros como temperatura, óxido-disuelto, potencial de Hidrógeno y otros permitirá establecer acciones a corto plazo para reducir y mitigar la contaminación del Lago Ypacaraí, convirtiéndolo de nuevo en un punto turístico importante en la sociedad, mejorando la calidad de agua de los efluentes del Lago y la calidad de vida de la fauna, flora y, por sobre todo, de las personas que habitan dentro de la cuenca del Lago Ypacaraí.

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

BIBLIOGRAFÍA

- [1] J. Ferber y G. Weiss, *Multi-agent systems: an introduction to distributed artificial intelligence*. Addison-Wesley Reading, 1999, vol. 1.
- [2] J. A. Shapiro, "Thinking about bacterial populations as multicellular organisms," *Annual review of microbiology*, vol. 52, no. 1, pp. 81–104, 1998.
- [3] M. Duarte, V. Costa, J. Gomes, T. Rodrigues, F. Silva, S. M. Oliveira, y A. L. Christensen, "Evolution of collective behaviors for a real swarm of aquatic surface robots," *PloS one*, vol. 11, no. 3, p. e0151834, 2016.
- [4] C. R. Kube y H. Zhang, "Collective robotics: From social insects to robots," *Adaptive behavior*, vol. 2, no. 2, pp. 189–218, 1993.
- [5] J.-L. Deneubourg, S. Goss, N. Franks, A. Sendova-Franks, C. Detrain, y L. Chrétien, "The dynamics of collective sorting robot-like ants and ant-like robots," in *Proceedings of the first international conference on simulation of adaptive behavior on From animals to animats*, 1991, pp. 356–363.
- [6] L. Steels, "Cooperation between distributed agents through self-organisation," in *EEE International Workshop on Intelligent Robots and Systems, Towards a New Frontier of Applications*. IEEE, 1990, pp. 8–14.
- [7] C. W. Reynolds, *Flocks, herds and schools: A distributed behavioral model*. ACM, 1987, vol. 21, no. 4.
- [8] E. Şahin, "Swarm robotics: From sources of inspiration to domains of application," in *International workshop on swarm robotics*. Springer, 2004, pp. 10–20.

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

- [9] Y. Diaz-Mercado, S. G. Lee, y M. Egerstedt, "Distributed dynamic density coverage for human-swarm interactions," in *2015 American Control Conference (ACC)*. IEEE, 2015, pp. 353–358.
- [10] S. Hauri, J. Alonso-Mora, A. Breitenmoser, R. Siegwart, y P. Beardsley, "Multi-robot formation control via a real-time drawing interface," in *Field and service robotics*. Springer, 2014, pp. 175–189.
- [11] M. Arzamendia, D. Gregor, D. G. Reina, y S. L. Toral, "An evolutionary approach to constrained path planning of an autonomous surface vehicle for maximizing the covered area of ypacarai lake," *Soft Computing*, vol. 23, no. 5, pp. 1723–1734, 2019.
- [12] M. Arzamendia, D. Gregor, D. Reina, y S. Toral, "A path planning approach of an autonomous surface vehicle for water quality monitoring using evolutionary computation," in *Technology for Smart Futures*. Springer, 2018, pp. 55–73.
- [13] G. López Moreira M, L. Hinegk, A. Salvadore, G. Zolezzi, F. Hölker, R. Monte Domecq S, M. Bocci, S. Carrer, L. De Nat, J. Escribá *et al.*, "Eutrophication, research and management history of the shallow ypacaraí lake (paraguay)," *Sustainability*, vol. 10, no. 7, p. 2426, 2018.
- [14] H. I. C. (CHI) y I. Hydroelectric. Sistema de monitoreo, control y estudios de la cuenca del lago ypacarai. [En línea]. Disponible: <http://hidroinformatica.itaipu.gov.py/gestiondecuenca/py/ypacarai/> [Fecha de consulta: noviembre 2019]
- [15] ABCcolor. El lago ypacaraí de nuevo es invadido por cianobacterias. [En línea]. Disponible: <https://www.abc.com.py/edicion-impresia/interior/el-lago-ypacarai-de-nuevo-es-invadido-por-cianobacterias-1785501.html> [Fecha de consulta: octubre 2019]

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

- [16] A. L. Christensen, S. Oliveira, O. Postolache, M. J. De Oliveira, S. Sargento, P. Santana, L. Nunes, F. J. Velez, P. Sebastião, V. Costa *et al.*, “Design of communication and control for swarms of aquatic surface drones.” in *ICAART (2)*, 2015, pp. 548–555.
- [17] R. Song, Y. Liu, J. Balbuena, F. Cuellar, y R. Bucknall, “Developing an energy effective autonomous usv for undertaking missions at the highlands of peru,” in *2018 OCEANS-MTS/IEEE Kobe Techno-Oceans (OTO)*. IEEE, 2018, pp. 1–7.
- [18] V. Bertram, “Unmanned surface vehicles-a survey,” *Skibsteknisk Selskab, Copenhagen, Denmark*, vol. 1, pp. 1–14, 2008.
- [19] G. Clauss, A. Kauffeldt, N. Otten *et al.*, “Agapas: Autonomous galileo-supported rescue vessel for persons overboard,” in *ASME 2009 28th International Conference on Ocean, Offshore and Arctic Engineering*. American Society of Mechanical Engineers Digital Collection, 2009, pp. 267–276.
- [20] M. Dunbabin, A. Grinham, y J. Udy, “An autonomous surface vehicle for water quality monitoring,” in *Australasian conference on robotics and automation (ACRA)*. Citeseer, 2009, pp. 2–4.
- [21] R. T. B.V. Wasteshark. [En línea]. Disponible: <https://www.ranmarine.io/wasteshark> [Fecha de consulta: octubre 2019]
- [22] Z. Liu, Y. Zhang, X. Yu, y C. Yuan, “Unmanned surface vehicles: An overview of developments and challenges,” *Annual Reviews in Control*, vol. 41, pp. 71–93, 2016.
- [23] J. Balbuena, D. Quiroz, R. Song, R. Bucknall, y F. Cuellar, “Design and implementation of an usv for large bodies of fresh waters at the highlands of peru,” in *OCEANS 2017-Anchorage*. IEEE, 2017, pp. 1–8.

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

- [24] M. C. McFarland, "Using bottom-up design techniques in the synthesis of digital hardware from abstract behavioral descriptions," in *Proceedings of the 23rd ACM/IEEE Design Automation Conference*. IEEE Press, 1986, pp. 474–480.
- [25] M. S. Grewal, L. R. Weill, y A. P. Andrews, *Global positioning systems, inertial navigation, and integration*. John Wiley & Sons, 2007.
- [26] A. West, F. Arvin, H. Martin, S. Watson, y B. Lennox, "Ros integration for miniature mobile robots," in *Lecture Notes in Computer Science*, 07 2018.
- [27] K. Ogata, *Modern Control Engineering*, 4th ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2001.
- [28] O. Souissi, R. Benatitallah, D. Duvivier, A. Artiba, N. Belanger, y P. Feyzeau, "Path planning: A 2013 survey," in *Proceedings of 2013 International Conference on Industrial Engineering and Systems Management (IESM)*. IEEE, 2013, pp. 1–8.
- [29] F. Peralta, "Vehículos autónomos de superficie: Estudio y comparación de planificadores locales de ruta," *AUGM Mendoza*, 2018.
- [30] S.-G. Cui, H. Wang, y L. Yang, "A simulation study of a-star algorithm for robot path planning," in *16th international conference on mechatronics technology, PP*, 2012, pp. 506–510.
- [31] T. H. Cormen, "Section 24.3: Dijkstra's algorithm," *Introduction to Algorithms*, pp. 595–601, 2001.
- [32] J. Barraquand, B. Langlois, y J.-C. Latombe, "Numerical potential field techniques for robot path planning," *IEEE transactions on systems, man, and cybernetics*, vol. 22, no. 2, pp. 224–241, 1992.

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

- [33] O. Adiyatov y H. A. Varol, "A novel rrt*-based algorithm for motion planning in dynamic environments," in *Mechatronics and Automation (ICMA), 2017 IEEE International Conference on.* IEEE, 2017, pp. 1416–1421.
- [34] Y. Liu, R. Bucknall, y X. Zhang, "The fast marching method based intelligent navigation of an unmanned surface vehicle," *Ocean Engineering*, vol. 142, pp. 363–376, 2017.
- [35] C. Blum y R. Gross, *Swarm Intelligence in Optimization and Robotics.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 1291–1309.
- [36] A. Sakai, D. Ingram, J. Dinius, K. Chawla, A. Raffin, y A. Paques, "Python-robotics: a python code collection of robotics algorithms," *arXiv preprint arXiv:1808.10703*, 2018.
- [37] C. White, *Markov decision processes.* Springer, 2001.
- [38] D. Silver y J. Veness, "Monte-carlo planning in large pomdps," in *Advances in neural information processing systems*, 2010, pp. 2164–2172.
- [39] P. H. Au, "Dynamic information disclosure," *The RAND Journal of Economics*, vol. 46, no. 4, pp. 791–823, 2015.
- [40] D. I. Inc. Digi xbee 3 zigbee 3. [En línea]. Disponible: <https://www.digi.com/products/embedded-systems/digi-xbee/rf-modules/2-4-ghz-modules/xbee3-zigbee-3> [Fecha de consulta: octubre 2019]
- [41] R. Faludi, *Building wireless sensor networks: with ZigBee, XBee, arduino, and processing.* .O'Reilly Media, Inc.", 2010.
- [42] L. ESPRESSIF SYSTEMS (SHANGHAI) CO. Esp8266ex. [En línea]. Disponible: <https://www.espressif.com/en/products/hardware/esp8266ex/overview> [Fecha de consulta: diciembre 2019]

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

- [43] A. Spiess. Esp8266-longterm-sensor-hourly. [En línea]. Disponible: <https://github.com/Sensorslot/ESP8266-Longterm-Sensor-Hourly> [Fecha de consulta: diciembre 2019]
- [44] AReResearch. Udp-ranger - a simple esp8266 range tester. [En línea]. Disponible: <http://www.areresearch.net/2016/03/udp-ranger-simple-esp8266-range-tester.html> [Fecha de consulta: diciembre 2019]
- [45] CNLohr. Esp8266 wifi range/distance tests. [En línea]. Disponible: <https://github.com/cnlohr/> [Fecha de consulta: diciembre 2019]
- [46] Libelium. 4g + gps shield for arduino and raspberry. [En línea]. Disponible: <https://www.cooking-hacks.com/documentation/tutorials/4g-gps-lte-wcdma-hspa-3g-gprs-shield-arduino-raspberry-pi-waspmote-tutorial/> [Fecha de consulta: diciembre 2019]
- [47] J. W. C. R. Website. Cell sizes. [En línea]. Disponible: <http://www.wirelesscommunication.nl/reference/chaptr04/cellplan/cellsize.htm> [Fecha de consulta: diciembre 2019]
- [48] G. Blog. An introduction to macrocells and small cells. [En línea]. Disponible: <https://www.slideshare.net/3G4GLtd/an-introduction-to-macrocells-small-cells> [Fecha de consulta: diciembre 2019]
- [49] M. Li, K. Lu, H. Zhu, M. Chen, S. Mao, y B. Prabhakaran, "Robot swarm communication networks: architectures, protocols, and applications," in *2008 Third International Conference on Communications and Networking in China*. IEEE, 2008, pp. 162–166.
- [50] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, y A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

- [51] A. Banks y R. Gupta. Mqtt version 3.1.1 plus errata 01. [En línea]. Disponible: <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html> [Fecha de consulta: noviembre 2019]
- [52] R. Axelrod y W. D. Hamilton, "The evolution of cooperation," *science*, vol. 211, no. 4489, pp. 1390–1396, 1981.
- [53] H. Ohtsuki, C. Hauert, E. Lieberman, y M. Nowak, "A simple rule for the evolution of cooperation on graphs and social networks," *Nature*, vol. 441, pp. 502–5, 05 2006.
- [54] B. Yang, Y. Ding, y K. Hao, "Area coverage searching for swarm robots using dynamic voronoi-based method," in *2015 34th Chinese Control Conference (CCC)*. IEEE, 2015, pp. 6090–6094.
- [55] Y. Liu, R. Song, R. Bucknall, y X. Zhang, "Intelligent multi-task allocation and planning for multiple unmanned surface vehicles (usvs) using self-organising maps and fast marching method," *Information Sciences*, vol. 496, pp. 180–197, 2019.
- [56] D. Brandtner y M. Saska, "Coherent swarming of unmanned micro aerial vehicles with minimum computational and communication requirements," in *2017 European Conference on Mobile Robots (ECMR)*. IEEE, 2017, pp. 1–6.
- [57] F. Aurenhammer y R. Klein, "Voronoi diagrams," *Handbook of computational geometry*, vol. 5, no. 10, pp. 201–290, 2000.
- [58] J. Cortes, S. Martinez, T. Karatas, y F. Bullo, "Coverage control for mobile sensing networks," *IEEE Transactions on robotics and Automation*, vol. 20, no. 2, pp. 243–255, 2004.
- [59] P. Marwedel, *Embedded system design*, 3rd ed. Springer, 2006, vol. 1.

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

- [60] R. P. Foundation. Raspberry pi 3 model b. [En línea]. Disponible: www.raspberrypi.org/products/raspberry-pi-3-model-b/ [Fecha de consulta: junio 2019]
- [61] I. Qualcomm Technologies. Dragonboard 410c development board. [En línea]. Disponible: <https://developer.qualcomm.com/hardware/dragonboard-410c> [Fecha de consulta: junio 2019]
- [62] N. Corporation. Harness ai at the edge with the jetson tx2 developer kit. [En línea]. Disponible: developer.nvidia.com/embedded/jetson-tx2-developer-kit [Fecha de consulta: junio 2019]
- [63] F. S. Foundation. Linux and the gnu system. [En línea]. Disponible: <https://www.gnu.org/gnu/linux-and-gnu.en.html> [Fecha de consulta: junio 2019]
- [64] P. S. Foundation. Python. [En línea]. Disponible: <https://www.python.org/> [Fecha de consulta: junio 2019]
- [65] F. Peralta. Cormoran simulator. an asv simulator for python. [En línea]. Disponible: <https://bitbucket.org/FedePeralta/cormoran-simulator/src/master> [Fecha de consulta: noviembre 2019]
- [66] A. Krogh. Snazzy maps. [En línea]. Disponible: <https://snazzymaps.com/> [Fecha de consulta: octubre 2019]
- [67] B. A. Francis y M. Maggiore, "Models of mobile robots in the plane," in *Flocking and Rendezvous in Distributed Robotics*. Springer, 2016, pp. 7–23.
- [68] G. Deng y L. Cahill, "An adaptive gaussian filter for noise reduction and edge detection," in *1993 IEEE Conference Record Nuclear Science Symposium and Medical Imaging Conference*. IEEE, 1993, pp. 1615–1619.

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

- [69] A. D. Team. Rover. [En línea]. Disponible: <http://ardupilot.org/rover/> [Fecha de consulta: noviembre 2019]
- [70] I. 3D Robotics. Dronekit. developer tool for drones. [En línea]. Disponible: <https://dronekit.io/> [Fecha de consulta: noviembre 2019]
- [71] L. Meier. Pixhawk. [En línea]. Disponible: <https://pixhawk.org/> [Fecha de consulta: noviembre 2019]
- [72] T. Q. Company. Get qt. [En línea]. Disponible: <https://www.qt.io/download> [Fecha de consulta: diciembre 2019]
- [73] nPerf. Mapa de cobertura 3g / 4g / 5g, paraguay. [En línea]. Disponible: <https://www.nperf.com/es/map/PY/-/-/signal/> [Fecha de consulta: diciembre 2019]

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

Anexo A: Tecnologías que utilizan Zigbee disponibles

Tabla 1 :Tecnologías disponibles que utilizan el protocolo ZigBee.

Manufacturera	URL	Componentes
Atmel	www.atmel.com	ICs, módulos, kits de des.
Cal. East. Lab.	www.cel.com	ICs, módulos
Digi Int.	www.digi.com	módulos, kits de des.
Freescale	www.freescale.com	ICs, kits de des.
Ember	www.ember.com	ICs, kits y entornos de des.
Microchip	www.microchip.com	módulos
Panasonic	www.panasonic.com	módulos
Telit	www.telit.com	módulos, entornos de des., USB
Texas Instruments	www.ti.com	ICs, kits y entornos de des.

Fuente: Elaboración propia.

Además de estos módulos también se puede simular la tecnología ZigBee para interconectar dispositivos. No obstante, algunos de los módulos presentados en la tabla implementan protocolos propietarios, es decir, disponibles exclusivamente para ellos.

Anexo B: Propiedad del Voronoi

Una de las propiedades de los Polígonos de Thiessen o Diagramas de Voronoi será enunciada, se asume que no puede haber cuatro puntos del conjunto original P en posición cocircular.

Propiedad 1. Sea P un conjunto de n sitios en el plano. Si todos los sitios son colineales, entonces $Vor(P)$ consiste de $n - 1$ líneas paralelas. De otra forma, $Vor(p)$ es conexo y sus aristas podrían ser segmentos o semi-líneas.

Que el diagrama sea conexo significa que existen vértices asignados a regiones de los generadores. Dando por hecho que existe una arista que una las regiones de los generadores más alejados entre si, por lo que, se puede concluir que, siempre y cuando los puntos generadores del Diagrama de Voronoi no sean colineales y tampoco corciculares, estos tendrán por lo menos 2 vecinos. Además, la librería provista para encontrar las regiones de Voronoi, realiza una triangulación Delaunay, que se realiza buscando crear triángulos con los generadores tal que ninguno de los generadores quede dentro de circunscripción de los triángulos generados. Con esta triangulación, toda arista o todo lado de un triángulo representa a vecinos adyacentes, y para generar por lo menos un triángulo circunscrito a generadores, se necesita por lo menos 3 que el resultado no sea trivial. Concluyendo, es imposible generar la triangulación Delaunay con solo 2 generadores.

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

Anexo C: Programas en lenguaje Python

En las siguientes páginas se encuentran los scripts completos en lenguaje Python de los programas Principal, Secundario y Auxiliar. El primero corresponde al programa encargado de comunicar al sub-sistema de guía con los demás, manteniendo el orden de prioridades, detectando errores y situándose en la capa más externa, es decir de menor decisión crucial del sub-sistema.

El Secundario se encarga de obtener los datos provistos por el primer programa, además de establecer los puntos a visitar, las rutas planificadas, así como las trayectorias a ejecutar. Se encarga de obtener la velocidad deseada del sistema con base en todas las informaciones provistas a él.

El programa Auxiliar es un script que es capaz de resolver el PODMP para un nivel de profundidad y una cantidad de acciones finitas, esto lo hace a través de hacer crecer el árbol e MCTS y luego se obtiene la mejor ruta a seguir según los parámetros pre-establecidos.

Programa 1: Principal. *Cormoran Auto*. Elaboración Propia.

```
1 import argparse
2 import json
3 import logging
4 import os
5 import threading
6 from time import sleep
7 import paho.mqtt.client as mqtt
8 import serial
9 from drone_v2b import Drone
10 parser = argparse.ArgumentParser(description='Cormoran Auto.')
11 parser.add_argument('--mqtt', dest='mqttarg', default="192.168.20.22,1883,60", help='127.0.0.1,1883,60')
12 parser.add_argument('--id', dest='idarg', default="-1", help='if id > -1 idarg > dronearg')
13 parser.add_argument('--drone', dest='dronearg', default='tcp:127.0.0.1:5760,0,1,Ypacarai',
14                     help='tcp:127.0.0.1:5760,3,0,Ypacarai')
15 parser.add_argument('--serial', dest='serialarg', default='COM13', help='COM13')
16 args = parser.parse_args()
17 mqtt_args = args.mqttarg.split(',')
18 for i in range(len(mqtt_args)):
19     try:
20         mqtt_args[i] = int(mqtt_args[i])
21     except ValueError:
22         pass
23 drone_args = args.dronearg.split(',')
24 for i in range(len(drone_args)):
25     try:
```

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

```
26     drone_args[i] = int(drone_args[i])
27 except ValueError:
28     pass
29 serial_args = args.serialarg
30 try:
31     id_arg = int(args.idarg)
32     if id_arg >= 0:
33         drone_args = ['tcp:127.0.0.1:{}'.format(5760 + 10 * id_arg), id_arg, 1, 'Ypacarai']
34 except ValueError:
35     pass
36 print(drone_args)
37 logging.basicConfig(filename='database_{}.log'.format(drone_args[1]), level=logging.INFO)
38 def calc_voronoi():
39     if not _running:
40         return
41     if len(database.close_agents) < 2:
42         if database.sensing_radius >= database.max_sensing_radius:
43             database.public_state = 3
44             return
45         else:
46             database.sensing_radius *= 2.0
47             client.publish("cooperation/requests",
48                             json.dumps({
49                                 "req_type": "position",
50                                 "id": database.drone_id,
51                                 "position": database.obtain_position("region"),
52                                 "radius": database.sensing_radius
53                             }))
54             print('calling again in 5 sec')
55             threading.Timer(5, calc_voronoi).start()
56             database.public_state = 1
57     else:
58         if database.create_voronoi_region():
59             database.sensing_radius *= 2.0
60             if database.sensing_radius >= database.max_sensing_radius:
61                 database.public_state = 3
62                 return
63             client.publish("cooperation/requests",
64                             json.dumps({
65                                 "req_type": "position",
66                                 "id": database.drone_id,
67                                 "position": database.obtain_position("region"),
68                                 "radius": database.sensing_radius
69                             }))
70             print('Distance', database.sensing_radius, ' is shorter, calling again in 5 sec')
71             threading.Timer(5, calc_voronoi).start()
72             database.public_state = 1
73         else:
74             database.public_state = 3
75
76
77 def publish_whenever_whatever(topic, message):
78     client.publish(topic, message)
79     logging.info("PUB: {}".format(message))
80
81
82 def on_connect(_client, _, __, rc):
```

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

```

83     global _running
84     _running = True
85     print("Connected with result code " + str(rc))
86     _client.subscribe("cooperation/requests")
87     _client.subscribe("cooperation/id" + str(database.drone_id))
88     str_con2 = str(database)
89     _client.publish("drone", str_con2)
90     logging.info("PUB: {}".format(str_con2))
91
92
93 def handle_mqtt_message(_client, _, msg):
94     try:
95         if "cooperation" in msg.topic:
96             data_received = json.loads(str(msg.payload, 'utf-8'))
97             if "requests" in msg.topic:
98                 if data_received["req_type"] == "position":
99                     if data_received["id"] != database.drone_id and data_received[
100                         "radius"] >= database.get_distance_metres(
101                             database.home_loc,
102                             aux_lat=float(data_received["position"]["lat"]),
103                             aux_lon=float(data_received["position"]["lon"])
104                         ):
105                         my_thread = threading.Thread(target=publish Whenever Whatever,
106                                                         args=("cooperation/id" + str(data_received["id"]),
107                                                             json.dumps(
108                                                                 {
109                                                                     "id": database.drone_id,
110                                                                     "position": database.obtain_position("region")
111                                                                 },
112                                                             )))
113                         my_thread.start()
114                     elif "id" + str(database.drone_id) in msg.topic:
115                         if database.add_agent(data_received["id"], data_received["position"]):
116                             return
117                         else:
118                             database.close_agents[data_received["id"]] = data_received["position"]
119                     except KeyError as e:
120                         print("data not understood: ", e)
121
122 def on_message(_client, user_data, msg):
123     msg_thread = threading.Thread(target=handle_mqtt_message, args=(_client, user_data, msg,))
124     msg_thread.start()
125     msg_thread.join()
126
127
128 def on_disconnect(_client, _, rc=0):
129     print("Disconnected result code " + str(rc))
130     _client.loop_stop()
131
132
133 def mqtt_thread(con_string, port, timeout):
134     global client
135     try:
136         client = mqtt.Client("id_{}".format(database.drone_id))
137         client.on_connect = on_connect
138         client.on_message = on_message
139         client.on_disconnect = on_disconnect

```

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

```

140     client.connect(con_string, port, timeout)
141     client.loop_forever()
142 except ConnectionRefusedError as e:
143     print("Could not connect to MQTT broker: ", e)
144     logging.error(e)
145     client = None
146
147
148 def serial_thread(con_string):
149     try:
150         with serial.Serial(con_string) as ser:
151             while not _running:
152                 pass
153             while _running:
154                 global serial_procedure, data
155                 if serial_procedure == 1:
156                     msg = 'B' + str(database.depth) + '\n'
157                     ser.write(msg.encode('utf-8'))
158                     ser.flush()
159                     echo = ser.readline().decode('utf-8')
160                     if echo == msg:
161                         print('sensing')
162                         serial_procedure = 2
163                 elif serial_procedure == 2:
164                     resp = json.loads(ser.readline().decode('utf-8'))
165                     if resp["result"] == "OK":
166                         data = resp["read"]
167                         serial_procedure = 3
168                     else:
169                         print(resp)
170                 else:
171                     sleep(5)
172             ser.close()
173     except serial.SerialException as e:
174         print("Could not open serial: ", e)
175         logging.error(e)
176
177
178 def serial_workaround():
179     global serial_procedure, data
180     serial_procedure = 2
181     sleep(2.5)
182     serial_procedure = 3
183     data = dict()
184     data["temp"] = 1
185
186
187 if __name__ == "__main__":
188     data = dict()
189     data["temp"] = 1
190     database = Drone(*drone_args)
191     _running = False
192     base_dir = os.path.dirname(os.path.abspath(__file__))
193     _mqtt_thread = threading.Thread(target=_mqtt_thread, args=(*_mqtt_args,))
194     _mqtt_thread.start()
195     serial_procedure = 0 # 0 = Wait, 1 = send start, 2 = wait finish, 3 = finished
196     _serial_thread = threading.Thread(target=_serial_thread, args=(*_serial_args,))

```

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

```

197     _serial_thread.start()
198     str_con = ""
199     try:
200         database.connect_and_arm()
201         while not database.vehicle.armed or not _running:
202             pass
203         while _running:
204             if database.private_state == 5:
205                 break
206             str_con = str(database)
207             print(str_con)
208             if client is not None:
209                 client.publish("drone", str_con)
210             logging.info("PUB: {}".format(str_con))
211             if database.public_state == 0:
212                 client.publish("cooperation/requests",
213                               json.dumps({
214                                   "req_type": "position",
215                                   "id": database.drone_id,
216                                   "position": database.obtain_position("region"),
217                                   "radius": database.sensing_radius
218                               }))
219             database.public_state = 1
220             threading.Timer(7, calc_voronoi).start()
221             print('calling calc in 7')
222             if database.private_state == 1:
223                 if serial_procedure == 0:
224                     print('drone sensing')
225                     serial_procedure = 1 # 1 = send start
226                     if not _serial_thread.is_alive():
227                         print(database.formal_gps2pix(database.vehicle.location.global_relative_frame))
228                         print("Serial is not available, stopping for 3 seconds")
229                         threading.Timer(3, serial_workaround).start()
230                     elif serial_procedure == 2:
231                         database.private_state = 2
232                     if database.private_state == 2:
233                         if serial_procedure == 3:
234                             database.update_measures(data)
235                             database.private_state = 3
236                             serial_procedure = 0
237                             sleep(0.1)
238                             database.update_measures(data)
239                             sleep(1)
240             except KeyboardInterrupt:
241                 print('KeyboardInterrupt')
242             database.vehicle.disarm()
243             database.vehicle.close()
244             str_con = str(database)
245             client.publish("drone", str_con)
246             logging.info("PUB: {}".format(str_con))
247             client.disconnect()
248             _running = False
249             database.generate_images('map/Ypacarai/ypacarai_final.yaml')
250             database.save_parameters('map/Ypacarai/ypacarai_final.yaml')
251             _mqtt_thread.join()
252             _serial_thread.join()

```

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

Programa 2: Secundario. *Drone*. Elaboración Propia.

```

1 import json
2 import logging
3 import os
4 import threading
5 import time
6 from time import sleep
7 import matplotlib.image as img
8 import numpy as np
9 import pymavlink.mavutil as mavutil
10 import yaml
11 from dronekit import connect, VehicleMode, LocationGlobalRelative
12 from mdccormoran.path_planners.path_planners import a_star
13 from scipy.spatial import Voronoi
14 from shapely.geometry import Polygon
15 from MCTS_region import MCTS
16
17
18 class Drone(object):
19     map_data: None
20     SLEEP_SEND_V = 0.25
21     SLEEP_CONNECT = 1.5
22     DEG2RAD = np.pi / 180
23     TILE_SIZE = 10.3333
24     FOUR_PI = (4 * np.pi)
25     SCALE = 1
26     screen_x0 = -57.38106014642857
27     screen_xscale = 1.030035714285734E-4
28     screen_y0 = -25.38342902978723
29     screen_yscale = 9.33042553191497E-5
30
31     def __init__(self, connection_string='tcp:127.0.0.1:5760', drone_id=3, energy_left=1, location='Ypacarai'):
32         self.drone_id = drone_id
33         self.depth = 100
34         self.energy_left = energy_left
35         self.connection_string = connection_string
36         self.vehicle = connect(connection_string, wait_ready=True)
37         self.desired_velocity = np.zeros((1, 2))
38         self.region_center = LocationGlobalRelative(lat=0, lon=0, alt=0)
39         self.close_agents = dict()
40         self.vor = None
41         self.region = None
42         self.region = []
43         self.waypoints = []
44         self.local_goal = LocationGlobalRelative(lat=0, lon=0)
45         # states: 0 = should update region, 1 = updating region, 2 = generating region, 3 = covering region
46         self.public_state = 0
47         self.sensing_radius = 1000
48         # states: 0 = driving, 1 = stopping, 2 = sensing, 3 = ready to drive
49         self.private_state = 0
50         self.number_of_updates = 0
51         self.max_updates = 15
52         self.parameter_maps = dict()
53         logging.log(logging.INFO, "initializing done")
54         if location != 'Ypacarai':
55             if location == 'Ciclovia':
56                 self.general_goals = []

```

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

```

57         self.obtain_map_data('map/Ciclovia/lago_ciclovia.yaml')
58         self.screen_xscale = 6.71875000024524e-07
59         self.screen_x0 = -57.5144784218750
60         self.screen_yscale = 5.83333333340856e-07
61         self.screen_y0 = -25.3391253333333
62         elif location == 'Citec':
63             self.max_updates = 0
64             self.general_goals = np.array([self.formal_gps2pix(
65                 self.get_location_metres(self.vehicle.location.global_relative_frame, 0, -17))])
66             self.obtain_map_data('map/Citec/citec.yaml')
67             self.screen_xscale = 6.209910629539905e-07
68             self.screen_x0 = -57.49157589195666
69             self.screen_yscale = 5.633358387369114e-07
70             self.screen_y0 = -25.295766333583874
71         else:
72             self.general_goals = []
73             print('Obtaining Maps...')
74             self.obtain_map_data('map/Ypacarai/ypacarai_final.yaml')
75             self.current_goal = 0
76             self.threads = []
77             self.max_sensing_radius = min(self.map_data.shape) * self.TILE_SIZE
78             print("-----", self.max_sensing_radius)
79             self.home_loc = self.vehicle.location.global_relative_frame
80
81     def connect_and_arm(self):
82         print('basic pre-arm checks')
83         while not self.vehicle.is_armable:
84             print(" waiting to be armable..")
85             sleep(self.SLEEP_CONNECT)
86             print("Arming motors")
87             self.vehicle.mode = VehicleMode("GUIDED")
88             self.vehicle.armed = True
89             while not self.vehicle.armed:
90                 sleep(self.SLEEP_CONNECT)
91             print("Ready!")
92             self.vehicle.wait_ready('autopilot_version')
93             self.threads = threading.Thread(target=self.dronekit_comm)
94             self.threads.start()
95             self.region_center = self.get_location_metres(self.vehicle.location.global_relative_frame, 0, 0)
96
97     def get_location_metres(self, original_location, d_north, d_east):
98         """
99         http://gis.stackexchange.com/questions/2951/algorithm-for-offsetting-a-latitude-longitude-by-some-amount-of-meters
100         """
101         earth_radius_eq = 6378137.0 # Radius of earth anp.round eq
102         earth_radius_po = 6356752.0 # Radius of earth crossing poles
103         d_lat = d_north / earth_radius_po
104         d_lon = d_east / (earth_radius_eq * np.cos(self.DEG2RAD * original_location.lat))
105         newlat = original_location.lat + (d_lat / self.DEG2RAD)
106         newlon = original_location.lon + (d_lon / self.DEG2RAD)
107         return LocationGlobalRelative(newlat, newlon, original_location.alt)
108
109     def obtain_next_goal(self):
110         return np.round(self.general_goals[self.current_goal]).astype(np.int64)
111
112     @staticmethod

```

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

```

113 def get_distance_metres(start_location, goal_location=None, aux_lat=None, aux_lon=None):
114     """
115     https://github.com/diudrones/ardunpilot/blob/master/Tools/autotest/common.py
116     """
117     if goal_location is not None:
118         dlat = goal_location.lat - start_location.lat
119         dlong = goal_location.lon - start_location.lon
120     else:
121         dlat = aux_lat - start_location.lat
122         dlong = aux_lon - start_location.lon
123     return np.sqrt((dlat * dlat) + (dlong * dlong)) * 1.113195e5
124
125 def add_agent(self, agent_id, position):
126     if str(agent_id) not in self.close_agents.keys():
127         self.close_agents[str(agent_id)] = position
128         return True
129     return False
130
131 def set_velocity_body(self, vx, vy, vz):
132     msg = self.vehicle.message_factory.set_position_target_global_int_encode(
133         0, 0, 0, mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT_INT,
134         0b000011111000111, 0, 0, 0, vy, vx, -vz, 0, 0, 0, 0, 0)
135     self.vehicle.send_mavlink(msg)
136     self.vehicle.flush()
137
138 def obtain_position(self, is_type="vehicle"):
139     if is_type == "vehicle":
140         return {"lat": self.vehicle.location.global_relative_frame.lat,
141                 "lon": self.vehicle.location.global_relative_frame.lon,
142                 "alt": self.vehicle.location.global_relative_frame.alt,
143                 "yaw": self.vehicle.attitude.yaw}
144     elif is_type == "goal":
145         return {"lat": self.local_goal.lat,
146                 "lon": self.local_goal.lon,
147                 "alt": self.local_goal.alt,
148                 "yaw": 0.0}
149     elif is_type == "region":
150         return {"lat": self.region_center.lat,
151                 "lon": self.region_center.lon,
152                 "alt": self.region_center.alt,
153                 "yaw": 0.0}
154
155 def formal_gps2pix(self, latlng, is_dict=False):
156     if is_dict:
157         screen_y = (latlng["lat"] - self.screen_y0) / self.screen_yscale
158         screen_x = (latlng["lon"] - self.screen_x0) / self.screen_xscale
159     else:
160         screen_y = (latlng.lat - self.screen_y0) / self.screen_yscale
161         screen_x = (latlng.lon - self.screen_x0) / self.screen_xscale
162     return [np.round(screen_x).astype(np.int64), np.round(screen_y).astype(np.int64)]
163
164 def formal_pix2gps(self, screen_pos):
165     screen_x = screen_pos.pop(0)
166     screen_y = screen_pos.pop(0)
167     world_y = self.screen_y0 + self.screen_yscale * screen_y
168     world_x = self.screen_x0 + self.screen_xscale * screen_x
169     return LocationGlobalRelative(lat=world_y, lon=world_x, alt=self.vehicle.location.global_relative_frame

```

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

```

170         .alt)
171
172     def dronekit_comm(self):
173         while self.vehicle.armed:
174             if self.public_state == 3:
175                 if self.general_goals is not None and len(self.general_goals) == 0:
176                     self.general_goals = self.obtain_pomdp_goals_if_need()
177                 if self.general_goals is None:
178                     self.set_velocity_body(0.0, 0.0, 0.0)
179                     self.private_state = 5
180                     break
181                 self.waypoints = a_star(self.map_data,
182                                         self.formal_gps2pix(self.vehicle.location.global_relative_frame),
183                                         self.vehicle.attitude.yaw - np.pi / 2,
184                                         self.obtain_next_goal()).tolist()
185                 self.local_goal = self.formal_pix2gps(self.waypoints.pop(0))
186             else:
187                 if self.private_state == 0 or self.private_state == 3:
188                     if self.private_state > 0:
189                         self.private_state = 0
190                         self.set_desired_velocity()
191                         self.set_velocity_body(self.desired_velocity[0], self.desired_velocity[1], 0.0)
192                         sleep(self.SLEEP_SEND_V)
193                     elif self.private_state == 5:
194                         break
195                     else:
196                         self.set_velocity_body(0, 0, 0.0)
197                         sleep(self.SLEEP_SEND_V)
198                 else:
199                     self.set_velocity_body(0, 0, 0.0)
200                     sleep(self.SLEEP_CONNECT)
201
202     def set_desired_velocity(self):
203         dxi = np.array([self.local_goal.lon - self.vehicle.location.global_relative_frame.lon,
204                        self.local_goal.lat - self.vehicle.location.global_relative_frame.lat])
205         if np.sum(np.abs(np.round(dxi, 4))) == 0:
206             if len(self.waypoints) == 0:
207                 self.current_goal += 1
208             if self.current_goal < len(self.general_goals):
209                 if self.private_state == 0 and (np.array(self.vehicle.velocity) <= 0.1).all():
210                     self.private_state = 1
211                 elif self.private_state == 3:
212                     self.private_state = 0
213                 if self.vehicle.mode != VehicleMode("GUIDED"):
214                     self.vehicle.mode = VehicleMode("GUIDED")
215                 else:
216                     self.current_goal -= 1
217                     self.desired_velocity = np.zeros((1, 2))[0]
218                     return
219                 self.waypoints = a_star(self.map_data,
220                                         self.formal_gps2pix(self.vehicle.location.global_relative_frame),
221                                         self.vehicle.attitude.yaw - np.pi / 2,
222                                         self.obtain_next_goal()).tolist()
223             else:
224                 if self.private_state == 0 and (np.array(self.vehicle.velocity) <= 0.1).all():
225                     self.current_goal = 0
226                     self.private_state = 1

```

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

```

226         self.public_state = 0
227         self.general_goals = []
228         return
229     elif self.private_state == 3:
230         self.private_state = 0
231         self.current_goal = 0
232         self.waypoints = a_star(self.map_data,
233                                self.formal_gps2pix(self.vehicle.location.global_relative_frame
234                                                    self.vehicle.attitude.yaw - np.pi / 2,
235                                                    self.obtain_next_goal()).tolist())
236     else:
237         self.current_goal -= 1
238         self.desired_velocity = np.zeros((1, 2))[0]
239         return
240     self.local_goal = self.formal_pix2gps(self.waypoints.pop(0))
241     dxi = np.array([self.local_goal.lon - self.vehicle.location.global_relative_frame.lon,
242                   self.local_goal.lat - self.vehicle.location.global_relative_frame.lat])
243     norma = np.linalg.norm(dxi)
244     dxi = dxi / norma
245     if (np.abs(dxi) > 1.0).any():
246         dxi /= np.max(dxi)
247     self.desired_velocity = dxi * 2
248
249 def __str__(self):
250     if self.vehicle.armed:
251         return json.dumps(
252             {
253                 "type": "drone",
254                 "id": self.drone_id,
255                 "online": self.vehicle.armed,
256                 "position": self.obtain_position(),
257                 "goal": self.obtain_position("goal"),
258                 "velocity": self.vehicle.velocity,
259                 "energy_left": self.energy_left,
260                 "parameters": self.obtain_param_for_pos(),
261                 "pri_pu_states": [self.private_state, self.public_state],
262                 "timestamp": time.time()
263             }
264         )
265     else:
266         return json.dumps({
267             "type": "drone",
268             "id": self.drone_id,
269             "online": self.vehicle.armed,
270             "position": self.obtain_position(),
271             "goal": {"lat": 0,
272                     "lon": 0,
273                     "alt": 0,
274                     "yaw": 0},
275             "velocity": self.vehicle.velocity,
276             "energy_left": self.energy_left,
277             "parameters": self.obtain_param_for_pos(),
278             "pri_pu_states": [self.private_state, self.public_state],
279             "timestamp": time.time()
280         })
281

```

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

```

282     def obtain_map_data(self, map_yaml_name):
283         with open(map_yaml_name, 'r') as stream:
284             try:
285                 map_yaml = yaml.load(stream, Loader=yaml.FullLoader)
286                 map_data = img.imread(os.path.join(os.path.dirname(map_yaml_name), map_yaml.get('image')))
287                 if map_yaml.get('negate') == 0:
288                     map_data = np.flipud(map_data[:, :, 0])
289                     self.map_data = 1 - map_data
290                 else:
291                     self.map_data = np.flipud(map_data[:, :, 0])
292                 print('main map obtained')
293                 parameters_names = map_yaml.get('parameter')
294                 for param in parameters_names:
295                     print('{} map obtained'.format(param))
296                     self.parameter_maps[param] = np.loadtxt(map_yaml.get(param), delimiter=",")
297                     print(np.size(self.parameter_maps[param]))
298             except yaml.YAMLError:
299                 self.map_data = None
300
301     def obtain_pomdp_goals_if_need(self):
302         self.sensing_radius = 1000
303         if self.number_of_updates < self.max_updates:
304             print('obatinining new goals: ', self.number_of_updates)
305             simulated = MCTS(None, 0, {"last_update": self.parameter_maps['tmp_update_data'],
306                                     "energy_left": self.energy_left}, dict(),
307                                     {"position": np.append([self.formal_gps2pix(self.vehicle.location.
308                                     global_relative_frame)],
309                                     [self.vehicle.attitude.yaw]),
309                                     "map_data": self.map_data,
310                                     "region": self.obtain_shapely_polygon()})
311             simulated.grow_tree()
312             beacons, e_reward = simulated.back_prop()
313             print(beacons)
314             logging.log(logging.INFO, "DRONE: beacons: {}".format(beacons))
315             self.number_of_updates += 1
316             if e_reward < 10:
317                 self.number_of_updates = self.max_updates + 1
318                 print("reward {} is too low".format(e_reward))
319                 logging.log(logging.WARNING, "DRONE: reward: {}".format(e_reward))
320                 return None
321             del simulated
322             return beacons
323         else:
324             return None
325
326     def update_measures(self, data):
327         for key in data:
328             if "{}_data".format(key) in self.parameter_maps:
329                 self.parameter_maps["{}_data".format(key)][
330                     self.formal_gps2pix(self.vehicle.location.global_relative_frame)[1],
331                     self.formal_gps2pix(self.vehicle.location.global_relative_frame)[0]] = data[key]
332                 logging.log(logging.INFO, "DRONE: DATA {} for pos {}: {}. Time:{}".format(key, self.formal_gps2pix(
333                     self.vehicle.location.global_relative_frame), data[key], time.time()))
334             if len(data) > 1:
335                 print("last update is not present yet")
336             else:
337                 aux = self.formal_gps2pix(self.vehicle.location.global_relative_frame)

```

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

```

338         self.parameter_maps["tmp_update_data"][aux[i] - 1:aux[i] + 2,
339         aux[0] - 1:aux[0] + 2] = 1546311600 - time.time()
340
341     def save_parameters(self, map_yaml_name):
342         with open(map_yaml_name, 'r') as stream:
343             map_yaml = yaml.load(stream, Loader=yaml.FullLoader)
344             parameters_names = map_yaml.get('parameter')
345             for param in parameters_names:
346                 np.savetxt(map_yaml.get(param), self.parameter_maps[param], delimiter=",", fmt='%.3f')
347
348     def obtain_param_for_pos(self):
349         pos = self.formal_gps2pix(self.vehicle.location.global_relative_frame)
350         result = dict()
351         for key in self.parameter_maps:
352             result[key] = self.parameter_maps[key][pos[1], pos[0]]
353
354         return result
355
356     def generate_images(self, map_yaml_name):
357         with open(map_yaml_name, 'r') as stream:
358             map_yaml = yaml.load(stream, Loader=yaml.FullLoader)
359             parameters_names = map_yaml.get('parameter')
360             for param in parameters_names:
361                 if "cond" in map_yaml.get(param):
362                     vmin = 100
363                     vmax = 250
364                 elif "od" in map_yaml.get(param):
365                     vmin = 6
366                     vmax = 9
367                 elif "or" in map_yaml.get(param):
368                     vmin = -3
369                     vmax = 3
370                 elif "ph" in map_yaml.get(param):
371                     vmin = 0
372                     vmax = 10
373                 elif "temp" in map_yaml.get(param):
374                     vmin = 23
375                     vmax = 35
376                 else:
377                     vmin = None
378                     vmax = 0
379                 img.imsave("{}_id{}.tiff".format(map_yaml.get(param), self.drone_id),
380                 np.flipud(self.parameter_maps[param]), cmap='hot',
381                 vmin=vmin,
382                 vmax=vmax)
383
384     def calc_voronoi(self):
385         self.public_state = 2
386         vor_list = []
387         for generator in self.close_agents.items():
388             vor_list.append(self.formal_gps2pix(generator[1], True))
389         vor_list.append(self.formal_gps2pix(self.region_center))
390         bounding_box = [0, self.map_data.shape[1], 0, self.map_data.shape[0]]
391         points_left = np.copy(vor_list)
392         points_left[:, 0] = bounding_box[0] - (points_left[:, 0] - bounding_box[0])
393         points_right = np.copy(vor_list)
394         points_right[:, 0] = bounding_box[1] + (bounding_box[1] - points_right[:, 0])

```

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

```

395     points_down = np.copy(vor_list)
396     points_down[:, 1] = bounding_box[2] - (points_down[:, 1] - bounding_box[2])
397     points_up = np.copy(vor_list)
398     points_up[:, 1] = bounding_box[3] + (bounding_box[3] - points_up[:, 1])
399     points = np.append(vor_list,
400                       np.append(np.append(points_left,
401                                           points_right,
402                                           axis=0),
403                                np.append(points_down,
404                                           points_up,
405                                           axis=0),
406                                axis=0),
407                       axis=0)
408     vor = Voronoi(points)
409     region = -1
410     for i in range(len(vor.points)):
411         if (self.formal_gps2pix(self.region_center) == vor.points[i]).all():
412             region = i
413             break
414
415     return vor, vor.vertices[vor.regions[vor.point_region[region]]]
416
417 def create_voronoi_region(self):
418     self.vor, self.region = self.calc_voronoi()
419     should_grow = True
420     max_dist = 0
421     for num in self.region:
422         if np.linalg.norm(np.subtract(np.array([num[0], num[1]]), self.formal_gps2pix(
423             self.region_center))) > max_dist:
424             max_dist = np.linalg.norm(
425                 np.subtract(np.array([num[0], num[1]]), self.formal_gps2pix(self.region_center)))
426     if max_dist * self.TILE_SIZE * 2 < self.sensing_radius:
427         should_grow = False
428     return should_grow
429 def obtain_shapely_polygon(self):
430     if len(self.region) == 0:
431         reg = Polygon(
432             [(0, 0), (0, np.size(self.map_data, 0)), (np.size(self.map_data, 1), np.size(self.map_data, 0))
433             ,
434              (np.size(self.map_data, 1), 0)])
435     else:
436         reg = Polygon(self.region)
437     logging.log(logging.INFO, "DRONE:reg is {}".format(reg))
438     return reg

```

Programa 3: Auxiliar. MCTS. Elaboración Propia.

```

1 import logging
2 import time
3 from copy import copy, deepcopy
4 import numpy as np
5 from mdcormoran.path_planners.path_planners import a_star
6 from shapely.geometry import Point
7
8
9 def simulate_random_action(state, parameter_maps, modif, current_reward, direction):
10     current_pos = copy(state["position"])

```

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

```

11     simulated_reward = copy(current_reward)
12     last_round_pos = np.round(current_pos[:2]).astype(np.int64)
13     map_data = state["map_data"]
14     modifiers = deepcopy(modif)
15     goal_pos = randomize_goal_position(current_pos, map_data, state["region"], direction)
16     if goal_pos is None:
17         return modifiers, current_pos, simulated_reward
18     if (last_round_pos == np.round(goal_pos[:2]).astype(np.int64)).all():
19         return modifiers, current_pos, simulated_reward - 100
20     path2follow = a_star(map_data, np.round(current_pos).astype(np.int64), current_pos[2],
21                          np.round(goal_pos).astype(np.int64)).tolist()
22     local_goal = path2follow.pop(0)
23     local_goal = np.ndarray.flatten(np.array(local_goal))
24     while True:
25         dxi = np.subtract(local_goal[:2], current_pos[:2])
26         if np.sum(np.abs(np.round(dxi))) < 2:
27             if len(path2follow) == 0:
28                 break
29             local_goal = path2follow.pop(0)
30             local_goal = np.ndarray.flatten(np.array(local_goal))
31             dxi = np.subtract(local_goal[:2], current_pos[:2])
32             norma = np.linalg.norm(dxi)
33             dxi = dxi / norma
34             dxu = np.zeros((2, 1))
35             dxu[0, :] = (np.cos(current_pos[2]) * dxi[0] + np.sin(current_pos[2]) * dxi[1])
36             dxu[1, :] = np.pi / 2 * np.arctan2(-np.sin(current_pos[2]) * dxi[0] + np.cos(current_pos[2]) * dxi[1],
37                                                dxu[0]) / (np.pi / 2)
38             xdist = 0.3 * np.cos(current_pos[2]) * dxu[0]
39             ydist = 0.3 * np.sin(current_pos[2]) * dxu[0]
40             current_pos[0] += xdist
41             current_pos[1] += ydist
42             current_pos[2] += 0.3 * dxu[1]
43             current_pos[2] = np.arctan2(np.sin(current_pos[2]), np.cos(current_pos[2]))
44             current_pos_round = np.round(current_pos[:2]).astype(np.int64)
45             if np.sum(current_pos_round - last_round_pos) != 0:
46                 last_round_pos = deepcopy(current_pos_round)
47                 obj = "{}, {}".format(last_round_pos[0], last_round_pos[1])
48                 if obj in modifiers:
49                     rew = np.exp(modifiers[obj] / 11381765)
50                 else:
51                     rew = np.exp((parameter_maps["last_update"][current_pos_round[1], current_pos_round[0]]) /
52                                11381765)
53                 if rew < 0.5:
54                     rew = -1
55                 simulated_reward += rew # / N + k * np.sqrt(2 * np.log(Np) / N)
56                 modifiers[obj] = 1560055093 - time.time()
57             return modifiers, current_pos, simulated_reward
58
59 def randomize_goal_position(c_pos, map_data, region, direction):
60     d = 500
61     aux = 250
62     while True:
63         if d < 10.0:
64             angle = np.random.randint(-1, 2)
65             g_pos = copy(c_pos)
66             g_pos[0] += aux * np.cos(g_pos[2] + angle * np.pi * 3 / 4)

```

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

```
67         g_pos[1] += aux * np.sin(g_pos[2] + angle * np.pi * 3 / 4)
68         g_pos[0] = np.clip(g_pos[0], 0, map_data.shape[1] - 1)
69         g_pos[1] = np.clip(g_pos[1], 0, map_data.shape[0] - 1)
70         if map_data[np.round(g_pos[1]).astype(np.int), np.round(g_pos[0]).astype(np.int)] == 0:
71             if region.contains(Point((g_pos[0], g_pos[1]))):
72                 logging.log(logging.INFO, "MCTS: strange gpos {} selected for cpos {}".format(g_pos, c_pos))
73     )
74     return g_pos
75     aux /= 2
76 else:
77     angle = direction - 2
78     g_pos = copy(c_pos)
79     g_pos[0] += d * np.cos(g_pos[2] + angle * np.pi / 4)
80     g_pos[1] += d * np.sin(g_pos[2] + angle * np.pi / 4)
81     g_pos[0] = np.clip(g_pos[0], 0, map_data.shape[1] - 1)
82     g_pos[1] = np.clip(g_pos[1], 0, map_data.shape[0] - 1)
83     if map_data[np.round(g_pos[1]).astype(np.int), np.round(g_pos[0]).astype(np.int)] == 0:
84         if region.contains(Point((g_pos[0], g_pos[1]))):
85             return g_pos
86     d /= 2
87
88 class MCTS:
89
90     def __init__(self, parent=None, depth=0, maps=None, modifiers=None, state=None, current_reward=0):
91         if modifiers is None:
92             modifiers = dict()
93         if state is None:
94             state = {"position": None, "map_data": None, "region": None}
95         self.parent = parent
96         self.children = [] # children of each node
97         self.depth = depth
98         self.parameter_maps = maps
99         self.reward = current_reward
100        self.state = state
101        self.modifiers = modifiers
102
103    def grow_tree(self, max_depht=3, max_leaves=5):
104        if self.depth < max_depht:
105            for k in range(max_leaves):
106                modifiers, final_pos, new_reward = simulate_random_action(self.state, self.parameter_maps,
107                                                                           self.modifiers,
108                                                                           self.reward, k)
109                leaf = MCTS(parent=self, depth=self.depth + 1, maps=self.parameter_maps, modifiers=modifiers,
110                           state={"position": final_pos, "map_data": self.state["map_data"],
111                                  "region": self.state["region"]},
112                           current_reward=new_reward)
113                self.children.append(leaf)
114                leaf.grow_tree()
115
116    def get_all_leaves(self):
117        if len(self.children) > 0:
118            all_l = []
119            for child in self.children:
120                ll = child.get_all_leaves()
121                if isinstance(ll, MCTS):
122                    all_l.append(ll)
```

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

```
123         else:
124             all_l.extend(l1)
125         return all_l
126     return self
127
128     def back_prop(self):
129         leaves = self.get_all_leaves()
130         selected = -1
131         e_reward = None
132         for leaf in leaves:
133             if e_reward is None or e_reward < leaf.reward:
134                 selected = leaf
135                 e_reward = leaf.reward
136         beacons = selected.state["position"]
137         beacons = np.expand_dims(beacons, 0)
138         while selected.parent is not None:
139             selected = selected.parent
140             if selected.parent is not None:
141                 beacons = np.concatenate([beacons, [selected.state["position"]]])
142         beacons = np.flipud(beacons)
143         print(e_reward)
144         if e_reward < 0:
145             print('warning, reward is less than 0')
146
147         return beacons, e_reward
```

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

Anexo D: Programa de Estación Base

Programa 4: Estación Base. Elaboración Propia

```

1 import json
2 import logging
3 import os
4 import sys
5 import threading
6
7 import paho.mqtt.client as mqtt
8 from PySide2.QtCore import QLineF, QRect, Slot, Signal, QObject, QPoint, QPointF
9 from PySide2.QtGui import QPen, QColor, QBrush, QPolygon, QPainter, QFont
10 from PySide2.QtWidgets import QApplication, QMainWindow, QGraphicsScene, QGraphicsItem, QGraphicsTextItem
11 from numpy import round, pi, int64
12
13 from ui_mainwindow import Ui_MainWindow
14
15 logging.basicConfig(filename='database.log', level=logging.INFO)
16
17 def obtaining_database():
18     global database, database_busy
19     database_busy = True
20     with open(base_dir + '/main-db.json', 'r') as f:
21         database = json.load(f)
22     database_busy = False
23
24
25 def updating_database():
26     global database, database_busy
27     database_busy = True
28     with open(base_dir + '/main-db.json', 'w') as f:
29         for i in range(len(database["clients"])):
30             database["clients"][i]["online"] = False
31         json.dump(database, f)
32     database_busy = False
33     database_busy = False
34
35
36 def on_connect(_client, _, __, rc):
37     print("Connected with result code " + str(rc))
38     _client.subscribe("drone")
39     _client.subscribe("database")
40
41
42 def handle_mqtt_message(_client, _, msg):
43     try:
44         if msg.topic == "drone":
45             message = str(msg.payload, 'utf-8')
46             logging.info("MSG: {}".format(message))
47             data_received = json.loads(message)
48             if "position" not in data_received or "goal" not in data_received or "online" not in data_received
49             or \
50                 "velocity" not in data_received or "energy_left" not in data_received:
51                 raise AssertionError
52             position = -1

```

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

```
52     for i in range(len(database["clients"])):
53         if database["clients"][i]["id"] == data_received["id"]:
54             position = i
55             break
56     if position == -1:
57         database["clients"].append(data_received)
58         print("adding new drone with id:", data_received["id"])
59         window.add_new_drone_id()
60     else:
61         database["clients"][position]["position"] = data_received["position"]
62         database["clients"][position]["goal"] = data_received["goal"]
63         database["clients"][position]["velocity"] = data_received["velocity"]
64         database["clients"][position]["energy_left"] = data_received["energy_left"]
65         database["clients"][position]["online"] = data_received["online"]
66         if window is not None:
67             window.update_data(position)
68     except KeyError as e:
69         print("data not understood: ", e)
70     except AssertionError as e:
71         print("data not valid: ", e)
72
73
74 def on_message(_client, user_data, msg):
75     msg_thread = threading.Thread(target=handle_mqtt_message, args=(_client, user_data, msg,))
76     msg_thread.start()
77     msg_thread.join()
78
79
80 def on_disconnect(_client, _, rc=0):
81     print("Disconnected result code " + str(rc))
82     _client.loop_stop()
83
84
85 def mqtt_thread():
86     global client
87     obtaining_database()
88     client = mqtt.Client("Basestation")
89     client.on_connect = on_connect
90     client.on_message = on_message
91     client.on_disconnect = on_disconnect
92     client.connect('192.168.20.22', 1883, 60)
93     _running = True
94     client.loop_forever()
95
96
97 class SignalSetPos(QObject):
98     signalSetPos = Signal(str)
99
100
101 class SignalAddNew(QObject):
102     signalAddNew = Signal(int)
103
104
105 class GoalView(QGraphicsTextItem):
106
107     def __init__(self, *args, **kwargs):
108         super().__init__(*args, **kwargs)
```

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

```

109     self.position = self.mapToParent(QPointF(0, 0))
110     self.blackPen = QPen(QColor(0, 0, 0))
111     self.paint_brush = QBrush(QColor(255, 0, 0))
112     self.font = QFont()
113     self.signalSetPos = SignalSetPos()
114     self.scaling_factor = 10
115     self.text = ""
116
117     def set_drone_id(self, string):
118         self.text = "G{}".format(string)
119
120     def paint(self, painter, option, widget=None):
121         painter.setPen(self.blackPen)
122         painter.setBrush(self.paint_brush)
123         self.font.setPixelSize(self.scaling_factor)
124         painter.setFont(self.font)
125         painter.scale(1, -1)
126         painter.drawText(QPoint(0, 0), self.text)
127
128
129     def boundingRect(self):
130         return QRect(0, 0, self.scaling_factor, self.scaling_factor)
131
132     def change_position(self, x, y, state):
133
134         if state == "ciclovía":
135             screen_xscale = 6.71875000024524e-07
136             screen_x0 = -57.5144784218750
137             screen_yscale = 5.83333333340856e-07
138             screen_y0 = -25.3391253333333
139         elif state == "ypacarai":
140             screen_x0 = -57.38106014642857
141             screen_xscale = 1.030035714285734E-4
142             screen_y0 = -25.38342902978723
143             screen_yscale = 9.33042553191497E-5
144         else:
145             screen_x0 = 0
146             screen_xscale = 0
147             screen_y0 = 0
148             screen_yscale = 0
149
150         screen_y = round((y - screen_y0) / screen_yscale).astype(int64)
151         screen_x = round((x - screen_x0) / screen_xscale).astype(int64)
152
153         self.position = QPointF(screen_x, screen_y)
154         self.signalSetPos.signalSetPos.emit('')
155
156     @Slot(str)
157     def do_change_pos(self):
158         self.setPos(self.position)
159
160
161 class DroneView(QGraphicsItem):
162     rd2dg = 180 / pi
163
164     def __init__(self, *args, **kwargs):
165         super().__init__(*args, **kwargs)

```

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

```

166         self.position = self.mapToParent(QPointF(0, 0))
167         self.online = False
168         self.angle = 0
169         self.blackPen = QPen(QColor(0, 0, 0))
170         self.paint_brush = QBrush(QColor(255, 0, 0))
171         self.signalSetPos = SignalSetPos()
172         self.scaling_factor = 10
173
174     def paint(self, painter, option, widget=None):
175         painter.setPen(self.blackPen)
176         painter.setBrush(self.paint_brush)
177
178         points = QPolygon([
179             QPoint(0, 0),
180             QPoint(self.scaling_factor, 0),
181             QPoint(self.scaling_factor, self.scaling_factor / 2),
182             QPoint(self.scaling_factor / 2, self.scaling_factor),
183             QPoint(0, self.scaling_factor / 2),
184             QPoint(0, 0)
185         ])
186
187         painter.drawPolygon(points)
188
189     def boundingRect(self):
190         return QRect(0, 0, self.scaling_factor, self.scaling_factor)
191
192     def change_position(self, x, y, angle, state):
193         if state == "ciclovía":
194             screen_xscale = 6.71875000024524e-07
195             screen_x0 = -57.5144784218750
196             screen_yscale = 5.83333333340856e-07
197             screen_y0 = -25.3391253333333
198         elif state == "ypacarai":
199             screen_x0 = -57.38106014642857
200             screen_xscale = 1.030035714285734E-4
201             screen_y0 = -25.38342902978723
202             screen_yscale = 9.33042553191497E-5
203         else:
204             screen_x0 = 0
205             screen_xscale = 0
206             screen_y0 = 0
207             screen_yscale = 0
208
209         screen_y = round((y - screen_y0) / screen_yscale).astype(int64)
210         screen_x = round((x - screen_x0) / screen_xscale).astype(int64)
211
212         self.position = QPointF(screen_x, screen_y)
213         self.angle = -round(angle * self.rd2dg)
214         self.signalSetPos.signalSetPos.emit('')
215
216     @Slot(str)
217     def do_change_pos(self):
218         self.setPos(self.position)
219         self.setRotation(self.angle)
220
221
222 class MainWindow(QMainWindow):

```

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

```

223     def __init__(self):
224         super(MainWindow, self).__init__()
225         self.ui = Ui_MainWindow()
226         self.ui.setupUi(self)
227
228         self.ui.cb_drone_id.currentIndexChanged.connect(self.on_cb_drone_id_change_event)
229         self.ui.checkBox.stateChanged.connect(self.on_follow_cb_changed)
230         self.scene = QGraphicsScene()
231         self.ui.graphicsView.setScene(self.scene)
232         self.ui.graphicsView.scale(1, -1)
233         self.prev_scale = 5
234         self.ui.zoomSlider.valueChanged.connect(self.on_slider_value_changed)
235         self.ui.graphicsView.setRenderHint(QPainter.Antialiasing)
236         self.scene.setSceneRect(QRect(0, 0, 1000, 1500))
237         my_pen = QPen(QColor(0, 200, 0))
238         top_line = QLineF(self.scene.sceneRect().topLeft(),
239                             self.scene.sceneRect().topRight())
240         left_line = QLineF(self.scene.sceneRect().topLeft(),
241                             self.scene.sceneRect().bottomLeft())
242         right_line = QLineF(self.scene.sceneRect().topRight(),
243                             self.scene.sceneRect().bottomRight())
244         bottom_line = QLineF(self.scene.sceneRect().bottomLeft(),
245                             self.scene.sceneRect().bottomRight())
246         self.scene.addLine(top_line, my_pen)
247         self.scene.addLine(left_line, my_pen)
248         self.scene.addLine(right_line, my_pen)
249         self.scene.addLine(bottom_line, my_pen)
250         self.drones = []
251         self.goals = []
252         self.state = "ypacarai"
253         self.ui.actionLago_Ypacarai.changed.connect(self.on_action_changed)
254         self.ui.actionLaguna_Ciclovia.changed.connect(self.on_action_changed)
255         self.signalAddNew = SignalAddNew()
256
257     def on_action_changed(self):
258         if self.state == "ypacarai" and self.ui.actionLaguna_Ciclovia.isChecked():
259             self.ui.actionLago_Ypacarai.setChecked(False)
260             self.state = "ciclovia"
261         elif self.state == "ciclovia" and self.ui.actionLago_Ypacarai.isChecked():
262             self.ui.actionLaguna_Ciclovia.setChecked(False)
263             self.state = "ypacarai"
264         print(self.state)
265
266     def on_follow_cb_changed(self):
267         if self.ui.checkBox.isChecked():
268             self.ui.graphicsView.centerOn(self.drones[self.ui.cb_drone_id.currentIndex()])
269             self.ui.checkBox.setText("True")
270         else:
271             self.ui.checkBox.setText("False")
272
273     def on_slider_value_changed(self):
274         value = self.ui.zoomSlider.value()
275         if value < self.prev_scale:
276             scale_factor = (self.prev_scale - value) / 1.2
277             self.ui.graphicsView.scale(scale_factor, scale_factor)
278         elif value > self.prev_scale:
279             scale_factor = (value - self.prev_scale) * 1.2

```

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

```

280         self.ui.graphicsView.scale(scale_factor, scale_factor)
281     self.prev_scale = value
282     for i in range(len(self.drones)):
283         if value < 8:
284             self.drones[i].scaling_factor = 50 / value
285             self.goals[i].scaling_factor = 70 / value
286
287     def fill_drone_id(self):
288         while database_busy:
289             pass
290         ids = []
291         for i in range(len(database["clients"])):
292             ids.append(str(database["clients"][i]["id"]))
293             aux = DroneView()
294             aux.signalSetPos.signalSetPos.connect(aux.do_change_pos)
295             aux.online = database["clients"][i]["online"]
296             aux.change_position(database["clients"][i]["position"]["lon"],
297                                database["clients"][i]["position"]["lat"],
298                                database["clients"][i]["position"]["yaw"],
299                                self.state)
300
301             aux_goal = GoalView()
302             aux_goal.set_drone_id(database["clients"][i]["id"])
303             aux_goal.signalSetPos.signalSetPos.connect(aux_goal.do_change_pos)
304             aux_goal.change_position(database["clients"][i]["goal"]["lon"],
305                                    database["clients"][i]["goal"]["lat"], self.state)
306
307             self.goals.append(aux_goal)
308             self.scene.addItem(self.goals[i])
309             self.drones.append(aux)
310             self.scene.addItem(self.drones[i])
311             self.ui.graphicsView.centerOn(self.drones[i])
312
313     self.ui.cb_drone_id.addItem(ids)
314
315     def add_new_drone_id(self):
316         while database_busy:
317             pass
318         ids = []
319         for i in range(len(database["clients"])):
320             if self.ui.cb_drone_id.findText(str(database["clients"][i]["id"])) == -1:
321                 ids.append(str(database["clients"][i]["id"]))
322                 aux = DroneView()
323                 aux.signalSetPos.signalSetPos.connect(aux.do_change_pos)
324                 aux.online = database["clients"][i]["online"]
325                 aux.change_position(database["clients"][i]["position"]["lon"],
326                                    database["clients"][i]["position"]["lat"],
327                                    database["clients"][i]["position"]["yaw"],
328                                    self.state)
329
330                 self.drones.append(aux)
331                 aux_goal = GoalView()
332                 aux_goal.set_drone_id(database["clients"][i]["id"])
333                 aux_goal.signalSetPos.signalSetPos.connect(aux_goal.do_change_pos)
334                 aux_goal.change_position(database["clients"][i]["goal"]["lon"],
335                                         database["clients"][i]["goal"]["lat"], self.state)
336                 self.goals.append(aux_goal)

```

SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO INFORMACIÓN DINÁMICA

```
337         self.signalAddNew.signalAddNew.connect(self.do_add_new)
338         self.signalAddNew.signalAddNew.emit(i)
339         break
340     self.ui.cb_drone_id.addItem(ids)
341
342     @Slot(str)
343     def do_add_new(self, i):
344         self.scene.addItem(self.drones[i])
345         self.scene.addItem(self.goals[i])
346
347     def on_cb_drone_id_change_event(self):
348         if database["clients"][self.ui.cb_drone_id.currentIndex()][ "online" ]:
349             self.ui.label_status.setText("online")
350         else:
351             self.ui.label_status.setText("offline")
352         self.ui.label_pos.setText( obtain_a_constant_str(self.ui.cb_drone_id.currentIndex(), "position"))
353         self.ui.label_goal.setText( obtain_a_constant_str(self.ui.cb_drone_id.currentIndex(), "goal"))
354         self.ui.label_vel.setText(str(database["clients"][self.ui.cb_drone_id.currentIndex()][ "velocity" ]))
355         self.ui.graphicsView.centerOn(self.drones[self.ui.cb_drone_id.currentIndex()])
356
357     def update_data(self, position):
358         try:
359             self.drones[position].change_position(database["clients"][position][ "position" ][ "lon" ],
360                                                  database["clients"][position][ "position" ][ "lat" ],
361                                                  database["clients"][position][ "position" ][ "yaw" ],
362                                                  self.state)
363
364             self.goals[position].change_position(database["clients"][position][ "goal" ][ "lon" ],
365                                                  database["clients"][position][ "goal" ][ "lat" ],
366                                                  self.state)
367
368             if database["clients"][position][ "online" ] != self.drones[position].online:
369                 if database["clients"][position][ "online" ]:
370                     self.drones[position].paint_brush = QBrush(QColor(0, 255, 0))
371                 else:
372                     self.drones[position].paint_brush = QBrush(QColor(255, 0, 0))
373                 self.drones[position].online = database["clients"][position][ "online" ]
374
375             if position == self.ui.cb_drone_id.currentIndex():
376                 if database["clients"][self.ui.cb_drone_id.currentIndex()][ "online" ]:
377                     self.ui.label_status.setText("online")
378                 else:
379                     self.ui.label_status.setText("offline")
380                 self.ui.label_pos.setText( obtain_a_constant_str(position, "position"))
381                 self.ui.label_goal.setText( obtain_a_constant_str(position, "goal"))
382                 self.ui.label_vel.setText(str(database["clients"][position][ "velocity" ]))
383                 if self.ui.checkBox.isChecked():
384                     self.ui.graphicsView.centerOn(self.drones[self.ui.cb_drone_id.currentIndex()])
385         except NameError as e:
386             print("not ready yet")
387
388     def closeEvent(self, event):
389         client.disconnect()
390         updating_database()
391         event.accept()
392
393
```

**SISTEMA DE GUÍA DE MÚLTIPLES DRONES DE SUPERFICIE ACUÁTICA UTILIZANDO
INFORMACIÓN DINÁMICA**

```
394 def obtain_a_constant_str(position, item):
395     yaw = str(round(database["clients"][position][item]["yaw"], 3))
396     aux = "'lat': " + str(round(database["clients"][position][item]["lat"], 7)) + "\n" + \
397         "'lon': " + str(round(database["clients"][position][item]["lon"], 7)) + "\n" + \
398         "'alt': " + str(round(database["clients"][position][item]["alt"], 7)) + "\n" + \
399         "'yaw': " + str(yaw)
400     return aux
401
402
403 if __name__ == "__main__":
404     client = None
405     base_dir = os.path.dirname(os.path.abspath(__file__))
406     _mqtt_thread = threading.Thread(target=_mqtt_thread)
407     _mqtt_thread.start()
408     app = QApplication(sys.argv)
409     window = MainWindow()
410     window.show()
411
412     window.fill_drone_id()
413     sys.exit(app.exec_())
```