UNIVERSIDAD DE LA REPÚBLICA URUGUAY

FACULTAD DE INGENIERÍA

Sistema de inspección de estructuras civiles utilizando drones autónomos

Informe de Proyecto de Grado presentado por

Nicolás Fripp, Agustín Queirolo, Agustín Santellán y Joaquín Valentín

en cumplimiento parcial de los requerimientos para la graduación de la carrera de Ingeniería en Computación de Facultad de Ingeniería de la Universidad de la República

Supervisor

Facundo Benavides

Montevideo, 22 de diciembre de 2023

Sistema de inspección de estructuras civiles utilizando drones autónomos por Nicolás Fripp, Agustín Queirolo, Agustín Santellán y Joaquín Valentín tiene licencia CC Atribución 4.0.

Agradecimientos

Agradecemos a nuestras familias y a nuestros seres queridos, sin los cuales este proyecto no habría sido posible. También, agradecemos a nuestro tutor por su disposición para con nosotros, y por último, agradecemos a Mauro Arzuaga, quien nos ayudó con la generación del modelo 3D de la Facultad de Ingeniería usado para el desarrollo.

Resumen

Este proyecto propone el uso de una flota de drones autónomos para llevar a cabo inspecciones edilicias frecuentes a la Facultad de Ingeniería, con el objetivo de modificar el proceso actual, mitigando sus riesgos de seguridad y disminuyendo sus tiempos.

Para la implementación del sistema, se utilizaron diferentes herramientas como ROS (Robotic Operating System) y PX4 para el control y monitoreo de los drones, y Gazebo como entorno simulador.

Uno de los puntos más importantes de la solución fue garantizar la eficiencia de las rutas de inspección. Para ello se utilizaron los algoritmos de A*, con el fin de encontrar el camino mas corto entre las zonas de inspección, y el Multiple Traveling Salesman Problem (MTSP) para resolver la asignación de drones a lugares.

Además, el sistema cuenta con un servidor, desarrollado en python, una base de datos no relacional, para la cual se utilizó MongoDB, y una interfaz de usuario desarrollada en React, utilizada para la definición y ejecución de las inspecciones.

Palabras clave: Drones, Inspección, Simulación, ROS, Gazebo, PX4

Índice general

Capítulo 1

Introducción

1.1. Motivación

Como todo edificio, la Facultad de Ingeniería (Universidad de la República) requiere de ciertos cuidados y mantenimientos frecuentes, que se incrementan con los años y el uso de las instalaciones, especialmente en sus fachadas, techos y canalones de desagüe, para poder mantener el edificio en buen estado. Con ese objetivo, se deben realizar inspecciones rutinarias del edificio, muchas de ellas en grandes alturas. Hoy en día estas inspecciones son realizadas por empleados de la Universidad de la República, sin contar con las estructuras y protecciones necesarias para desarrollar estas actividades de forma segura. La solución a esta problemática tiene un costo muy alto ya que implica nuevas estructuras en los techos y la obtención de equipos de protección personal, así como la mano de obra para la implementación.

Observando esta realidad, en conjunto con el interés del equipo de introducirse en el campo de la robótica, surge como solución intermedia el uso de drones autónomos que se encarguen de la inspección visual de los puntos de interés del edificio, reduciendo así la exposición del equipo de mantenimiento al riesgo actual. La solución sería parcial dado que no elimina el trabajo manual de mantenimiento que es necesario en tareas como reparaciones y limpiezas de desagües.

Dado el tamaño de la infraestructura a inspeccionar resulta importante contar con múltiples drones para paralelizar las tareas y permitir inspecciones más rápidas y extensas.

1.2. Objetivos

El proyecto se enfocará en elaborar un sistema que controle una flota de drones semi-autónomos en un entorno de simulación. Éste deberá permitir la grabación de videos de las distintas zonas de la facultad que posteriormente serán utilizados por parte del equipo de mantenimiento para realizar un análisis

del estado de la infraestructura. Además de generar una planificación eficiente y segura de caminos para los robots mediante la implementación e integración de diferentes algoritmos, el producto incluirá una interfaz de usuario que soportará la configuración, lanzamiento y supervisión de inspecciones.

1.3. Organización General del Documento

Capítulo 2. Introducción de conceptos necesarios para la comprensión de la implementación.

Capítulo 3. Desarrollo del análisis, diseño e implementación del sistema. Decisiones tomadas y proceso de desarrollo.

Capítulo 4. Planificación de pruebas. Ejecución de experimentos. Análisis de resultados.

Capítulo 5. Conclusiones y trabajo futuro.

Capítulo 2

Marco de trabajo

Índice

El foco de este capítulo será la presentación de conceptos necesarios para la comprensión de la solución elaborada, así como la descripción del estado del arte en soluciones a problemas similares y el entorno utilizado durante la implementación y evaluación del sistema.

2.1. La solución a alto nivel

Como se comenta en la introducción, las inspecciones al edificio de la facultad son llevadas a cabo por el equipo de mantenimiento. La inspección manual de las zonas de interés sin el equipo de protección adecuado ponen en riesgo la integridad física de los operarios. Para mitigar estos riesgos y reducir el tiempo de inspecciones, se propone el uso de múltiples drones.

El problema resulta complejo, pero puede dividirse en una serie de desafíos que resultan más sencillos, para luego combinarlos y lograr una solución completa.

En primer lugar, se identifica el problema de representación de la realidad, tanto el modelado de los datos como la preparación del entorno de simulación, que será abordado a partir de la sección 3.3.

Luego, en la sección 3.5.1, se mostrará cómo la representación de la Facultad servirá de entrada a los algoritmos encargados de planificar caminos entre los diferentes lugares de interés, evitando los obstáculos conocidos del entorno.

También se deberá distribuir eficientemente las tareas entre los drones de la flota con el fin de evitar desequilibrios en la asignación de áreas de inspección, garantizando que ningún dron tenga una carga excesiva mientras otro carece de tareas. La solución a este problema será elaborada en la sección 3.5.3 y utilizará los caminos entre los diferentes lugares para tomar estas decisiones.

Habiendo resuelto los desafíos anteriores, se estará en condiciones de enviar a la flota en una misión que visite los lugares de interés, pero no es suficiente para considerarlo una inspección. Por ello, en la sección 3.5.2 se resolverá la planificación de los caminos de inspección acorde a los diferentes tipos de zonas identificados.

Por último, en la sección 3.6, se abordará el desarrollo de una interfaz que permita a los usuarios controlar y monitorear el sistema sin requerir de entrenamientos extensos.

Si bien el problema planteado y la solución implementada en este proyecto son específicos de la Facultad de Ingeniería, se busca desarrollar una solución genérica y extensible, para que a futuro pueda ser utilizada en otros servicios universitarios.

2.2. Marco Teórico

2.2.1. UAV

Un UAV, del inglés *Unmanned Aerial Vehicle*, es un vehículo aéreo no tripulado, también conocido como dron. Se trata de un dispositivo que puede volar sin la necesidad de un piloto a bordo y puede ser equipado con una multitud de sensores que hacen posible un gran número de casos de uso como la seguridad, seguimiento de objetivos, cine, recreación y entrega de paquetes, entre otros.

Existen varios tipos de UAVs[1], que varían en cuanto a su forma y sus grados de libertad (*degrees of freedom* o DOF) controlables, que cambian su utilidad en diferentes casos de uso:

- **Fixed-Wing (Ala fija)**: Un dron de ala fija (Figura 2.1) posee un ala rígida y se rige por los mismos principios que un avión, que a través de su geometría (y variaciones) e interacción con el flujo de aire genera el empuje vertical. Esto los hace eficientes en términos de energía y muy útiles para aplicaciones que requieran cobertura de grandes distancias. Este tipo de dron posee tres grados de libertad controlables, su actitud o cabeceo (*pitch*), alabeo (*roll*) y giro (*yaw*), que en su combinación le otorgan seis

[1] `https://www.auav.com.au/articles/drone-types/`

grados de libertad, pero no permiten, por ejemplo, que se traslade hacia un costado, o quede estacionario en una posición. Ésta es su principal desventaja en casos de uso en los que se necesita mayor maniobrabilidad, además de necesitar un gran espacio libre para despegar y aterrizar.

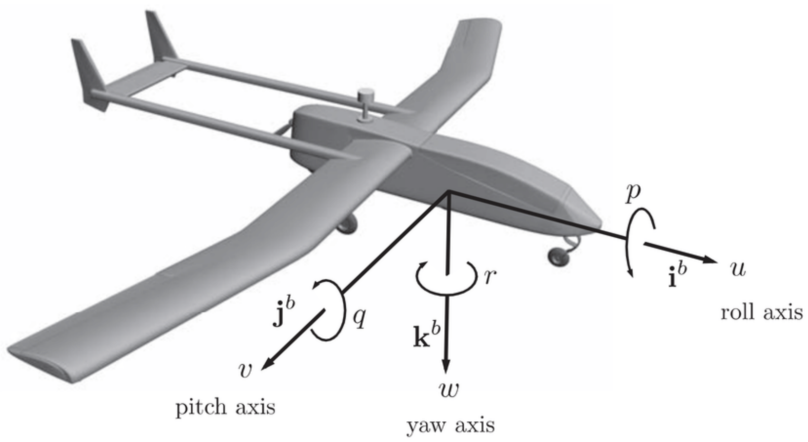

Figura 2.1: UAV de ala fija y sus respectivos grados de libertad controlables. Extraído de (Fari, 2017).

- **Mono-Rotor (*Single-Rotor*)**: Se asemejan a los helicópteros en estructura y diseño, contando con un rotor que genera empuje vertical, además de un rotor de cola para controlar la dirección y la estabilidad. Como ventaja frente a los Multi-Rotor son más eficientes en términos de energía, mientras que tienen la desventaja de requerir mucho mantenimiento y cuidado por su complejidad mecánica (Figura 2.2), ya que debe tener mecanismos para controlar el ángulo de ataque de las aspas del rotor vertical. Por su diseño, presentan cuatro grados de libertad controlables: arriba/abajo (*heave*), izquierda/derecha (*sway*), atrás/adelante (*surge*) y giro (*yaw*); los que se pueden apreciar en la Figura 2.3 que representa un helicóptero.
- **Híbridos de despegue y aterrizaje vertical (*Vertical Take-Off and Landing* o VTOL)**: Los drones híbridos VTOL (Figura 2.4) combinan los beneficios de los diseños de ala fija y de rotores. Este tipo de dron tiene rotores capaces de ejercer fuerza tanto vertical como horizontal variando su orientación conectados a las alas, lo que le permite despegar y aterrizar como un dron de rotores y luego hacer la transición a un vuelo aerodinámico. Por este motivo, este diseño no necesita de amplios espacios para sus inicios y fines de vuelo, pero mantiene el control de vuelo de los de ala fija.
- **Multi-Rotor**: Se les llama de esta forma porque tienen más de un ro-

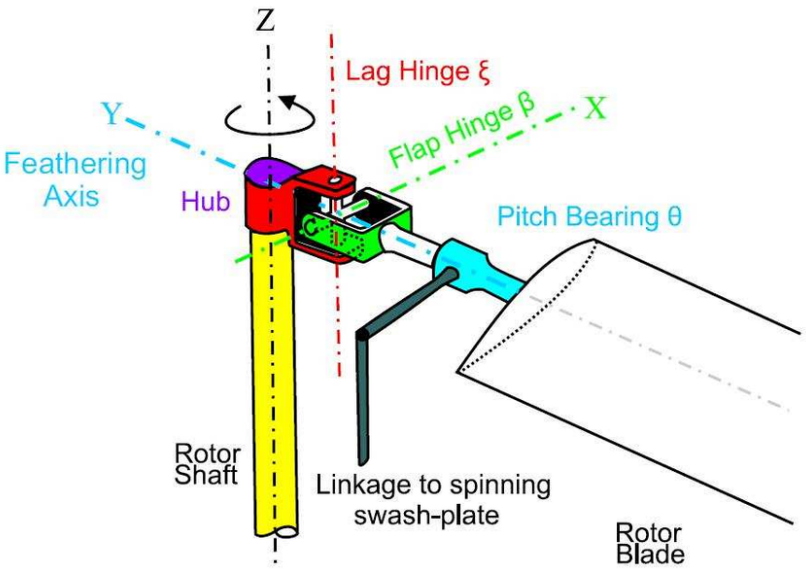

Figura 2.2: Esquema del mecanismo de control de hélice de un dron mono-rotor. Extraído de (Marichal y cols., 2013).

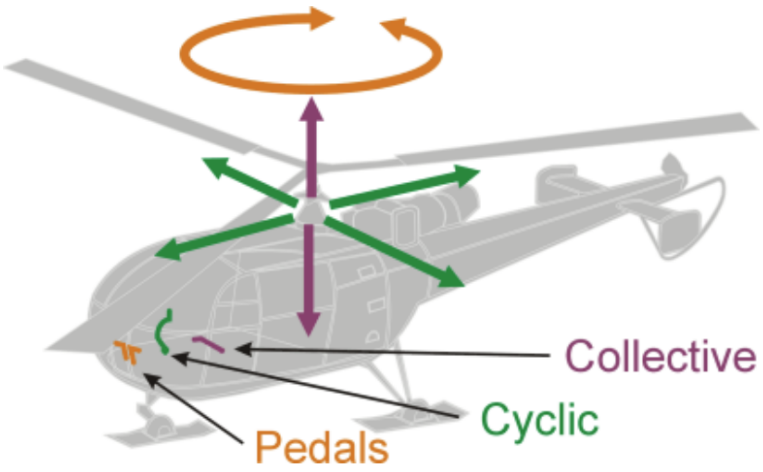

Figura 2.3: Helicóptero y sus respectivos grados de libertad controlables. Extraído de (*Helicopter Control*, s.f.).

Figura 2.4: Dron KUS-TR de tipo VTOL (Yu, Heo, Jeong, y Kwon, 2016).

tor. En esta categoría se pueden encontrar: tricópteros (tres rotores), cuadricópteros (cuatro rotores) y hexacópteros (seis rotores), entre otros. Los más comunes son los cuadricópteros. Las principales ventajas en comparación a los demás tipos es su sencillez mecánica y su popularidad, que hacen fácil su obtención y mantenimiento, además de presentar un muy buen nivel de control. Por otro lado, su principal desventaja es que tiene un alto consumo de energía, por lo que los tiempos de vuelo usando una carga de su batería son menores. Los cuadricópteros, que serán los utilizados en el trabajo, tienen los mismos grados de libertad controlables que los drones mono-rotor (Figura 2.5).

Los UAVs pueden operar en tres modos de autonomía en relación a su *Ground Control Station* (GCS) con la que interactúa un humano según Wang y cols. (Wang y cols., 2020):

- **Human-in-the-loop**: Los UAVs no tienen la capacidad de operar independientemente y son controlados en tiempo real por la GCS. El control de parte de un operario es total, por lo que el control de vuelo, estabilidad y la finalización exitosa de la misión dependen fuertemente de la habilidad del piloto, además de la conexión con la GCS.
- **Human-on-the-loop**: En este modo, también conocido como semi-autónomo, el control es compartido entre la GCS y los UAVs. Estos últimos son capaces de tomar el control en algunas ocasiones como, por ejemplo, evitar una colisión, pero el control principal en el vuelo es de la GCS. Una pérdida momentánea de la conexión con la terminal de control no es crítica ya que la estabilidad del vuelo es gestionada por parte del robot.
- **Autónomo**: Como su nombre lo indica, los UAVs son capaces de cumplir sus tareas y tomar decisiones de corrección de trayectoria sin la necesidad de control directo humano posterior a la definición de los objetivos salvo que una emergencia lo requiera.

Figura 2.5: Cuadricóptero IRIS (*Iris Quadcopter UAV*, s.f.).

El uso de un solo UAV permite la realización de diversas tareas. Sin embargo, el uso de múltiples drones en simultáneo puede tener varios beneficios (Skorobogatov, Barrado, y Salamí, 2020). Estos son:

- Eficiencia: Los tiempos operacionales de una misión se pueden ver reducidos al utilizar múltiples drones.
- Paralelización: Una flota de UAVs puede realizar acciones en distintos puntos geográficos al mismo tiempo.
- Tolerancia a las fallas: En el caso de un solo UAV, una falla en su funcionamiento implica el término de la misión. Sin embargo, al usar una flota de múltiples drones, la falla de un UAV se puede mitigar, en algunos casos, mediante un algoritmo que le pueda asignar a los UAVs restantes las responsabilidades del UAV perdido.

2.2.2. Paradigmas de robótica

Los enfoques, o paradigmas, constituyen modelos que delinean el modo de operación de un robot. Estos paradigmas son definidos a partir de como interactúan las primitivas. En este contexto, se define primitiva como una acción fundamental que puede realizar un robot. Cada primitiva desempeña una función en la que se procesa un conjunto de información o datos de entrada para generar una salida.

Las primitivas robóticas se dividen en tres categorías: Sensar, Planificar y Actuar. Cada una de estas primitivas cumple una función específica. La primitiva "Sensar" toma datos de sensores como entrada y genera información sensada como salida. Por otro lado, la primitiva "Planificar" procesa información como entrada y produce directivas como salida. Finalmente, la primitiva "Actuar" toma información sensada o directivas como entrada y emite comandos a los actuadores como salida.

Para clasificar los sistemas, se definen tres tipos de paradigmas (Murphy, 2000):

1. **Paradigma Jerárquico o Deliberativo (Figura 2.6):** Caracterizado por planificar antes de actuar, este paradigma encuentra aplicación cuando un robot necesita utilizar un mapa para alcanzar su destino. En este caso, el robot puede sensar para ubicarse en el mapa, calcular un plan y luego ejecutar los movimientos necesarios para cumplirlo.

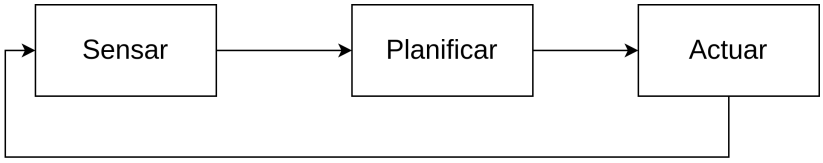

Figura 2.6: Paradigma reactivo.

2. **Paradigma Reactivo (Figura 2.7):** Este enfoque, basado en comportamientos, carece de un componente de planificación. Se observa comúnmente en la naturaleza, siendo comparable a un reflejo del ser humano ante la presencia de un estímulo, prescindiendo de una etapa de planificación.

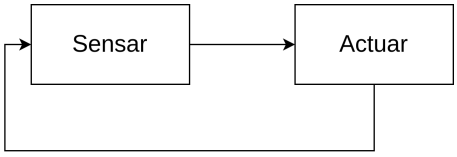

Figura 2.7: Paradigma reactivo.

3. **Paradigma Híbrido (Figura 2.8):** Este enfoque combina elementos del paradigma reactivo y jerárquico, dependiendo de la lógica del algoritmo. Tomando como referencia el modelo jerárquico, si el robot se enfrentase a obstáculos desconocidos en el mapa, seguiría una trayectoria de colisión. No obstante, mediante el paradigma híbrido, el robot podría, además de ejecutar un plan preestablecido, realizar continuos análisis sensoriales durante la ejecución para ajustar el plan en curso y evitar colisiones con obstáculos no previstos.

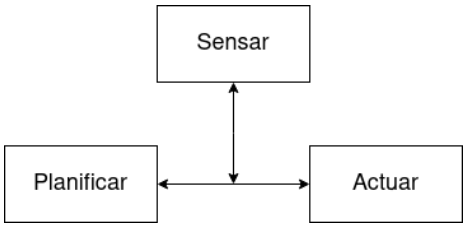

Figura 2.8: Paradigma híbrido.

El desarrollo del paradigma reactivo, mediante la inspiración en la naturaleza, trajo la capacidad a los robots de funcionar en entornos dinámicos o con más incógnitas (terrenos nuevos, obstáculos móviles o imprevistos) en los que no se podía realizar una planificación determinista. Más adelante, la aparición del paradigma híbrido permitió resolver los problemas del anterior, añadiendo planes que habilitaban a optimizar las tareas del robot. Para el caso del proyecto, el uso de este último paradigma es una necesidad, ya que permite optimizar los traslados de los robots a la vez que éstos pueden reaccionar a obstáculos imprevistos y evitarlos.

2.2.3. Problemas de Optimización

Para llevar a cabo la inspección de las distintas zonas de la Facultad de Ingeniería, se requiere la creación de caminos para cada dron involucrado en la tarea. Estos caminos, permitirán que los drones partan desde un punto de despegue, recorran múltiples ubicaciones para llevar a cabo la inspección de cada una de ellas, y finalmente aterricen en el punto de despegue. Dado que el tiempo de vuelo de los drones es muy limitado, es necesario que los caminos generados minimicen la distancia recorrida.

Con este propósito, se modelará la realidad como un problema de optimización. Al resolver este problema, se obtendrá una ruta eficiente para que los drones lleven a cabo la inspección en el menor tiempo posible. A continuación se presentarán algunos algoritmos apropiados para la realidad del problema.

Travelling Salesman Problem

El *Travelling Salesman Problem* (TSP) es un problema de optimización combinatorial que se plantea como sigue: dado un conjunto finito de ciudades y las distancias entre cada par de ciudades, se propone encontrar la ruta más corta que visite cada ciudad exactamente una vez y regrese a la ciudad de origen. Esta ruta debe ser lo más corta posible, es decir, minimizar la suma total de las distancias entre las ciudades visitadas.

El TSP se puede modelar usando grafos de manera muy natural. En un grafo, las ciudades se representan como nodos y las distancias entre las ciudades se representan como aristas con pesos. La ruta más corta que recorre todas las ciudades y regresa a la ciudad de origen se puede representar como un ciclo

hamiltoniano en el grafo, es decir, un camino que pasa por todos los nodos exactamente una vez y vuelve al nodo de origen.

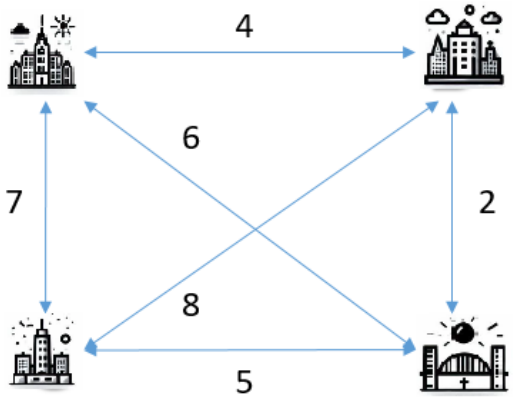

Figura 2.9: Travelling Salesman Problem.

El TSP puede resolverse usando Programación Lineal Entera, una técnica de optimización matemática que se utiliza para resolver problemas modelados con una función objetivo sujeta a un conjunto de restricciones, en los que algunas o todas las variables de decisión son enteras. Para ello, el problema es planteado usando la formulación de Miller-Tucker-Zemlin (Miller, Tucker, y Zemlin, 1960). Sea x_{ij} igual a 1 si existe un camino desde la ciudad i hasta la ciudad j, y 0 en caso contrario, para el conjunto de ciudades $0, ..., n$. Sean u_i para $i = 1, ..., n$ variables artificiales y sea c_{ij} la distancia desde la ciudad i a la ciudad j. Entonces el modelo de Programación Lineal Entera puede ser escrito como:

$$\text{mín} \sum_{i=0}^{n} \sum_{j \neq i, j=0}^{n} c_{ij} x_{ij} \tag{2.1}$$

$$x_{i,j} \in \{0, 1\} \qquad i, j = 1, \ldots, n; \tag{2.2}$$

$$\sum_{i=0, i \neq j}^{n} x_{ij} = 1 \qquad j = 0, \ldots, n \tag{2.3}$$

$$\sum_{j=0, j \neq i}^{n} x_{ij} = 1 \qquad i = 0, \ldots, n \tag{2.4}$$

$$u_i - u_j + n x_{ij} \leq n - 1 \qquad 1 \leq i \neq j \leq n. \tag{2.5}$$

Donde (2.1) es la función objetivo. Las restricciones (2.3) y (2.4) marcan que cada ciudad es visitada por el viajero una única vez. Finalmente, la ecuación con

las variables artificiales u_i sirven para evitar la generación de subciclos dentro del camino resultado. Estas variables se encargan de etiquetar el orden en el que los nodos son visitados.

Un ciclo hamiltioniano de costo mínimo es una solución a la formulación previa. Esto se puede hacer utilizando fuerza bruta, examinando todas las posibles rutas y seleccionando la de menor peso. Sin embargo, este enfoque no es práctico para grafos grandes debido a su alto costo computacional. El TSP es un problema NP-difícil, lo que significa que no hay un algoritmo conocido que pueda resolver el problema de manera eficiente (en tiempo polinomial). Por lo tanto, se utilizan algoritmos heurísticos y aproximaciones para encontrar soluciones subóptimas en tiempos admisibles.

Dado que en el proyecto se plantea utilizar una flota de múltiples drones, resolver TSP no será suficiente para realizar la planificación de los caminos. Si la cantidad de drones es mayor a uno, hay que modelar el problema como el Multiple Travelling Salesman Problem (MTSP).

Multiple Travelling Salesman Problem

El Multiple Travelling Salesman Problem (MTSP) es una generalización de TSP (Bektas, 2006), en donde se tienen varios vendedores y se busca encontrar múltiples rutas que recorran un conjunto de ciudades de manera óptima. Cuando el número de vendedores es 1, nos encontramos con el problema de TSP. Cuando hay múltiples vendedores, el objetivo es encontrar una solución que visite todos los nodos minimizando la suma total de las distancias recorridas por todos los vendedores.

Este problema se puede modelar de manera similar a TSP, modificando las restricciones sobre las cantidades de arcos salientes y entrantes al nodo de salida (Ecuación 2.7 y Ecuación 2.8) y utilizando grafos en los que el recorrido de cada vendedor se representa como un ciclo en el grafo. La solución óptima consiste en encontrar un conjunto de ciclos de menor peso total. Su formulación matemática es la siguiente:

$$\text{mín} \sum_{i=1}^{n} \sum_{j \neq i, j=1}^{n} c_{ij} x_{ij} \tag{2.6}$$

$$\sum_{i=2}^{n} x_{i1} = m \tag{2.7}$$

$$\sum_{j=2}^{n} x_{1j} = m \tag{2.8}$$

$$\sum_{i=2, i \neq j}^{n} x_{ij} = 1 \qquad j = 1, \ldots, n; \tag{2.9}$$

$$\sum_{j=2, j \neq i}^{n} x_{ij} = 1 \qquad i = 1, \ldots, n; \tag{2.10}$$

$$u_i - u_j + 1 \leq (n-1)(1 + x_{ij}) \qquad 2 \leq i \neq j \leq n; \tag{2.11}$$

$$0 \leq u_i \leq n \qquad 2 \leq i \leq n; \tag{2.12}$$

$$x_{ij} \in \{0, 1\} \qquad i, j = 1, \ldots, n; \tag{2.13}$$

$$u_i \in \mathbf{Z} \qquad i = 2, \ldots, n. \tag{2.14}$$

Donde las restricciones 2.7 y 2.8 indican que en la ciudad de origen parten y arriban m vendedores. Las restricciones 2.9 y 2.10 indican que cada ciudad, excepto la primera, es recorrida exactamente por un vendedor, y por último, las variables artificiales que al igual que en TSP, sirven para evitar la generación de subciclos dentro del camino resultado.

2.2.4. Algoritmos de búsqueda de caminos más cortos

La búsqueda de caminos más cortos es un problema fundamental en la teoría de grafos y una tarea esencial en numerosas aplicaciones prácticas, que abarcan desde la planificación de rutas en sistemas de navegación hasta la optimización de redes de comunicación y la resolución de problemas logísticos complejos. En el contexto de este trabajo, se necesita un algoritmo de este tipo para encontrar el camino mas corto entre dos lugares y, de esta forma, optimizar el tiempo de ejecución de las misiones de inspección.

A continuación se presentarán varios algoritmos de búsqueda de caminos más cortos en grafos, los cuales se definen como $G = \{V, E\}$, siendo V el conjunto vértices y E el de aristas.

Algoritmo de Dijkstra

El algoritmo de Dijkstra es un algoritmo de búsqueda que se utiliza para encontrar el camino más corto en un grafo ponderado entre un nodo de origen y todos los demás nodos. Funciona examinando todos los nodos adyacentes al nodo de origen y actualizando la distancia acumulada desde el nodo de origen a cada nodo a medida que se expande la búsqueda. Es un algoritmo "Greedy". Es decir, en cada paso de su ejecución elige la mejor opción, sin considerar los siguientes pasos.

A continuación se encuentra el pseudocódigo del algoritmo de Dijkstra(Kleinberg y Tardos, 2005):

Algorithm 1 Algoritmo de Dijkstra

procedure DIJKSTRA(G, s)
  $S \leftarrow \emptyset$ ▷ Conjunto de nodos explorados
  for each $u \in V$ **do**
    $d(u) \leftarrow \infty$ ▷ Inicializa distancias
  end for
  $S \leftarrow \{s\}$ ▷ Inicializar con nodo fuente
  $d(s) \leftarrow 0$
  while $S \neq V$ **do**
    Selecciona nodo $v \in S$ con al menos una arista en S tal que
    $d'(v) = \text{mín}_{e=(u,v):u \in S}\, d(u) + s_e$ es mínimo
    Agregar v a S
    $d(v) \leftarrow d'(v)$
  end while
end procedure

La complejidad computacional de este algoritmo es de orden $O(n^2)$ donde n es el número de nodos en el grafo, pero utilizando una cola de prioridad, se puede lograr una implementación de orden $O(m * \ln n)$, donde m es la cantidad de aristas.

Algoritmo A*

El algoritmo A* es una generalización del algoritmo de Dijkstra que tiene en cuenta una función heurística que estima la distancia restante hasta el nodo destino. Esta función de heurística permite al algoritmo A* expandir nodos que están más cerca del nodo destino, lo que aumenta en algunas instancias la eficiencia del algoritmo en comparación con Dijkstra.

El algoritmo A* utiliza una función de evaluación: $f(n) = g(n)+h'(n)$ donde $h'(n)$ representa el valor heurístico del nodo a evaluar desde el actual, n, hasta el final, y $g(n)$ el coste real del camino recorrido para llegar a dicho nodo, n, desde el nodo inicial.

La complejidad del algoritmo A* depende en gran medida de la calidad de la función de heurística $h'(n)$ utilizada. Dado que Dijkstra es un caso particular A* que usa la heurística nula (que no aporta información), si la función de heurística es adecuada y da una buena estimación de la distancia restante, el algoritmo A* puede ser más eficiente en algunas instancias que Dijkstra, aunque son igual de eficientes en el peor caso.

Una buena heurística para el algoritmo A* debe cumplir con dos propiedades: debe ser admisible y consistente (Hart, Nilsson, y Raphael, 1968). Una heurística es admisible si siempre subestima la distancia restante hasta el nodo de destino. Esto garantiza que A* no se expandirá por nodos que no son óptimos y que siempre encontrará la solución óptima si existe. Una heurística es consistente si se cumple: $\forall (N, A) \in E : h'(N) - h'(A) + w(N, A) \leq h'(A)$, siendo (N, A) una arista, y siendo w el peso de la arista. Esto garantiza que A* siempre se

expandirá en la dirección correcta hacia el nodo de destino.

Una de las heurísticas más comunes para el algoritmo A* es la distancia euclidiana. Esta heurística es admisible y consistente, y se utiliza comúnmente en problemas de búsqueda de camino en entornos tridimensionales. La distancia euclidiana entre dos puntos en un espacio 3D se calcula mediante la fórmula:

$$\text{Distancia Euclidiana} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

Donde: x_1, y_1, y z_1 son las coordenadas del primer punto, y x_2, y_2, y z_2 son las coordenadas del segundo punto.

Existen otras heurísticas posibles además de la distancia euclidiana, por ejemplo la distancia de Manhattan, que se basa en movimientos en cuadrículas o la distancia de Chebyshev (Figura 2.10), que permite movimientos diagonales en cuadrículas, entre otras(Patel, 2023).

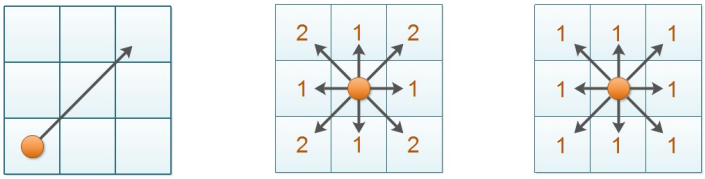

Figura 2.10: Representación en dos dimensiones de Distancia Euclidiana, Distancia de Manhattan y Distancia de Chebyshev.

A continuación se encuentra el pseudocódigo para la implementación del algoritmo A* (Paliwal, 2023):

2.2.5. Representaciones del espacio tridimensional

Con el objetivo de contar con un entorno de simulación cercano a la realidad, se trabajó con diversas representaciones de la Facultad de Ingeniería, cada una con diferentes fines.

Nube de puntos

La representación en nubes de puntos consiste en describir el entorno en una colección de vértices en el espacio tridimensional, sin conocer aristas. Usualmente, este tipo de información es obtenida mediante el uso de sensores de distancia, como por ejemplo LiDAR (del inglés "*Light Detection And Ranging*"), y es muy usada en robots que necesitan representar entornos previamente desconocidos.

Para el desarrollo del proyecto, se colaboró con un estudiante de agrimensura para generar una nube de puntos (Figura 2.11) de la Facultad de Ingeniería mediante un vuelo exploratorio de un dron teleoperado y equipado con sensores LiDAR.

Algorithm 2 Algoritmo A*

```
procedure AStar(Grafo G, nodo inicio, nodo objetivo)
    ListaAbierta ← [inicio]
    ListaCerrada ← []
    g(inicio) = 0
    h(inicio) = heurística(inicio, objetivo)
    f(inicio) = g(inicio) + h(inicio)
    while ListaAbierta ≠ [] do
        n ← nodo con menor f en ListaAbierta
        if n = objetivo then
            return ListaCerrada
        else
            ListaAbierta ← ListaAbierta \ {n}
            ListaCerrada ← ListaCerrada ∪ {n}
            for all n' ∈ vecinos de n do
                costo_nn' = g(n) + heurística(n, n')
                g(n') = costo_nn'
                if n' ∈ ListaAbierta and costo_nn' < g(n') then
                    ListaAbierta ← ListaAbierta \ {n'}
                else if n' ∈ ListaCerrada and costo_nn' < g(n') then
                    ListaCerrada ← ListaCerrada \ {n'}
                else if n' ∈ ListaCerrada then
                    continue
                else
                    ListaAbierta ← ListaAbierta ∪ {n'}
                    f(n') = g(n') + h(n')
                end if
            end for
        end if
    end while
end procedure
```

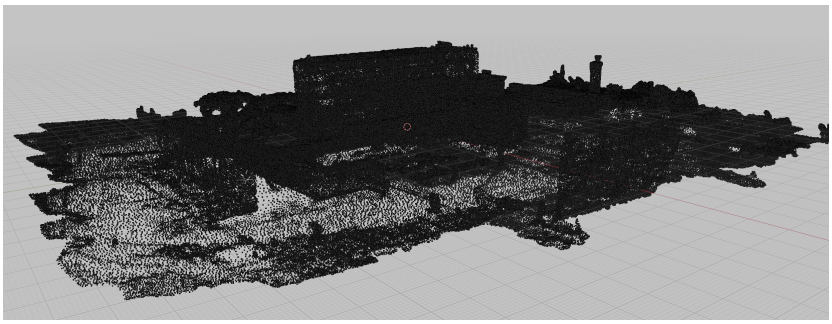

Figura 2.11: Facultad de Ingeniería representada como una nube de puntos.

Mallas tridimensionales

Brindando más información que las nubes de puntos, las mallas tridimensionales que representan a un objeto almacenan la posición de sus vértices, las aristas que los conectan, y en algunos formatos los vectores normales de sus caras. Además, algunos formatos permiten el almacenamiento de información de texturas, lo que los hace muy útiles a la hora de simular entornos.

Con el objetivo de obtener este tipo de representación, se recurrió a la técnica conocida como fotogrametría. Ésta consiste en capturar múltiples imágenes en ángulos variados a un objetivo, que en conjunto con la nube de puntos, servirán de entrada para un algoritmo encargado de generar la malla apropiada (Figura 2.12).

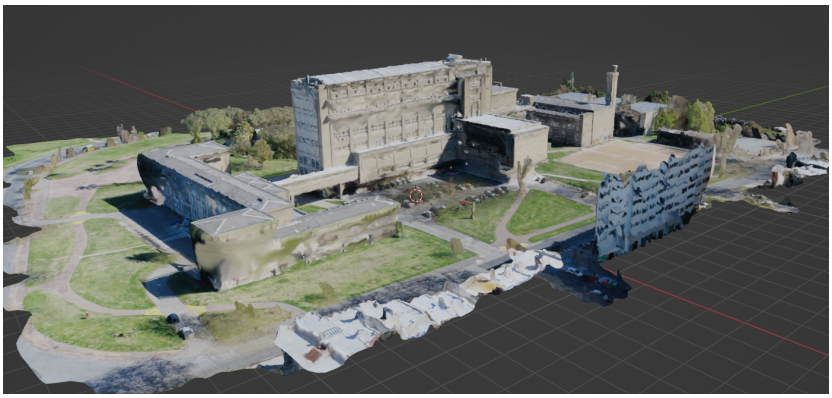

Figura 2.12: Facultad de Ingeniería representada como malla tridimensional.

Grillas de ocupación

Las grillas de ocupación (en inglés "*occupancy grid*") son una representación más simple del entorno que almacena información de relevancia para la planificación de movimientos de un robot: cuáles secciones del espacio se encuentran

libres para ocuparlas y cuáles no. Para esto, el espacio se divide en una cuadrícula de tamaño acorde al entorno y el robot, buscando una partición en la que cada espacio sea el suficiente para que el robot pueda ser contenido junto con un margen de seguridad. Además, ya que cada celda es modelada como un nodo en la planificación de caminos, la elección de su tamaño tiene impacto directo en el rendimiento de los algoritmos, por lo que se debe buscar un balance entre la precisión y el rendimiento.

Esta representación (Figura 2.13) fue generada a partir de la nube de puntos obtenida y utilizando un tamaño que brinda un buen margen de seguridad a los robots.

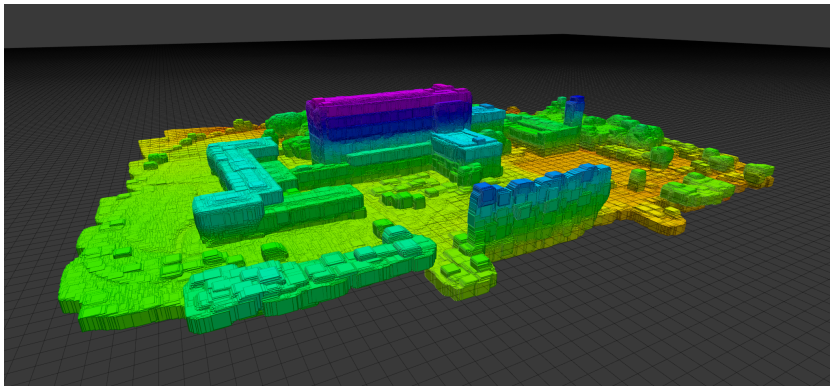

Figura 2.13: Facultad de Ingeniería representada como una grilla de ocupación. La altura de los cubos es representada con una escala de colores.

2.3. Revisión de Antecedentes

Los UAVs se han popularizado en la investigación académica y comercial debido a sus variadas aplicaciones. Durante las primeras instancias del trabajo, se realizó una investigación y análisis del estado del arte sobre drones y los conceptos y prácticas que los rodean. Entre lo investigado, se destacan la planificación de caminos, el control de drones utilizando el *framework ROS (Robot Operating System)* y el estudio de proyectos previos con similitudes temáticas.

2.3.1. Planificación de Caminos

Según Latombe (1991), todo sistema que lidie con agentes móviles necesita alguna estrategia para generar las instrucciones de movimiento. Cuando se quiere elaborar un sistema autónomo, se vuelve necesario brindarle a los robots la capacidad de realizar esta tarea. Al problema de generación de estas instrucciones se lo conoce como Planificación de Movimiento (del inglés *Motion Planning*) o Planificación de Caminos (*Path Planning*).

Para resolver este problema, existen diferentes acercamientos que pueden categorizarse según la forma de representar la realidad:

1. Hoja de ruta (*Roadmap*). Este acercamiento se basa en la construcción de una hoja de ruta en el espacio libre de obstáculos que sirve de caminos estandarizados. Por esto, la planificación puede reducirse en encontrar tres caminos, los dos que conectan al origen y al destino a la hoja de ruta y el camino estandarizado que conecta los otros dos puntos.

2. Descomposición de celdas. Estos métodos son los más estudiados y consiste en dividir el espacio libre de manera que conseguir un camino para dos posiciones dentro de una celda sea sencillo.

3. Campos de potencial. Se divide el espacio en una fina grilla de configuraciones y se busca en ella los caminos. El nombre surge de que se puede tratar al robot como una partícula que se mueve influenciada por un campo de potencial, con cada configuración generando una fuerza de atracción a ella.

Otras fuentes (Yang y cols., 2016) proponen otra categoría, los algoritmos basados en la naturaleza como los Algoritmos Evolutivos, Redes Neuronales u optimización por colonia de hormigas, que se inspiran en los seres vivos para obtener las soluciones a los problemas.

Todos estos algoritmos necesitan información acerca del entorno en el que el robot se moverá y sus obstáculos, que puede ser provista por el usuario, programas de diseño o ser obtenida mediante el uso de sensores. Si se tiene información completa del entorno, la planificación puede realizarse totalmente previa al movimiento del robot, mientras que en el caso contrario será necesario intercalar planificación y ejecución. De poseer una mayor incertidumbre, será importante realizar una planificación que tenga en cuenta varias contingencias.

El problema a solucionar puede tornarse más complejo, como por ejemplo por la presencia de múltiples agentes móviles, como otros robots u obstáculos, ya que podría introducir la necesidad de coordinar movimientos, o hacer que la planificación cambie con el tiempo. Otras razones incluyen la complejidad de las articulaciones y mecanismos que los agentes pueden presentar. Esto muestra claramente que la planificación de movimientos es en realidad un conjunto de problemas, que resultan ser en mayor o menor medida variaciones entre ellos (Latombe, 1991).

Planificación de Caminos de Cobertura

La planificación de caminos de cobertura, conocida como *Coverage Path Planning* (CPP), es definida como el procedimiento de calcular una ruta viable que garantice la inspección exhaustiva de una estructura específica (Almadhoun, Taha, Seneviratne, y Zweiri, 2019). Este enfoque considera un conjunto de puntos predefinidos que el UAV debe recorrer con el fin de cubrir por completo el área de interés y realizar una inspección integral.

Una de las tareas mas importantes en CPP es la generación de puntos para recorrer, o *viewpoints*. Estos *viewpoints* se utilizan para planificar el recorrido de un UAV de manera que se cubra la mayor parte del área de interés de manera eficiente. Cuanto más eficiente sea el camino creado a partir de los *viewpoints*, menor será el tiempo del recorrido.

Existen dos maneras de generar *viewpoints*, que dependen de si se basan o no en modelos. Los métodos basados en modelos dependen de un modelo de referencia previamente creado y procesado, mientras que los no basados en modelos no tienen conocimiento previo del ambiente y utilizan sensores para tomar decisiones y monitorear en tiempo real.

Una estrategia basada en modelos para generar *viewpoints* consiste en dividir la ubicación a inspeccionar en figuras geométricas, tales como cuadrados, triángulos o hexágonos.

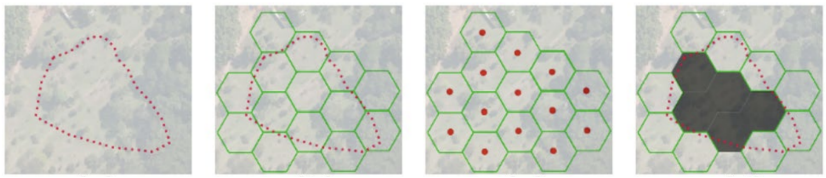

Figura 2.14: Pasos de la descomposición hexagonal de una ubicación.

Como se observa en la figura 2.14, el área es dividida en hexágonos y el centroide de cada hexágono es hallado, para finalmente descartar las zonas con obstáculos que no se pueden inspeccionar.

Existen varios enfoques para la generación de caminos de cobertura que se pueden clasificar, según Ahmed (2021), en: basados en grillas, geométricos, basados en recompensas, Next Best View (NBV) o incrementales aleatorios.

En la estrategia basada en grillas planteada por Avellar, Pereira, Pimenta, y Iscold (2015) se propone realizar una etapa de descomposición de un plano bidimensional en filas de barrido. Como se observa en la figura 2.15, la dirección de barrido (*Sweep direction*) óptima es la paralela a la dimensión lineal más chica de la zona.

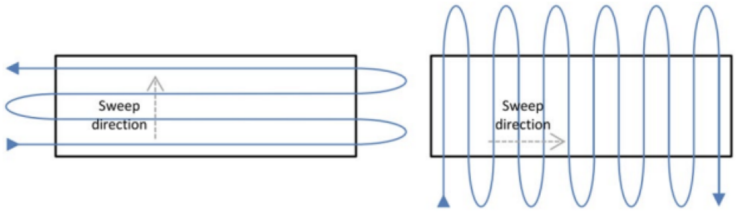

Figura 2.15: Descomposición de un plano en filas de barrido.

Los métodos geométricos utilizan grafos de visibilidad para generar el camino. Estos grafos incluyen un conjunto de puntos y obstáculos, con los nodos representando lugares y las aristas son caminos que no pasan por obstáculos.

Los enfoques basados en recompensas son usados en muchos trabajos recientes en CPP con varios UAVs debido a sus ventajas en capacidad de aprendizaje y procesamiento en paralelo. Los métodos que caen dentro de esta categoría más populares son Redes Neuronales, métodos inspirados por la naturaleza y algoritmos híbridos.

2.3.2. Proyectos Similares

El grupo de Sistemas Multi-Robot de la Universidad Técnica Checa (CTU MRS) investiga e implementa diversos sistemas robóticos complejos (2023). MRS se desenvuelve en la seguridad de sistemas críticos, monitoreo del ambiente por flotas de UAVs y estabilización de robots, entre otros.

Dentro de los proyectos desarrollados por dicho grupo, se destaca *"Dronument: Aerial Autonomy for Documentation of Historical Monuments"* [2]. Este utiliza una flota de UAVs autónomos para realizar una inspección del interior de edificios históricos.

Python Robotics (Haviland y Corke, 2022) es una colección de algoritmos para la robótica escritos en Python. En el repositorio de Zhou(2020) se implementan algoritmos de *Path Planning* conocidos y usados para la robótica en Python mientras que en el de Lu (2022) expone varios algoritmos de *Path Planning* y optimización de trayectorias en MATLAB. GAAS(2023) es un programa de código abierto diseñado para el vuelo autónomo de UAVs. *XTDrone* (2023) es un simulador de UAVs basado en PX4, ROS y Gazebo. Soporta una amplia gama de UAVs y ha implementado varias funciones como detección y seguimiento de objetos y planificación de caminos en dos y tres dimensiones, entre otros.

En la Facultad de Ingeniería también se han realizado otros proyectos de grado con temática de robótica, tal como en el de Cabrera, Castro y Salmini

[2] `https://mrs.felk.cvut.cz/projects/dronument`

(2021) que se centra en la implementación de un enjambre de múltiples drones navegando de forma autónoma, para el cual se desarrollaron acciones de control que involucran planificación de trayectorias, algoritmos de seguimiento y de evasión. En el proyecto de Menéndez, Bruzzone, Salmantón y Corazza (2022) se presentan 3 estrategias para equipos de múltiples robots móviles, una estrategia con enfoque cooperativo y dos con enfoque independiente.

Capítulo 3

Implementación de la solución

Índice

En esta sección se describe la solución implementada. Se brinda una descripción de cómo el proyecto fue gestionado y las decisiones clave tomadas a lo largo del proyecto, y detalles del análisis y diseño de la solución. Además, se detalla la implementación del modelo tridimensional de la estructura, la planificación de los caminos, los nodos ROS y la integración de estos aspectos en el simulador, así como detalles de la interfaz de usuario y el *backend* de la aplicación. Por último, se detalla la implementación de los requerimientos no funcionales del proyecto.

3.1. Gestión del Proyecto

Se utilizó la metodología Kanban[1]. Para ello, se tuvieron reuniones semanales entre el grupo de forma tal de identificar y agregar tareas a la pila de trabajo. Para la gestión del proyecto se utilizó Notion (Ver Anexo B.1.3).

Por otro lado, se utilizó GitLab para el manejo de versiones del código (Ver Anexo B.1.3). En ésta línea, se siguieron ciertos estándares de calidad del código. Se utilizó el linter ESLint[2] para seguir buenas prácticas de código y, además, se realizaron revisiones de código entre miembros del equipo, de forma tal de mantener la prolijidad del código.

Se creó una página web (Fripp, Queirolo, Santellán, y Valentín, 2023) utilizando GitLab Pages donde se puede encontrar información sobre el proyecto.

3.2. Análisis y Diseño

3.2.1. Requisitos

Con el objetivo de obtener los requisitos se estuvo en contacto con Adrián Santos, asistente académico y Director de Plan de Obras y Mantenimiento de la Facultad de Ingeniería. Luego de la primera entrevista se realizó un documento de requerimientos, que fue validado y llevó a la creación de una segunda versión. Los requerimientos acordados entre el equipo del proyecto y el cliente son los siguientes.

Requisitos funcionales

- Crear un sistema de flota de drones que permita recorrer lugares y grabar videos de los mismos.
- Crear una interfaz gráfica que permita seleccionar diferentes lugares previamente establecidos para recorrer.
- Iniciar recorridos de los drones utilizando la interfaz gráfica.
- Poder monitorear inspecciones en tiempo real.

[1] `https://swsblog.stanford.edu/blog/agile-project-management-and-its-flavors-where-does-scrum-end-and-kanban-begin.html`

[2] `https://eslint.org/`

- Las inspecciones deben ser cancelables en todo momento.
- El sistema debe alertar al usuario en caso de que las condiciones climáticas no sean viables realizar el vuelo.

Requisitos no funcionales

- No requerir de entrenamiento en manejo de drones para utilizar el sistema.
- Garantizar la seguridad física de las personas que acuden a la Facultad de Ingeniería.
- Garantizar la eficiencia de los caminos generados.
- Garantizar vuelos sin colisiones.
- Se debe tener mecanismos que prohíban ataques maliciosos al sistema.
- El sistema debe ser extensible a otros entornos.
- La interfaz gráfica debe ser intuitiva.

Otras funcionalidades de interés que no son parte del alcance del proyecto:

- Realizar verificaciones previas al vuelo como batería, almacenamiento interno, conexión con la GCS.
- Reasignar tareas de los drones en tiempo real en caso de falla.
- Hacer la inspección con cámaras térmicas.
- Detectar anomalías en la fachada o techo en tiempo real, mediante técnicas de visión por computadora.

3.2.2. Diseño

Con el objetivo de facilitar el mantenimiento y la extensión del sistema se optó por diseñar el sistema de forma modular, donde cada componente tiene sus responsabilidades claras y acotadas. En la Figura 3.1 se muestran los componentes utilizados.

Estos componentes se pueden encontrar distribuidos físicamente en distintas computadoras o en la nube (Figura 3.2), e interactúan entre sí a través de internet.

Los componentes de interfaz de usuario y API se muestran en cajas separadas ya que se encuentran dentro del mismo servidor pero es posible separarlos.

En el proceso de desarrollo, debido a que el entorno fue simulado, todos los componentes (a excepción del de base de datos que se encuentra en la nube) fueron ejecutados en la misma computadora. Esto es posible debido a las abstracciones para el manejo de comunicación entre nodos sobre IP que brinda ROS.

El diagrama de flujo de la Figura 3.3 muestra cómo es el proceso necesario para obtener un conjunto de caminos validos, partiendo de la entrada del usuario: un conjunto de lugares y otro de drones. En el mismo se distingue una sección que es el Planificador de Transiciones. Este componente se distingue del resto por ejecutarse de forma *offline*, es decir, en un momento anterior al resto

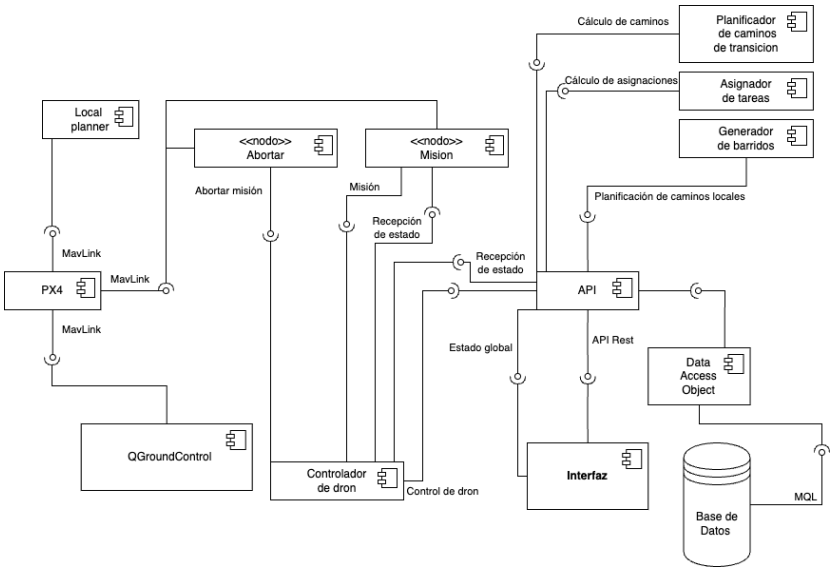

Figura 3.1: Diagrama de componentes del sistema

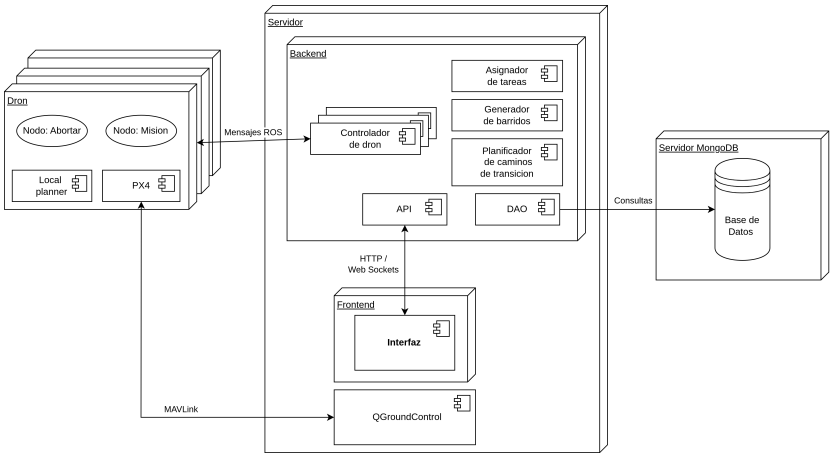

Figura 3.2: Diagrama de despliegue del sistema

del flujo planteado. La ejecución del Planificador de Transiciones forma parte del proceso de configuración inicial del sistema, para habilitar su uso. El resultado del Planificador de Transiciones es guardado en la base de datos, siendo éstos los caminos generados y las distancias de los mismos para la matriz de costos.

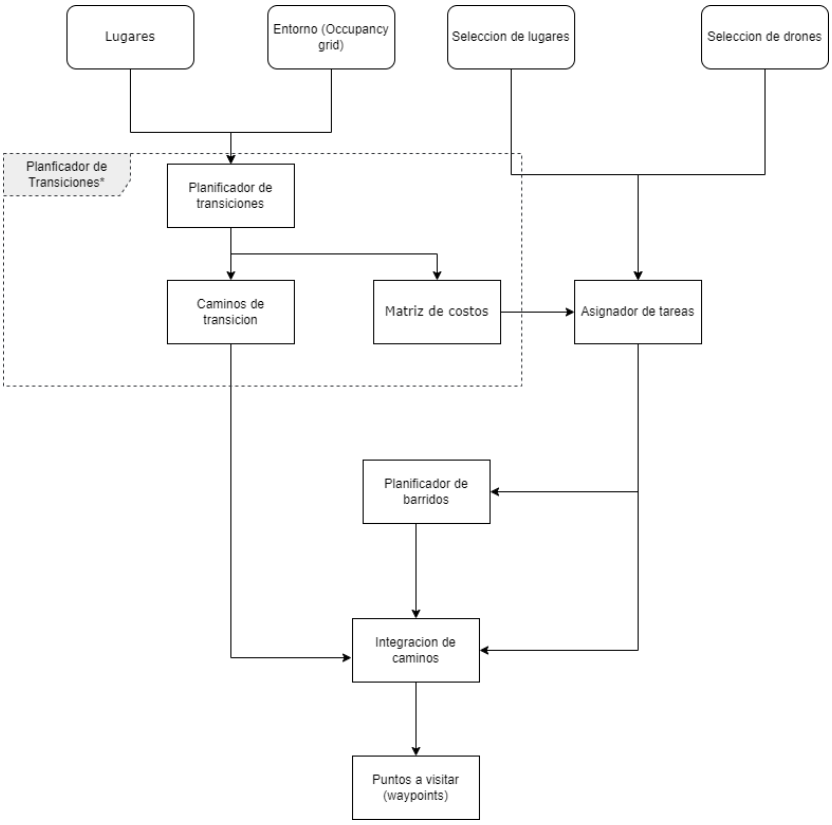

Figura 3.3: Diagrama de flujo de los componentes para la generación de caminos

Por otro lado, de forma tal de poder vincular los diagramas realizados anteriormente, con la ejecución de una misión se realizó un diagrama de secuencia (Figura 3.4). Por temas de simplicidad en el diagrama se omitieron ciertos componentes, como el controlador del dron. Una vez que se obtiene el plan de misión, dicho componente envía frecuentemente información del dron a la API, tal como la posición actual, que luego será retornada a la interfaz de usuario.

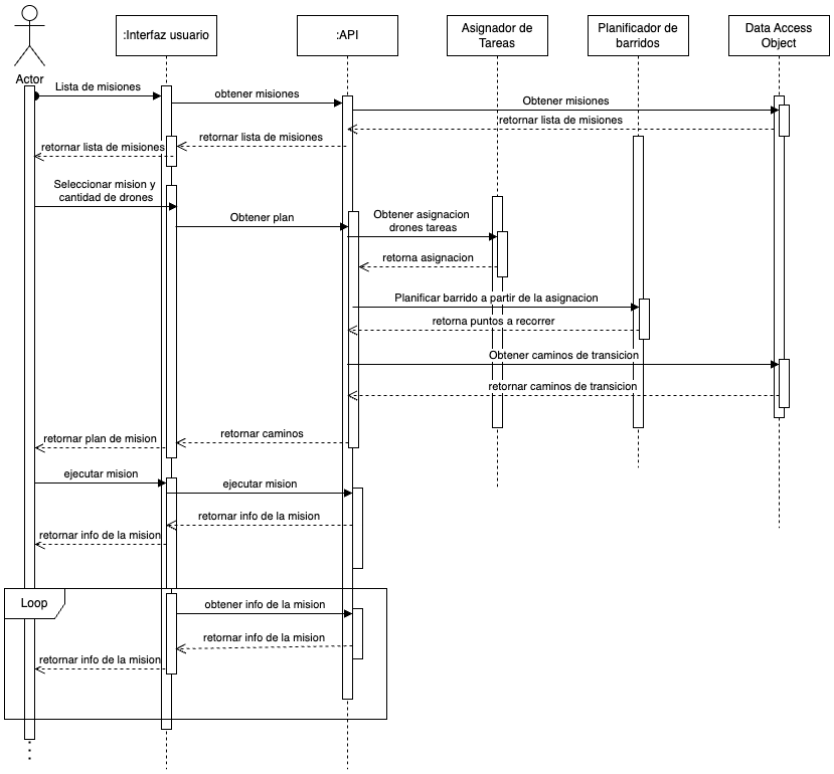

Figura 3.4: Diagrama de secuencia para el caso de uso ejecutar misión

3.3. Modelo Tridimensional de la Estructura

Se utilizó un dron para tomar fotos y obtener un modelo tridimensional de la Facultad de Ingeniería. A pesar de que ya existían modelos tridimensionales de la facultad al iniciar el proyecto, ninguno se adecuaba completamente a las necesidades del equipo. Es por esto que se contactó a un estudiante de Agrimensura, Mauro Arzuaga, que ya había creado un modelo de la facultad usando fotogrametría. El escaneo del edificio y su entorno cercano fue realizado utilizando sensores *LiDAR* y cámaras, siendo su resultado una nube de puntos en formato LAS, y un modelo en formato OBJ con su respectiva textura en formato mapa de Albedo.

Si bien el modelo resultante era correcto, fue necesario modificar su posicionamiento para facilitar su integración con el sistema. Con el objetivo de poder utilizar coordenadas globales, se debió mover tanto el modelo como la nube de puntos para posicionar el origen relativo del modelo en un punto conocido. Este punto fue elegido estratégicamente para que sea fácil obtener su latitud y longitud. El mismo se ubica en la intersección de las fachadas Norte y Este del Salón

de Actos, junto con el piso. Esta tarea se realizó con el programa *Blender*[3].

Por otra parte, la nube de puntos obtenida originalmente era demasiado densa para lo que se necesitaba. Se optó por su simplificación, haciéndose un recorte de una gran cantidad de puntos que no aportaban mas información. El resultado fue una reducción cercana a 100 veces su anterior tamaño. Esta simplificación se realizó con la herramienta *Subsample* de *CloudCompare*[4]. A su vez, se exportó la nube resultante en formato *Point Cloud Data* (PCD), que es el formato estándar y abierto[5].

A partir del modelo obtenido se produjo un octree con el objetivo de utilizarlo como entrada para la evitación de obstáculos provista por *PX4 Avoidance* en su *Global Planner*. Luego de repetidos intentos, no se logró implementar correctamente debido a la poca flexibilidad del paquete con respecto a la modificación de los marcos de referencia, por lo que no fue posible desacoplar la posición del octree de la del dron al momento de conectarlo. Por este motivo se decidió utilizar una implementación de A* que recibe como entrada una grilla de ocupación generada a partir de la nube de puntos obtenida en el vuelo exploratorio.

3.4. Definición de lugares de inspección

Las zonas de inspección fueron modeladas como un conjunto de vértices "*waypoints*", que denotan los vértices del área a inspeccionar, y un nombre.

La generación de los puntos fue hecha mediante un proceso de inspección manual en el entorno simulado utilizando un dron. Particularmente, se optó por controlar el vuelo del dron utilizando un *joystick*. Se posicionó al dron en los vértices de los lugares a inspeccionar, y se consultó su posición a través de los mensajes con información de posicionamiento de MAVLink que son constantemente publicados en el tópico de ROS "global_position/global"[6]. Una vez sensados estos datos, fueron guardados en el arreglo "*waypoints*" de la representación correspondiente del lugar en la base de datos. Este mecanismo no fue automatizado dentro del sistema, por lo que para la adición de nuevos lugares, o la modificación de los mismos, debería resolverse con el mismo método o similares. Se realizó esta tarea para todos las zonas de inspección de interés (ver Anexo C.1).

3.5. Planificación de Caminos

Para la planificación de rutas en el contexto del proyecto, se identifican tres problemas independientes. Al primero de estos problemas, lo llamaremos "asignación de tareas" y se refiere a la asignación estratégica de los drones a los lugares elegidos para inspeccionar.

[3] `https://www.blender.org/`
[4] `https://www.cloudcompare.org/`
[5] `https://pointclouds.org/`
[6] `https://wiki.ros.org/mavros#mavros.2FPlugins.Published_Topics-2`

El segundo problema, en adelante “planificación local”, se enfoca en la planificación del mejor camino de inspección de cada lugar por parte de un dron.

Por último, al tercer problema lo llamaremos “planificación de transiciones”. Éste se concentra en determinar la manera más eficiente de desplazarse de un lugar a otro sin colisionar con el entorno conocido previo al vuelo.

A continuación, detalla la solución de estos problemas, y luego el método para combinarlos.

3.5.1. Planificación de Transiciones

Como se mencionó anteriormente, la planificación de transiciones atiende el desafío de diseñar una buena ruta para permitir que el dron alcance cada lugar. Dicha ruta debe evitar los obstáculos conocidos del entorno como los edificios y los árboles. Para abordar esta problemática, se ha optado por utilizar una implementación del algoritmo A*, conocido por su capacidad para determinar, en un tiempo computacional razonable, una trayectoria de costo mínimo entre dos nodos en un grafo ponderado. La implementación utilizada se basa en una grilla de ocupación, que se obtiene de la representación tridimensional de la Facultad de Ingeniería.

El algoritmo usado requiere la definición de una heurística que ayude a llegar a la solución. En este caso, se ha optado por usar la distancia euclidiana, que representa la distancia entre los puntos o nodos en línea de vista.

Los orígenes y destinos de los caminos y los obstáculos que deben sortear los drones no son muy cambiantes ya que se tratan de puntos relacionados a los edificios y los mismos edificios respectivamente. Es por esto que se implementó la persistencia de los caminos obtenidos por el algoritmo en la base de datos, permitiendo al sistema consultar por caminos previamente generados en lugar de ejecutar el algoritmo en cada una de las planificaciones de inspecciones, ahorrando complejidad en las mismas.

3.5.2. Planificación Local

Una vez que el UAV ha llegado al lugar a inspeccionar, se debe realizar una inspección grabando un video para su posterior análisis. El recorrido del UAV debe ser eficiente, de forma de optimizar el uso de su batería. Se distinguen tres tipos distintos de lugares de inspección, según las indicaciones del usuario final, que por sus características fueron abordados con diferentes estrategias: canalones, techos y fachadas.

Para crear el camino de puntos, o *waypoints*, que el UAV debe recorrer se implementó un *script* en Python. Este *script* toma como entrada los puntos que constituyen a los vértices del techo, junto con la posición actual o esperada del dron, a la que se le llama `comingFrom`, de manera de comenzar el barrido en el punto más cercano a esta. A los efectos de las ejecuciones de misiones, `comingFrom` siempre es un punto de una zona de inspección, por lo que la distancia es calculada utilizando los caminos generados por el planificador de transiciones.

Canalones

La planificación de inspección de canalones resulta una tarea trivial, ya que éstos se encuentran sobre el perímetro de las zonas de inspección, por lo que la lista de puntos a recorrer será la misma que define a la zona, comenzando por el punto más cercano a `comingFrom` (Figura 3.5).

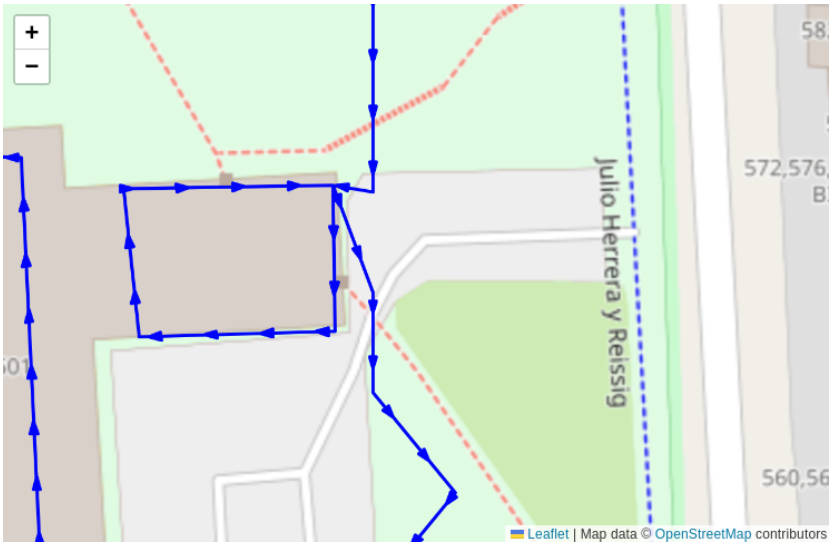

Figura 3.5: Barrido creado por el planificador local para el canalón del techo del Salón de Actos

Techos

Se representa a un techo como un cuadrilátero. Se usará una estrategia de barrido en donde el dron irá de lado a lado recorriendo todo el plano. Para recorrer este plano, se intentará disminuir la cantidad de veces que el dron realiza una curva, ya que aquí es donde el dron demora mas tiempo y debe acelerar más, lo que consume más energía. Por lo tanto el UAV irá en dirección paralela al largo del rectángulo como se puede observar en la figura de abajo.

Figura 3.6: Dirección de barrido de un techo

Para construir la lista de puntos, se particionan los dos lados más cortos del cuadrilátero de manera que se respete una separación máxima de forma de garantizar la buena cobertura de toda el área con los videos. Esta distancia es calculada en base a la separación del robot a la zona de inspección (d) y el ángulo de la cámara (α). Con estos dos valores, ya que se usa una cámara que apunta hacia abajo, se puede obtener la distancia horizontal que cubre la imagen (f) usando la ecuación 3.1 que se deduce de la Figura 3.7 a la que se multiplica por una constante menor a uno (c) para obtener la distancia de separación de barridos (s) (ecuación 3.2). El parámetro de ángulo de la cámara, la constante de separación y la distancia a la superficie son configurables en código.

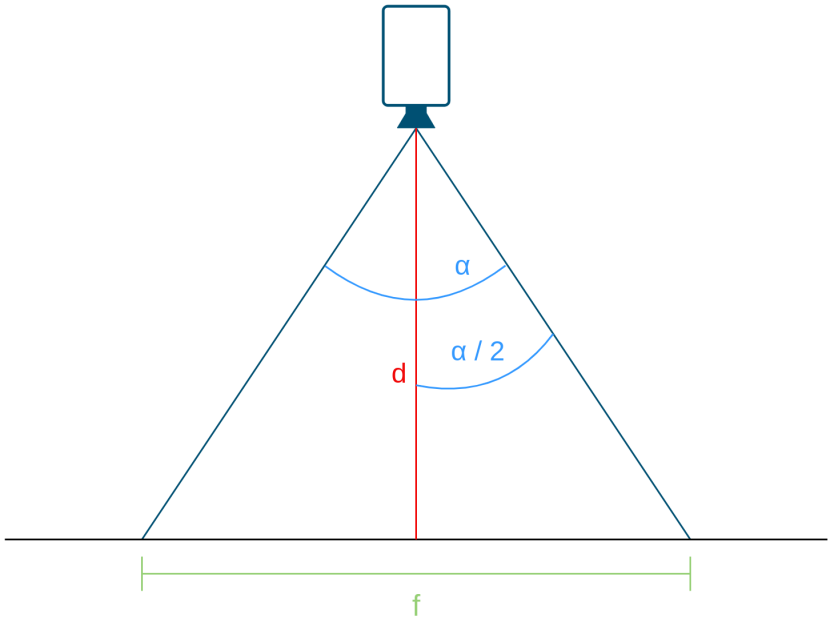

Figura 3.7: Representación del ángulo de la cámara apuntando a una superficie

$$f = 2 * \tan(\alpha/2) * d \tag{3.1}$$

$$s = f * c \tag{3.2}$$

Con el objetivo de que la implementación sea más general, el planificador local no restringe la forma de la zona por fuera de la cantidad de vértices. Esto hace más complicada la planificación, ya que debe poder hacer el cubrimiento de zonas irregulares como trapecios. Para resolver esto, las particiones se hacen en secuencia, comenzando con el segundo lado más corto y siguiendo con el más corto, particionando el primero en segmentos de largo igual a la distancia de separación y luego el segundo en la misma cantidad de segmentos que el

anterior, garantizando que la separación en el lado más corto es a lo sumo igual a la configurada. El último paso de la planificación consiste en intercalar en una lista los extremos de los segmentos obtenidos para cada lado, generando así un recorrido serpenteante (Figura 3.8).

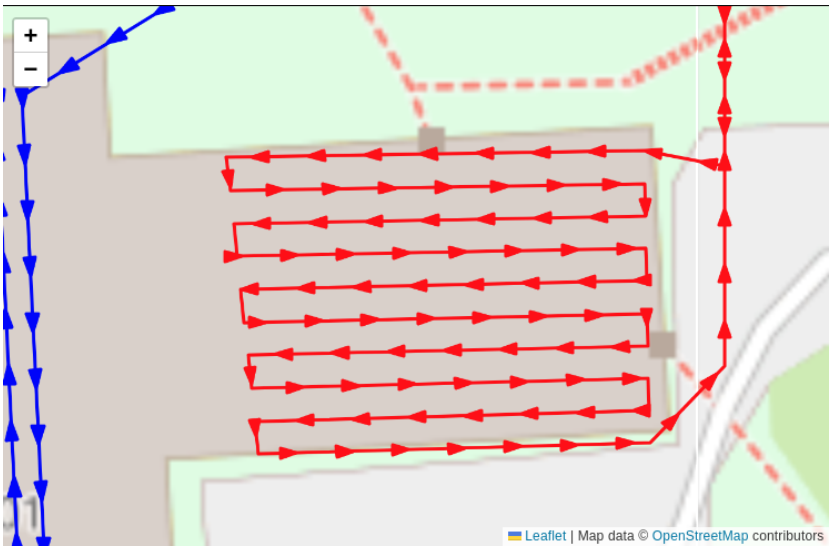

Figura 3.8: Barrido creado por el planificador local para el techo del Salón de Actos

Todas las particiones fueron realizadas con una interpolación lineal, que trae como ventaja la compatibilidad con techos inclinados, ya que genera un barrido que acompaña la inclinación que definen los vértices. Por esto mismo, no resulta posible acompañar las inclinaciones de techos de múltiples aguas, ya que no se tiene información adicional al perímetro, pero tiene solución al marcar las aguas como zonas de inspección independientes.

Fachadas

El barrido de una fachada es muy similar al caso de los techos, de hecho, el barrido resultante tiene la misma forma pero vertical, si uno lo viera de frente vería el mismo patrón. Que la superficie a recorrer sea vertical trae otro aspecto a consideración. En este caso, no todas las direcciones de movimiento tienen el mismo costo energético para los robots. Dada la forma en que los UAVs generan su empuje vertical, es más costoso para ellos ganar altitud que perderla, por lo que la primera decisión es que el barrido se va a hacer en sentido horizontal sin importar la dimensión de la fachada, ya que resulta más económico realizar más curvas frente a más ascensos y descensos. Siguiendo el mismo razonamiento, la planificación siempre hará que el dron tenga que descender durante la inspección, por lo que esta comenzará en uno de los puntos más altos de la fachada. En una

vista vertical (Figura 3.9), el barrido se ve como una línea.

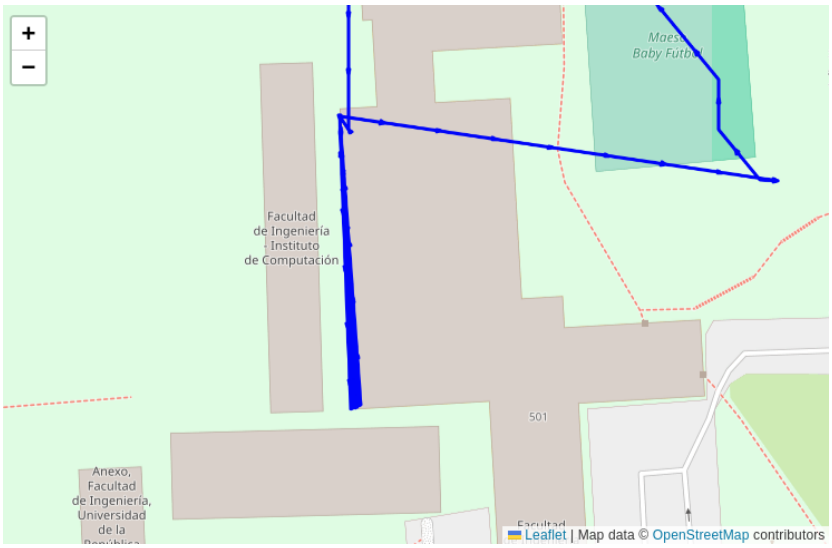

Figura 3.9: Barrido de fachada creado por el planificador local para la fachada este de la Biblioteca

3.5.3. Asignación de tareas

El problema de la asignación de zonas a inspeccionar a los drones fue separado en tres casos, dependiendo de la cantidad de drones y lugares elegidos. Siendo n el numero de drones elegidos y m la de lugares a inspeccionar, los casos son:

1. Igual cantidad de drones que lugares ($n = m$)
2. Más drones que lugares ($n > m$)
3. Menos drones que lugares ($n < m$)

La asignación de la solución a implementar se da según que condiciones se cumplen, en el orden que se listaron.

Las primeras dos soluciones son triviales. La asignación de las tareas es directa, siendo una correspondencia de dron a lugar. Esto se realizó sin distinción del tipo de dron, ya que se asumió que las condiciones de los mismos son iguales.

En el caso de tener más drones que lugares seleccionados, se resuelve de igual forma, pero dejando al menos un dron sin asignación. De todas formas, este caso no es presentado en la interfaz de usuario, ya que se limita la cantidad de drones según la cantidad de lugares elegidos.

A continuación, se presenta el problema con más transcendencia dentro del problema de la asignación de tareas. Se modeló el problema de asignación de

drones a lugares como un problema MTSP, en donde cada dron se equipara conceptualmente a un vendedor en el contexto del problema, y las zonas a inspeccionar se corresponden a las ciudades a visitar.

A continuación se detallará como se implementó la solución a este problema.

Multiple Traveling Salesman Problem

Para poder resolver el problema de MTSP se utilizó la Programación Lineal Entera (PLE) para definir el problema. La formulación de MTSP usando PLE se puede encontrar en la sección (2.2.3) del Marco de Trabajo de este documento. Dada la naturaleza del problema es necesario el uso de una herramienta para realizar el modelado del problema. Para ello se utilizó CVXPY[7], una biblioteca de Python para modelar y resolver problemas de optimización convexa. La PLE es un caso especial de la optimización convexa, ya que las restricciones de enteros se pueden representar mediante un conjunto de restricciones convexas.

Se utilizó una implementación propuesta en un repositorio de la biblioteca para la resolución del problema. En el mismo, se traslada la definición matemática presentada en el Marco Teórico, a través del uso de PLE, a un conjunto de definiciones de variables, restricciones y la función objetivo.

Es necesario el uso de un *solver* para resolver los problemas de programación lineal. Un *solver* es un programa especializado que se utiliza para encontrar el valor óptimo de una función sujeta a un conjunto de restricciones. La biblioteca no incluye uno propio, sino que se integra con varios externos, de forma de enfocarse en proporcionar una interfaz de alto nivel y expresiva para definir los problemas de optimización, delega la tarea de resolver el problema. Considerando la lista de *solvers* disponibles se había optado por usar MOSEK[8], debido a que es capaz de resolver problemas de estas condiciones. En una etapa más tardía del desarrollo, debido al manejo de licencias que se debía hacer para poder usar esta herramienta, se decidió utilizar el *solver* SCIPY[9]. Éste es de código abierto y es capaz de resolver problemas con las restricciones planteadas, por lo que no generó ninguna desventaja.

La introducción de múltiples agentes trae consigo una nueva problemática: el balance de cargas. Una solución balanceada tiene varias ventajas aplicables a nuestro problema. Una de ellas es el tiempo de ejecución total de la misión: una distribución equitativa de zonas de inspección puede ayudar a la minimización del tiempo final, en comparación con una distribución donde algunos drones quedan inactivos. Si bien esta estrategia implica recorrer una distancia total mayor, la contraparte no es una forma óptima de utilizar los recursos, ya que no se paralelizan las tareas todo lo posible. Una ventaja de la paralelización que puede pasar desapercibida es que el desgaste de los drones es repartido, disminuyendo su necesidad de mantenimiento y alargando su vida útil.

Para ayudar a balancear las soluciones alcanzadas por el MTSP, se exploraron algunas alternativas. Investigando sobre el tema, se encontró un artículo

[7] `https://www.cvxpy.org/`
[8] `https://www.mosek.com/`
[9] `https://scipy.org/`

que explora el balanceo de MTSP y a partir de la distinción por vendedor de las variables x_{ij}, crea una nueva función objetivo que se define como la suma de costos de los *tours*, que representan los ciclos que recorre cada vendedor (de Castro Pereira, Solteiro Pires, y de Moura Oliveira, 2023). Los investigadores de este artículo resuelven el problema utilizando la técnica *Ant Colony Optimization* y se realizó un intento de modelar el problema en PLE. Dicho intento falló por una restricción en las dimensiones de las variables que permite CVXPY ya que era necesario agregar una dimensión a la variable de la función objetivo de MTSP y la biblioteca no soporta esa cantidad de dimensiones. CVXPY permite, sin embargo, resolver problemas multi-objetivo.

Luego de ese intento, se optó por modificar las restricciones de la implementación para condicionar el espacio de soluciones. En la formulación del problema se usa una variable artificial u_i que simboliza el orden en el que el lugar i fue visitado por un dron. Agregando una cota superior al valor de u_i se logra limitar la cantidad de lugares que puede visitar un dron, o el largo de su ciclo. Al asegurar una cota balanceada, la asignación de cantidad de lugares a cada dron resulta también balanceada. Esta modificación ocurre en la ecuación 3.8, donde se cambia la cota superior del largo de los ciclos de $n-1$ a $\lceil (n-1)/m \rceil$. A continuación se plantea la formulación del problema nuevamente para facilidad de lectura.

$$\text{mín} \sum_{i=1}^{n} \sum_{j \neq i, j=1}^{n} c_{i,j} x_{i,j} \tag{3.3}$$

$$\sum_{i=2}^{n} x_{i,1} = m \tag{3.4}$$

$$\sum_{j=2}^{n} x_{1,j} = m \tag{3.5}$$

$$\sum_{i=2, i \neq j}^{n} x_{i,j} = 1 \qquad j = 1, \ldots, n; \tag{3.6}$$

$$\sum_{j=2, j \neq i}^{n} x_{i,j} = 1 \qquad i = 1, \ldots, n; \tag{3.7}$$

$$u_i - u_j + 1 \leq \lceil (n-1)/m \rceil (1 + x_{i,j}) \qquad 2 \leq i \neq j \leq n; \tag{3.8}$$

$$0 \leq u_i \leq n \qquad 2 \leq i \leq n; \tag{3.9}$$

$$x_{i,j} \in \{0, 1\} \qquad i, j = 1, \ldots, n; \tag{3.10}$$

$$u_i \in \mathbf{Z} \qquad i = 2, \ldots, n. \tag{3.11}$$

Matriz de Costos

Para abordar la resolución de MTSP, es necesario modelar la realidad mediante un grafo conexo, donde los lugares a inspeccionar se representan como nodos y las aristas se ponderan con el costo de desplazarse de un lugar a otro. La implementación de la solución tiene como entrada a una matriz $C_{2\text{x}2}$ donde el elemeto C_{ij} representa el costo de trasladarse del lugar i al j. Esta matriz se ha denominado matriz de costos.

Cada entrada C_{ij} de la matriz se construye en nuestro caso como la distancia a recorrer en el trayecto de i hacia j. El problema modelo no tiene en cuenta el tamaño de las ciudades a recorrer, la distancia entre dos zonas es única. En la realidad, no obstante, la distancia entre dos zonas variará dependiendo de los vértices de partida y llegada. Para poder representar la realidad en una forma aplicable al problema, se debió elegir uno de estos trayectos para obtener el costo de la arista, que en este caso fue el de distancia mínima. Por este motivo, es posible que la planificación obtenida usando esta aproximación en los costos difiera ligeramente de la óptima, lo que se verá en la sección 4. Los caminos son obtenidos de la base de datos, ya que fueron previamente calculados por el planificador de transiciones.

$$a_{ij} = \text{distancia}(i, j)$$

3.5.4. Combinación de caminos

Luego de resolver cada uno de los problemas anteriores, se está en condiciones de realizar una planificación completa de una misión de inspección. Para ello, las soluciones a estas planificaciones más pequeñas son combinadas por nuestro componente *Router* de la siguiente manera:

En primer lugar, con los lugares seleccionados, se ejecuta el asignador de tareas para conseguir los lugares que cada dron debe visitar y su orden. Luego, se itera sobre cada asignación para conseguir de la base de datos el camino hacia el siguiente lugar y el camino de inspección. Como la generación de los caminos de inspección depende únicamente del punto de entrada a la zona, el punto de salida de la misma queda definido independiente de la siguiente tarea. Esto se traduce en que el camino de transición siguiente se selecciona en este momento, lo que permite tomar el mejor hacia la siguiente zona. Debido a esto, el costo real de la misión puede diferir ligeramente del costo planificado en la asignación de tareas.

3.6. Interfaz de Usuario

Se desarrolló una interfaz de usuario en React que permite ejecutar las diferentes funcionalidades del sistema, presentadas a continuación.

3.6.1. Funcionalidades

Iniciar Sesión

Antes de poder acceder a la pagina principal de la aplicación, el usuario deberá autenticarse. Para esto deberá ingresar su correo electrónico y contraseña (Figura 3.10).

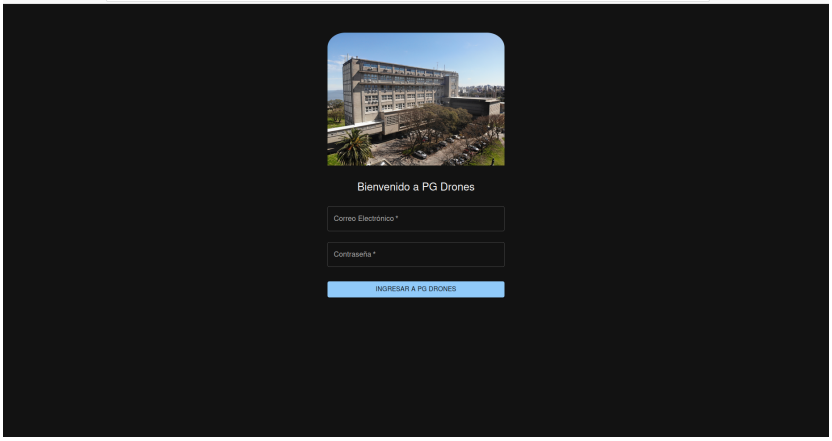

Figura 3.10: Página de inicio de sesión

Se introdujo un mecanismo de autenticación con el propósito de ofrecer una plataforma segura y confiable para el usuario. El usuario deberá realizar un proceso de inicio de sesión en el sistema previo a utilizarlo. Una vez que el usuario se ha autenticado, su sesión permanecerá activa incluso en caso de cerrar el navegador. No obstante, transcurrido un período determinado, la sesión de autenticación expirará, requiriendo así que el usuario proceda a autenticarse nuevamente. Esta funcionalidad se ha implementado empleando *JSON Web Tokens* (JWTs)[10].

Cerrar Sesión

Para poder cerrar la sesión, el usuario deberá hacer clic en el icono de *avatar*, situado en el encabezado de página, y posteriormente deberá hacer clic en el botón "Cerrar sesión".

Crear Misión

Para crear una misión, el usuario debe acceder a la página principal (Figura 3.11) y luego hacer clic en el botón "Crear Misión", donde será redirigido a la página de Crear Misión (Figura 3.12).

[10] `https://jwt.io/introduction`

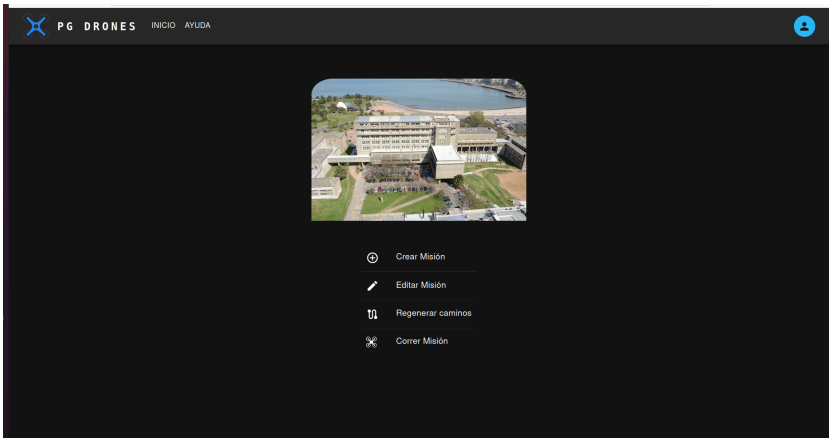

Figura 3.11: Página principal

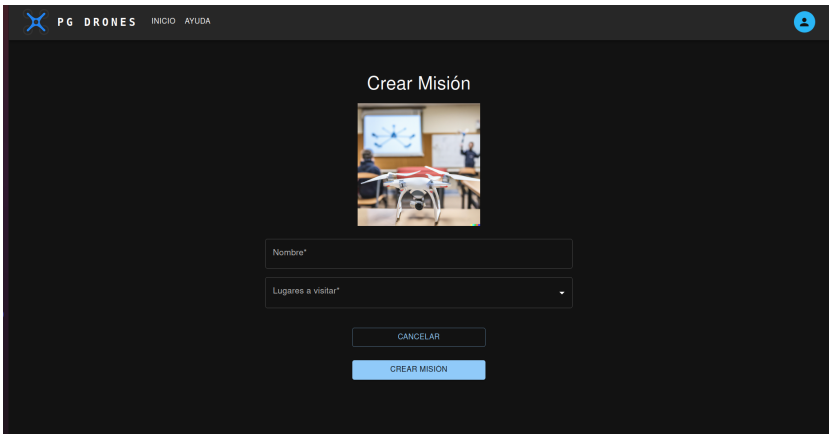

Figura 3.12: Página Crear Misión

Allí, deberá proporcionar el nombre de la misión y una lista no vacía de lugares a visitar, ambos campos son obligatorios y se indican como tal en la interfaz.

Los lugares que la misión visitará se seleccionan de una lista predeterminada brindada por el usuario final en la sección C.1.

Una vez que el usuario confirma que la configuración de la misión es correcta, se mostrará una ventana modal (Figura 3.13) que indicará si la creación de la misión se realizó con éxito o si se produjo algún error en el proceso.

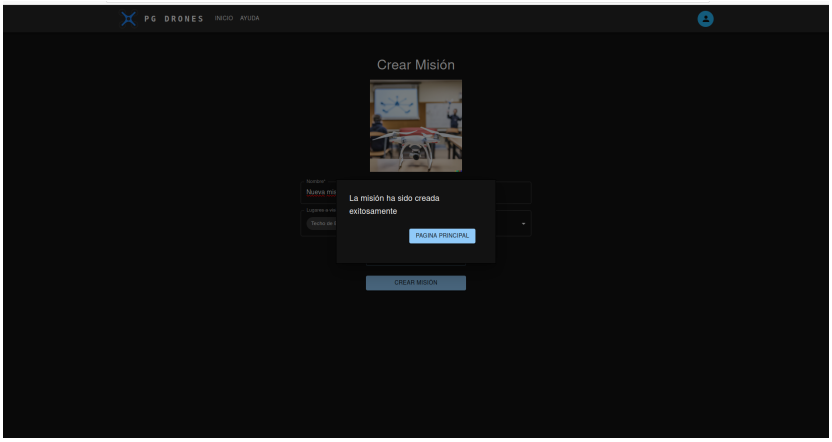

Figura 3.13: Modal que indica el éxito al agregar una misión

Editar Misión

Para editar una misión existente, el usuario debe acceder a la página principal (Figura 3.11) y luego hacer clic en el botón “Editar Misión”. Luego, el usuario será redirigido a una página donde se mostrará un listado de todas las misiones del sistema (Figura 3.14).

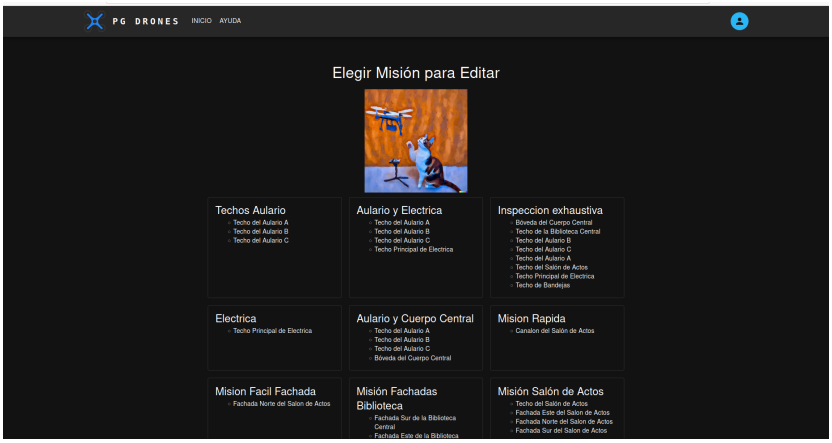

Figura 3.14: Lista de misiones para editar

Al seleccionar una misión para editar, se presentará la página de editar misión (Figura 3.15), donde el usuario puede modificar el nombre y los lugares de la misma.

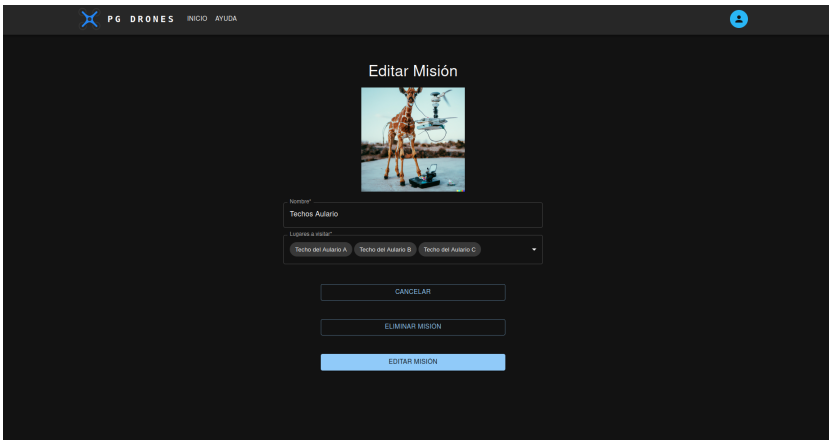

Figura 3.15: Página de editar misión

Una vez confirmados los cambios, se mostrará una ventana modal indicando si la edición se realizó con éxito o si se produjo algún error durante el proceso.

Ejecutar y supervisar misión

Para ejecutar una misión, el usuario debe hacer clic en el botón "Correr misión", desde la página principal (Figura 3.11). A continuación, deberá elegir la misión que desea ejecutar (Figura 3.16) y posteriormente indicar la cantidad de drones que se utilizarán (Figura 3.17).

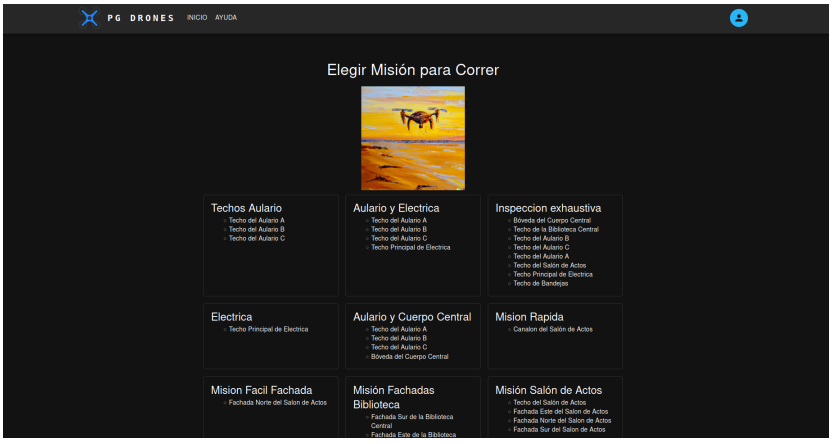

Figura 3.16: Página seleccionar misión para ejecutar

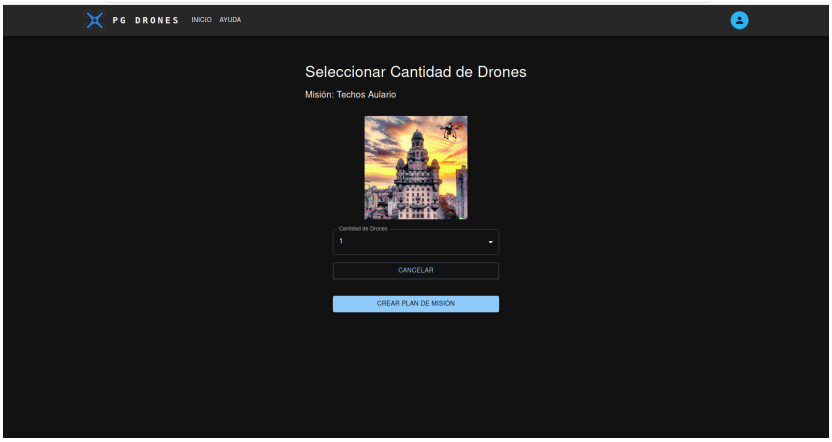

Figura 3.17: Página para seleccionar la cantidad de drones a utilizar en la misión

Con esa información, el servidor resolverá la asignación de tareas para cada dron y su respectivo camino, y se le presentará el plan de ejecución al usuario, mostrando el camino que recorrerá cada dron (Figura 3.18).

Figura 3.18: Página que muestra la planificación de la misión

El usuario podrá aceptar el plan de misión o cancelar la misión. En caso de aceptar el plan, el usuario será redirigido a la página de supervisar misión (Figura 3.19), donde podrá finalmente iniciar la misión, y además, donde se mostrará información detallada sobre el progreso de la misma.

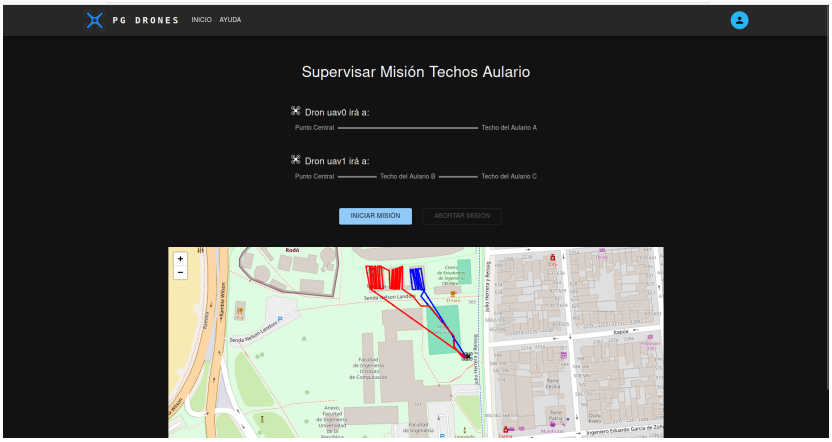

Figura 3.19: Página de supervisar misión, misión aún sin iniciarse

Durante la ejecución de la misión, el usuario podrá seguir su progreso, observando qué dron está inspeccionando cada lugar en tiempo real, y además se podrá observar la posición actual de cada dron (Figura 3.20).

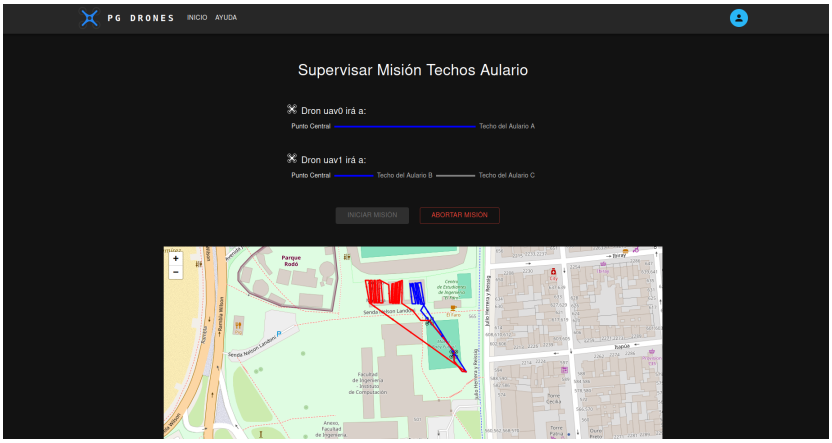

Figura 3.20: Página de supervisar misión, misión en progreso

Cuando la misión haya finalizado, se le notificará al usuario por medio de una ventana modal.

Abortar misión

El usuario tiene la posibilidad de abortar una misión en progreso. Para ello debe hacer clic en el botón “Abortar misión” en la página de supervisar misión (Figura 3.20).

Reanudar supervisión de misión en curso

En caso de que la página web sea interrumpida, o se abandone la página de supervisar misión, el usuario tendrá la capacidad de volver a supervisar la misión que se encuentra en progreso. Para ello el usuario debe hacer clic en el botón de "Supervisar misión en progreso", que se encuentra en el encabezado de la página (Figura 3.21)

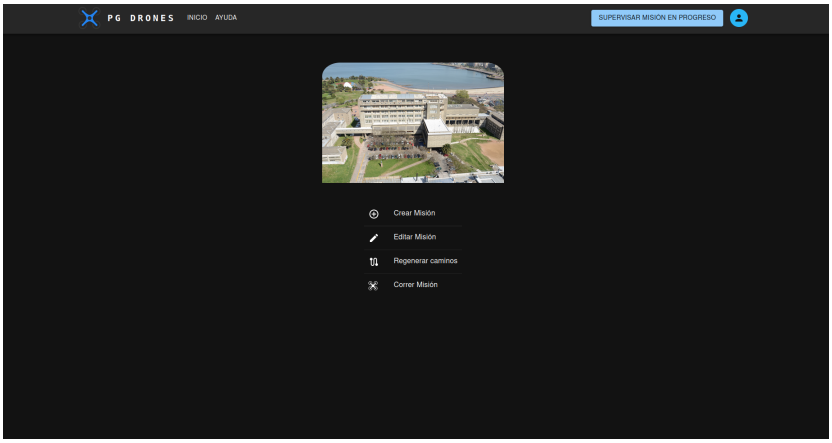

Figura 3.21: Supervisar misión en progreso

Regenerar caminos

El objetivo de esta funcionalidad es el de regenerar todos los caminos de transición libre de colisiones entre lugares. Esta funcionalidad es muy importante en caso de que se modifiquen los puntos que representan un lugar, o incluso en caso de agregar un nuevo lugar en el sistema. Para utilizar dicha funcionalidad, desde la página principal (Figura 3.11), el usuario debe hacer clic en el botón "Regenerar caminos"

Sección de Ayuda

Se ha incorporado una sección de ayuda en la interfaz, que se encuentra disponible en el encabezado de la página. Esta sección proporciona información sobre cómo usar las distintas funcionalidades de la página.

3.6.2. Características

En esta sección se explicarán algunas características que tiene la interfaz gráfica.

Intuitiva

Se ha intentado crear una interfaz intuitiva y fácil de utilizar. Para esto se estableció un orden en las funcionalidades, lo que hace que el usuario pueda seguir un flujo lógico al interactuar con el sistema.

Responsiva

La interfaz es responsiva, adecuándose a varios tamaños de pantalla. Esto permite a los usuarios acceder al sistema desde una mayor variedad de dispositivos y mejora la calidad de la experiencia de usuario.

3.6.3. Validación de la Interfaz de Usuario

Con el objetivo de validar que la interfaz de usuario cumplía con los objetivos, se realizaron dos instancias de demostraciones con el cliente, quien se mostró conforme con el trabajo realizado.

3.7. Backend

El backend de la aplicación consiste en un servidor que se conecta con el frontend mediante una API REST y websockets, y una base de datos no relacional. A continuación se entrará en detalle sobre cada uno de estos componentes, mencionando detalles de su diseño e implementación.

3.7.1. Servidor

Se implementó un servidor en Python, el cual provee una API REST y Web Sockets que permite la comunicación con la interfaz de usuario. Además, el servidor obtiene datos de la base de datos, y ejecuta los algoritmos de planificación descritos anteriormente

API REST

Se ha utilizado Flask para implementar una API REST que permita a la interfaz de usuario conectarse con el resto del sistema. La elección fue por su simplicidad, ya que permite crear APIs en pocas líneas de código.

A continuación se listarán todos los *endpoints* que se implementaron.

- `/places/get_all` - Obtener todos los lugares que se pueden visitar.
- `/missions/insert_mission` - Crear una nueva misión
- `/missions/get_all` - Obtener todas las misiones existentes.
- `/missions/get_by_id/<string:mission_id>` - Obtener datos de una misión.

- `/missions/edit_mission` - Editar una misión.
- `/missions/delete_mission` - Eliminar una misión.
- `/missions/get_by_id/` - Obtener la misión por id.
- `/missions/get_mission_plan` - Obtener el plan de una misión, generado mediante los algoritmos de planificación implementados.
- `/missions/start_mission` - Empezar una misión.
- `/missions/abort_mission` - Abortar una misión en progreso.
- `/users/validate` - Validar un usuario para proveer autenticación.
- `/users/logout` - Cerrar sesión de un usuario.
- `/paths/regenerate` - Regenerar caminos de transición.

Web Sockets

Una gran ventaja de los Web Sockets es que permiten la comunicación en tiempo real entre el cliente y el servidor sin la necesidad de tener que realizar solicitudes HTTP adicionales cada vez que se desea actualizar la información. Se implementó un evento, global_status_update que mantiene información actual de la misión y del estado de los drones en tiempo real.

3.8. Base de Datos

Se decidió utilizar MongoDB (Ver AnexoB.1.2). A continuación se procederá a describir los datos almacenados en la base de datos y a justificar la elección de una base de datos no relacional en lugar de una relacional.

Descripción de las Colecciones

- Property - Conjunto de propiedades donde se va a realizar la inspección. Por el momento, el sistema cuenta de una única propiedad, la Facultad de Ingeniería.
- Place - Lugares específicos dentro de la propiedad que se desea inspeccionar. Dentro de los atributos mas importantes, se encuentran:
  - waypoints: Es una colección contiene los cuatro puntos que definen el perímetro del lugar. Cada uno de estos *waypoints* está compuesto de: latitud, longitud y altitud.
  - yaw: Es el ángulo con el cual el dron debe posicionarse para realizar el barrido en el lugar y que la cámara apunte al lugar correcto. Es útil únicamente en fachadas, ya que de lo contrario los videos son grabados utilizando la cámara cenital.

- Mission - Es un conjunto de lugares que se desean inspeccionar. Entre los atributos mas importantes se destacan:
  - places: Conjunto de lugares de se deben recorrer en la misión.
  - created_date_time: Fecha y hora que se creó la misión.
- Procedure - Aquí se guardan los datos de cada misión que se ejecuta. Entre los atributos mas importantes se tiene:
  - start_time: Fecha y hora que la misión se inició.
  - end_time: Fecha y hora de finalización de la misión.
  - success: Booleano que indica si la misión terminó con éxito o si fue abortada.
  - plan: Es un objeto que contiene la misión que se ejecutó, qué drones fueron parte y cual ruta siguió cada uno de ellos.
- Usuario - Credenciales de usuario para poder autenticarse. La contraseña se encuentra encriptada en la base de datos.

Justificación de una Base de Datos No Relacional

Las bases de datos relacionales son las mas frecuentes en la industria ya que permiten una representación normalizada de los datos y también cuentan con un lenguaje de consulta estandarizado en SQL. A pesar de esto se optó utilizar una base de datos no relacional y documental para almacenar los datos en este proyecto.

Las bases de datos no relacionales son mas flexibles y adaptables, ya que no se rigen por un esquema estricto con tablas y restricciones como en las relacionales. Esto las vuelve ideales en proyectos en evolución constante, donde los requerimientos y la estructura de los datos pueden cambiar con frecuencia, como en este caso. Idealmente el sistema implementado será usado, mantenido y mejorado durante cierto tiempo. Esto lleva a concluir que la estructura de los datos puede cambiar una vez que el proyecto se entregue y el sistema cambie de manos y, por ende, una base de datos no relacional como MongoDB sea adecuada. Además, al principio del proyecto, no se tenía conocimiento sobre los datos que serían necesarios almacenar. Por lo tanto, adoptar este enfoque nos resultó muy útil durante el desarrollo del sistema.

3.9. Simulación

Gazebo

Existen muchas herramientas de simulación para robótica, y más específicamente para robots voladores, con diferentes ventajas y desventajas. Para el trabajó se eligió Gazebo dada su popularidad en la comunidad de ROS y la disponibilidad de documentación en internet. Como se explica en el anexo de

tecnologías (Ver Anexo B.1.1), se trata de una plataforma que permite el uso de modelos de robots y la creación de entornos virtuales y posee un buen motor físico.

En este programa, se generó un entorno usando el modelo que representa a la Facultad de Ingeniería y su terreno, ubicado en sus coordenadas geográficas reales. Este enfoque permitió crear un entorno de simulación que reproduce las condiciones y lecturas de sensores (incluyendo la posición de GNSS) de una inspección real. Por esto, fue usado para ejecutar inspecciones a lo largo del desarrollo y la verificación.

La generación de videos fue posible a través de esta herramienta ya que tiene la capacidad de generar cámaras y publicar las imágenes obtenidas a los tópicos de ROS correctos, facilitando también el futuro uso de drones reales, ya que no es necesario cambiar la implementación de esta funcionalidad.

RViz

Debido a la generalidad de la herramienta RViz, se utilizó a lo largo del desarrollo para visualizar varios datos. Entre ellos se encuentran la posición del dron, las imágenes obtenidas por las cámaras, los octrees generados en tiempo real por el sistema PX4-Avoidance (Ver Anexo B.2), la grilla de ocupación y los caminos generados.

RQT

Para comprender y resolver problemas relacionados a los marcos de referencia, o *frames*, que ocurrieron al definir nuevos sensores en los drones y al momento de intentar trabajar con octrees y Octomap (Ver Anexo B.2), se usó la herramienta de rqt "rqt_tf_tree", que permitió visualizar en forma de grafos los diferentes *frames* y sus interacciones, permitiendo identificar fallas en las definiciones o transformaciones.

De este paquete de ROS, también se usó "rqt_graph" con el fin de verificar el correcto funcionamiento de los tópicos y los diferentes nodos que se suscribían o publicaban a ellos.

Grabación de Videos Durante la Inspección

Se estableció como requisito la necesidad de capturar registros visuales de los lugares sujetos a inspección. Para cumplir con este objetivo, se implementó una funcionalidad específica de grabación de videos. Con el propósito de ofrecer una gestión ordenada y distinguida de los videos generados, se ha establecido un formato de nomenclatura para los archivos de video resultantes.

Por cada dron que forme parte del sistema, se procede a la grabación de dos videos simultáneos. Los nombres de estos archivos de video están estructurados de la siguiente manera: **tiempo-de-inspección_drone-id_dirección-de-la-camara**. La variable **dirección-de-la-camara** puede tomar dos valores: **forward** (hacia adelante) o **downward** (hacia abajo), en función de la orientación de la cámara correspondiente.

Con el objetivo de optimizar los recursos, tanto de memoria del dron como el tiempo del encargado de revisar el video, la grabación se activa exclusivamente durante el período en que el dron se encuentra realizando una inspección efectiva de un lugar, es decir, no se graban videos en los momentos de transición entre lugares de inspección.

3.9.1. PX4 Autopilot

Protocolos de comunicación

En este contexto, PX4 Autopilot y QGroundControl, dos elementos centrales en la operación de UAVs, utilizan el protocolo MAVLink (*Micro Air Vehicle Link*) para establecer una comunicación.

MAVLink, un estándar de comunicación reconocido, es el protocolo predominante en el entorno de PX4 y en la mayoría de las plataformas de control de vuelo de drones. MAVLink facilita la transferencia de datos entre el dron y la estación terrestre, así como con otros dispositivos compatibles con este protocolo.

PX4 Avoidance

PX4 Avoidance es una funcionalidad incorporada a la plataforma PX4 Autopilot que ofrece a los UAVs la capacidad de identificar y evadir obstáculos de manera autónoma durante el vuelo, basándose en la visión artificial (Computer vision) y utilizando el algoritmo 3DVFH* (Baumann, 2018).

La funcionalidad de PX4 Avoidance se basa en un conjunto de sensores, tal como cámaras, que proporcionan datos en tiempo real sobre el entorno del dron. Estos datos se utilizan para construir un mapa tridimensional del espacio y detectar obstáculos potenciales. Con esta información el UAV puede ajustar su ruta de vuelo de manera autónoma para evitar colisiones con obstáculos detectados.

Esta herramienta fue utilizada en este proyecto de forma tal de evitar colisiones en tiempo real utilizando un único dron. Sin embargo, no funciona con múltiples drones, y adaptar el código para permitirlo hubiese requerido de mucho tiempo, por lo que se omitió el uso de esta herramienta para este caso.

Archivos de lanzamiento

En el contexto de ROS, los “archivos de lanzamiento” o “launch files” son documentos en formato XML empleados para la inicialización y configuración de nodos individuales y sistemas robóticos en su conjunto. Estos archivos encapsulan información esencial sobre cuáles nodos deben ser activados, las respectivas configuraciones que deben recibir y cómo se conectan estos componentes. Además, permiten incluir otras cosas, como el mundo que se desea inicializar.

PX4 Autopilot proporciona una variedad de ejemplos de archivos de lanzamiento. Estos ejemplos han sido utilizados como punto de partida para la

creación de archivos de lanzamiento personalizados para la aplicación en cuestión.

Se tienen dos "launch files"distintos, uno usado cuando se quiere simular solamente un dron y otro para simular multiples drones. Mientras que "launch-single-uav.launch" utiliza el *local planner* provisto por PX4-Avoidance, "launch-multi-uava.launch" no lo hace, por lo explicado en la sección PX4 Avoidance. Ambos archivos utilizan el mundo "fing.world", que contiene todos los datos necesarios para inicializar la facultad como un modelo en Gazebo.

3.9.2. Definición de los UAVs

PX4 Autopilot brinda muchas definiciones de modelos en *Simulation Description Format* (SDF), dentro de los que se encuentran algunos UAVs, permitiendo su integración y utilización en simulaciones. Uno de los ejemplos en esta colección es el modelo denominado Iris Quadcopter, que representa un dron desarrollado por 3D Robotics.

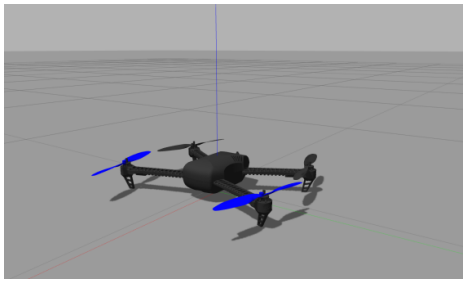

Figura 3.22: Modelo del UAV Iris en Gazebo

A partir de los requisitos del sistema se identificó una necesidad particular: la incorporación de dos cámaras en el dron para fines de inspección y detección de obstáculos. Una de estas cámaras debía estar orientada hacia adelante, con el propósito de posibilitar la evasión de obstáculos y la inspección de fachadas, mientras que la otra debía dirigirse hacia abajo, permitiendo la inspección de techos y canalones. Por este motivo, ninguna de las variaciones disponibles del modelo Iris Quadcopter proporcionado por PX4 resultó adecuada para la tarea. En consecuencia, se optó por modificar el modelo del Iris Quadcopter, agregando ambas cámaras. Como resultado, se incluyeron nuevos tópicos de comunicación para las cámaras adicionales, y cada misión realizada con este UAV generó dos flujos de video independientes, uno correspondiente a cada una de las cámaras.

3.9.3. Controlador de Dron

La mayoría de la nueva lógica implementada para controlar el dron se encuentra en un nodo de ROS llamado *DroneController*. Este nodo se encarga de

la publicación de los *waypoints*, iniciar y abortar misiones, enviar actualizaciones del estado del dron y la configuración del componente PX4 Avoidance para la evitación de obstáculos.

3.9.4. Controlador de Misión

En la estación de control en tierra, el componente principal es la clase denominada "MissionManager". Esta entidad se ocupa del control y ejecución de misiones aéreas con UAVs.

En particular, las responsabilidades del "MissionManager" comprenden la capacidad de iniciar y administrar las misiones de vuelo, así como la coordinación de la grabación de video durante la ejecución de dichas misiones. Otras funciones son la configuración y gestión de puntos de referencia (*waypoints*), la transición del UAV al modo de vuelo de misión automática para seguir una serie de *waypoints*, el procesamiento de imágenes capturadas por las cámaras del dron, y la programación de la grabación de video en función de los *waypoints* que marcan el inicio y la finalización de la grabación.

Adicionalmente, el "MissionManager" cumple un papel crucial en la publicación de actualizaciones referentes al estado de la misión mediante el uso de ROS, lo que permite una supervisión continua y en tiempo real de la ejecución de la misión. Asimismo, esta entidad posee la capacidad de ejecutar acciones de aborto o finalización de la misión, según las circunstancias lo requieran, manteniendo una comunicación fluida con el sistema de control de vuelo del dron a través de los protocolos MAVLink y ROS.

Estas funcionalidades permiten una gestión completa y automatizada de las misiones de vuelo.

Capítulo 4

Experimentación

Índice

El sistema implementado desempeña diversas tareas con el objetivo de facilitar la inspección de las instalaciones de la Facultad de Ingeniería. En primer lugar, se ha desarrollado un sistema de planificación autónoma de rutas para los drones empleados. Esta funcionalidad se basa en un planificador que consta de tres componentes principales: el asignador de tareas, el planificador de transiciones y el planificador local. La interacción entre estos módulos permite la generación eficiente de trayectorias para los drones, asegurando una buena cobertura de los puntos a inspeccionar.

Una vez planificadas las rutas, el sistema procede a llevar a cabo las inspecciones propiamente dichas. Para ello, se ha empleado el simulador Gazebo, el cual simula de manera realista los escenarios y los objetos presentes en la facultad. Durante las inspecciones, uno o más drones son utilizados para examinar los techos, fachadas y canalones de los edificios. Esta etapa del proceso tiene como finalidad detectar posibles anomalías o daños de los espacios universitarios.

Además, se ha tenido en cuenta la usabilidad del sistema, permitiendo al usuario interactuar con el mismo a través de una interfaz gráfica intuitiva y amigable. Esta interfaz proporciona control sobre las operaciones del sistema, permitiendo al usuario crear misiones, aprobar las rutas y supervisar las inspecciones en tiempo real.

Se han llevado a cabo pruebas para comprobar el correcto funcionamiento de todas las funcionalidades del sistema y evaluar su desempeño en diferentes escenarios. A continuación se encuentran detalladas las pruebas realizadas y el posterior análisis de los resultados.

4.1. Evaluación de la Asignación de Tareas

4.1.1. Asignación de tareas a un dron

Para verificar que la planificación se realice eficientemente se ejecutaron pruebas automatizadas de planificación de caminos para una serie de misiones definidas en el sistema. Para cada caso, se obtendrá la métrica de distancia recorrida para la asignación generada por MTSP y para asignaciones aleatorias, seleccionadas del conjunto de permutaciones de lugares posibles. El fin es compararlas y verificar el correcto funcionamiento del asignador de tareas. Se limitó el número de permutaciones a veinte, de forma de mantener la legibilidad del documento y evitar la computación excesiva de casos.

En esta sección los resultados de las comparaciones serán mostrados en tablas que contienen los órdenes de visita de las zonas de inspección y el costo asociado, así como el porcentaje de desvío de cada caso en comparación con el mejor.

Misión Techos Aulario

La Misión Techos Aulario consiste en realizar una inspección de los tres techos del edificio del Aulario Polifuncional “José Luis Massera”. Se denominan los techos Aulario A, Aulario B y Aulario C; siguiendo la nomenclatura interna de la facultad. El camino dado por el planificador ha sido:

1. Punto de Despegue
2. Techo del Aulario A
3. Techo del Aulario C
4. Techo del Aulario B

Figura 4.1: Asignación de tareas - Misión Techos Aulario - 1 dron.

Si bien el orden de visita generado no parece ser el mejor a primera vista, el cómputo de los costos para las otras permutaciones realizado en la Tabla 4.1.1 muestra que es la asignación de menor costo para la misión.

Drones	Costo total (m)	Tipo	Desvío vs Mejor (%)	Secuencia
1	894.29	MTSP	0	1,2,3,4,1
1	910.84	Aleatorio	1.85	1,2,4,3,1
1	958.40	Aleatorio	7.16	1,3,2,4,1
1	910.00	Aleatorio	1.75	1,3,4,2,1
1	898.39	Aleatorio	0.45	1,4,3,2,1
1	951.96	Aleatorio	6.44	1,4,2,3,1

Tabla 4.1: Comparación de los costos de diferentes órdenes de visita para la misión Techos Aulario.

Misión Techos Aulario y Eléctrica

Esta misión consiste en los tres techos del Aulario de la prueba anterior y el techo principal del edificio del Instituto de Ingeniería Eléctrica. El orden de visita dado por el planificador ha sido:

1. Punto de Despegue
2. Techo del Aulario A
3. Techo del Aulario B
4. Techo del Aulario C
5. Techo Principal de Eléctrica

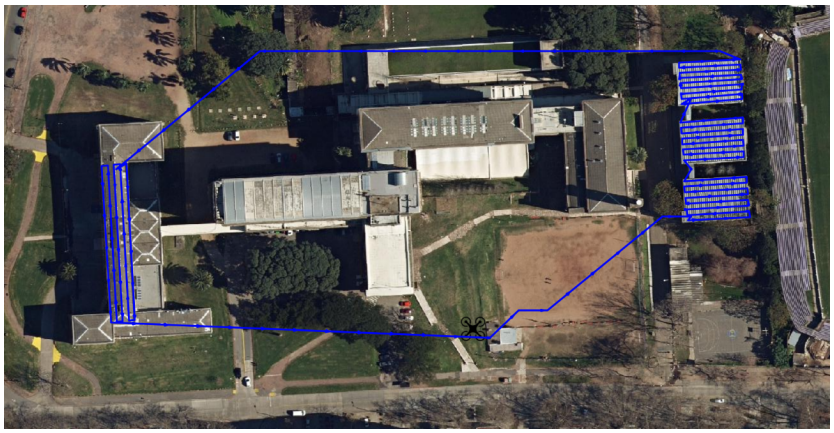

Figura 4.2: Asignación de tareas - Misión Techos Aulario y Eléctrica - 1 dron.

En la Tabla 4.2 se puede apreciar cómo en este caso la solución encontrada por el planificador es la mejor dentro de las permutaciones aleatorias analizadas.

Drones	Costo total (m)	Tipo	Desvío vs Mejor (%)	Secuencia
1	1351.34	MTSP	0.00	1,2,3,4,5,1
1	1431.9	Aleatorio	5.96	1,4,2,3,5,1
1	1625.02	Aleatorio	20.25	1,4,2,5,3,1
1	1627.69	Aleatorio	20.45	1,3,2,5,4,1
1	1548.02	Aleatorio	14.55	1,2,5,4,3,1
1	1412.57	Aleatorio	4.53	1,5,4,2,3,1
1	1607.58	Aleatorio	18.96	1,4,5,3,2,1
1	1446.68	Aleatorio	7.05	1,3,4,2,5,1
1	1623.13	Aleatorio	20.11	1,3,5,2,4,1
1	1458.29	Aleatorio	7.91	1,4,3,2,5,1
1	1607.14	Aleatorio	18.92	1,2,3,5,4,1
1	1605.69	Aleatorio	18.82	1,4,3,5,2,1
1	1612.81	Aleatorio	19.34	1,4,5,2,3,1
1	1592.02	Aleatorio	17.81	1,2,4,5,3,1
1	1545.68	Aleatorio	14.38	1,3,4,5,2,1
1	1442.58	Aleatorio	6.75	1,2,4,3,5,1
1	1605.69	Aleatorio	18.82	1,2,5,3,4,1
1	1453.47	Aleatorio	7.55	1,5,3,2,4,1
1	1386.01	Aleatorio	2.56	1,3,2,4,5,1
1	1351.34	Aleatorio	0.00	1,2,3,4,5,1
1	1364.17	Aleatorio	0.94	1,5,4,3,2,1

Tabla 4.2: Comparación de los costos de diferentes órdenes de visita para la misión Techos Aulario y Eléctrica.

En este caso, resulta que hay otro orden de inspección que tiene un costo igual al que genera el planificador, dentro de un margen de error despreciable.

Misión Techos

La misión Techos consiste en inspeccionar todos los techos de interés de la facultad. En este caso resulta especialmente importante la limitación aplicada a la cantidad de permutaciones analizadas, ya que el total es de 8!, que son 40320.

La solución encontrada por el asignador de tareas resulta la mejor dentro de las permutaciones analizadas, como se puede ver en la Tabla 4.3, y su orden de visita es:

1. Punto Central
2. Techo del Aulario A
3. Techo del Aulario B
4. Techo del Aulario C
5. Techo de Bandejas
6. Techo de la Biblioteca Central
7. Bóveda del Cuerpo Central
8. Techo Principal de Eléctrica
9. Techo del Salón de Actos

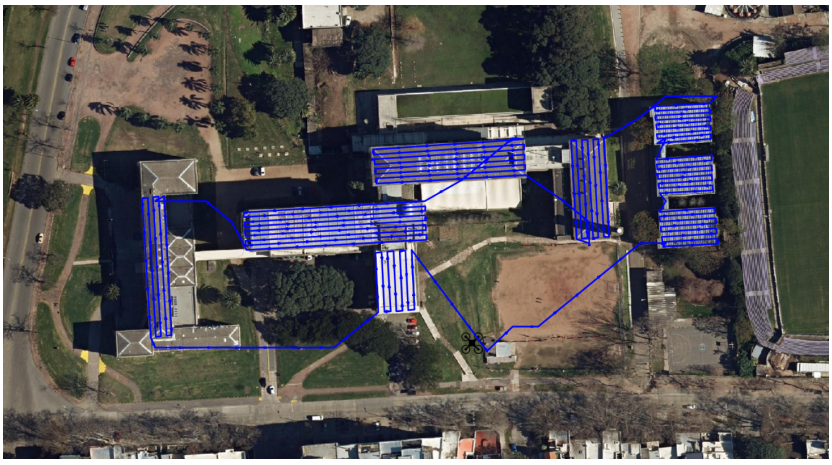

Figura 4.3: Distribución de tareas - Misión Techos - 1 dron.

Drones	Costo total (m)	Tipo	Desvío vs Mejor (%)	Secuencia
1	2929.56	MTSP	0	1,2,3,4,5,6,7,8,9,1
1	3152.23	Aleatorio	7.60	1,5,4,2,6,3,8,7,9,1
1	3176.89	Aleatorio	8.44	1,4,3,5,7,8,9,6,2,1
1	3178.00	Aleatorio	8.48	1,7,8,3,6,4,5,2,9,1
1	3235.35	Aleatorio	10.43	1,3,2,5,6,7,8,4,9,1
1	3275.89	Aleatorio	11.82	1,4,8,7,9,6,5,2,3,1
1	3362.79	Aleatorio	14.78	1,5,9,8,2,3,4,7,6,1
1	3425.01	Aleatorio	16.91	1,2,8,9,3,4,6,5,7,1
1	3348.01	Aleatorio	14.28	1,4,3,5,6,2,7,9,8,1
1	3429.89	Aleatorio	17.07	1,8,9,2,7,3,4,5,6,1
1	3472.01	Aleatorio	18.51	1,3,7,4,5,9,8,6,2,1
1	3514.90	Aleatorio	19.98	1,5,6,7,9,2,4,8,3,1
1	3525.12	Aleatorio	20.32	1,4,2,8,3,6,5,9,7,1
1	3530.46	Aleatorio	20.51	1,5,9,7,4,3,8,6,2,1
1	3558.90	Aleatorio	21.48	1,6,9,2,5,7,4,3,8,1
1	3560.01	Aleatorio	21.52	1,9,2,7,4,6,8,3,5,1
1	3582.79	Aleatorio	22.29	1,7,4,3,8,2,5,9,6,1
1	3601.01	Aleatorio	22.92	1,3,5,7,9,4,8,2,6,1
1	3635.90	Aleatorio	24.11	1,5,8,3,9,7,2,4,6,1
1	3640.46	Aleatorio	24.26	1,7,6,3,9,2,5,8,4,1
1	3646.90	Aleatorio	24.48	1,6,3,8,2,5,7,4,9,1

Tabla 4.3: Comparación de distancias para diferentes asignaciones de tareas en la misión Techos.

Misión Aulario y Cuerpo Central

Esta misión consiste en los tres techos del Aulario y la Bóveda del Cuerpo Central. El orden de visita dado por el planificador ha sido:

1. Punto de Despegue
2. Techo del Aulario A
3. Techo del Aulario B
4. Techo del Aulario C
5. Bóveda del Cuerpo Central

Este caso resulta interesante ya que la planificación generada no es la mejor. Esto se debe a lo mencionado en la sección3.5.3 sobre la elección de los caminos usados para la matriz de costos. Al elegir el camino mínimo entre zonas para representar el costo de las aristas del problema modelo MTSP, es posible que el camino real sea uno diferente. La planificación presenta una aumento en el costo de un 2,41 % o de aproximadamente 37 metros. A pesar de no ser óptimo, resulta aceptable y es el valor más bajo de aumento registrado en los casos observados. El resultado puede observarse en la Tabla 4.4.

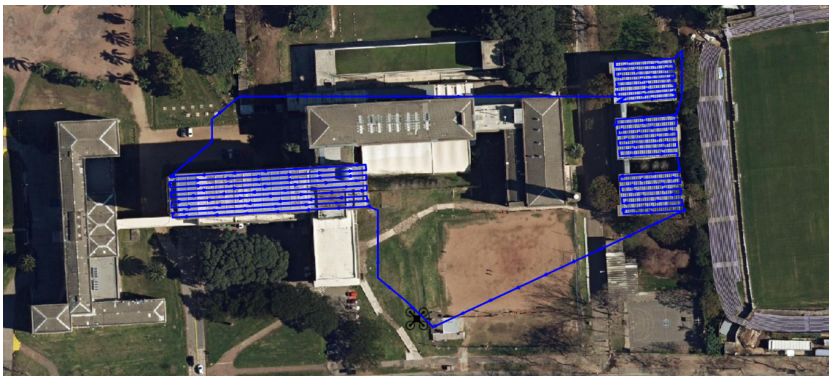

Figura 4.4: Asignación de tareas - Misión Techos Aulario y Cuerpo Central - 1 dron.

Drones	Costo total (m)	Tipo	Desvío vs Mejor (%)	Secuencia
1	1556.89	MTSP	2.41	1,2,3,4,5,1
1	1785.68	Aleatorio	17.46	1,3,2,5,4,1
1	1758.34	Aleatorio	15.67	1,5,2,4,3,1
1	1556.89	Aleatorio	2.41	1,5,3,4,2,1
1	1597.11	Aleatorio	5.06	1,3,4,5,2,1
1	1796.68	Aleatorio	18.19	1,4,2,5,3,1
1	1645.67	Aleatorio	8.25	1,2,4,5,3,1
1	1736.34	Aleatorio	14.22	1,3,2,4,5,1
1	1729.11	Aleatorio	13.74	1,4,2,3,5,1
1	1520.11	Aleatorio	0.00	1,5,4,3,2,1
1	1614.56	Aleatorio	6.21	1,2,3,5,4,1
1	1756.45	Aleatorio	15.54	1,3,4,2,5,1
1	1700.67	Aleatorio	11.87	1,5,2,3,4,1
1	1556.89	Aleatorio	2.41	1,2,3,4,5,1
1	1731.78	Aleatorio	13.92	1,5,4,2,3,1
1	1783.01	Aleatorio	17.29	1,4,5,2,3,1
1	1573.23	Aleatorio	3.49	1,4,5,3,2,1
1	1719.68	Aleatorio	13.12	1,4,3,2,5,1
1	1586.11	Aleatorio	4.34	1,4,3,5,2,1
1	1628.23	Aleatorio	7.11	1,2,5,3,4,1
1	1780.34	Aleatorio	17.11	1,3,5,2,4,1

Tabla 4.4: Comparación de los costos de diferentes órdenes de visita para la misión Aulario y Cuerpo Central.

4.1.2. Asignación de tareas a múltiples drones

Para la planificación asignación con múltiples UAVs se plantea el problema como un Multiple Traveling Salesman Problem y se espera que el algoritmo implementado cree un camino balanceado y ordenado. A continuación se mostrarán caminos que serán recorridos por varios UAVs con el objetivo de verificar que las asignaciones generadas por el algoritmo son apropiadas y eficientes. Se analizará el costo asociado a cada planificación realizada por el asignador de tareas.

Misión Techos Aulario

Para esta misión, que consiste en los tres techos del Aulario, el planificador dio como resultado la siguiente asignación de tareas:

Asignación 1:

1. Punto de Despegue
2. Techo del Aulario A

Asignación 2:

1. Punto de Despegue
2. Techo del Aulario B
3. Techo del Aulario C

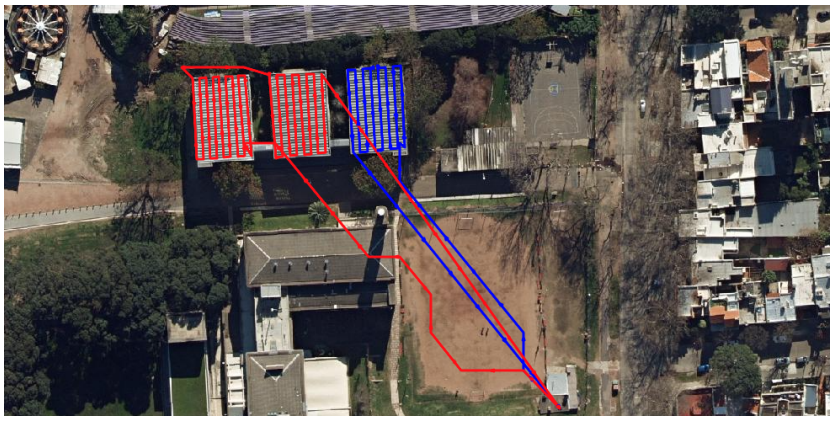

Figura 4.5: Asignación de tareas - Misión Techos Aulario - 2 drones.

El costo total de la misión es de 1082.85 metros recorridos. Este costo representa un aumento de un 21.08 % con respecto a la ejecución con un solo dron. El costo del primer camino es de 618.8 metros, mientras que el costo del segundo es de 984.2 metros. En promedio, el costo de los viajes son de 801.5 metros. Esto supone un decremento de un 10.37 % en el costo promedio que debe afrontar el dron con respecto a la planificación con un dron. Esto significa que, a pesar de que la suma de todos los caminos es mayor de que si lo hiciera un solo dron, el tiempo final de misión es menor. Por otra parte, la diferencia de costos entre drones, en este caso, supone una diferencia de 365.4 metros.

Misión Techos Aulario y Eléctrica

Esta misión, utilizada previamente para probar caminos con un solo dron, consiste en los tres techos del Aulario y, también, el techo principal del edificio del Instituto de Ingeniería Eléctrica. Se pedirá al algoritmo que genere asignaciones teniendo en cuenta que la misión será ejecutada con 2 drones. Las asignaciones generadas por el planificador son:

Asignación 1:

1. Punto de Despegue
2. Techo del Aulario B
3. Techo del Aulario A

Asignación 2:

1. Punto de Despegue
2. Techo del Aulario C
3. Techo Principal de Eléctrica

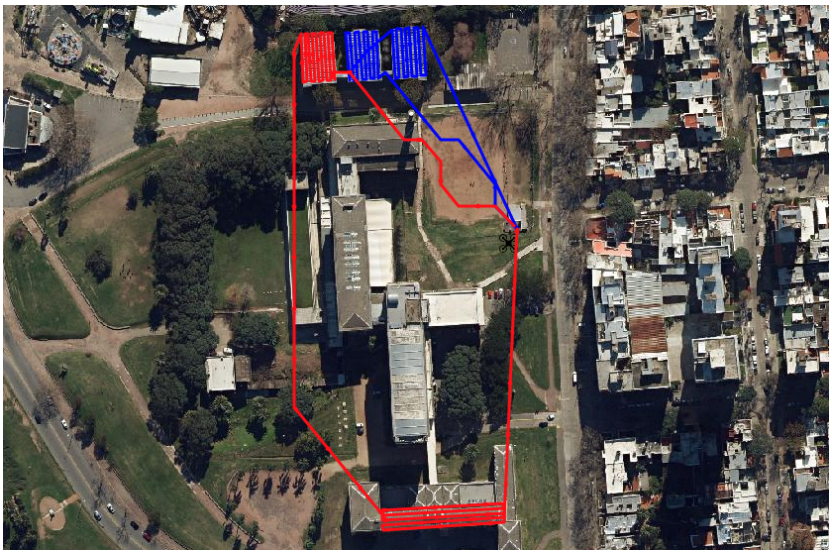

Figura 4.6: Asignación de tareas - Misión Techos Aulario y Eléctrica - 2 drones.

Se puede observar que la repartición de tareas es equitativa, dando dos zonas de inspección por dron. El costo total de la misión es de 1603.02 metros, mientras que los costos individuales son de 618.80 metros para el camino 1 y de 984.21 metros para el camino 2. El aumento del costo total, con respecto a la utilización de un único dron, es del 18.62 %, mientras que el costo promedio del dron en este caso es de 801.50 metros, lo que representa un decremento de 40.68 % en el costo por dron.

Misión Techos

La misión Techos consiste en inspeccionar todos los techos de interés de la facultad. Se pedirá al algoritmo que cree asignaciones para la misión con 2

drones y 3 drones.

Las asignaciones creadas para la versión con 2 drones son:

Asignación 1:

1. Punto de Despegue
2. Techo de Bandejas
3. Techo del Aulario C
4. Techo del Aulario B
5. Techo del Aulario A

Asignación 2:

1. Punto de Despegue
2. Techo del Salón de Actos
3. Techo de la Biblioteca Central
4. Bóveda del Cuerpo Central
5. Techo Principal de Eléctrica

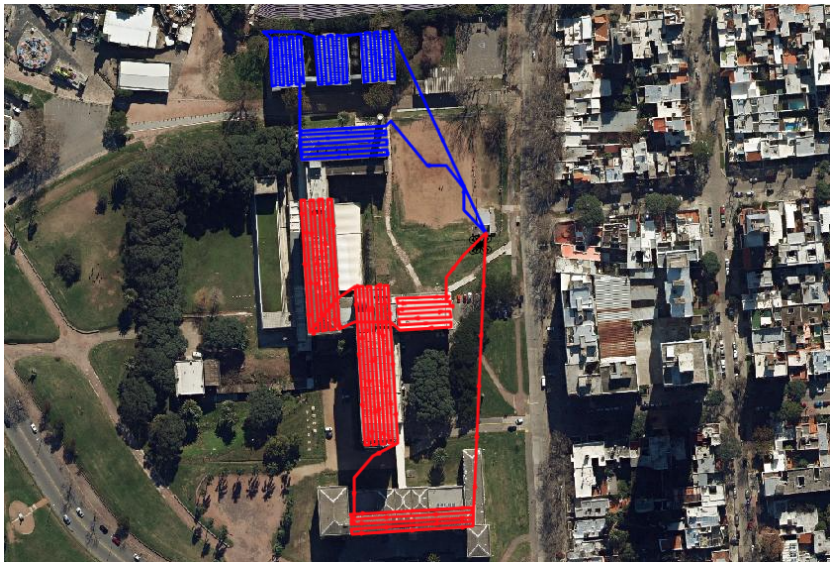

Figura 4.7: Asignación de tareas - Misión Techos - 2 drones.

Se observa como todos los lugares de un mismo camino se encuentran cerca entre si. En cuanto al costo de la misión, es de 3039.71 metros. El costo de los caminos es de 1238.03 metros y 1801.67 metros respectivamente.

Este experimento demostró cómo el planificador puede repartir equitativamente la carga de la inspección en múltiples drones y como asigna a los lugares a cada camino para reducir el tiempo de traslado entre ellos y, por ende, el tiempo de ejecución de la misión.

Las asignaciones creadas para la versión con 3 drones son:

Asignación 1:

1. Punto de Despegue
2. Techo Principal de Eléctrica
3. Bóveda del Cuerpo Central

Asignación 2:

1. Punto de Despegue
2. Techo de Bandejas

3. Techo de la Biblioteca Central
4. Techo del Salón de Actos

Asignación 3:

1. Punto de Despegue
2. Techo del Aulario A
3. Techo del Aulario B
4. Techo del Aulario C

Figura 4.8: Asignación de tareas - Misión Techos - 3 drones.

Con la asignación utilizando tres drones, en lugar de dos, se puede observar que sigue siendo balanceada en cuanto a cantidad de lugares. Con respecto al costo total de la misión, es de 3198.89 metros, un 5.2 % mas costosa que para dos drones. De todas formas, el costo por dron (969.98, 1334.61 y 894.29 metros por camino) es de 1066.3 metros en promedio, lo que implica un decremento de un 42.5 % de carga por dron en comparación con el uso de dos drones.

4.2. Planificación local de caminos

Es necesario que el algoritmo genere los *waypoints* a recorrer por el dron en la inspección de un lugar específico. Para todos los lugares se mantienen los cuatro vértices los delimitan. Mediante el algoritmo descrito en la sección 3.5.2, se generan los *waypoints* dinamicamente, teniendo en cuenta qué vértice es más conveniente usar primero.

4.2.1. Inspección de Canalones

Para realizar la inspección de un canalón se necesita recorrer el perímetro del lugar objetivo, como se muestra en la Figura 4.9.

Figura 4.9: Planificación Local - Misión Canalón de Bandejas - 1 dron.

4.2.2. Inspección de Techos

A continuación se presentan imágenes que muestran algunos caminos de inspección de techos. Estos caminos con forma de barrido verifican que siempre se cubre el área en su totalidad.

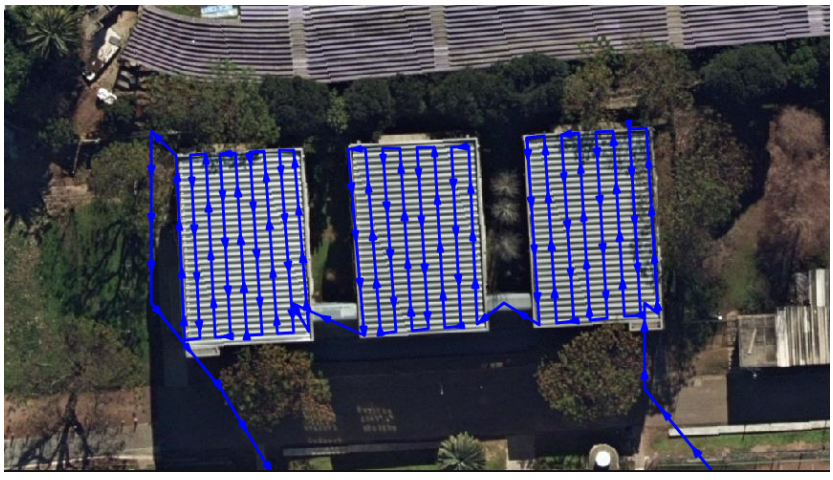

Figura 4.10: Planificación Local - Techos Aulario.

Figura 4.11: Planificación Local - Techo de Bandejas.

Figura 4.12: Planificación Local - Techo Principal de Eléctrica.

También se agregó una prueba en una porción de un techo del edificio del Instituto de Ingeniería Eléctrica que tiene una forma geométrica distinta, con el fin de comprobar la flexibilidad del planificador local. El resultado se puede ver en la Figura 4.13.

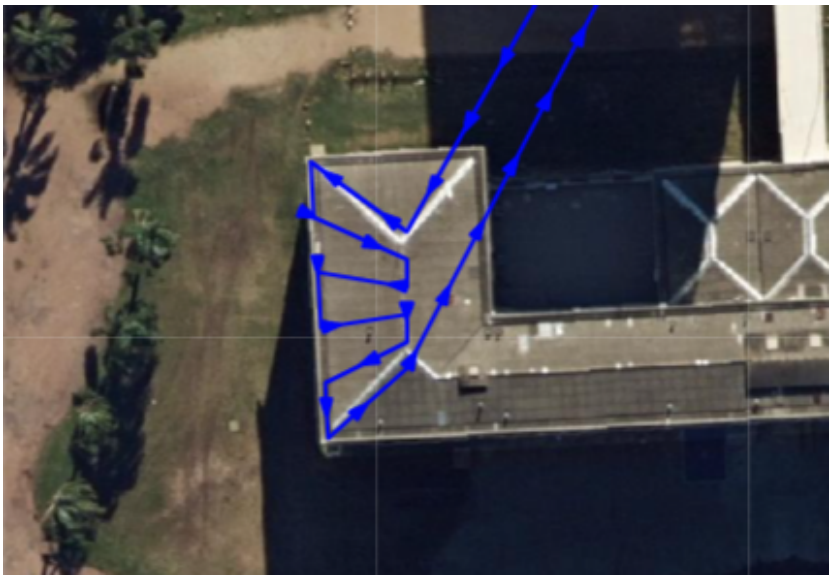

Figura 4.13: Planificación Local - Techo con forma de trapecio.

4.2.3. Inspección de Fachadas

El camino del barrido de la inspección se puede observar en el mapa (Figura 4.14). También se puede observar, desde otro punto de vista, en la herramienta RViz (Figura 4.15). Se puede verificar que cumple las condiciones definidas en la implementación, en la sección 3.5.2.

Figura 4.14: Planificación Local - Misión Fachada Norte del Salón de Actos - 1 dron.

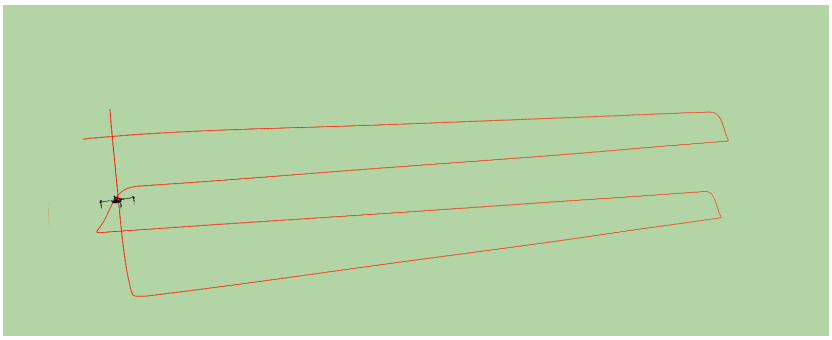

Figura 4.15: Planificación Local - Recorrido del dron en RViz de la misión Fachada Norte del Salón de Actos.

4.3. Simulaciones

Para verificar que todos los componentes del sistema funcionen correctamente, se realizaron pruebas utilizando el simulador Gazebo. El mundo creado para la simulación es simple. Contiene un modelo tridimensional de la Facultad de Ingeniería y no se han incorporado ciertas funcionalidades de Gazebo como, por ejemplo, el viento. Se utiliza un modelo de dron denominado *Iris Two Depth Camera* cuyas especificaciones se han descrito en la sección 3.9.2.

4.3.1. Entorno de hardware

Las simulaciones fueron ejecutadas en una computadora de las características de la Tabla 4.5.

CPU	Intel i5 10600K
GPU	Nvidia 3070
Memoria RAM	32GB
Sistema operativo	Ubuntu 20.04 LTS

Tabla 4.5: Características de la computadora de prueba.

4.3.2. Entorno de software

Las versiones de los programas y paquetes necesarios para ejecutar el sistema se detallan en la Tabla 4.6.

Gazebo	11.13.0
Python	3.8.10
ROS 1	Noetic
Sistema operativo	Ubuntu 20.04 LTS

Tabla 4.6: Programas y paquetes utilizados para la simulación.

4.3.3. Pruebas en simulación

Se han realizado una serie de pruebas de integración en el simulador, en las que se ejecutan casos de uso de la misma manera que haría un usuario. Con estas pruebas, se pudo verificar el correcto funcionamiento de las misiones en el entorno simulado.

Todos los experimentos de este tipo fueron documentados en video, tanto los videos obtenidos por la cámara de los robots como la grabación de la pantalla que muestra la interfaz de usuario, el entorno simulado en Gazebo y la visualización de las imágenes de las cámaras en RViz en simultáneo.

Estos videos pueden ser encontrados en una lista de reproducción[1] en YouTube, creada con el propósito de hacer disponibles estos documentos.

Las grabaciones muestran misiones de inspección de los diferentes tipos de zonas, realizadas por uno y múltiples drones.

A lo largo de las pruebas anteriores se verificó que los drones puedan evadir los obstáculos correctamente. Esto ha sido exitoso, con los drones evitando las colisiones, gracias al sistema de evasión de obstáculos PX4 Avoidance utilizado en el proyecto. Además, se probó que los caminos de transición generados por el planificador son correctos, no causan ninguna colisión.

4.3.4. Limitaciones de la simulación

Gazebo es un simulador muy potente y personalizable. No obstante, cuenta con algunas limitaciones que vale la pena mencionar.

La física de la simulación puede no ser completamente precisa. Aunque Gazebo tiene un modelo físico detallado, puede haber discrepancias entre la simulación y la realidad debido a la simplificación inherente al uso de modelos.

Hay limitaciones en el modelado de los robots. Su dinámica puede ser limitada si no se cuenta con un modelo detallado del robot.

Asimismo, puede ser computacionalmente costoso. Simular un gran número de robots y sensores en un escenario puede requerir mucho poder de cómputo, lo que puede ser un desafío para algunas máquinas.

4.4. Integración con Interfaz de Usuario

Se ha implementado una pagina web que funcione como interfaz de usuario para que el usuario pueda interactuar con los drones y la simulación. Es necesario probar que la integración ha sido implementada correctamente. Para eso se han probado los tres flujos principales que se pueden realizar desde la interfaz: Crear Misión, Editar Misión y Correr Misión.

[1] `https://www.youtube.com/playlist?list=PLpfRg59Dpd_zyefNuPLGlPdpxwAxb0cFq`

4.4.1. Crear Misión

Para probar que este caso de uso funcione correctamente se probó el flujo. Aquí el usuario entra a la página web, hace click en el botón de "Crear Misión", ingresa el nombre de la misión y los lugares a visitar y confirma la creación. Casos borde en este flujo que han sido probados, incluyen que el nombre de la misión no haya sido ingresado o que no se haya seleccionado ningún lugar a inspeccionar.

4.4.2. Editar Misión

Para probar el caso de uso de Editar Misión, el usuario entra a la página web, hace clic en el botón de "Editar Misión", selecciona la misión que desearía editar. Luego, cambia el nombre y los lugares a visitar y confirma la edición. Los casos borde que se probaron incluyen el cambio del nombre a un texto vacío y que la selección de lugares para inspeccionar sea vacía.

4.4.3. Correr Misión

Este caso de uso es el más complejo e importante de todos. Aquí el usuario debe entrar a la página web, hacer clic en el botón de "Correr Misión", seleccionar la misión que desea ejecutar, seleccionar la cantidad de drones para la misión, validar la ruta que genera el algoritmo de planificación e iniciar la misión. Una vez iniciada la misión, el usuario podrá hacer seguimiento del estado de la misión, viendo en qué lugar se encuentra cada dron directamente desde la página web. Una vez que todos los drones terminaron la misión, el usuario es notificado.

4.4.4. Cancelación de misión

El usuario puede cancelar la misión mientras esta se está desarrollando. Para eso debe hacer clic en el botón de "Cancelar Misión". En ese momento los drones abortan sus tareas de inspección y vuelven al punto de despegue inmediatamente.

4.5. Análisis de resultados

Se ha comprobado que el componente de asignación de tareas es capaz de asignar lugares a los drones de forma eficiente. Las tareas asignadas por este componente para un dron son coherentes y nunca se desvían fuertemente de la solución óptima. Cuando se cuenta con múltiples drones, la solución obtenida es balanceada y disminuye el tiempo de misión en relación con un solo robot. Al analizar las distancias de los caminos resultantes, se puede observar como los caminos generados no son excesivamente largos y su realización por un dron real es factible.

Al observar los caminos de barrido generados por el planificador local, se observa que estos brindan un camino con el cual es posible inspeccionar la totalidad de un techo o fachada. Además, es capaz de realizar caminos para figuras geométricas variadas.

En la pruebas que involucran la simulación con Gazebo y la interacción con la interfaz de usuario se comprueba el correcto funcionamiento del sistema entero. Se observa cómo se puede interactuar fácilmente con la interfaz de usuario y como la inspección es realizada exitosamente, sin colisiones con los obstáculos y grabando los videos solicitados.

Capítulo 5

Conclusiones y Trabajo A Futuro

5.1. Conclusiones

Este proyecto fue desarrollado con el objetivo de realizar un prototipo de un sistema que permita la inspección mediante una flota de drones autónomos de la Facultad de Ingeniería.

En primer lugar, se utilizaron las tecnologías ROS Noetic, Gazebo 11 y PX4 Autopilot para poder simular múltiples drones que se puedan desplazar por el entorno mencionado. Se obtuvo mediante fotogrametría un modelo tridimensional de la facultad, que fue insertado en la simulación para poder visualizar la inspección. El componente PX4 Avoidance es utilizado para habilitar a los drones a evadir obstáculos en tiempo real.

Para abordar el problema de la planificación de caminos, este se subdividió en tres problemas distintos: la asignación de tareas, la planificación local y la planificación de transiciones; que al combinarse permitieron resolver el problema completo. La asignación de tareas se encargó de decidir qué lugares debía inspeccionar cada dron, y en qué orden. Para esto se modeló la realidad como un grafo y el problema como el *Multiple Traveling Salesman Problem*, que fue resuelto usando Programación Lineal Entera y logró una asignación óptima en la mayoría de los casos y aceptable en algunos, a pesar de la aproximación elegida para la matriz de costos. La planificación local se encargó de crear caminos de inspección para cada lugar, que resultaron adecuados al tipo y cubrieron correctamente las superficies. El planificador de transiciones, mediante el uso del algoritmo A* con la heurística de distancia euclidiana, se encargó de crear un camino entre lugares, que demostraron ser eficientes y evitar colisiones con los obstáculos conocidos.

Además, se implementó una interfaz de usuario intuitiva y responsiva que permite al usuario crear, editar, ejecutar y monitorear misiones. Al monitorear una misión, el usuario podrá observar la posición de los drones en tiempo real

y poder cancelar la misión. Además, la interfaz cuenta con un componente de autenticación para mejorar la seguridad del sistema.

También se implementó un servidor Flask en Python que se comunica con la interfaz mediante una *API Rest* y *websockets*. El servidor además realiza consultas sobre una base de datos documental en MongoDB. Para cada dron se tiene un controlador, que cumple el rol de representación del dron en el servidor. Éste se conecta mediante mensajes de ROS a PX4 Autopilot, permitiendo el manejo de la misión.

Se han realizado pruebas para verificar la correctitud del sistema. Se han ejecutado múltiples y diversas misiones simuladas, en donde uno o múltiples drones recorren techos, canalones y fachadas de la facultad grabando videos. Estas pruebas muestran que los requisitos originales del proyecto han sido cumplidos.

Haber transitado el proceso de desarrollo del proyecto ha sido una experiencia de aprendizaje muy valiosa. Trabajar con UAVs es muy complejo y tiene una curva de aprendizaje muy grande. Sin embargo, se logró aprender sobre robótica autónoma y UAVs en profundidad, probar herramientas e implementar el prototipo de un sistema de inspección. Se pudo comprender las posibilidades y limitaciones de las herramientas utilizadas.

Se considera que la elección de algoritmos y problemas modelo fue adecuada ya que dio buenos resultados. Sin embargo, podría haberse dedicado más tiempo a la prueba de diferentes técnicas antes de seleccionarlas. De esa manera, el equipo hubiera podido experimentar con algoritmos como *Ant Colony Optimization* para la resolución del problema de asignación de tareas, y tal vez llegar a una solución que brinde resultados más balanceados.

También, se vivieron las consecuencias de haber elegido, en primera instancia, tecnologías más nuevas como Rust y ROS 2. Ésta elección tenía mucho sentido desde el punto de vista de aprendizaje de nuevas tecnologías, pero se aprendió a prestarle más importancia al impacto que puede tener en el proceso de desarrollo la falta de documentación o soporte de la comunidad para los casos de uso objetivos. Esto último, combinado con la falta de experiencia del equipo en el área, hizo que la resolución de problemas demandara mucho tiempo, por lo que se descartaron las tecnologías en favor de aquellas con más soporte, ROS Noetic y Python/C++. En el futuro, el equipo tendrá menos problemas de esta índole y dedicará más tiempo al análisis completo de las tecnologías antes de comenzar la implementación de un proyecto.

5.2. Trabajo a futuro

Desde la concepción del proyecto se planteó que el sistema implementado sea eventualmente completado y utilizado para realizar inspecciones en la Facultad de Ingeniería y, posiblemente, en otros edificios de la Universidad de la República. A continuación se hablará sobre el trabajo necesario para terminar la implementación del sistema y, también, sobre posibles mejoras.

Migración a drones reales

Se debería implementar la conexión entre un dron y el sistema implementado y realizar pruebas que verifiquen la seguridad antes de poder utilizar el sistema. Para poder utilizar los componentes implementados para la comunicación con los robots, éstos deberán tener una placa *Pixhawk*[1] para ser compatibles con PX4 Autopilot.

El dron simulado actualmente cuenta con dos cámaras, una que apunta hacia adelante y otra hacia abajo. Esto es una limitante a la hora de conseguir un dron comercial que cumpla con estas características. Por lo tanto, se podría implementar el manejo de un *gimbal*. Un *gimbal* consiste en una estructura pivotante que permite que la cámara se mueva independientemente de los movimientos del dron. Esto permitiría utilizar una sola cámara y se encuentra presente en muchos de los modelos comerciales disponibles.

También se debería agregar una restricción al modelo de MTSP para tener en cuenta la capacidad de la batería de los robots, con el objetivo de no generar planificaciones que no sean factibles para los drones que se van a usar. Además, los niveles de batería podrían ser verificados al momento de lanzar una misión de forma de evitar que el usuario comience la inspección sin tener carga suficiente. Otra verificación que podría realizarse es del almacenamiento interno disponible, para evitar que los videos de las inspecciones queden incompletos.

Interfaz gráfica

Se ha implementado una interfaz gráfica intuitiva y amigable, validada por el usuario final. No obstante, cuando se empiece a utilizar el sistema, el usuario podría pedir nuevos cambios a la interfaz. También se detallarán cambios que podrían mejorar la experiencia del usuario.

Cuando el usuario está configurando la misión que desea correr y debe seleccionar la cantidad de drones que serán parte de la inspección, se podría implementar una validación de que esa cantidad de drones se encuentra conectada al sistema.

Autenticación

Cuando un usuario intenta autenticarse en la página, no se ofrecen algunas funcionalidades comúnmente proporcionadas. Se podría implementar un botón de "mostrar/ocultar contraseña" para permitir que los usuarios vean lo que están escribiendo y eviten errores de escritura. Se podría implementar la opción de restablecer la contraseña en caso de que el usuario la olvide, por medio de un correo electrónico o mensaje de texto con un enlace temporal para restablecer la contraseña. El formulario de autenticación podría ser validado por la interfaz previo a ser enviado, validando que los campos no sean vacíos y el correo electrónico tenga el formato adecuado.

[1] `https://pixhawk.org/`

Para mejorar la seguridad de la autenticación, se puede considerar la implementación de autenticación de dos factores, como el uso de SMS o de aplicaciones de autenticación como Google Authenticator.

Seguridad informática

ROS es un sistema distribuido y, por lo tanto, es importante considerar la seguridad al usarlo para conectar sistemas. Para la puesta en producción del sistema podría ser de interés implementar algunas medidas preventivas.

Los drones, actualmente, quedan expuestos dentro de la red en la que se encuentren. Un ataque particular a este sistema de drones son los ataques de interceptación de comunicaciones, del tipo *Man in the Middle*. Estos ataques permitirían a malos actores controlar el UAV de forma no autorizada. Una red privada, que deje aislado el sistema sería una buena opción para evitar ataques. La implementación de un cifrado de extremo a extremo, utilizando protocolos de seguridad específicos como el estándar de cifrado AES (*Advanced Encryption Standard*) para comunicaciones inalámbricas, contribuiría a mitigar el riesgo de ataques y garantizar la protección de la comunicación entre las partes.

Por defecto, un *ROS Master Node* responderá a cualquier solicitud de cualquier dispositivo en la red con el que se puede conectar. La pagina web de ROS[2] sugiere el uso de una red aislada o un cortafuegos para restringir el acceso a él y utilizar extensiones para agregar seguridad.

Planes de contingencia ante errores

Se puede mejorar el manejo de errores del sistema. Actualmente, el usuario puede abortar la misión en cualquier momento. Una mejora a esta funcionalidad es la capacidad de abortar solamente un dron, y no la misión en su totalidad. Con esto, una reasignación de las tareas pendientes al resto de los drones sería una funcionalidad deseable. Esta reasignación implicaría que se deba planificar los caminos de los drones restantes en tiempo real. Este mecanismo también podría entrar en acción si un dron se desconectara durante el transcurso de la inspección.

Representación del espacio tridimensional

El espacio tridimensional de la Facultad de Ingeniería se representó como una grilla de ocupación para la planificación de caminos de transición. Este podría ser representado con un octree para reducir el número de nodos y mejorar el uso de la memoria y tiempo de ejecución del algoritmo A*.

PX4 Avoidance con múltiples drones

La funcionalidad de evasión de obstáculos provista por PX4 Avoidance no funciona cuando se emplean múltiples drones en el sistema. Si se desease im-

[2] `https://wiki.ros.org/Security`

plementar este sistema con esas condiciones, es aconsejable buscar alternativas que permitan la evitación de colisiones de los drones en tiempo de vuelo.

Balanceo de tiempos de vuelo por dron

Como se menciona en la Sección 3.5.3, la implementación actual no considera los costos individuales de cada recorrido, sino que busca minimizar el costo total del sistema. El rendimiento del sistema con múltiples drones se vería beneficiado de un algoritmo especializado en buscar una repartición equitativa de las cargas entre los *tours*, en relación a las distancias recorridas y tiempos de vuelo de cada dron.

Cámara térmica

La utilización de una cámara térmica no fue parte del alcance del proyecto. Sin embargo, una cámara de este estilo proveería de mayor información sobre la condición de los edificios a inspeccionar. Podría resultar de interés para inspeccionar las condiciones de la aislación térmica de la Facultad y realizar relevamientos que permitan resolver problemas, aumentando la eficiencia energética de las instalaciones. Muchos drones comerciales actuales tienen cámaras térmicas ya equipadas.

Computer vision

La utilización de *computer vision*, o el análisis de imágenes o videos para extraer información significativa, tampoco formó parte del alcance del proyecto. Sin embargo, se podría utilizar para detectar anomalías en los edificios de forma automática, sin la necesidad de tener que ver los videos grabados por el dron durante la inspección.

Referencias

Almadhoun, R., Taha, T., Seneviratne, L., y Zweiri, Y. (2019). A survey on multi-robot coverage path planning for model reconstruction and mapping. *SN Applied Sciences*, *1*(847). doi: 10.1007/s42452-019-0858-x

Avellar, G. S. C., Pereira, G. A. S., Pimenta, L. C. A., y Iscold, P. (2015). Multi-uav routing for area coverage and remote sensing with minimum time. *Sensors*. doi: 10.3390/s151127783

Baumann, T. (2018). *Obstacle avoidance for drones using a 3dvfh* algorithm* [Tesis de maestría].

Bektas, T. (2006). The multiple traveling salesman problem: an overview of formulations and solution procedures. *Omega*, *34*(3), 209-219. doi: https://doi.org/10.1016/j.omega.2004.10.004

Cabrera Núñez, G., Castro Ibarburu, A., y Salmini Rodríguez, S. (2021). *Endra : Enjambre de drones autónomos* (Tesis de grado). Udelar, Montevideo.

Czech Technical University in Prague, M.-r. S. G. (2023). *Mrs uav system - open source platform for uav research.* `https://mrs.felk.cvut.cz/`. (Consultado el 28/11/2023)

de Castro Pereira, S., Solteiro Pires, E. J., y de Moura Oliveira, P. B. (2023). Ant-balanced multiple traveling salesmen: Aco-bmtsp. *Algorithms*, *16*(1). doi: 10.3390/a16010037

Fari, S. (2017). *Guidance and control for a fixed-wing uav* (Tesis Doctoral). doi: 10.13140/RG.2.2.24973.28641

Fripp, N., Queirolo, A., Santellán, A., y Valentín, J. (2023). *Página web del proyecto de grado.* `https://pginspecciondrones.pages.fing.edu.uy/pgPages/`.

Generalized Intelligence. (2023). *Gaas - navigation algorithms.* `https://github.com/generalized-intelligence/GAAS/tree/main/algorithms/src/Navigation`. (Consultado el 29/11/2023)

Hart, P. E., Nilsson, N. J., y Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*.

Haviland, J., y Corke, P. (2022). *Python robotics.* `https://pythonrobotics.io/`. (Consultado el 28/11/2023)

Helicopter control. (s.f.). Descargado de `https://www.hubschrauberflug.de/en/docs/show/25/helicopter-control` (Consultado el 01/12/2023)

Iris quadcopter uav. (s.f.). Descargado de https://www.arducopter.co.uk/iris-quadcopter-uav.html (Consultado el 01/12/2023)

Kleinberg, J., y Tardos, É. (2005). *Algorithm design.* Boston, MA: Pearson/Addison-Wesley.

Latombe, J.-C. (1991). *Robot motion planning.* Springer.

Lu, Y. (2022). *Quadrotor control, path planning and trajectory optimization.* https://github.com/yrlu/quadrotor. (Consultado el 29/11/2023)

Marichal, N., Tomas-Rodriguez, M., Hernandez, A., Castillo-Rivera, S., y Campoy, P. (2013, 06). Vibration reduction for vision systems on board unmanned aerial vehicles using a neuro-fuzzy controller. *Journal of Vibration and Control*, *20*, 2243-2253. doi: 10.1177/1077546313479632

Menéndez, G., Bruzzone, L., Salmantón, A., y cols. (2022). *Análisis de estrategias de path planning para equipos de múltiples robots móviles* (Tesis de grado). Udelar, Montevideo.

Miller, C. E., Tucker, A. W., y Zemlin, R. A. (1960). Integer programming formulation of traveling salesman problems. *J. ACM*. doi: 10.1145/321043.321046

Murphy, R. R. (2000). *Introduction to ai robotics.* London, England: Bradford Books.

Nafis Ahmed, K. C., Chaitali J. Pawase. (2021). Distributed 3-d path planning for multi-uavs with full area surveillance based on particle swarm optimization. *Applied Sciences*.

Paliwal, P. (2023). A survey of a-star algorithm family for motion planning of autonomous vehicles. En *Ieee international students' conference on electrical, electronics and computer science.* Mumbai, India.

Patel, A. (2023). *Heuristics.* https://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html. (Consultado el 19/11/2023)

Shaun, R. (2023). *Xtdrone.* https://github.com/robin-shaun/XTDrone/blob/master/README.en.md. (Consultado el 29/11/2023)

Skorobogatov, G., Barrado, C., y Salamí, E. (2020). Multiple uav systems: A survey. *Unmanned Systems*, *08*(02), 149-169. doi: 10.1142/S2301385020500090

Wang, H., Zhao, H., Zhang, J., Ma, D., Li, J., y Wei, J. (2020). Survey on unmanned aerial vehicle networks: A cyber physical system perspective. *IEEE Communications Surveys Tutorials*, *22*(2), 1027-1070. doi: 10.1109/COMST.2019.2962207

Yang, L., Qi, J., Song, D., Xiao, J., Han, J., y Xia, Y. (2016). Survey of robot 3d path planning algorithms. *Journal of Control Science and Engineering*, *2016*, 7426913. doi: 10.1155/2016/7426913

Yu, S., Heo, J., Jeong, S., y Kwon, Y. (2016, 01). Technical analysis of vtol uav. *Journal of Computer and Communications*, *04*, 92-97. doi: 10.4236/jcc.2016.415008

Zhou, H. (2020). *Pathplanning.* https://github.com/zhm-real/PathPlanning. (Consultado el 29/11/2023)

Anexo A

Glosario

- **UAV** (*Unmanned Aerial Vehicle*): Vehículo aéreo no tripulado. Sinónimo de dron.
- ***Quadcopter***: Tipo de UAV con cuatro rotores.
- **LiDAR** (*Light Detection and Ranging*): Sensor que utiliza pulsos láser para medir distancias y generar mapas tridimensionales.
- **ROS** (*Robot Operating System*): Sistema operativo de robots utilizado para control y comunicación entre componentes robóticos.
- **Gazebo**: Entorno de simulación para robots.
- **PX4 Autopilot**: Plataforma de código abierto para control de vehículos no tripulados.
- **PX4 Avoidance**: Sistema de evasión de obstáculos para vehículos aéreos.
- **QGroundControl**: Interfaz de usuario para configurar y controlar vehículos aéreos no tripulados.
- **GCS** (*Ground Control Station*): Sistema de control terrestre para operar y monitorear vehículos no tripulados.
- ***Waypoint***: Punto de referencia utilizado en navegación para definir rutas de vuelo.
- ***Gimbal***: Dispositivo mecánico que estabiliza y orienta cámaras u otros sensores.
- **MAVLink** (*Micro Air Vehicle Link*): Protocolo de comunicación ligero para sistemas aéreos no tripulados.
- **MAVROS**: Paquete de ROS para la comunicación entre ROS y sistemas basados en MAVLink, como autopilotos de drones.

- **GNSS** (*Global Navigation Satellite System*): Sistema de navegación por satélite que incluye GPS, GLONASS, entre otros.
- **RViz**: Herramienta de visualización 3D para robots y datos sensoriales.
- **RQT**: Marco de trabajo de ROS visualización de datos en interfaces gráficas.
- **Grafo**: Estructura matemática que representa relaciones entre objetos.
- **TSP** (*Travelling Salesman Problem*): Problema del vendedor viajero que busca la ruta más corta que visita todos los puntos exactamente una vez.
- **MTSP** (*Multiple Travelling Salesman Problem*): Problema similar al TSP pero para múltiples vendedores viajeros.
- **PLE** (Programación Lineal Entera): Busca soluciones óptimas en problemas con variables enteras.
- **A*** (A-Estrella): Algoritmo de búsqueda heurística utilizado para encontrar la ruta más corta en un grafo.
- **Heurística**: Método de solución aproximada que guía hacia la solución óptima.
- **JWT** (*JSON Web Token*): Token de seguridad utilizado para la autenticación en aplicaciones web.
- **Nube de puntos**: Conjunto de puntos tridimensionales que representan la geometría de un entorno.
- **Grilla de ocupación**: Representación de un entorno que indica qué áreas están ocupadas o libres.
- **Malla tridimensional**: Estructura que divide el espacio en celdas tridimensionales para representar un entorno.
- **Octree**: Estructura de datos jerárquica utilizada para representar información tridimensional de manera eficiente.
- **Octomap**: Biblioteca para construir y manipular mapas tridimensionales utilizando octrees.
- **CVXPY**: Biblioteca de Python para el modelado y optimización convexa.
- **MOSEK**: Software de optimización para problemas matemáticos complejos.
- **SCIPY**: Otro software de optimización para problemas matemáticos complejos.

Anexo B

Tecnologías

B.1. Herramientas y tecnologías

B.1.1. Simulación y control

ROS[1]

ROS, del inglés *Robot Operating System*, es un conjunto de herramientas y bibliotecas de código abierto diseñado para el desarrollo de *software* para robots. Dentro de estas herramientas se incluye la abstracción de hardware, control de dispositivos de bajo nivel, un sistema de mensajería para permitir a diferentes componentes del robot comunicarse entre sí, un sistema de planificación de movimiento para controlar el movimiento del robot, mantenimiento de paquetes y otras funcionalidades comúnmente usadas. También provee herramientas y librerías para obtener, construir, escribir y ejecutar código en distintos robots.

ROS se utiliza comúnmente en la investigación y desarrollo de robots móviles, y se ha convertido en un estándar en la comunidad de robótica. Gracias a las abstracciones que ofrece este *framework*, es posible el desarrollo de la lógica de control del robot sin concentrarse en la programación de bajo nivel.

ROS utiliza un sistema de versionado basado en distribuciones, donde cada distribución representa una versión específica del sistema con mejoras y actualizaciones. Para la implementación de la solución se eligió usar la distribución Noetic. Es la última versión estable de las que se catalogan dentro de ROS 1 y está diseñada para proporcionar un soporte completo para Ubuntu 20.04.

En 2017 se introdujo una nueva versión de ROS, conocida como ROS 2, con múltiples cambios que marcaron una diferencia notoria con respecto a las versiones anteriores, por lo que fue necesario enfatizarlo en su nombre. En general, ROS 2 ofrece un conjunto más completo y mejorado de herramientas y características en comparación con ROS, lo que lo convierte en una opción más avanzada. Sin embargo, ROS sigue siendo más popular y una opción válida para el desarrollo.

[1] `https://www.ros.org`

Dado que ROS 2 es la última versión de ROS, inicialmente se decidió utilizarlo para la implementación de la solución, empezando con la distribución *Foxy*. Sin embargo, en una fase temprana del desarrollo se encontraron varias dificultades para su uso de forma correcta. Una de las mayores limitantes es la disponibilidad de paquetes y librerías con esta versión, que si estaban soportadas por ROS 1 en ese momento. Uno de los paquetes que no cuentan con un soporte para ROS 2 es PX4 Avoidance. Otra razón para desestimar el uso de ROS 2 fue la cantidad de documentación y discusión que se puede encontrar en internet. ROS 1 lleva siendo desarrollado y utilizado por la comunidad desde hace más tiempo, por lo que muchos de los problemas que se pueden encontrar en la implementación o integración de los diferentes módulos ya había sido transcurrido y discutido anteriormente para esa versión. Después de una evaluación de las ventajas y desventajas de las dos versiones, se tomó la decisión de cambiar a ROS 1 en su ultima distribución: *Noetic*. Con esta versión, el equipo pudo avanzar en la implementación.

RVIZ [2]

RVIZ es una herramienta de visualización 3D que forma parte de ROS. Permite la visualización de datos sensoriales como nubes de puntos, mapas y trayectorias. Además, puede utilizarse para la representación de modelos, tanto en 2 como en 3 dimensiones. La capacidad de RVIZ para brindar una representación visual en tiempo real de las interacciones y los datos generados por los nodos de ROS permite la comprensión y evaluación del comportamiento de los robots, siendo de gran utilidad para la depuración del sistema .

Gazebo[3]

Gazebo es un simulador 3D de robots de código abierto. Es una herramienta que permite experimentar el comportamiento de los robots en un entorno virtual, y brinda la posibilidad de representar la realidad con un gran nivel de fidelidad. Gazebo se puede utilizar con varios sistemas operativos de robots, siendo el entorno de simulación por defecto en los proyectos con ROS.

Hay una amplia variedad de simuladores de robots disponibles similares a Gazebo, tanto gratuitos como de pago. Algunos ejemplos de simuladores de robots populares son V-REP, Webots, CoppeliaSim, MORSE y MSRS *(Microsoft Robotics Studio)*. Estos simuladores varían en cuanto a su enfoque y características, pero en general ofrecen funcionalidades similares a Gazebo.

Se ha elegido el uso de Gazebo sobre los otros simuladores debido a su integración con ROS y PX4, siendo parte de la configuración base de los mismos. Ademas, su interfaz de usuario es intuitiva y fácil de usar, es gratuito y de código abierto, y cuenta con una gran comunidad utilizándolo y brindando soporte.

[2] `http://wiki.ros.org/rviz`
[3] `https://gazebosim.org/`

PX4[4]

PX4 es un sistema de control de vuelo para drones. Dentro de sus funcionalidades ofrece la capacidad de volar en modo autónomo, el soporte para diferentes tipos de sensores y la integración con diferentes sistemas de navegación y posicionamiento. Además, al ser de código abierto, es posible modificar y mejorar el sistema según sus necesidades.

PX4, ROS y Gazebo pueden trabajar juntos en proyectos de drones para proporcionar un sistema completo que abarca desde el control de vuelo hasta la programación del comportamiento del dron y la simulación en un entorno virtual. Además, PX4 permite la simulación del sistema ejecutando el código los drones en una computadora, sin necesitar del hardware real. Esto se conoce como *Software in the Loop*.

PX4 Avoidance[5]

PX4 Avoidance es un proyecto de PX4 que incluye tres planificadores que permiten a los drones detectar y evitar obstáculos mientras vuelan. Esta función se basa en la utilización de sensores para detectar obstáculos y calcular rutas de evasión en tiempo real. De esta manera, los drones pueden volar de manera más segura en entornos dinámicos y complejos.

Los planificadores del proyecto son: *local planner*, *global planner* y *safe landing planner*. Dada la realidad del problema, se utilizó el *local planner* como estrategia reactiva para la evasión de obstáculos.

El *local planner* se basa en el algoritmo 3DVFH* (Baumann, 2018). El algoritmo genera un histograma polar 2D a partir de la nube de puntos obtenida por la cámara, que se utilizará para modificar el movimiento del UAV en caso de encontrar un obstáculo. De esta forma se pueden evitar colisiones en tiempo real.

QGroundControl[6]

QGroundControl es una aplicación de *software* libre y de código abierto que se utiliza para controlar y monitorizar sistemas de UAVs. QGroundControl es compatible con una amplia variedad de sistemas de UAVs que utilizan MAVLink, incluyendo PX4. PX4 y QGroundControl han sido implementados por la misma empresa: Dronecode. Esto conlleva que haya un gran nivel de integración entre ambos productos y puedan ser utilizados juntos facilmente.

Se puede utilizar esta herramienta para una amplia variedad de propósitos, como configurar los parámetros de vuelo de un UAV (su altura máxima, su velocidad máxima o su tiempo máximo de vuelo entre otros), programar una ruta de vuelo para que el UAV la siga de manera autónoma, monitorear en tiempo real el estado del UAV, establecer comunicaciones seguras entre el UAV y la *Ground Control Station* (GCS), entre otros.

[4] `https://px4.io/`
[5] `https://github.com/PX4/PX4-Avoidance`
[6] `http://qgroundcontrol.com/`

B.1.2. Implementación

Ubuntu[7]

Ubuntu es un sistema operativo de código abierto basado en Linux. Específicamente, para utilizar la versión Noetic de ROS se utilizó la versión 20.04 de Ubuntu.

Python[8]

Python es un lenguaje de programación potente y fácil de aprender. Tiene estructuras de datos eficientes de alto nivel y un enfoque simple pero efectivo para la programación orientada a objetos. Es muy popular entre los usuarios de ROS, y cuenta con librerías altamente mantenidas, como lo es rospy[9] , la cual se utilizo en este proyecto.

Flask[10]

Flask es un web framework de código abierto escrito en Python. Flask se caracteriza por su diseño ligero y modular, lo que lo hace fácil de usar y extender. Por esta razón se utilizó para implementar el servidor del proyecto. Con Flask se implementó una API REST y conexión con *WebSockets*.

C++[11]

C++ es un lenguaje de programación de propósito general de bajo nivel y de código abierto. Es uno de los lenguajes de programación más utilizados y se caracteriza por su flexibilidad y potencia.

Si bien C++ es una excelente opción para el desarrollo de aplicaciones de robots ya que posee un alto rendimiento, también es un lenguaje más complejo y difícil de aprender que otros lenguajes disponibles, como Python. Además, debe manejarse la memoria con mucho cuidado. C++ fue utilizado en la implementación de algunos nodos, como por ejemplo, el nodo encargado de realizar la transición de caminos.

WebSockets

En una aplicación en tiempo real, los WebSockets pueden utilizarse para enviar actualizaciones de estado, notificaciones, mensajes de chat, resultados de búsqueda en tiempo real, datos de sensores, entre otros. Al utilizar WebSockets en lugar de solicitudes HTTP, se pueden reducir significativamente los tiempos de espera y mejorar la eficiencia de la comunicación.

---

[7] `https://ubuntu.com/`
[8] `https://docs.python.org/3/`
[9] `http://wiki.ros.org/rospy`
[10] `https://flask.palletsprojects.com/en/3.0.x/`
[11] `https://cplusplus.com/`

React[12]

React es un framework web de código abierto creado por Meta / Facebook. Se utiliza ampliamente en el desarrollo de aplicaciones web y móviles, desde sitios pequeños y simples hasta aplicaciones complejas. React se basa en *JavaScript* y es una excelente opción para facilitar el desarrollo de aplicaciones de página única.

MongoDB[13]

MongoDB es un sistema de gestión de bases de datos no relacionales de código abierto. MongoDB es *NoSQL*, lo que significa que no utiliza un esquema fijo para almacenar los datos, sino que permite una gran flexibilidad en el almacenamiento y el acceso a los datos. Es una buena opción para sistemas que estén en constante desarrollo o evolución, motivo por el cual se optó por utilizarlo.

B.1.3. Gestión de proyecto

Hugo[14]

Hugo es un generador de sitios web estáticos de código abierto escrito en Go. Para ello, utiliza plantillas y contenido escrito en formato *Markdown*. Esto permite a los desarrolladores crear sitios web rápidos, seguros y escalables de manera sencilla. Se utilizó para generar el sitio web del presente proyecto, en donde se encuentra una breve descripción del mismo (Fripp y cols., 2023).

Notion[15]

Notion es una plataforma de productividad que permite organizar y gestionar proyectos de una manera sencilla. La plataforma cuenta con herramientas como bases de datos, tablas, listas de tareas, calendarios, wikis, entre otras, que permiten crear y compartir contenido y colaborar con otros usuarios. En este trabajo se utilizó para guardar documentos, notas de reuniones, e incluso para la asignación y seguimiento de las tareas.

GitLab[16]

GitLab es una plataforma de código abierto para el desarrollo de *software*. Una de las principales características de GitLab es el control de versiones, es decir, permite a los desarrolladores guardar diferentes versiones de su código y hacer seguimiento de los cambios que se realizan. Además, se utilizó la funcionalidad de GitLab Pages, que permite a los usuarios publicar páginas web estáticas directamente desde sus repositorios de GitLab.

[12] `https://es.react.dev/`
[13] `https://www.mongodb.com/`
[14] `https://gohugo.io/`
[15] `https://www.notion.so/`
[16] `https://about.gitlab.com/`

MAVLink[17]

Es un protocolo de comunicación ligero diseñado para la transmisión de datos entre UAVs y la estación de tierra. Este protocolo se utiliza comúnmente en sistemas de vehículos aéreos no tripulados, como ArduPilot y PX4.

MAVLink se basa en un formato de paquete simple que permite la transmisión de datos importantes entre la aeronave como lo son la posición, estado de la batería, sensores y comandos de control, entre otros.

MAVROS, Es un conjunto de nodos ROS que se utilizan para interactuar con UAVs que siguen el protocolo MAVLink. MAVROS proporciona una interfaz entre ROS y MAVLink, permitiendo a los desarrolladores integrar drones con sistemas basados en ROS.

Algunas de las funciones de MAVROS incluyen la publicación de información del estado del vehículo, la suscripción a comandos de control y la gestión de servicios como el armado y desarmado del vehículo. Esencialmente, MAVROS actúa como un puente entre la infraestructura de ROS y la comunicación MAVLink utilizada por el sistema de control del dron.

B.2. Tecnologías exploradas y no usadas

Rust[18]

Rust es un lenguaje de programación de alto nivel y de código abierto. Es un lenguaje compilado ,y su principal ventaja es que proporciona un buen manejo de memoria. Es relativamente nuevo y actualmente no tiene soporte oficial de la plataforma ROS, pero hay un esfuerzo por parte de la comunidad para llevarlo a cabo. Inicialmente, se consideró la adopción de dicha herramienta en el proyecto. Sin embargo, tras un período de evaluación, se determinó que las librerías existentes no estaban lo suficientemente completas, lo que llevó a no seguir adelante con esta opción y concentrarse en tecnologías ya conocidas y soportadas oficialmente, como Python y C++.

Docker[19]

Docker es una plataforma de contenedores que permite a los desarrolladores desplegar y ejecutar aplicaciones de manera eficiente y portátil en cualquier entorno. Un contenedor es una forma de empaquetar y aislar una aplicación con todas sus dependencias, bibliotecas y otros componentes necesarios para su ejecución, de manera que se pueda ejecutar de manera consistente en cualquier entorno. Esto permite a los desarrolladores crear y desplegar aplicaciones de manera rápida, ya que los contenedores se pueden transportar fácilmente entre diferentes entornos de desarrollo, pruebas y producción.

[17] `https://mavlink.io/en/`
[18] `https://www.rust-lang.org/es`
[19] `https://www.docker.com/`

Octree

El octree es una estructura de datos en forma de árbol que se emplea para organizar y almacenar información en un espacio tridimensional. Cada nodo del octree representa un cubo que delimita una porción de dicho espacio, y a su vez, cada cubo se subdivide en ocho subcubos más pequeños, todos ellos de igual tamaño, mediante un proceso recursivo. Este proceso de subdivisión prosigue hasta que se alcanza una determinada profundidad en el árbol o hasta que cada cubo contenga un número máximo predefinido de elementos.

En aplicaciones donde se requiere un almacenamiento y acceso eficiente a datos en un espacio tridimensional, los octrees son ampliamente utilizados. Un octree puede ajustarse dinámicamente a diferentes niveles de detalle o resolución. A medida que la geometría es mas simple, se pierde resolución y detalle, pero mejora la eficiencia computacional. Son aplicados en el almacenamiento y acceso de objetos en tiempo real en videojuegos o en la búsqueda de rutas en entornos tridimensionales.

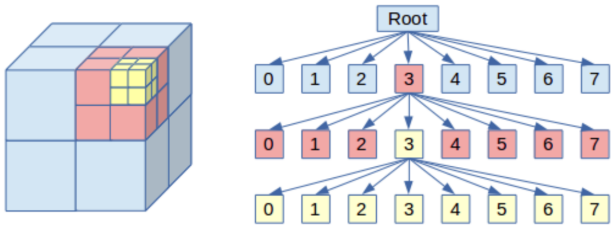

Figura B.1: Representación de un Octree.

Un octomap, derivado del concepto de octree previamente mencionado, constituye una representación de un entorno tridimensional mediante la estructura de datos de un octree. La generación de un octomap se lleva a cabo a través de la exploración minuciosa del entorno mediante la utilización de uno o varios sensores, como un láser o una cámara, con el objetivo de recopilar datos acerca de la ocupación de cada cubo dentro del octree. Con base en la información capturada, los cubos pueden ser designados como espacios libres si el sensor detecta que esta vacío, o como ocupados si el sensor detecta la presencia de algún obstáculo en el interior del cubo.

En el ámbito de la robótica y la navegación autónoma, los octomaps se utilizan para representar entornos tridimensionales y para realizar búsquedas de caminos en entornos dinámicos. Una de las principales ventajas de los octomaps es que permiten un acceso rápido a la información de ocupación de un cubo en el espacio tridimensional mediante la búsqueda en un octree.

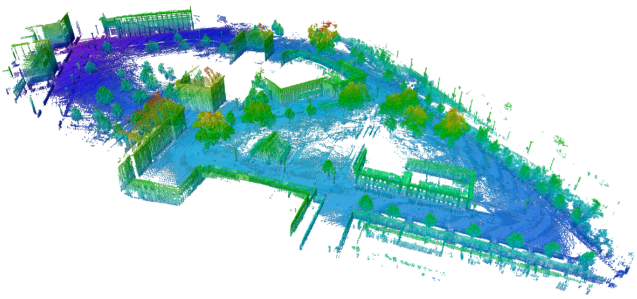

Figura B.2: Visualización de un octomap.

Anexo C

Datos de la realidad

C.1. Zonas de inspección

Tabla C.1: Lugares y los puntos aproximados que los definen, junto con el largo del camino de inspección.

Inicio de tabla		
Nombre	Largo (m)	Punto (lat, long, alt)
Punto Central	0	(-34.91789, -56.16607, 33.181)
Bóveda del Cuerpo Central	544.26	(-34.91812,-56.16659,49.8),
		(-34.91813,-56.16677,49.8)
		(-34.91883,-56.16672,49.8)
		(-34.91882,-56.16654,49.8)
Techo de la Biblioteca Central	593.57	(-34.91774,-56.16690,31.1)
		(-34.91832,-56.16686,31.1)
		(-34.91774,-56.16707,31.2)
		(-34.91833,-56.16702,31.1)
Techo de Bandejas	359.14	(-34.91756,-56.16706,22.1)
		(-34.91754,-56.16661,22.2)
		(-34.91741,-56.16660,22.3)
		(-34.91743,-56.16707,23.4)
Techo del Salón de Actos	207.004	(-34.91817,-56.16626,35.5)
		(-34.91831,-56.16625,35.4)
		(-34.91832,-56.16653,35.4)
		(-34.91818,-56.16654,35.4)
Techo del Aulario A	175.02	(-34.91700,-56.16674,22.4)
		(-34.91699,-56.16657,22.4)
		(-34.91721,-56.16656,22.4)
		(-34.91722,-56.16672,22.4)
		—

Continuación C.1		
Nombre	Largo (m)	Punto (lat, long, alt)
Techo del Aulario B	222.2	(-34.91701,-56.16682, 22.4)
		(-34.91702,-56.16699,22.4)
		(-34.91723,-56.16697,22.4)
		(-34.91722,-56.16681,22.4)
Techo del Aulario C	242.487	(-34.91702,-56.16707, 22.4)
		(-34.91702,-56.16723,22.4)
		(-34.91723,-56.16705,22.4)
		(-34.91724,-56.16722,22.4)
Techo Principal de Eléctrica	223.162	(-34.91911,-56.16614,28.5)
		(-34.91919,-56.16613,28.5)
		(-34.91922,-56.16679,28.5)
		(-34.91914,-56.16680,28.5)
Fachada Este del Salon de Actos	61.591	(-34.91816,-56.16622,29.5)
		(-34.91830,-56.16622,29.5)
		(-34.91830,-56.16622,23.5)
		(-34.91816,-56.16622,23.5)
Fachada Norte del Salon de Actos	112.819	(-34.91814,-56.16626,29.5)
		(-34.91816,-56.16656,29.5)
		(-34.91814,-56.16626,23.5)
		(-34.91816,-56.16656,23.5)
Fachada Sur del Salon de Actos	104.071	(-34.91835, -56.16652,29.5)
		(-34.91834,-56.16624,29.5)
		(-34.91835,-56.16652,23.5)
		(-34.91834,-56.16624,23.5)
Fachada Sur de la Biblioteca Central	98.119	(-34.91836, -56.16688,29.5)
		(-34.91837, -56.16704, 29.5)
		(-34.91837, -56.16704, 21)
		(-34.91836,-56.16688,21)
Fachada Este de la Biblioteca Central	328.095	(-34.91831,-56.16680,29.5)
		(-34.91773,-56.16683,29.5)
		(-34.91831,-56.16680,25.0)
		(-34.91773,-56.16683,25.0)
Fachada Oeste de la Biblioteca Central	152.855	(-34.91775,-56.16712,29.5)
		(-34.91833,-56.16708,29.5)
		(-34.91775,-56.16712,21.0)
		(-34.91833,-56.16708,21.0)
Canalón de Bandejas	126.497	(-34.91756,-56.16706,22.1)
		(-34.91754,-56.16660,22.2)
		(-34.91741,-56.16707,22.3)
		(-34.91743, -56.16661,23.4)
Canalón del Cuerpo Central	180.9	(-34.91812,-56.16659,49.8)
		(-34.91813,-56.16677,49.8)
		(-34.91883,-56.16672,49.8)

Continuación C.1		
Nombre	Largo (m)	Punto (lat, long, alt)
		(-34.91882, -56.16654,49.8)
Canalón de la Biblioteca Central	163.982	(-34.91774,-56.16690, 31.1)
		(-34.91832,-56.16686,31.1)
		(-34.91774,-56.16707,31.2)
		(-34.91833, -56.16702,31.1)
Canalón del Salón de Actos	86.862	(-34.91817,-56.16626,35.4)
		(-34.91831,-56.16625,35.4)
		(-34.91832,-56.16653,35.4)
		(-34.91818, -56.16654,35.4)
Canalón Principal de Eléctrica	163.837	(-34.91911,-56.16614,28.5)
		(-34.91919,-56.16613,28.5)
		(-34.91922,-56.16679,28.5)
		(-34.91914,-56.16680,28.5)
Fin de tabla		