UNIVERSIDAD DE ANTIOQUIA

IMPLEMENTACIÓN DE ALGORITMOS DE PLANIFICACIÓN DE RUTAS DE COBERTURA PARA UN VEHÍCULO AÉREO NO TRIPULADO EN UN AMBIENTE SIMULADO

Nelson Augusto Benítez Montoya

Universidad de Antioquia
Facultad de Ingeniería
Departamento de Ingeniería Electrónica y Telecomunicaciones
Medellín, Colombia
2021

Implementación de algoritmos de planificación de rutas de cobertura para un vehículo aéreo no tripulado en un ambiente simulado

Nelson Augusto Benítez Montoya

Trabajo de grado presentado como requisito para optar al título de:
Ingeniero Electrónico

Asesor:
Álvaro Jaramillo Duque, Ph.D en Ingeniería Eléctrica

Universidad de Antioquia
Facultad de Ingeniería
Departamento de Ingeniería Electrónica y Telecomunicaciones
Medellín, Colombia
2021

Tabla de contenido

Lista de figuras

Lista de tablas

Resumen

El mundo se encuentra actualmente en una transición hacia las energías renovables, siendo la energía solar una de las fuentes más prometedoras y usadas. Sin embargo, los paneles solares fotovoltaicos presentan grandes inconvenientes para su correcto funcionamiento debido a que la suciedad y los efectos causados por el medio ambiente afectan la potencia de salida. Por esta razón, el mantenimiento periódico es esencial para alargar la vida útil y garantizar un buen desempeño de estos sistemas. El uso de vehículos aéreos no tripulados (VANTs, UAVs del inglés: Unmanned Aerial Vehicles) para diagnosticar los problemas presentes en los paneles, favorecen el diagnóstico rápido y efectivo como también la planeación de las tareas de limpieza y mantenimiento.

Este trabajo de grado se propone implementar algoritmos de rutas de cobertura para áreas que contienen instalaciones fotovoltaicas, como también evaluar dichas implementaciones en un ambiente simulado. Se toman en cuenta los estudios existentes en la literatura relacionados con los diferentes enfoques empleados en problemas de planificación de rutas de cobertura, especialmente aquellos que utilizan VANTs. En las rutas se generarán patrones de vuelo de geometrías simples y soluciones basadas en descomposición celular exacta y descomposición celular aproximada del área de interés. Algunas métricas de rendimiento, que se suelen aplicar para evaluar el éxito de la cobertura, fueron calculadas en varios casos.

Palabras clave: VANT, ruta de cobertura, ambiente simulado, métricas.

Introducción

El Grupo de Manejo Eficiente de la Energía (GIMEL) adscrito a la Facultad de Ingeniería de la Universidad de Antioquia, realiza investigación, asesoría y consultoría, orientadas al manejo eficiente de la energía. Mantiene una interacción fructífera con el sector energético y en general con los sectores productivos regionales y nacionales, así como con el sistema nacional de ciencia, tecnología e innovación.

Actualmente, el grupo investiga métodos novedosos que permitan la detección automática de suciedad y deterioros causados por el ambiente en paneles fotovoltaicos. La detección temprana de los factores anteriores, en conjunto con el mantenimiento adecuado, permite que las pérdidas de potencia generadas en estos sistemas sean mínimas.

El grupo también posee una base de datos fotográfica de instalaciones fotovoltaicas en las que, previamente, definió el área de interés que se desea recorrer para detectar dichas fallas mediante el uso de cámaras instaladas en Vehículos Aéreos No Tripulados. Las áreas incluyen medianas y grandes extensiones de terreno, lo que justifica el uso de VANTs o drones, como se conocen comúnmente.

Además, para garantizar la detección de las fallas o suciedad, la ruta que debe seguir el VANT deberá recorrer en su totalidad las áreas de interés ya establecidas. Dicho problema ya ha sido investigado y abordado mediante la planeación de la ruta de cobertura (CPP en inglés) y consiste en determinar una trayectoria que garantice que el vehículo o robot, que llevará a cabo la tarea, recorrerá todos los puntos de un área de interés al menos una vez. Algunas de las aplicaciones de la CPP se han dado en agricultura [1], limpieza de pisos [2], monitoreo de vida salvaje [3] y procesos industriales como pintura o lijado [4].

Por lo anterior, el grupo de investigación determinó que es necesario implementar y evaluar rutas de cobertura para inspeccionar sistemas fotovoltaicos. Para evaluar los algoritmos y las rutas obtenidas se considerarán diferentes áreas a inspeccionar y se utilizarán métricas que permitan evaluar el desempeño y, por ende, determinar el mejor de los algoritmos en los casos considerados. Dicha implementación debe permitir el desarrollo de pruebas de una forma rápida y a bajo costo mediante el trabajo de grado de un estudiante de pregrado.

Este trabajo de grado busca que, mediante la utilización de software libre de simulación de drones, se puedan validar las rutas de cobertura para dar solución a la necesidad que actualmente tiene el proyecto de investigación del grupo GIMEL. Dicho ambiente de simulación se seleccionará con base en los trabajos de investigación previos que se hallen en la literatura y la documentación existente.

Objetivos

Objetivo General

Evaluar algoritmos de rutas de cobertura en VANTs, mediante su implementación en el lenguaje de programación Python y posterior validación en ambientes de simulación de cuadricópteros, para seleccionar los que tienen mejor desempeño para la cobertura de un área fotovoltaica.

Objetivos específicos

- Seleccionar, al menos, dos algoritmos de rutas de cobertura que puedan ser validados en el ambiente de simulación con el fin de evaluar su desempeño utilizando las métricas de longitud de la trayectoria cubierta, tiempo de recorrido, porcentaje de recorrido del área total y redundancia de puntos recorridos.

- Implementar los algoritmos seleccionados en el lenguaje de programación Python para su posterior validación en la plataforma de simulación.

- Analizar las plataformas de simulación de robots aéreos existentes para seleccionar una que permita evaluar las rutas de cobertura obtenidas con los algoritmos implementados en Python.

- Proponer una interfaz que permita la selección de los algoritmos de cobertura y sus parámetros de simulación, por parte el usuario, para que facilite su uso y las pruebas en diferentes casos.

1. Marco Teórico

1.1. Python

Python es un lenguaje de programación interpretado cuya filosofía hace hincapié en la legibilidad de su código. Se trata de un lenguaje de programación multiparadigma, ya que soporta parcialmente la orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, dinámico y multiplataforma. Es administrado por la Python Software Foundation. Posee una licencia de código abierto, denominada Python Software Foundation License [5].

1.2. API

Una API es un conjunto de definiciones y protocolos que se utiliza para desarrollar e integrar el software de las aplicaciones [6]. API significa interfaz de programación de aplicaciones.

Las API permiten que productos y servicios se comuniquen con otros, sin necesidad de saber cómo están implementados. Esto simplifica el desarrollo de las aplicaciones y permite ahorrar tiempo y dinero. Las API otorgan flexibilidad; simplifican el diseño, la administración y el uso de las aplicaciones, y proporcionan oportunidades de innovación, lo cual es ideal al momento de diseñar herramientas y productos nuevos (o de gestionar los actuales) [6].

1.3. QT

Qt es utilizado por las principales empresas y desarrolladores mundiales en todo el mundo, y la tecnología permite a sus clientes ofrecer experiencias de usuario excepcionales y avanzar en sus iniciativas de transformación digital [7]. Qt logra esto a través de su marco de software multiplataforma para el desarrollo de aplicaciones y dispositivos, tanto bajo licencias comerciales como de código abierto [7].

Aproximadamente un millón de desarrolladores en todo el mundo utilizan la tecnología Qt. Permite un código de software único en todos los sistemas operativos, plataformas y tipos de pantalla, desde computadoras de escritorio y sistemas integrados hasta aplicaciones críticas para el negocio, sistemas en vehículos, dispositivos portátiles y dispositivos móviles conectados al Internet de las cosas [7].

1.4. Ubuntu 20.04 LTS

Ubuntu es una de las distribuciones del sistema operativo Linux de software libre y código abierto. Ubuntu 20.04 LTS (del inglés, Long Term Support) es la última versión que se ha lanzado al mercado y su nombre oficial es Focal Fossa [8]. Puede correr en computadores de escritorio y servidores. Está orientado al usuario promedio, con un fuerte enfoque en la facilidad de uso y en mejorar la experiencia del usuario y normalmente distribuido bajo una licencia libre o de código abierto [8].

Su patrocinador, Canonical, ofrece el sistema de manera gratuita, y se financia por medio de servicios vinculados al sistema operativo y vendiendo soporte técnico. Además, al mantenerlo libre y gratuito, la empresa es capaz de aprovechar los desarrolladores de la comunidad para mejorar los componentes de su sistema operativo [8].

Canonical, además de mantener Ubuntu, provee una versión orientada a servidores, Ubuntu Server; una versión para empresas, Ubuntu Business Desktop Remix; una para televisores, Ubuntu TV; otra versión para tabletas, Ubuntu Tablet; también Ubuntu Phone y una para usar el escritorio desde teléfonos inteligentes, Ubuntu for Android [8].

Cada seis meses se publica una nueva versión de Ubuntu. Esta recibe soporte por parte de Canonical durante nueve meses por medio de actualizaciones de seguridad, parches para bugs críticos y actualizaciones menores de programas. Las versiones LTS, que se liberan cada dos años [8], reciben soporte durante cinco años en los sistemas de escritorio y de servidor.

1.5. ROS

ROS es un marco flexible para escribir software de robots con una colección de herramientas, bibliotecas y convenciones que tienen como objetivo simplificar la tarea de crear un comportamiento robótico complejo y robusto en una amplia variedad de plataformas robóticas [9]. ROS se creó desde cero para fomentar el desarrollo colaborativo de software de robótica. Por ejemplo, un laboratorio podría tener expertos en mapeo de ambientes interiores y podría contribuir con un sistema de clase mundial para producir mapas. Otro grupo podría tener expertos en el uso de mapas para navegar, y otro grupo podría haber descubierto un enfoque de visión por computadora que funciona bien para reconocer objetos pequeños. ROS fue diseñado específicamente para que grupos como estos colaboren y se basen en el trabajo de los demás, como se describe a lo largo del sitio web [9].

1.5.1. Nodos ROS

La unidad base en ROS se llama nodo. Los nodos se encargan de manejar dispositivos o algoritmos informáticos, cada nodo para una tarea separada. Los nodos pueden comunicarse

entre sí mediante temas o servicios. También pueden publicar o subscribirse a un tema o tópico y pueden proveer o usar un servicio. Usan una librería cliente de ROS para comunicarse con los otros nodos [10].

1.5.2. Tópicos o temas ROS

En ROS, el tema es un flujo de datos que se utiliza para intercambiar información entre nodos. Se utilizan para enviar mensajes frecuentes de un determinado tipo o clase. Esto podría ser una lectura del sensor o la velocidad objetivo del motor. Cada tema se registra con un nombre único y con un tipo de mensaje definido. Los nodos pueden conectarse con el tema para publicar mensajes o suscribirse a ellos. Para un tema determinado, un nodo no puede publicarlo y suscribirse al mismo tiempo, pero no hay restricciones en la cantidad de nodos diferentes que publican o suscriben [11].

1.5.3. Servicios ROS

La comunicación por servicios se asemeja al modelo cliente-servidor. En este modo, un nodo (el servidor) registra el servicio en el sistema. Posteriormente, cualquier otro nodo puede solicitar ese servicio y obtener respuesta. A diferencia de los temas o tópicos, los servicios permiten la comunicación bidireccional, ya que una solicitud también puede contener algunos datos [12].

1.5.4. Maestro ROS

El maestro ROS (ROS Master en inglés) proporciona servicios de nomenclatura y registro al resto de nodos del sistema ROS. Realiza un seguimiento de los editores y suscriptores de temas y servicios [13]. El papel del maestro es permitir que los nodos ROS individuales se ubiquen entre sí como se observa en Figura 1 y Figura 2. Una vez que estos nodos se han localizado, se comunican entre sí de igual a igual como se puede ver en la Figura 3.

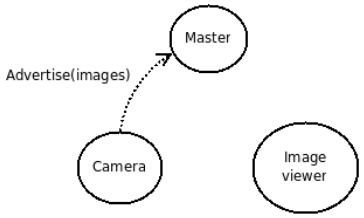


Figura 1. Publicación de un tópico en ROS, tomada de [13]

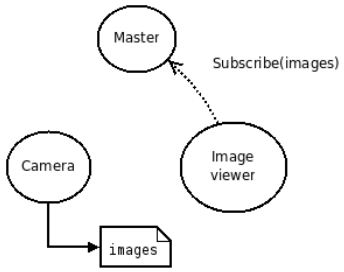


Figura 2. Suscripción a un tópico en ROS, tomada de [13]

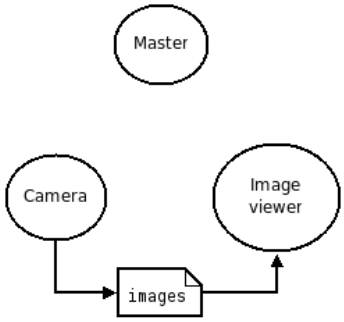


Figura 3. Comunicación entre tópicos en ROS, tomada de [13]

1.5.5. Librerías cliente

Las librerías cliente de ROS permiten que los nodos sean escritos en diferentes lenguajes de programación para comunicarse [14]. Por ejemplo, se tienen las siguientes librerías cliente:

- rospy: librería cliente en Python.
- roscpp: librería cliente en C++.

1.5.6. roslaunch

La herramienta "roslaunch" es utilizada para lanzar fácilmente múltiples nodos ROS de forma local y remota a través de SSH, así como para configurar parámetros en el servidor de parámetros[15]. Incluye opciones para reaparecer automáticamente los procesos que ya han muerto. La herramienta "roslaunch" toma uno o más archivos de configuración XML (con la extensión. launch) que especifican los parámetros para establecer y los nodos para iniciar, así como las máquinas en las que deben ejecutarse [15].

Para lanzar un nodo se usa la terminal con los siguientes comandos.

```
$ roslaunch package_name file.launch
```

1.5.7. roscore

La herramienta "roscore" es una colección de nodos y programas que son requisitos previos de un sistema basado en ROS. Se debe tener un "roscore" en ejecución para que los nodos ROS se comuniquen [16].

Si usa "roslaunch", automáticamente iniciará "roscore" si detecta que aún no se está ejecutando (a menos que se proporcione el argumento - -wait) [16].

La herramienta "roscore" iniciará: un maestro ROS, un servidor de parámetros ROS y un nodo de registro "rosout".

1.6. Drones o VANTs

Los vehículos aéreos no tripulados (VANTs) son aeronaves con autopropulsión que operan a control remoto o de forma automática [17]. Estas plataformas son operadas de forma remota por un ser humano, pero también puede realizar vuelos programados y automatizados que se pueden ejecutar utilizando sistemas inteligentes integrados con los sensores a bordo. Los vehículos aéreos no tripulados se usan cada vez más en una amplia gama de aplicaciones entre ellas vigilancia [18], agricultura inteligente [19], fotogrametría [20], gestión de desastres, seguridad civil [21] e inspección de líneas eléctricas [22].

1.7. Ambiente de simulación

La simulación ha sido reconocida como una importante herramienta de investigación desde principios del siglo XX [23]. Al principio, la simulación fue ante todo una herramienta de investigación académica. Con el desarrollo de las computadoras se ha impulsado la simulación a nuevos niveles. Es una herramienta poderosa que respalda el diseño, la planificación, el análisis y las decisiones en diferentes áreas de investigación y desarrollo. Por supuesto, la robótica como rama tecnológica moderna no es la excepción [23].
La simulación es el proceso de diseñar un modelo de un sistema físico real o teórico, ejecutando el modelo, y permitiendo analizar el resultado de la ejecución. Ser capaz de simular abre una amplia gama de opciones para resolver muchos problemas de forma creativa. Se puede investigar, diseñar, visualizar y probar un objeto incluso si no existe; permite ver los resultados de un sistema aún por construir. Es posible que sus soluciones fallen o incluso exploten, pero solo en simulación. Usando las herramientas de simulación se pueden evitar

lesiones y daños, también se evitan cambios innecesarios en el diseño después de que la producción de piezas ya haya comenzado [23]. Adicionalmente la simulación permite el desarrollo de pruebas de una forma rápida y a bajo costo: comparado con un proceso típico de construir un modelo o robot, donde después de cada fallo, se debe reparar o reconstruir el modelo.

1.7.1. Ambientes de simulación disponibles

Para este trabajo de grado, se consideraron los siguientes softwares de simulación gratuitos disponibles.

1.7.1.1. Webots

Es un simulador de robot móvil tridimensional de código abierto. Originalmente se desarrolló como una herramienta de investigación para varios algoritmos de control en robótica móvil. Desde diciembre de 2018, Webots se lanza como un software de código abierto bajo la licencia Apache 2.0 [24].

Para usar este simulador, se debe tener un conocimiento mínimo en robótica móvil, en programación C, C++, Java, Python o MATLAB, y en VRML97 (Lenguaje de modelado de realidad virtual).

Aunque permite simular drones o VANTs, dentro de la documentación existente consultada, no se pudo establecer que pudiera simular drones existentes en el mercado.

1.7.1.2. Gazebo

Es una plataforma gratuita de código abierto que se puede utilizar para diseñar, desarrollar, probar y visualizar casi cualquier tipo de robot [25]. Gazebo se ejecuta en Linux, Windows y Mac, y tiene soporte integrado para ROS (Robot Operation System) y Player .

También viene con algunos modelos de robots como PR2, DX, Irobot Create y TurtleBot [25], por lo que puede comenzar rápidamente incluso si no tiene sus propios modelos de robot. También es compatible con una amplia gama de sensores, y puede simular ruido y fallas del sensor para simular con precisión problemas del mundo real. Incluso se puede ejecutar Gazebo en la nube e interactuar con el simulador utilizando un simple navegador web. Esto es perfecto si desea trabajar en una computadora portátil, ya que puede alquilar una instancia de AWS para probar y diseñar rápidamente un robot.

El simulador Gazebo permite probar rápidamente algoritmos, diseñar robots, llevar a cabo pruebas de regresión y entrenar sistemas de inteligencia artificial usando escenarios realistas [25]. Gazebo ofrece la habilidad de simular poblaciones de robots de manera eficiente y precisa en ambientes internos y externos complejos. Al alcance de la mano se tiene un motor de física robusto [26], gráficos de alta calidad y convenientes interfaces programáticas y gráficas. Lo mejor de todo es que Gazebo es software libre con una comunidad vibrante [26].

1.7.1.3. Microsoft Robotics Developer Studio

Se puede utilizar para crear, probar y desarrollar una amplia gama de robots en un entorno simulado en 3D. Dado que está fabricado por Microsoft, solo se ejecuta en el sistema operativo Windows. Tiene soporte para la mayoría de las plataformas de robots como LEGO Mindstorms, VEX, y también tiene soporte para diferentes tipos de sensores [27].

Lamentablemente, Microsoft cerró su división de investigación en robótica y hay muy poco soporte para el MRDS. Microsoft Robotics Developer Studio Versión 4 es la versión final que lanzaron, y no parece que Microsoft actualice el MRDS en ningún momento en el futuro previsible.

1.7.2. Selección del ambiente de simulación

Para llevar a cabo la simulación de este trabajo, se seleccionó Gazebo por las siguientes razones: es software libre y puede ser descargado de la web de gazebo [14]; dicho ambiente de simulación posee amplia documentación; además, es el más mencionado y usado en simulación en los artículos científicos consultados que tratan las rutas de cobertura en VANTs; también permite modelar VANTs comerciales mediante el uso del software PX4 autopilot que se puede hallar en la web de PX4 [26]; permite la visualización de la ruta simulada a través del software QGroundControl que se puede descargar de la web de Dronecode [28]. Además, la interacción entre PX4, QgroundControl y Gazebo se logra mediante ROS (del inglés Robot Operating System) que presenta abundante documentación que se pueden hallar en la web de ROS [29].

1.7.3. Softwares adicionales al ambiente de simulación

Para cumplir con los objetivos propuestos en el trabajo de grado, fue necesario hacer uso de softwares adicionales al ambiente de simulación Gazebo. Dichas necesidades se hicieron evidentes en el proceso de investigación de los ambientes disponibles y las dependencias asociadas para su correcto funcionamiento.

1.7.3.1. Sistema operativo

El sistema operativo seleccionado fue Ubuntu 20.04 LTS. Las razones para dicha selección fueron las siguientes:

- Las versiones de sistemas operativos compatibles y soportados para el desarrollo de PX4 son Ubuntu 18.04 LTS (Bionic Beaver) y 20.04 (Focal Fossa) [30].
- Ubuntu 18.04 LTS trae instalado por defecto la versión 2.x de Python que dejó de tener soporte en enero de 2021 [5].

1.7.3.2. ROS

Dentro del proceso investigativo de los ambientes de simulación disponibles, se halló que una de las dependencias necesarias para el proceso de simulación con Gazebo es ROS (del inglés Robot Operating System). ROS permite la interacción de Gazebo con PX4 y los algoritmos diseñados en Python. Las instrucciones de instalación para Ubuntu 20.04 LTS se pueden hallar en la web de ROS [29].

Otros motivos para seleccionar ROS fueron los siguientes:

- En la literatura consultada, la mayoría de las implementaciones de rutas de cobertura utilizaron ROS.
- Posee una amplia documentación en línea, con códigos y ejemplos para crear cada uno de los módulos de un proyecto.
- Es un software de código abierto.

1.7.3.3. QGroundControl

Para validar el recorrido del VANT para cada una de las rutas obtenidas, se seleccionó el software QGroundControl. Este se puede descargar de la web de Dronecode [28]. Las razones principales para dicha elección fueron:

La herramienta QGroundControl proporciona control de vuelo completo y configuración del vehículo para vehículos con motor PX4 o ArduPilot. Además, proporciona un uso fácil y directo para principiantes, al mismo tiempo que ofrece soporte de funciones de alta gama para usuarios experimentados [31]. Sus características principales son:

- Instalación y configuración completa de vehículos impulsados por ArduPilot y PX4 Pro.

- Soporte de vuelo para vehículos que ejecutan PX4 y ArduPilot (o cualquier otro piloto automático que se comunique mediante el protocolo MAVLink).

- Planificación de misión para vuelo autónomo.

- Visualización del mapa de vuelo que muestra la posición del vehículo, la ruta de vuelo, los puntos de referencia y los instrumentos del vehículo.

- Transmisión de video con superposiciones de pantalla de instrumentos.

- Soporte para la gestión de múltiples vehículos.

- Se ejecuta en Windows, OS X, plataformas Linux, dispositivos iOS y Android.

1.7.3.4. PX4

Para simular un dron en Gazebo hay dos alternativas: diseñar el modelo desde cero o usar modelos ya terminados y probados de terceros. Al ser uno de los objetivos del trabajo de grado la evaluación de rutas de cobertura mediante la simulación de robots aéreos existentes se optó por usar un software de terceros como lo es PX4.

PX4 posee las siguientes características:

- PX4 es un software de control de vuelo de código abierto para drones y otros vehículos no tripulados.

- Proporciona un conjunto flexible de herramientas para que los desarrolladores de drones compartan tecnologías para crear soluciones personalizadas para aplicaciones de drones [32].

- PX4 proporciona un estándar para brindar soporte de hardware y software para drones, brindando un ecosistema para construir y mantener hardware y software de manera escalable.

- PX4 es parte de Dronecode, una organización sin fines de lucro administrada por Linux Foundation para fomentar el uso de software de código abierto en vehículos voladores. Dronecode también aloja QGroundControl, MAVLink y el SDK [32].

- Posee amplia documentación en línea y ejemplos de implementación y simulación con Gazebo [33].

1.8. Ruta de cobertura

La planificación de la ruta de cobertura (en inglés Coverage Path Planning, CPP) es la tarea de determinar una ruta que pasa por todos los puntos de un área o volumen de interés evitando obstáculos.

Los algoritmos de rutas de cobertura se pueden clasificar como fuera de línea o en línea. Dicha clasificación fue propuesta originalmente en [34]. Los algoritmos fuera de línea se basan solo en información, y se supone que el medio ambiente se conoce de antemano. Sin embargo, asumir un conocimiento previo completo del medio ambiente puede ser poco realista en muchos escenarios. Por otro lado, los algoritmos en línea no asumen un conocimiento previo completo del medio ambiente a ser cubierto y utilizan las mediciones de sensores en tiempo real para cubrir el espacio objetivo. Por lo tanto, estos algoritmos también son llamados algoritmos de cobertura basados en sensores.

Uno de los enfoques más comunes para resolver el problema de CPP en entornos poligonales 2D es emplear una estrategia de divide y vencerás. En entornos complejos, a menudo es inviable y computacionalmente inmanejable determinar una ruta de cobertura óptima para todo un entorno en un solo paso. En su lugar, muchos autores emplean un paso de descomposición en el que dividen el entorno en un conjunto de celdas, calculan una ruta óptima o casi óptima para cada celda y luego se concatenan dichas rutas. Los métodos presentados en [35] que emplean el enfoque de partición son referidos como métodos exactos de descomposición en celdas en la literatura.

1.8.1. Métodos exactos de descomposición en celdas

Los métodos exactos de descomposición en celdas particionan el espacio libre (es decir, el espacio libre de obstáculos) en regiones simples y no superpuestas llamadas celdas. La unión de todas las celdas llena exactamente el espacio libre. Estas regiones, donde no hay obstáculos, son "fáciles" de cubrir y pueden ser barridas por el robot con movimientos simples. Por ejemplo, cada celda podría cubrirse usando un patrón de ida y vuelta [36].

Normalmente, un planificador basado en la descomposición exacta en celdas genera una ruta de cobertura en dos pasos. El primer paso consiste en realizar algún tipo de descomposición del espacio de trabajo, que resulta en un conjunto de celdas que conforman el espacio de trabajo original. El segundo paso implica calcular una ruta para cada región por separado [37].

1.8.2. Métodos basados en cuadrículas

Los métodos basados en cuadrículas utilizan una representación del entorno descompuesto en una colección de celdas cuadriculadas uniformemente. Esta representación de cuadrícula fue propuesta por primera vez en [38], para trazar un mapa de un ambiente interior utilizando un sonar montado en un robot móvil. En esta representación, cada cuadrícula tiene un valor asociado que indica si hay un obstáculo o si es un espacio libre. El valor puede ser binario o una probabilidad. Normalmente, cada celda es un cuadrado, pero también se pueden usar diferentes formas de celda, por ejemplo, triángulos.

Como las representaciones en cuadrícula solo aproximan la forma de la región de destino y sus obstáculos. En el trabajo presentando en [34] se clasificaron los métodos basados en cuadrículas como descomposiciones en celdas aproximadas. Como resultado de esta representación aproximada, la mayoría de los métodos basados en cuadrículas son tipo "resolución completa", es decir, su integridad depende de la resolución del mapa de cuadrícula.

Es fácil crear un mapa de cuadrícula, ya que se puede representar como una matriz, donde cada elemento contiene información de ocupación asociada con una celda. Por otro lado, es sencillo marcar las áreas cubiertas en un mapa de cuadrícula. Como resultado, las representaciones basadas en cuadrículas son las más utilizadas para algoritmos de cobertura. No obstante, los mapas de cuadrículas sufren un crecimiento exponencial del uso de memoria porque la resolución permanece constante independientemente de la complejidad del medio ambiente. Además, requieren una localización precisa para mantener la coherencia del mapa [39].

Por estas razones, los métodos de cobertura basados en cuadrículas son adecuados para operaciones de robots móviles en interiores, ya que el tamaño del área a cubrir suele ser relativamente pequeño.

1.9. Métodos usados para obtener la ruta de cobertura.

Para el trabajo de grado se implementaron 3 métodos para calcular la CPP: método BCD, método STC y método Wavefront.
En la literatura consultada, estos métodos han sido descritos como algoritmos usados en VANTs, como se describe en [40] y [41]. Además, estos métodos han sido usados como punto de partida para generar rutas tomando en cuenta otras métricas como la energía, tal y como se describe en [42] y [43].

1.9.1. Ruta de cobertura Boustrophedon Cell Decomposition (BCD)

En [44] se propuso la técnica de descomposición exacta en celdas boustrophedon. La palabra "boustrophedon" viene del griego antiguo que significa literalmente "el camino del buey". Significa que el patrón que se logra es el mismo que genera un buey cuando arrastra un arado de un lado a otro. Este método asume que el área posee obstáculos poligonales y que el terreno es conocido a priori, siendo este clasificado como un método offline. El objetivo de la técnica es separar un polígono en un número mínimo de polígonos convexos, Figura 4; luego la trayectoria es calculada en cada uno de estos polígonos con patrón de ida y vuelta donde el robot sigue siempre la misma dirección (de norte a sur o viceversa),Figura 5.

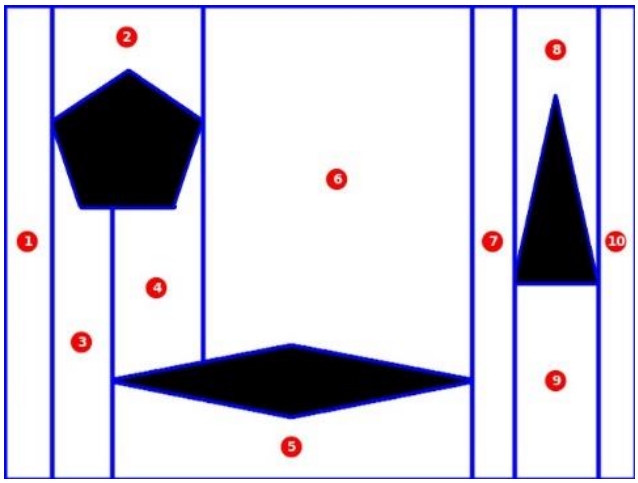


Figura 4. Partición del espacio usando el método BCD, gráfica tomada de [45]

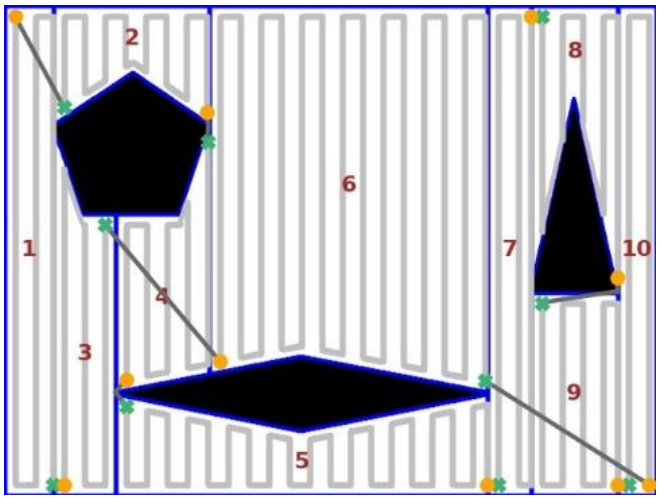


Figura 5. Rutas calculadas en cada partición realizada por método BCD, gráfica tomada de [45]

La ruta final se obtiene al unir las rutas generadas individualmente en cada uno de los polígonos, como se observa en la Figura 5, empezando en el polígono marcado con el número 1 y terminando en el polígono marcado con el número 10.

1.9.2. Ruta de cobertura Wavefront

En [46] se propuso el primer método basado en celdas para una planeación de ruta de cobertura. En el método fuera de línea, se usó una representación en celdas y se aplicó un algoritmo completo de planeación de ruta de cobertura. El método requiere de una celda de inicio y una celda objetivo. Una transformación de distancia que propaga un frente de onda desde el objetivo hasta el inicio se utiliza para asignar un número específico a cada elemento de la cuadrícula. Es decir, el algoritmo primero asigna un 0 al objetivo y luego un 1 a todas sus celdas circundantes. Luego, todas las celdas vecinas no marcadas a las marcadas con 1 son etiquetados con un 2. El proceso se repite incrementalmente hasta que el frente de onda alcanza la celda de inicio, tal como se ve en la Figura 6.

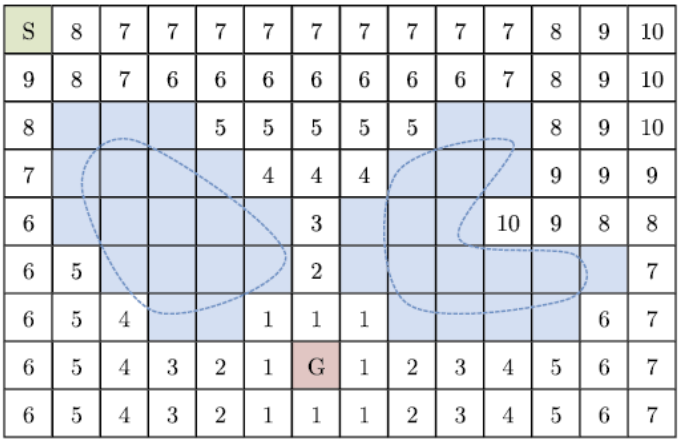


Figura 6. Transformación de distancia para la selección del punto de inicio S y el punto objetivo G, imagen tomada de [41]

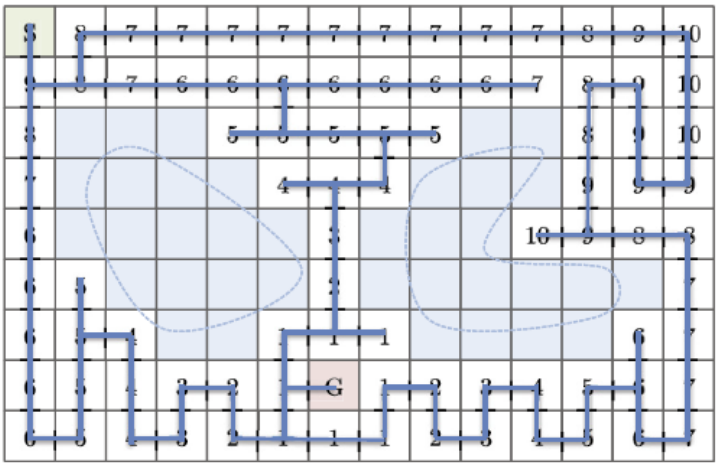


Figura 7. Ruta generada por el método Wavefront, imagen tomada de [41]

Una vez que se calcula la transformación de distancia, una ruta de cobertura puede ser encontrada comenzando en la celda de inicio y seleccionando la celda vecina con la etiqueta más alta que no ha sido visitada, Figura 7. Si dos o más celdas no visitadas y vecinas comparten la misma etiqueta, una de ellas se selecciona al azar.

1.9.3. Ruta de cobertura Spanning Tree Coverage (STC)

Este método consiste en subdividir el espacio de trabajo en un mapa de celdas y seguir una trayectoria sistemática en espiral. Esta trayectoria sistemática se genera al seguir un árbol de expansión del mapa de cuadrícula parcial que el robot construye de forma incremental, utilizando sus sensores integrados. El robot es capaz de cubrir cada celda de la cuadrícula con precisión una vez, y realizar una ruta de cobertura completa [41].

El algoritmo STC funciona de la siguiente manera. Se usan dos tamaños de celdas diferentes. Las células más grandes (las llamadas megaceldas) se dividen en cuatro celdas más pequeñas, que son del mismo tamaño que el robot, Figura 8. Para realizar la cobertura, el robot ejecuta el siguiente procedimiento. Comenzando en la celda actual, el robot elige una nueva dirección de viaje seleccionando la primera nueva megacelda en el espacio libre en sentido antihorario. Luego, un nuevo borde de árbol de expansión se genera desde la megacelda actual a la nueva [41]. El algoritmo se llama de forma recursiva, tal como se ve en la Figura 9. La recursividad se detiene solo cuando la celda actual no tiene vecinos nuevos (una megacelda se considera vieja si al menos una de sus cuatro celdas más pequeñas está cubierta, de lo contrario se considera nueva).

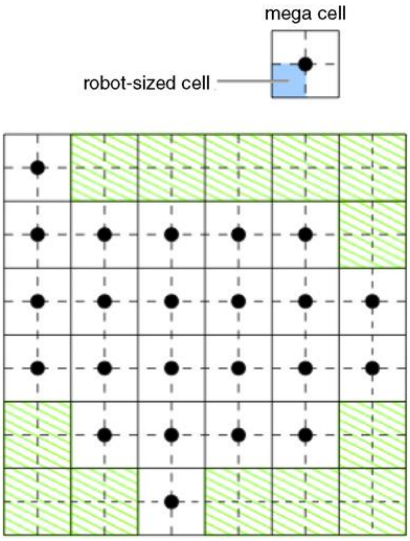


Figura 8. Descomposición aproximada de celdas en megaceldas y celdas del tamaño del robot, imagen tomada [41]

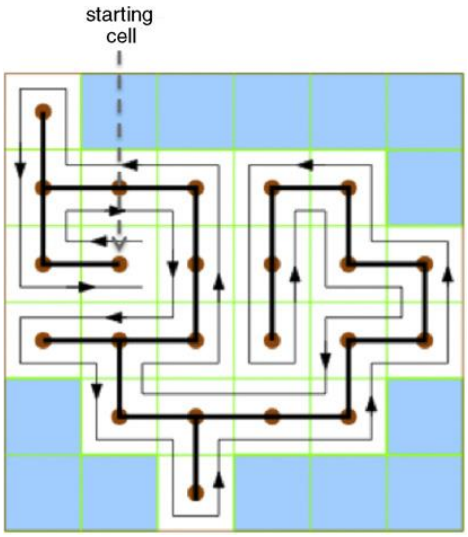


Figura 9. Trayectoria de cobertura con el algoritmo STC, imagen tomada de [41]

Como resultado de esta recursividad, el robot se mueve a lo largo de un lado del árbol de expansión hasta que llega al final del árbol. En ese punto, el robot se da la vuelta para atravesar el otro lado del árbol. Vale la pena notar que, cuando se completa la cobertura, el robot regresa a la celda de inicio, Figura 9, facilitando su acumulación y almacenamiento. Por otro lado, el método STC nunca visita más de una vez cualquiera de las celdas pequeñas, minimizando el tiempo de cobertura.

1.10. Métricas implementadas

Dentro de los objetivos trazados, se estableció el considerar métricas que ayuden a seleccionar la ruta que debe seguir el VANT. Las siguientes métricas fueron implementadas.

1.10.1. Longitud de la trayectoria cubierta

Es la longitud de la trayectoria total cubierta por el vehículo aéreo desde el punto inicial hasta el objetivo. Para una trayectoria en el plano x-*y*, compuesto de *n* puntos, y asumiendo el punto inicial como $(x_1, f(x_1))$ y el objetivo como $(x_2, f(x_2))$, se puede calcular como:

$$P_L = \sum_{i=1}^{n-1} \sqrt{(x_{i+1} - x_i)^2 + \left(f(x_{i+1}) - f(x_i)\right)^2} \quad (1)$$

En la ecuación (1), $(x_i, f(x_i))$ para $i = 1,2,3,\dots,n$ son los n puntos de la trayectoria en coordenadas cartesianas [47].

Para la ruta de cobertura implementada en este trabajo de grado se usaron coordenadas GPS. Por lo tanto, la longitud de la trayectoria se calculó usando la distancia entre coordenadas GPS de la ruta de cobertura calculada.

1.10.2. Tiempo de recorrido

Es el tiempo total para recorrer la longitud de la trayectoria a la velocidad establecida para el robot.

$$T_t = \frac{v}{P_L} \quad (2)$$

En la ecuación (2), v representa la velocidad del dron en su desplazamiento a cada uno de los puntos que conforma la ruta y P_L la longitud del recorrido entre cada uno de los puntos. En este trabajo, para calcular el valor teórico del tiempo de vuelo, la velocidad es constante al recorrer los puntos que componen la ruta calculada debido a que no se cuenta con datos reales que informen en que porcentaje se reduce la velocidad en cada uno de los giros.

1.10.3. Porcentaje de cobertura del área total

Mide la cantidad de puntos efectivamente recorridos, del total de puntos que contiene el área a recorrer.

$$C_{\%} = \frac{P_v}{P_T} \quad (3)$$

En la ecuación (3), P_v y P_T corresponden, respectivamente, a la cantidad de puntos totales visitados y cantidad total de puntos del área.

1.10.4. Redundancia de puntos recorridos

Corresponde a la cantidad de puntos que se visitan más de una vez del total de puntos que contiene la trayectoria.

$$R_{\%} = \frac{Lon_{Pt}}{P_r} \quad (4)$$

En la ecuación (4), Lon_{Pt} y P_r corresponden, respectivamente, a la cantidad total de puntos que contiene la trayectoria y los puntos visitados más de una vez.

1.10.5. Cantidad de giros de la ruta

Corresponde a la cantidad de cambios de dirección que posee la ruta de cobertura calculada. Esta métrica es adicional a las establecidas en los objetivos de este trabajo de grado y se puede establecer mediante la ecuación (5).

$$P_T = \sum_{i=1}^{n} P_i(\Delta\emptyset) \quad (5)$$

En la ecuación (5), $P_i(\Delta\emptyset)$ para $i = 1,2,3,\dots,n$ son los puntos que pertenecen a la ruta de cobertura donde la dirección de la trayectoria cambia.

2. Metodología

Se realizó una búsqueda de algoritmos de rutas de cobertura en artículos científicos que describieran los métodos usados en VANTS, como los descritos en [40] y [41], además de ejemplos de implementaciones como los descritos en [45] y [1]. Además, se consultó la documentación en línea de los distintos softwares. Mediante la información obtenida se procedió a establecer los distintos módulos y submódulos necesarios para simular los algoritmos mediante una interfaz gráfica.

Después se implementó una API en el lenguaje de programación Python que incluye cada uno de los submódulos necesarios; el módulo que los integra y, además, despliega la interfaz gráfica. Se seleccionó dicho lenguaje de programación por su facilidad de aprendizaje e implementación de dichos algoritmos, como también que es uno de los lenguajes soportados para usar en ROS. Para esto se adelantaron los siguientes pasos.

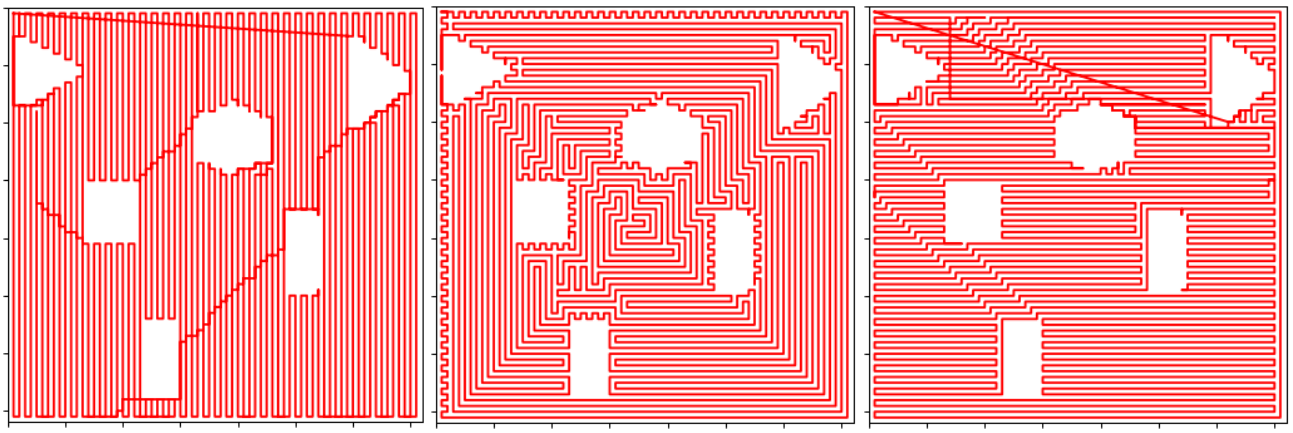
2.1. Nodos ROS

Se implementó una clase en Python llamada "nodo_ROS.py" que contiene los servicios, temas o tópicos y los nodos necesarios para que el VANT se pueda simular usando ROS, Gazebo y el software PX4. En esta clase se implementa todas las funciones necesarias para que el dron simulado pueda despegar, ejecutar automáticamente una ruta calculada, consultar datos de su estado y los datos necesarios para realizar en línea el cálculo del tiempo de simulación, longitud del recorrido, entre otros cálculos. Dicha función está en el repositorio GitHub del proyecto [48].

Además, se implementaron los nodos Maestros para la simulación. Estos nodos se pueden hallar en la carpeta "Master ROS" en el GitHub del proyecto [48]. Cada uno de los archivos allí contenidos pertenece a uno de los escenarios que se pueden simular.

2.2. Algoritmos de rutas de cobertura

Las rutas de cobertura implementadas en software pueden ser consultadas en la carpeta "CoveragePathPlanning/cpp_algorithms" que se encuentra en el repositorio GitHub del proyecto [48]. Dichas implementaciones usaron los algoritmos descritos en la sección 1.9. El resultado de aplicar los algoritmos a una región que contiene áreas que no se deben recorrer, se puede observar en la Figura 10.



(a) Algoritmo BCD *(b) Algoritmo Wavefront* *(c) Algoritmo STC*

Figura 10. Rutas calculadas por los distintos algoritmos

2.3. Interfaz gráfica

La interfaz gráfica, ver Figura 11, se implementó en QT. Una vez diseñada, se exporto a código Python. Dicho archivo se encuentra en el repositorio GitHub del proyecto [48] y se llama "coverage.py".

2.4. Cálculo de las rutas de cobertura

Para el cálculo de las rutas de cobertura del área predeterminada, se creó el archivo "readCoords.py" que puede ser hallado en el repositorio GitHub del proyecto [48].
En este archivo implementa la lectura del archivo que contiene las coordenadas GPS del área, el cálculo de la ruta de cobertura, la generación del archivo que contiene la ruta que seguirá el VANT y el cálculo de las métricas de dicha ruta.

2.5. Integración de módulos

Para realizar la simulación completa de las rutas de cobertura, se implementó la clase gui en el archivo "gui.py" que puede ser hallada en el repositorio GitHub del proyecto [48]. Este archivo contiene la integración de cada uno de los módulos implementados y al ser ejecutado, mostrará la interfaz gráfica de la Figura 11.

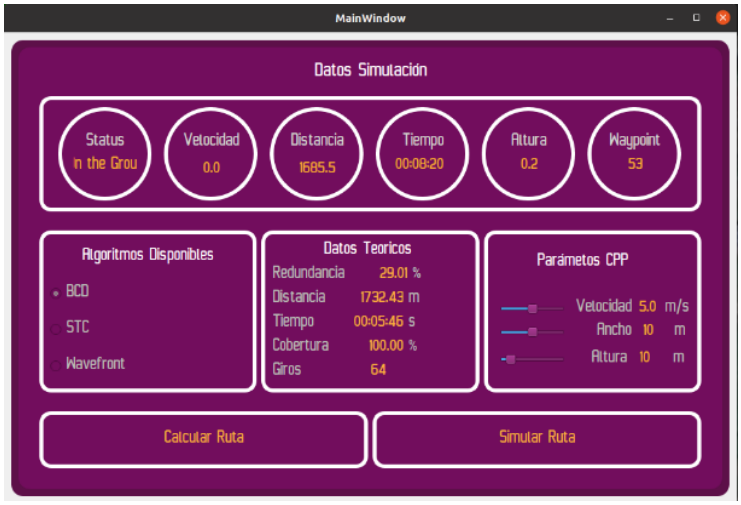


Figura 11. Interfaz gráfica que integra los distintos módulos

La interfaz de la Figura 11, permite:

- **Visualizar los datos de simulación:** Se configuraron *Status, Velocidad, Distancia, Tiempo, Altura y Waypoint*. *Status* permite saber el estatus del VANT en simulación (On the Ground, Taking Off, In the Air, Landing); *Velocidad* informa la velocidad del VANT a cada instante en la simulación; *Distancia* informa la distancia recorrida desde el momento en que el VANT está en el aire; *Tiempo* informa el tiempo transcurrido desde el despegue del VANT; finalmente *Waypoint* informa el último punto de la trayectoria que se visitó.

- **Seleccionar el algoritmo:** Esta selección se puede realizar en el recuadro "Algoritmos Disponibles". Posee las opciones BDC (véase 1.9.1), STC (véase 1.9.3) y Wavefront (véase 1.9.2).

- **Seleccionar los parámetros:** Estos parámetros se pueden seleccionar en el recuadro "Parámetros CPP". Los parámetros disponibles son *Velocidad, Ancho y Altura*. *Velocidad* establece la velocidad deseada en la ruta a calcular; *Ancho* establece la distancia entre las líneas de la trayectoria a calcular; *Altura* establece de la altura deseada para la ruta a calcular.

- **Visualizar las métricas de la ruta calculada:** En el recuadro "Datos Teóricos" se visualizan las métricas calculadas de la ruta de cobertura calculada. *Redundancia* informa el porcentaje de redundancia de la ruta calculada, (véase 1.10.4); *Distancia* informa la distancia de recorrido calculada de la ruta de cobertura, (véase 1.10.1); *Tiempo* informa el tiempo aproximado para recorrer la ruta calculada, (véase 1.10.2); *Cobertura* informa el porcentaje de cobertura de la ruta calculada para el área dada, (véase 1.10.3); y finalmente *Giros* establece la cantidad de giros que posee la ruta calculada, (véase 1.10.5).

- **Botones para calcular y simular la ruta:** Se tienen dos botones. El botón "Simular Ruta" permite inicializar la simulación de una ruta previamente calculada. El botón "Calcular Ruta" permite calcular una ruta, previa selección de uno de los algoritmos (BDC, STC, Wavefront) y los parámetros (Velocidad, Ancho, Altura).

2.6. Diagrama de integración

Se integró ROS con PX4 y el simulador Gazebo, usando un nodo de la librería MAVROS (ver **Anexo *B***, paso 7) llamado MAVLink, ver Figura 12.

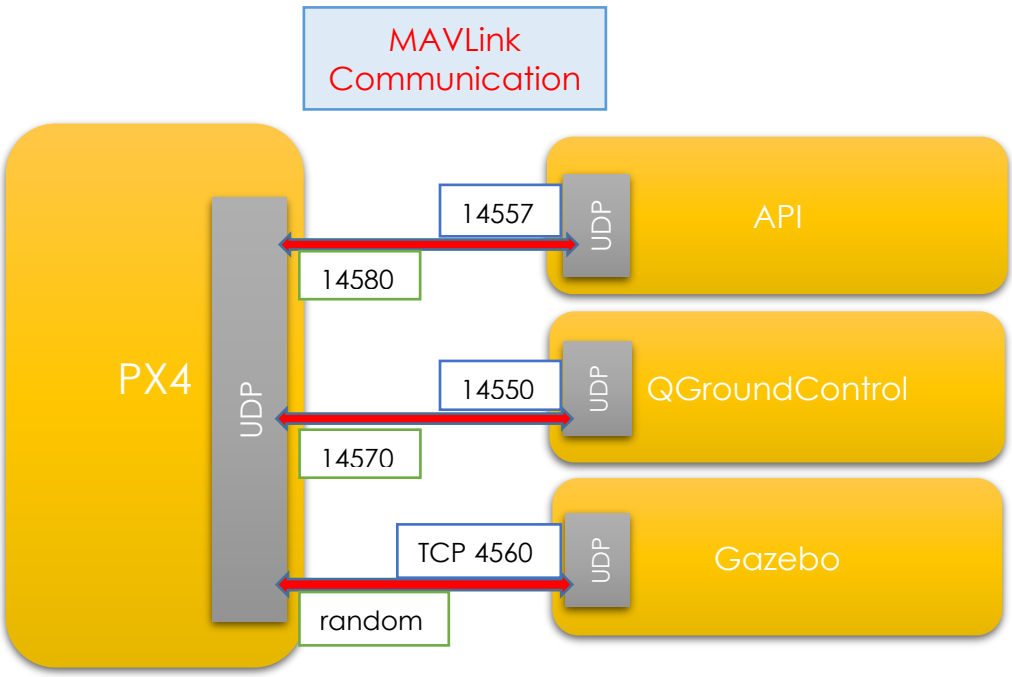


Figura 12. Integración PX4, ROS, QGroundControl y Gazebo

La integración de ROS y Gazebo con PX4 sigue el patrón mostrado en la Figura 12 donde:

- PX4 se comunica con el simulador Gazebo para recibir los datos de los sensores desde el escenario simulado y para enviar los valores de motores y actuadores.

- PX4 se comunica con QGroundControl y con la API (Maestro ROS, interfaz gráfica) para enviar la telemetría desde el escenario simulado, además para recibir comandos.

- Toda la comunicación se realiza usando puertos UDP.

- La API diseñada es la encargada de enviar los comandos necesarios para la simulación; además de recibir los datos de la telemetría que se usan para mostrar y calcular los datos de simulación mostrados en la Figura 11.

- QGroundControl recibe la telemetría, la despliega en pantalla y permite visualizar la ruta que sigue el VANT en simulación, ver Figura 13. Los puntos amarillos indican cada uno de los puntos que posee la ruta de cobertura calculada; las líneas rojas indican la trayectoria seguida por el VANT; el recuadro inferior muestra los valores de la telemetría.

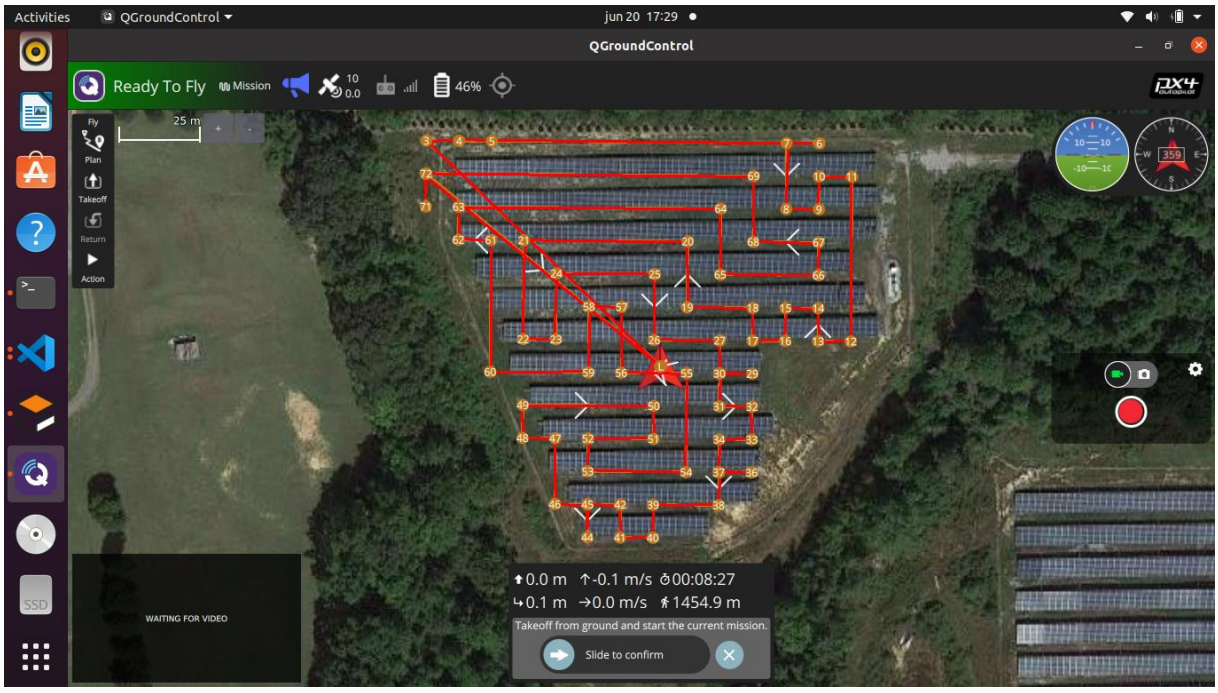


Figura 13. Aspecto simulación de ruta en QGroundControl

2.7. Aspectos importantes en la implementación

En el proceso de implementación en Python se tuvieron en cuenta los siguientes aspectos:

- De cada sistema fotovoltaico, al que se le genera o calcula una ruta de cobertura, se debe tener una foto en blanco y negro. La parte blanca corresponde al área a recorrer, sistema fotovoltaico; el área negra representa el resto del área que no se recorrerá.

- La foto del sistema fotovoltaico a recorrer se debe ubicar en la carpeta "CoveragePathPlanning/fotos". Esta se debe poseer extensión "png" y debe llamarse "mascara.png". El archivo se puede consultar en el repositorio GitHub del proyecto [48].

- La foto que toma el algoritmo no debe poseer pixeles blancos aislados. En caso de que los posea, no se podrá generar el archivo con las rutas de cobertura, y se generará un error al tratar de calcular las rutas de cobertura.

- La foto debe ser de muy buena calidad. Esta sufre cambios en su tamaño cuando se calculan las rutas de cobertura; estos cambios los determina el ancho entre líneas deseado para la ruta.

- La carpeta "CoveragePathPlanning/test_maps" no debe poseer archivos inicialmente. Allí solo se almacenará la foto que sufrió cambios en su tamaño y se le asignará en cada simulación el mismo nombre.

- De cada foto se deben tener las coordenadas GPS que corresponden a sus esquinas. Estas deben estar en un archivo llamado "coords.csv" que se debe leer para el cálculo de la ruta de cobertura del área en cuestión. Dicho archivo se encuentra en el repositorio GitHub del proyecto [48].

- El cálculo de los tiempos de simulación inicia cuando se detecta que el dron despega y termina una vez detectado el aterrizaje.

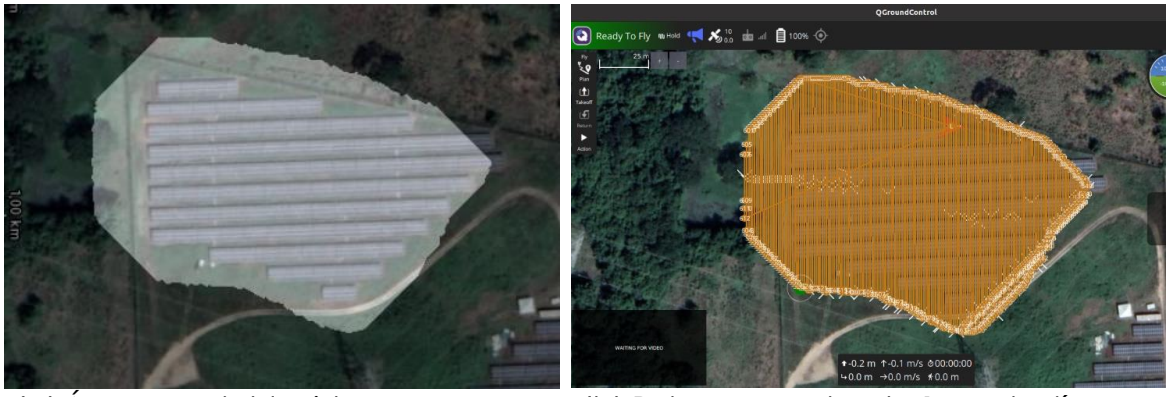
3. Resultados de simulación

Las simulaciones realizadas evaluaron cada uno de los algoritmos en dos plantas solares: Celsia Bolívar y South Winston. El proceso de simulación realizado se puede consultar en el **Anexo F.**

3.1. Celsia Bolívar

Celsia bolívar es una planta solar ubicada en el departamento de Bolívar, Colombia, propiedad de la empresa Celsia. Para dicha planta se preestableció el área de la Figura 14(a).

Para evaluar que el área delimitada para la planta Celsia Bolívar, Figura 14(a), se estableció una distancia entre líneas de un metro para calcular una ruta de cobertura, Figura 14(b). El resultado obtenido permitió establecer que las áreas son las mismas.



(a) Área preestablecida *(b) Ruta con ancho de 1m entre líneas*

Figura 14. Área a recorrer planta Celsia Bolívar

Los resultados obtenidos en simulación con los distintos algoritmos están consolidados en la Tabla 1.

Planta	Algoritmo	Distancia (m)				Tiempo			Redundancia %	Cobertura %	Cantidad de Giros	Bateria	
		Entre líneas	Teórica	Simulación		Teórico	Simulación					Autonomía (s)	Porcentaje Restante %
				Qground Control	Interfaz		Qground Control	Interfaz					
Celsia Bolivar	BCD	10	1732.43	1672.0	1685.5	0:05:46	0:08:23	0:08:20	29.01	100	64	900	46
Celsia Bolivar	STC	10	1521.33	1410.8	1443.0	0:05:04	0:08:36	0:08:36	6.11	100	86	900	43
Celsia Bolivar	Wavefront	10	1546.06	1430.5	1459.8	0:05:09	0:08:52	0:08:51	6.11	100	94	900	43

Tabla 1. Resultados simulación Celsia Bolívar

En [40] se describen las métricas de desempeño de cada uno de los algoritmos implementados. Las métricas comunes para estos algoritmos son: longitud de la trayectoria, tiempo de recorrido

Por lo tanto, se pudo establecer con estas métricas que la mejor ruta se obtuvo con el algoritmo STC. De la Tabla 1, los valores teóricos y simulados de la longitud de la trayectoria corresponden a dicho algoritmo. En cuanto a los tiempos de recorrido, el valor teórico estableció que dicho algoritmo también poseía el mejor tiempo, mientras que la simulación demostró que es el segundo mejor tiempo.

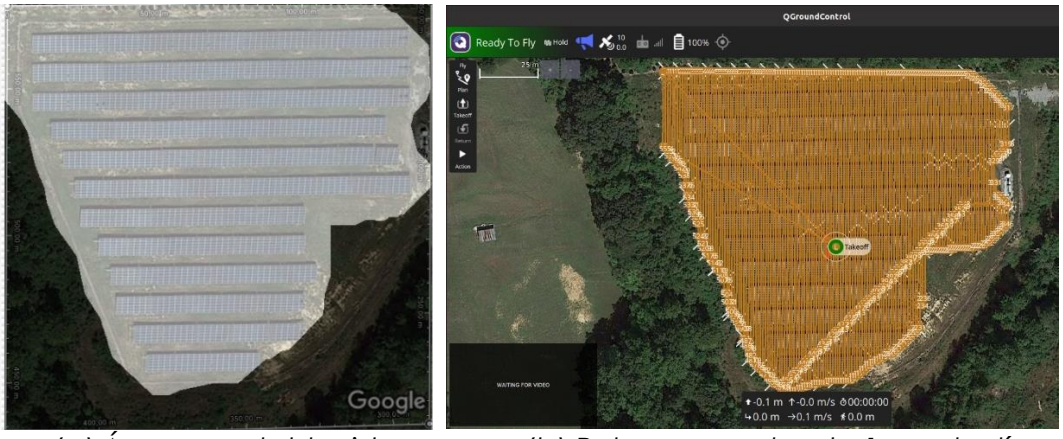
Tomando en cuenta los demás datos de la Tabla 1, el algoritmo STC comparte con el algoritmo Wavefront el menor porcentaje de redundancia, aunque la ruta posee menos cantidad de giros que la ruta calculada con el algoritmo Wavefront. Estas métricas también indican que la mejor ruta es la que genera el algoritmo STC.

También en [40] se establece que los algoritmos que toman en cuenta la energía, su principal métrica es el consumo energético. Tomando el ítem de batería de la Tabla 1, el algoritmo STC comparte el puesto con el algoritmo Wavefront en el consumo energético. Ambas rutas consumieron el 57% de la batería, un 3% más que su contraparte BCD.

3.2. South Winston

South Winston es una planta solar está ubicada en los Estados Unidos. Sus coordenadas GPS son 36.02961731613414, -80.30389653877222.

Para la planta South Winston se preestableció el área que se puede apreciar en la Figura 15(a). Su correspondiente ruta con ancho entre líneas de 1m se puede apreciar en la Figura 15(b) y permite establecer que ambas áreas son iguales.



(a) Área preestablecida *(b) Ruta con ancho de 1m entre líneas*

Figura 15. Área a recorrer planta South Winston

Para la planta South Winston se realizaron simulaciones con anchos entre líneas de 5 y 10 metros. Se usó un dron con autonomía de 1500 segundos y se simularon las rutas calculadas con cada uno de los algoritmos disponibles. La Tabla 2 consolida los resultados obtenidos con un ancho entre líneas de 10 metros y la Tabla 3 los resultados con un ancho entre líneas de 5 metros.

Planta	Algoritmo	Distancia (m)				Tiempo			Redundancia %	Cobertura %	Cantidad de Giros	Bateria	
		Entre líneas	Teórica	Simulación		Teórico	Simulación					Autonomía (s)	Porcentaje Restante %
				Qground Control	Interfaz		Qground Control	Interfaz					
South Winston	BCD	10	1851.75	1787.8	1800.8	0:06:10	0:08:47	0:08:46	35.61	100	65	1500	41
	STC	10	1541.49	1454.9	1475.6	0:05:08	0:08:27	0:08:26	4.58	99.24	71	1500	46
	Wavefront	10	1508.38	1438.8	1455.3	0:05:01	0:07:32	0:07:31	2.27	100	56	1500	51

Tabla 2. Resultados simulación South Winston ancho entre líneas 10m

De la Tabla 2 se puede establecer que el algoritmo de mejor desempeño es Wavefront. La longitud teórica y simulada de la ruta es la menor; el tiempo de recorrido teórico y el simulado también es el menor. Las métricas desempeño descritas en [40] para cada uno de los algoritmos implementados, establece que las métricas comunes para estos algoritmos son: longitud de la trayectoria, tiempo de recorrido, confirmando la afirmación previa.

Tomando en cuenta los demás datos de la Tabla 2, vemos que el algoritmo Wavefront también posee los valores más bajos de las métricas redundancia, cantidad de giros, y cobertura. En cuanto al consumo de batería, la ruta establecida con el algoritmo Wavefront es la que menor cantidad de batería consumió, 10% menos que el algoritmo BCD, y 5% menos que el algoritmo BCD.

Planta	Algoritmo	Distancia (m)				Tiempo			Redundancia %	Cobertura %	Cantidad de Giros	Bateria	
		Entre líneas	Teórica	Simulación		Teórico	Simulación					Autonomía (s)	Porcentaje Restante %
				Qground Control	Interfaz		Qground Control	Interfaz					
South Winston	BCD	5	3171.06	3055.1	3077.2	0:10:34	0:14:45	0:14:43	19.08	100	127	1500	42
	STC	5	2839.71	2609.7	2670.3	0:09:27	0:15:54	0:15:54	2.51	99.81	177	1500	40
	Wavefront	5	2813.99	2551.2	2612.7	0:09:22	0:16:09	0:16:02	1.54	100	234	1500	38

Tabla 3. Resultados simulación South Winston ancho entre líneas 5m

Los datos en la Tabla 3 permitieron evaluar las rutas simuladas con las métricas comunes (longitud y tiempo de vuelo del recorrido) establecidas en [40]. No fue posible determinar la mejor ruta. Se obtuvo que la ruta con menor longitud tanto teórica como simulada es la que usa el algoritmo Wavefront. En cuanto a los tiempos de recorrido, el teórico establece que la ruta con el algoritmo Wavefront es la menor, mientras que en simulación se halló que es la ruta con mayor tiempo de recorrido. El menor tiempo de recorrido se obtuvo con el algoritmo BCD.

Al incluir las métricas adicionales de la Tabla 3, la mejor ruta es la generada con el algoritmo STC. Esta ruta se ubica en segundo lugar en todas las métricas. La longitud de recorrido; el tiempo de vuelo teórico y simulado; redundancia, cobertura, cantidad de giros y porcentaje de batería restante también se ubican en segundo lugar. Además, la diferencia en la longitud

con su contraparte Wavefront es de sólo 25.72 metros en el cálculo teórico; mientras que en la simulación es de 58.5 metros en QGroundControl, y de 57.6 metros en la interfaz.

4. Análisis de resultados

La Tabla 4 resume los datos hallados para cada una de las plantas y sus respectivas simulaciones.

Planta	Algoritmo	Distancia (m)				Tiempo			Redundancia %	Cobertura %	Cantidad de Giros	Bateria	
		Entre líneas	Teórica	Simulación		Teórico	Simulación					Autonomía (s)	Porcentaje Restante %
				Qground Control	Interfaz		Qground Control	Interfaz					
Celsia Bolivar	BCD	10	1732.43	1672.0	1685.5	0:05:46	0:08:23	0:08:20	29.01	100	64	900	46
	STC	10	1521.33	1410.8	1443.0	0:05:04	0:08:36	0:08:36	6.11	100	86	900	43
	Wavefront	10	1546.06	1430.5	1459.8	0:05:09	0:08:52	0:08:51	6.11	100	94	900	43
South Winston	BCD	5	3171.06	3055.1	3077.2	0:10:34	0:14:45	0:14:43	19.08	100	127	1500	42
	STC	5	2839.71	2609.7	2670.3	0:09:27	0:15:54	0:15:54	2.51	99.81	177	1500	40
	Wavefront	5	2813.99	2551.2	2612.7	0:09:22	0:16:09	0:16:02	1.54	100	234	1500	38
	BCD	10	1851.75	1787.8	1800.8	0:06:10	0:08:47	0:08:46	35.61	100	65	1500	41
	STC	10	1541.49	1454.9	1475.6	0:05:08	0:08:27	0:08:26	4.58	99.24	71	1500	46
	Wavefront	10	1508.38	1438.8	1455.3	0:05:01	0:07:32	0:07:31	2.27	100	56	1500	51

Tabla 4. Consolidado de resultados simulaciones

De la Tabla 4 se puede deducir lo siguiente:

- Los algoritmos que hallan la ruta con menor recorrido en los cálculos teóricos son los mismos que obtienen el menor recorrido en simulación. Para el caso de Celsia, el algoritmo STC; para South Winston, el algoritmo Wavefront.

- El menor tiempo teórico calculado en cada uno de los algoritmos para las distintas plantas, no siempre corresponde al menor tiempo de simulación.

- Mayor cantidad de giros en una ruta no siempre significa menor consumo energético. Es el caso del algoritmo Wavefront en Celsia Bolívar y el algoritmo STC en South Winston ambos con ancho entre líneas de 10m.

- Los algoritmos BCD y Wavefront obtienen las mejores rutas tanto teóricas como simuladas.

- Al duplicar el ancho entre líneas de las rutas, los valores obtenidos en longitud de la trayectoria y el tiempo de vuelo prácticamente se duplicaron, apuntando a una posible relación lineal entre este parámetro y las métricas obtenidas.

- Con la interfaz implementada se logra determinar de antemano las mejores rutas para un área predeterminada a cubrirse con un VANT; estas pueden ser corroboradas mediante la simulación.

- Las métricas adicionales permiten elegir la mejor ruta calculada y simulada cuando los valores teóricos y simulados de longitud y tiempo del recorrido no permiten que se establezca.

5. Conclusiones

- La solución implementada permite seleccionar, simular y evaluar una ruta en base a múltiples parámetros de entrada, ahorrando tiempo en la selección del recorrido a realizar por parte de un VANT.

- Aunque el método desarrollado toma como entrada una foto denominada máscara, dicha máscara debe estar libre de pixeles aislados.

- Los resultados se pueden ver afectados por la generación de la máscara, el cambio del tamaño de dicha máscara en el procesamiento, la cantidad de giros y la distancia entre líneas que posee la ruta calculada, como también los métodos usados para el cálculo de los distintos valores de nuestra solución y el usado por el software QGroundControl.

- Las métricas calculadas en este trabajo son las más usadas en la literatura para evaluar la calidad de una ruta de cobertura dado un área o ambiente conocido de antemano.

- Aunque no era objetivo de este trabajo de grado, se incluyó una métrica para la cantidad de giros de las distintas rutas calculas. Los resultados obtenidos en simulación confirmaron que, en general, a mayor cantidad de giros, no siempre se genera mayor consumo energético por parte del VANT al cubrir un área determinada.

- Usar métricas adicionales permite la selección de una ruta alternativa cuando las métricas principales no permiten establecer la mejor ruta.

6. Referencias Bibliográficas

[1] J. Jin and L. Tang, "Optimal Coverage Path Planning for Arable Farming on 2D Surfaces," *Transactions of the ASABE*, vol. 53, no. 1, pp. 283–295, 2010.

[2] J. S. Oh, Y. H. Choi, J. B. Park, and Y. F. Zheng, "Complete coverage navigation of cleaning robots using triangular-cell-based map," *IEEE Transactions on Industrial Electronics*, vol. 51, no. 3, pp. 718–726, Jun. 2004, doi: 10.1109/TIE.2004.825197.

[3] K. Shah, G. Ballard, A. Schmidt, and M. Schwager, "Multidrone aerial surveys of penguin colonies in Antarctica," 2020. [Online]. Available: http://robotics.sciencemag.org/
[4] M. Hassan, D. Liu, S. Huang, and G. Dissanayake, "Task oriented area partitioning and allocation for optimal operation of multiple industrial robots in unstructured environments," in *2014 13th International Conference on Control Automation Robotics and Vision, ICARCV 2014*, 2014, pp. 1184–1189. doi: 10.1109/ICARCV.2014.7064473.
[5] Python.org, "About Python," Jun. 29, 2021. https://www.python.org/ (accessed Jul. 01, 2021).
[6] Redhat, "¿QUÉ ES UNA API?," 2021. https://www.redhat.com/es/topics/api/what-are-application-programming-interfaces (accessed Jul. 03, 2021).
[7] QT Group, "About US," 2021. https://www.qt.io/company (accessed Jul. 03, 2021).
[8] Canonical, "Ubuntu," Jun. 29, 2021. https://canonical.com/ (accessed Jul. 01, 2021).
[9] ROS.org, "What is ROS?," 2021. https://www.ros.org/about-ros/ (accessed Jul. 01, 2021).
[10] ROS.org, "Nodes," 2021. http://wiki.ros.org/Nodes (accessed Jul. 01, 2021).
[11] ROS.org, "Topics," 2021. http://wiki.ros.org/Topics (accessed Jul. 01, 2021).
[12] ROS.org, "Services," Jul. 01, 2021. http://wiki.ros.org/Services (accessed Jul. 01, 2021).
[13] ROS.org, "ROS Master," Jun. 29, 2021. http://wiki.ros.org/Master (accessed Jul. 01, 2021).
[14] ROS.org, "Client Libraries," 2021. http://wiki.ros.org/Client%20Libraries (accessed Jul. 01, 2021).
[15] ROS.org, "roslaunch," 2021. http://wiki.ros.org/roslaunch (accessed Jul. 01, 2021).
[16] ROS.org, "roscore," 2021. http://wiki.ros.org/roscore (accessed Jul. 01, 2021).
[17] F. Greenwood Id, E. L. Nelson Id, and P. G. Greenough, "Flying into the hurricane: A case study of UAV use in damage assessment during the 2017 hurricanes in Texas and Florida," 2020, doi: 10.1371/journal.pone.0227808.
[18] N. Basilico and S. Carpin, "Deploying teams of heterogeneous UAVs in cooperative two-level surveillance missions," in *IEEE International Conference on Intelligent Robots and Systems*, Dec. 2015, vol. 2015-December, pp. 610–615. doi: 10.1109/IROS.2015.7353435.
[19] P. Lottes, R. Khanna, J. Pfeifer, R. Siegwart, and C. Stachniss, "UAV-based crop and weed classification for smart farming," in *Proceedings - IEEE International Conference on Robotics and Automation*, Jul. 2017, pp. 3024–3031. doi: 10.1109/ICRA.2017.7989347.
[20] A. Cesetti *et al.*, "A Visual Global Positioning System for Unmanned Aerial Vehicles Used in Photogrammetric Applications," *J Intell Robot Syst*, vol. 61, pp. 157–168, 2011, doi: 10.1007/s10846-010-9489-5.
[21] I. Maza *et al.*, "Experimental Results in Multi-UAV Coordination for Disaster Management and Civil Security Applications," *J Intell Robot Syst*, vol. 61, pp. 563–585, 2011, doi: 10.1007/s10846-010-9497-5.
[22] W. Chang, J. G. Yang, Z. L. Yu, L. Cheng, and C. Zhou, "Development of A Power Line Inspection Robot with Hybrid Operation modes.," *Proceedings of the 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 973–978, Sep. 2017.
[23] L. Žlajpah, "Simulation in robotics," *Mathematics and Computers in Simulation*, vol. 79, no. 4, pp. 879–897, Dec. 2008, doi: 10.1016/j.matcom.2008.02.017.
[24] Cybertronics, "Webots User Guide," 2021. https://cyberbotics.com/doc/guide/foreword (accessed Jul. 01, 2021).

[25] Gazebo.org, "What is Gazebo?," 2021. http://gazebosim.org/tutorials?tut=guided_b1&cat=#WhatisGazebo? (accessed Jul. 01, 2021).
[26] PX4 Ardupilot, "PX4," Jun. 22, 2021. https://docs.px4.io/master/en/ (accessed Jun. 30, 2021).
[27] Microsoft, "Simulating The World With Microsoft Robotics Studio," 2021. https://docs.microsoft.com/en-us/archive/msdn-magazine/2008/june/robotics-simulating-the-world-with-microsoft-robotics-studio (accessed Jul. 01, 2021).
[28] Dronecoede, "QGroundControl Download and Install," Jun. 29, 2021. https://docs.qgroundcontrol.com/master/en/getting_started/download_and_install.html (accessed Jul. 01, 2021).
[29] ROS.org, "Ubuntu install of ROS Noetic," Jun. 21, 2021. http://wiki.ros.org/noetic/Installation/Ubuntu (accessed Jul. 01, 2021).
[30] Guia de Usuario PX4., "Guia de Usuario PX4.," Jun. 29, 2021. https://docs.px4.io/master/en/dev_setup/dev_env_linux_ubuntu.html (accessed Jun. 30, 2021).
[31] DroneCode, "QGROUNDCONTROL," 2021. http://qgroundcontrol.com/ (accessed Jul. 01, 2021).
[32] PX4.io, "What Is PX4?," 2021. https://px4.io/software/software-overview/ (accessed Jul. 01, 2021).
[33] PX4.io, "Ubuntu Development Environment," 2021. https://docs.px4.io/master/en/dev_setup/dev_env_linux_ubuntu.html (accessed Jul. 01, 2021).
[34] H. Choset, "Coverage for robotics-A survey of recent results," Kluwer Academic Publishers, 2001.
[35] S. Brown, "Coverage Path Planning and Room Segmentation in Indoor Environments using the Constriction Decomposition Method," 2017.
[36] E. Galceran and M. Carreras, "A survey on coverage path planning for robotics," *Robotics and Autonomous Systems*, vol. 61, pp. 1258–1276, 2013, doi: 10.1016/j.robot.2013.09.004.
[37] J. C. Latombe, *Robot Motion Planning*. 2012.
[38] H. Moravec and A. Elfes, "High Resolution Maps from Wide Angle Sonar," 1985.
[39] J. A. Castellanos, J. D. Tardos, and G. Schmidt, "Building a global map of the environment of a mobile robot: the importance of correlations," in *Proceedings of International Conference on Robotics and Automation*, 1997, pp. 1053–1059. doi: 10.1109/ROBOT.1997.614274.
[40] T. M. Cabreira, L. B. Brisolara, and P. R. Ferreira, "Survey on Coverage Path Planning with Unmanned Aerial Vehicles," 2019, doi: 10.3390/drones3010004.
[41] E. Galceran and M. Carreras, "A survey on coverage path planning for robotics," *Robotics and Autonomous Systems*, vol. 61, no. 12, pp. 1258–1276, Dec. 2013, doi: 10.1016/j.robot.2013.09.004.
[42] T. M. . . Cabreira, P. R. Ferreira Jr, C. di Franco, and G. C. Buttazzo, "Grid-based Coverage Path Planning with Minimum Energy over Irregular-shaped Areas with UAVs," Jun. 2019.
[43] T. M. Cabreira, C. di Franco, P. R. Ferreira Jr, and G. C. Buttazzo, "Energy-Aware Spiral Coverage Path Planning for UAV Photogrammetric Applications," *IEEE ROBOTICS AND AUTOMATION LETTERS*, vol. 3, no. 4, Oct. 2018.

[44] H. Choset and P. Pignon, "Coverage Path Planning: The Boustrophedon Cellular Decomposition," *Proceedings of International Conference on Field and Service Robotics*, 1997.
[45] Wen-Chieh Tung and Jing-Sin Liu, "SOLUTION OF AN INTEGRATED TRAVELING SALESMAN AND COVERAGE PATH PLANNING PROBLEM BY USING A GENETIC ALGORITHM WITH MODIFIED OPERATORS," *IADIS International Journal on Computer Science & Information Systems*, vol. 14, no. 2, pp. 95–114, 2019.
[46] A. Zelinsky, R. A. Jarvis, J. C. Byrne, and S. Yuta, "Planning Paths of Complete Coverage of an Unstructured Environment by a Mobile Robot," in *Proceedings of International Conference on Advanced Robotics*, 1993, pp. 533–538.
[47] N. Munoz Ceballos, J. Valencia, and A. Alvarez Giraldo, "Simulation and Assessment Educational Framework for Mobile Robot Algorithms," *Journal of Automation, Mobile Robotics & Intelligent Systems*, vol. 8, no. 1, pp. 53–59, 2014.
[48] Nelson Benítez, "Github Proyecto." Medellín, Jun. 10, 2021. Accessed: Jul. 01, 2021. [Online]. Available: https://github.com/NelsonMontoya/GUI
[49] Gazebo.org, "Construcción de escenario en Gazebo," Jun. 29, 2021. http://gazebosim.org/tutorials?tut=build_world (accessed Jul. 01, 2021).
[50] Intelligentquads, "Intelligent Quads," Jun. 29, 2021. http://intelligentquads.com/ (accessed Jul. 01, 2021).

7. Anexos

Anexo A. Instalación individual de ROS.

A la fecha de escritura de este informe, las instrucciones para instalar ROS en Ubuntu 20.04 LTS a través de la consola se consultaron en [29] y son:

1. **Configure su computadora para aceptar software de packages.ros.org**

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
```

2. **Configurar las llaves.**

```
sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-key C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
```

3. **Primero, asegúrese de que el índice de su paquete Ubuntu esté actualizado**

```
sudo apt update
```

4. **Elegir la instalación completa**

```
sudo apt install ros-noetic-desktop-full
```

5. **Configuración del entorno**

```
source /opt/ros/noetic/setup.bash
echo "source /opt/ros/noetic/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

6. Dependencias para la construcción de paquetes

```
sudo apt install python3-rosdep python3-rosinstall python3-rosinstall-generator python3-wstool
build-essential
```

7. Inicializar rosdep

```
sudo apt install python3-rosdep
sudo rosdep init
rosdep update
```

Anexo B. Proceso de verificación e instalación Gazebo en conjunto con ROS

Para la instalación se debe usar el archivo "ubuntu_noetic_ros.sh" que contiene todas las instrucciones completas para instalar ROS, Gazebo y todas las dependencias necesarias para su funcionamiento. Este archivo se puede hallar en el GitHub del proyecto [48] en la carpeta "Instalación ROS".

Pasos instalación con archivo "ubuntu_noetic_ros.sh"

1. Verificar la carpeta donde se descargó el archivo como se observa en la Figura 16.

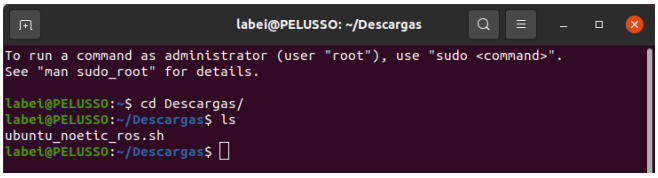


Figura 16. Ubicación archivo "ubuntu_noetic_ros.sh"

2. Iniciar el proceso de instalación como se observa en la Figura 17.

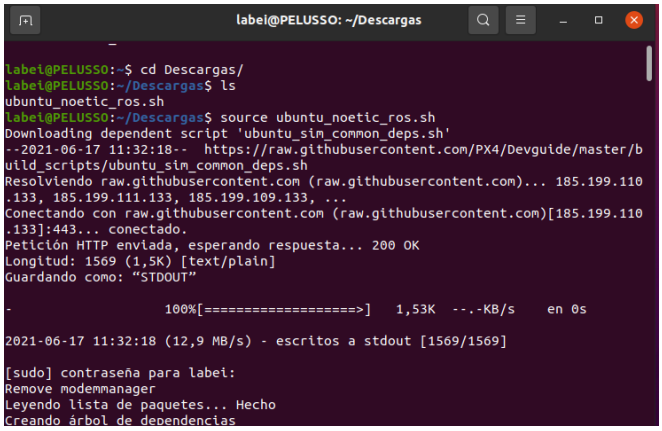


Figura 17. Ejecución del proceso de instalación Gazebo en conjunto con ROS

3. Aceptar continuar con el proceso cada que se le solicite, proceso que se observa en la Figura 18.

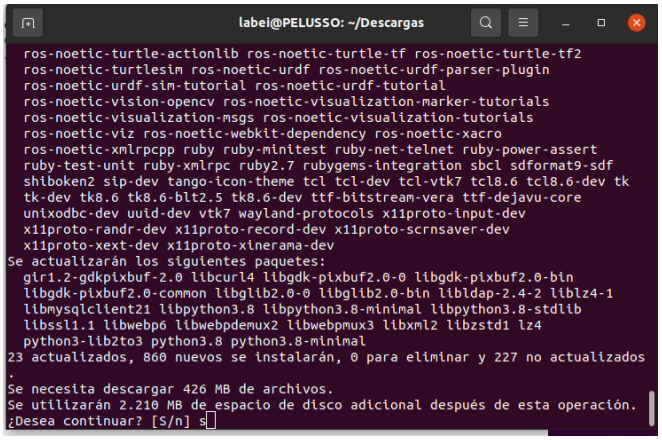


Figura 18. Continuación del proceso de instalación

.

4. Al terminar el proceso, verificar que las dependencias se hayan instalado, ver Figura 19.

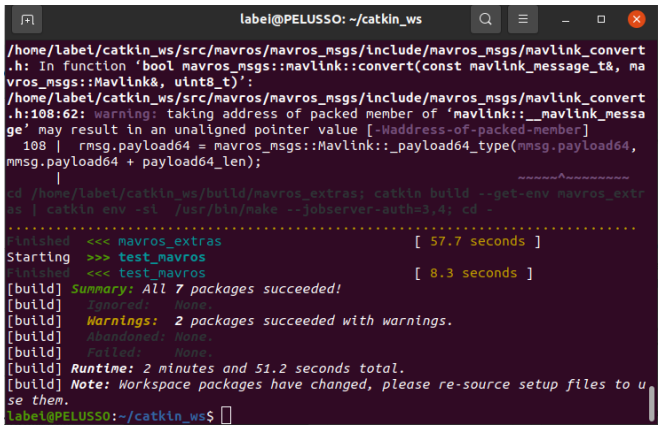


Figura 19. Proceso de instalación de ROS y Gazebo terminado

5. Verificar que Gazebo se encuentra instalado, tal como se ve en la Figura 20.

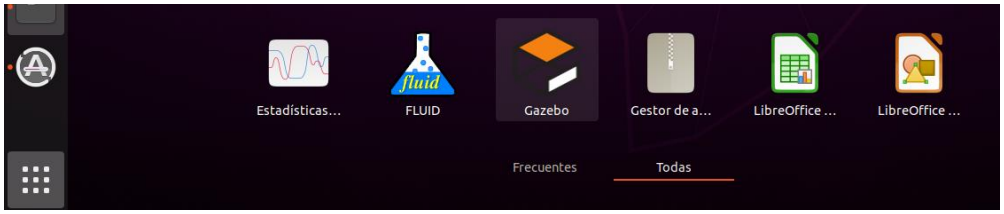


Figura 20. Verificación de la instalación de Gazebo

6. Una vez instalado ROS, Gazebo y todas las dependencias necesarias, se debe tener una carpeta en el espacio de usuario con el nombre de "catkin_ws", tal como se ve en la Figura 21.

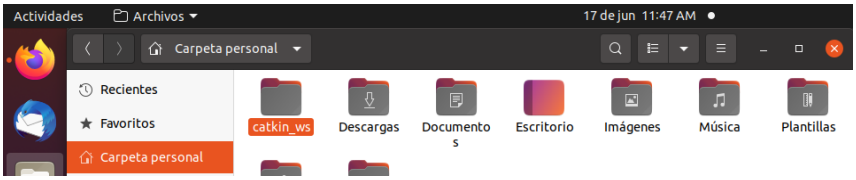


Figura 21. Verificación de la creación del ambiente de trabajo

7. Validar en la carpeta "catkin_ws" que las dependencias "mavlink" y "mavros" se hayan instalado, tal como se ve en la Figura 22.

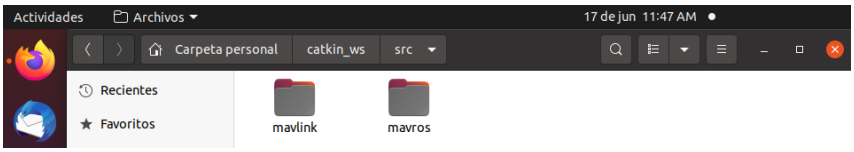


Figura 22. Verificación de la instalación de las dependencias necesarias.

Anexo C. Proceso de Instalación y verificación de PX4

La instalación se realiza de acuerdo con los pasos descritos en la página web de PX4 [33].

1. **Descargar el código fuente:** En la terminal, se escribe lo siguiente comando, tal como se ve en la Figura 23:

```
git clone https://github.com/PX4/PX4-Autopilot.git --recursive
```

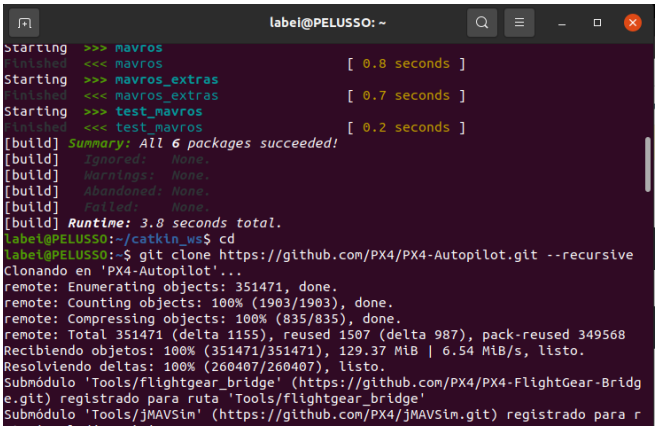


Figura 23. Descarga del código fuente PX4

2. **Ejecutar el archivo ubuntu.sh**

   En la consola se escribe el siguiente comando para instalar PX4 y sus dependencias, tal como se ve en la Figura 24.

```
bash ./PX4-Autopilot/Tools/setup/ubuntu.sh
```

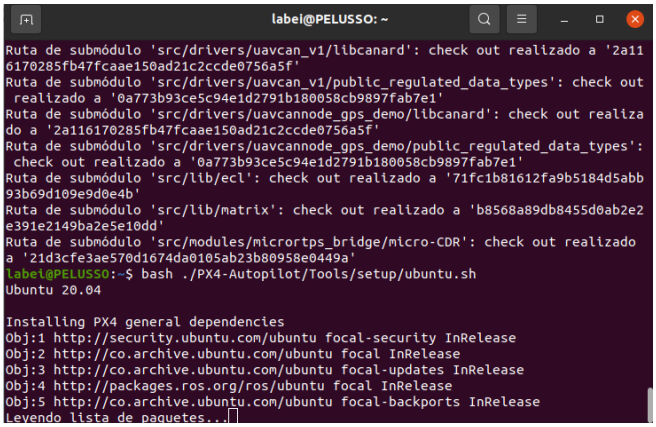


Figura 24. Ejecución del archivo "ubuntu.sh"

3. Reiniciar

Una vez se ha instalado todos los paquetes del PX4, se debe reiniciar el computador, como se observa en la Figura 25.

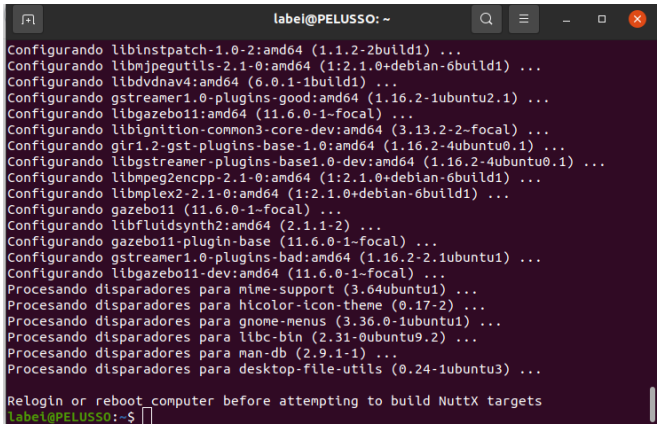


Figura 25. Proceso de instalación PX4 terminado

4. Validar carpeta PX4

Una vez reiniciado el sistema operativo, se debe ubicar la carpeta PX4. Debe estar en el espacio del usuario con el nombre PX4-Autopilot, tal como se ve en la Figura 26.

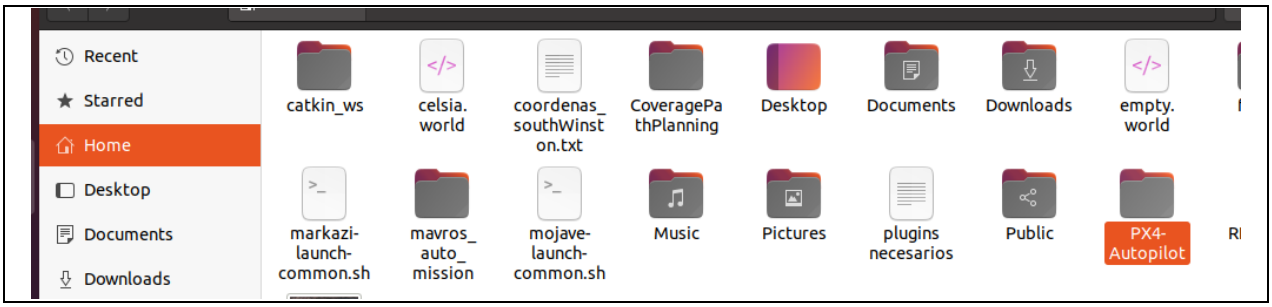


Figura 26. Ubicación de la carpeta PX4-Autopilot

5. Simular VANT o dron

Para instalar esta herramienta, se debe asegurar que ya se haya instalado Gazebo y ROS. De ser así, debe escribir en consola lo siguiente, tal como se ve en la Figura 27.

```
cd PX4-Autopilot/
make px4_sitl gazebo
```

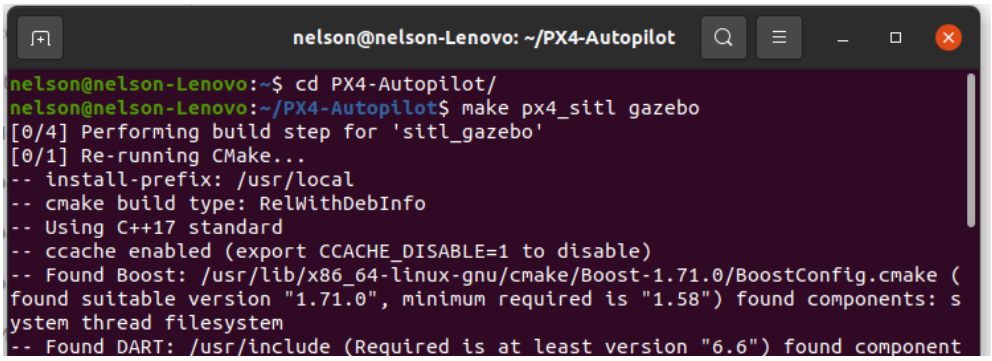


Figura 27. Proceso de compilación y creación de un dron usando PX4

6. Verificar simulación

Al ejecutar los comandos anteriores, Figura 27, se construye un dron por defecto con todos sus parámetros para que se pueda simular en Gazebo. También debe desplegarse automáticamente Gazebo, como se puede apreciar en la Figura 28.

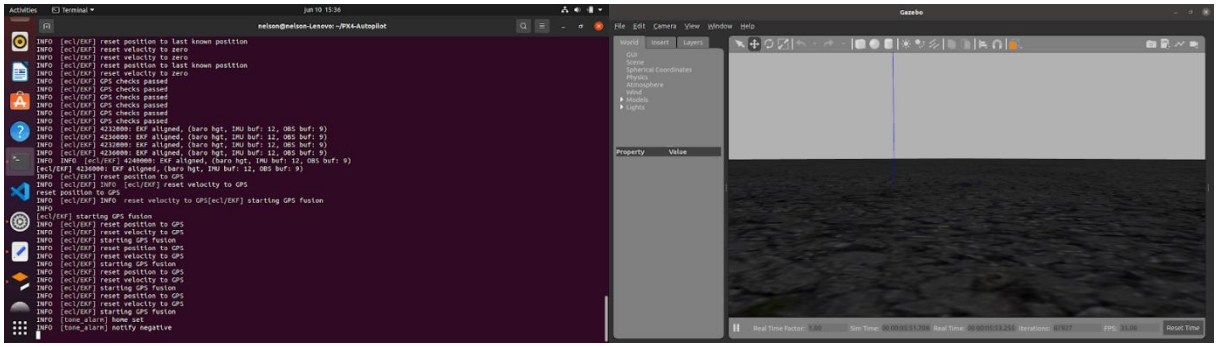


Figura 28. Ambiente de simulación Gazebo desplegado en conjunto con PX4

7. Despegar dron

Se escribe en la terminal lo siguiente, tal como se ve en la Figura 29.

```
commander takeoff
```

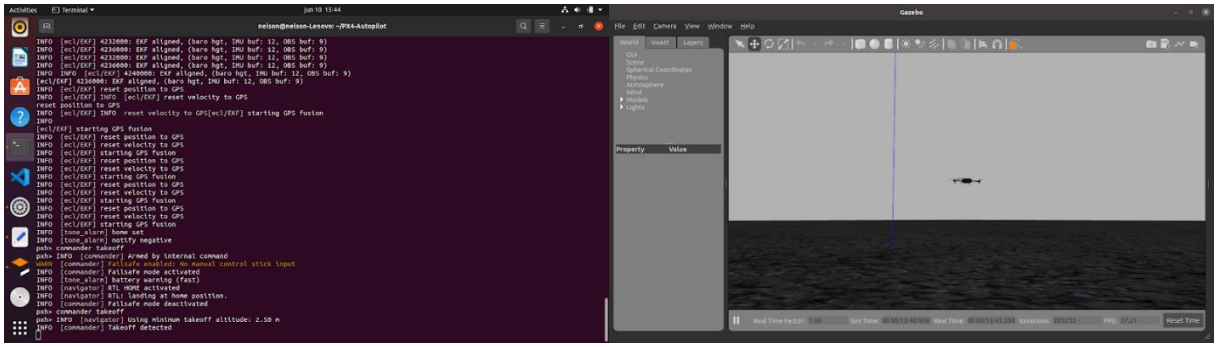


Figura 29. Simulación de despegue en el ambiente Gazebo.

Con esta instrucción, el dron debe despegar hasta una altura de 2.5 metros y aterrizar nuevamente. En la Figura 29 se puede apreciar parte de proceso.

Una vez simulado el dron, con los pasos anteriormente descritos, se puede afirmar que se realizó la instalación del ambiente de simulación y todas sus dependencias, de forma correcta.

Anexo D. Creación de escenarios Gazebo

Para crear escenarios de simulación que podrán ser visualizados en Gazebo, se puede seguir el tutorial que se encuentra en la página web de Gazebo[49].

Para el trabajo de grado, y debido a que se simularán escenarios que contienen paneles fotovoltaicos, se usó una aplicación en línea que ayuda en la generación de escenarios [50].

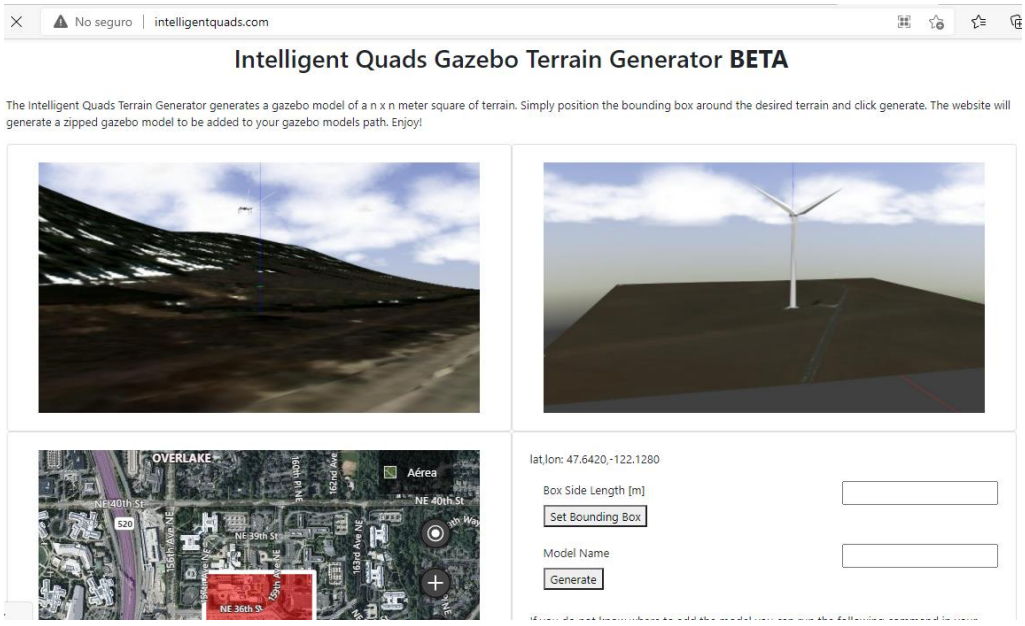


Figura 30. Página web para creación de escenarios

Tal como se ve en la Figura 30, se pueden generar escenarios realistas a partir de fotos satelitales. Se tomaron varias plantas de generación solar fotovoltaica para construir los escenarios.

Generación de archivos gráficos

- Para la creación del escenario, se debe establecer la ubicación, el Box Side Lenght y el nombre del modelo que se generará, tal como se ve en la Figura 31.

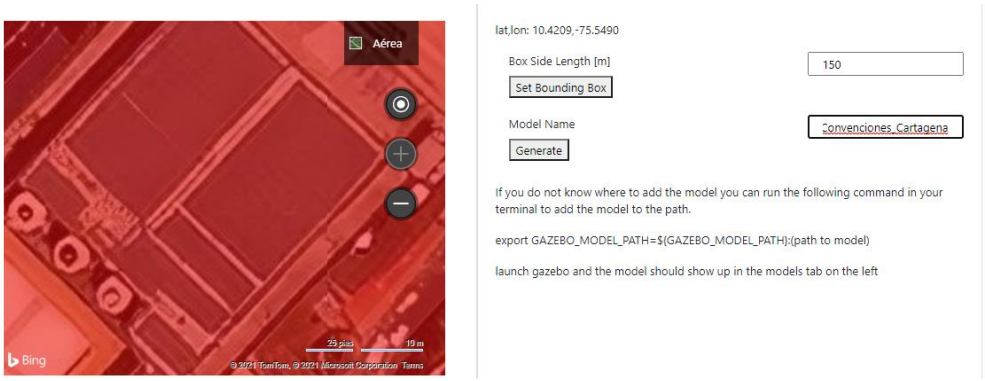


Figura 31. Proceso de creación de escenarios

Una vez establecidos estos datos, se pulsa el botón" Generate", que se visualiza en la Figura 31.

- El archivo generado es de extensión "zip". Dentro del archivo se encuentran los siguientes archivos, tal como se ve en la Figura 32.

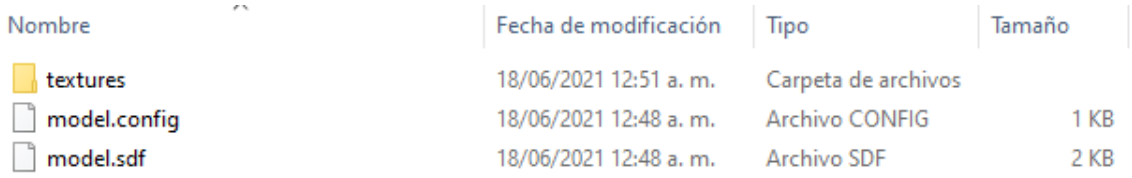


Figura 32. Contenido carpeta de escenarios

Dentro de la carpeta "textures", se encuentran los siguientes archivos, como se observa en la Figura 33. Estos archivos corresponden a la foto aérea del lugar elegido y una foto en blanco y negro que describe la altura del terreno (del inglés, heightmap) en el sitio.

Figura 33. Archivos gráficos del escenario creado

Mover archivos a carpeta PX4

Se deben mover los archivos que se visualizan en Figura 32 y Figura 33 a la carpeta "PX4-Autopilot/Tools/sitl_gazebo/models". El objetivo es que dicho modelo del terreno lo pueda tomar Gazebo al momento de simular el dron y desplegarlo como el escenario que corresponde a dicha simulación.

Anexo E. Proceso de instalación de la herramienta QGroundControl

- Ingresar en la terminal los siguientes comandos.

```
sudo usermod -a -G dialout $USER
sudo apt-get remove modemmanager -y
sudo apt install gstreamer1.0-plugins-bad gstreamer1.0-libav gstreamer1.0-gl -y
```

- Descargar la aplicación QGroundControl, véase Figura 34, de la página web de Dronecode [28].

Figura 34. Archivo QGroundControl.AppImage

- Modificar los permisos para permitir la ejecución usando la terminal con los siguientes comandos:

```
chmod +x ./QGroundControl.AppImage
./QGroundControl.AppImage
```

Una vez ejecutados los comandos, se debe desplegar la aplicación como se observa en la Figura 35.

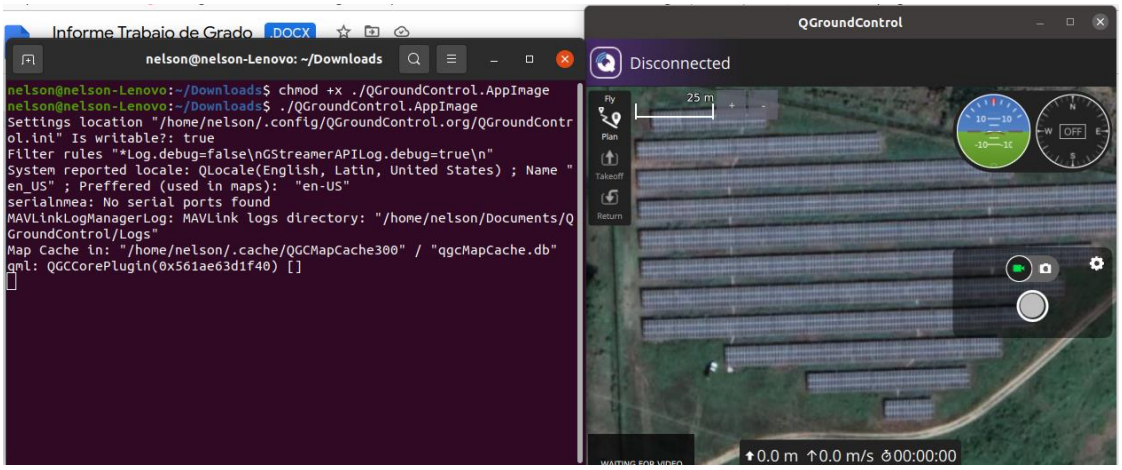


Figura 35. Ejecución de QGroundControl

Anexo F. Proceso de simulación

Para la simulación se debe seguir los siguientes pasos:

1. Lanzar el archivo con extensión ".sh" que corresponde al escenario elegido, en este caso el archivo "celsia-launch-common.sh", tal como se ve en la Figura 36. En el GitHub del proyecto [48] se pueden consultar los escenarios disponibles para simulación en la carpeta "Master ROS".

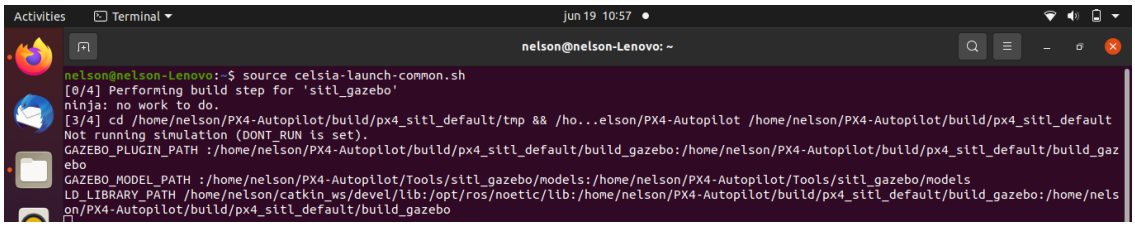


Figura 36. Lanzamiento del maestro ROS

Al lanzar el escenario, se ejecuta automáticamente el maestro ROS, se establece el escenario y sus coordenadas GPS, se ejecuta Gazebo y se inicializan los nodos de comunicación necesarios para la simulación, ver Figura 37.

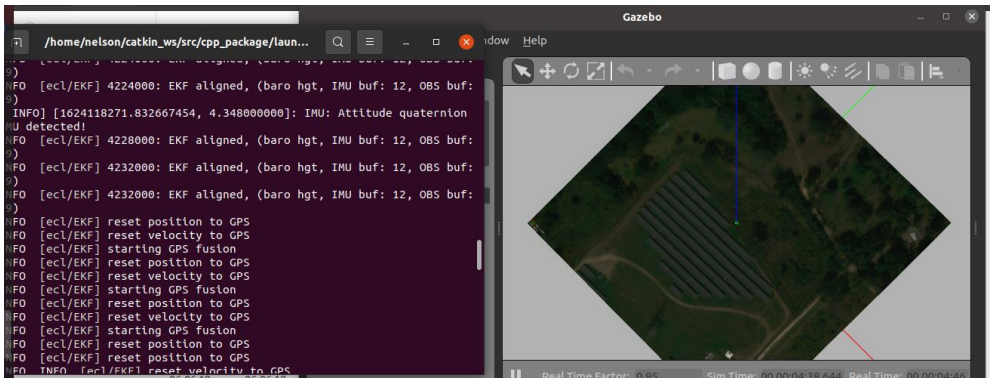


Figura 37. Ambiente de Simulación Gazebo con el escenario desplegado

2. Abrir la aplicación QGroundControl

Hacer doble clic en la aplicación "QGroundControl.AppImage". Se despliega el ambiente gráfico, tal como se ve en la Figura 38.

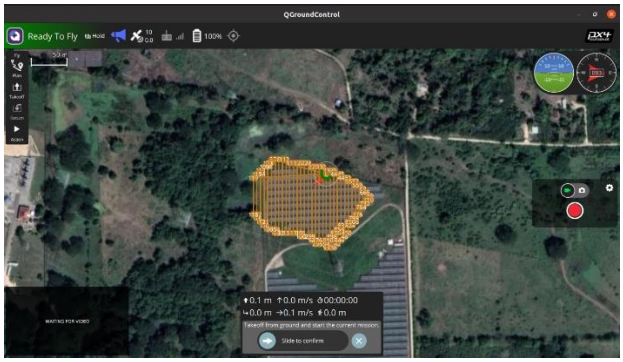


Figura 38. QGroundControl desplegado

3. Ejecutar el archivo "gui.py"

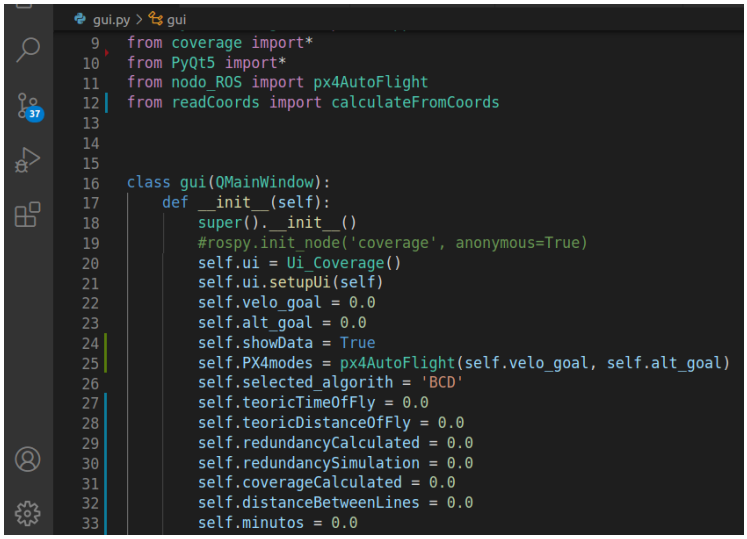


Figura 39. Archivo gui.py a ejecutarse en VS Code

Al ejecutar dicho archivo, Figura 39, se despliega la interfaz gráfica que permite calcular las rutas de cobertura e inicializar la simulación, Figura 40.

4. Seleccionar el algoritmo y los parámetros que se desean para realizar el cálculo de la ruta de cobertura, tal como se ve en la Figura 40, en el recuadro "Parámetros CPP.

Figura 40. Interfaz gráfica del proyecto

Para el ejemplo anterior, Figura 40, se seleccionó el algoritmo BCD para calcular una ruta donde el VANT tendrá una velocidad deseada de 5 m/s, distancia entre las líneas de la ruta de 5m y una altura de 10m.

5. Calcular la ruta.

Al presionar el botón "Calcular Ruta", se despliega una imagen de cómo será la ruta de cobertura a ejecutar por el dron. Dicha imagen se puede guardar para referencia, véase Figura 41.

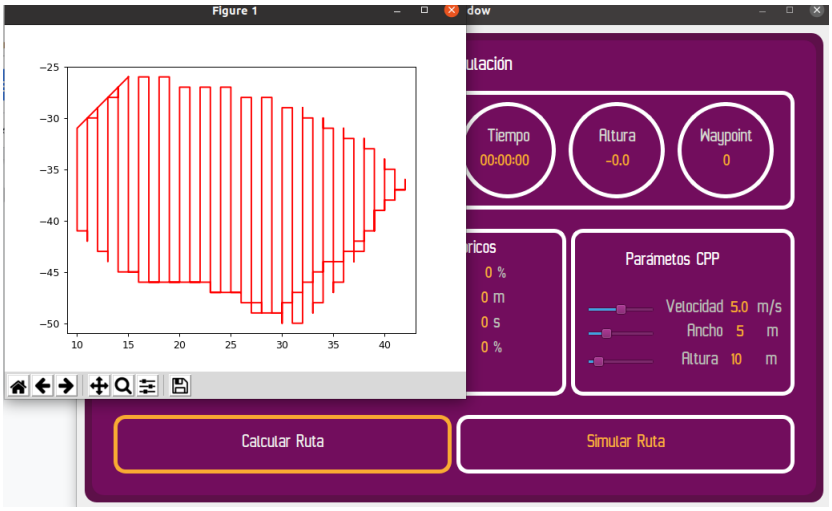


Figura 41. Cálculo de la ruta de cobertura

Para continuar con el cálculo, se debe cerrar la ventana con la gráfica.

6. Verificar métricas teóricas.

Las métricas teóricas calculadas se deben desplegar en el recuadro "Datos Teóricos". Las métricas calculadas son: redundancia, distancia del recorrido, tiempo del recorrido, cobertura del área y la cantidad de giros que posee la ruta calculada, como se observa en la figura 43. Si no se despliegan dichas métricas, esto significa que no se pudo calcular la ruta de cobertura y la razón es que la imagen aportada, y que describe el área a cubrir, posee pixeles aislados.

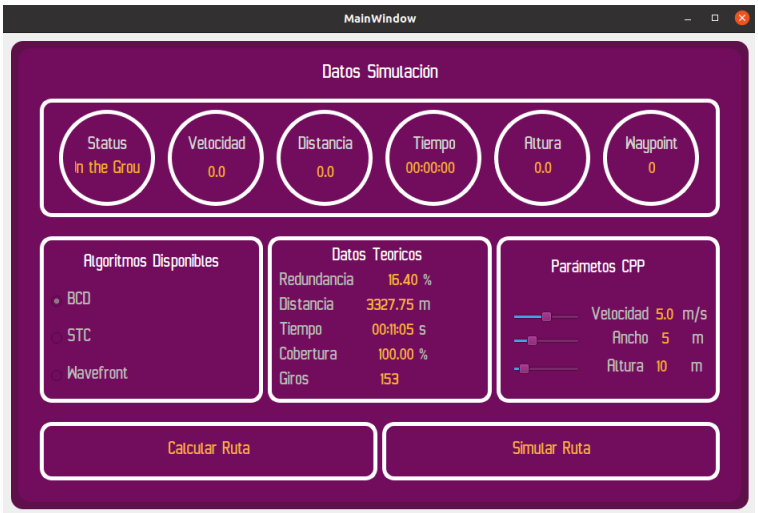


Figura 42. Interfaz con datos de la ruta de cobertura calculada

En la Figura 42, se pueden visualizar los datos teóricos calculados. Redundancia:16.40%; distancia del recorrido: 3327.75 metros; tiempo aproximado de recorrido:11 minutos y 5 segundos; cobertura calculada: 100% del área; giros:153.

7. Simular Ruta

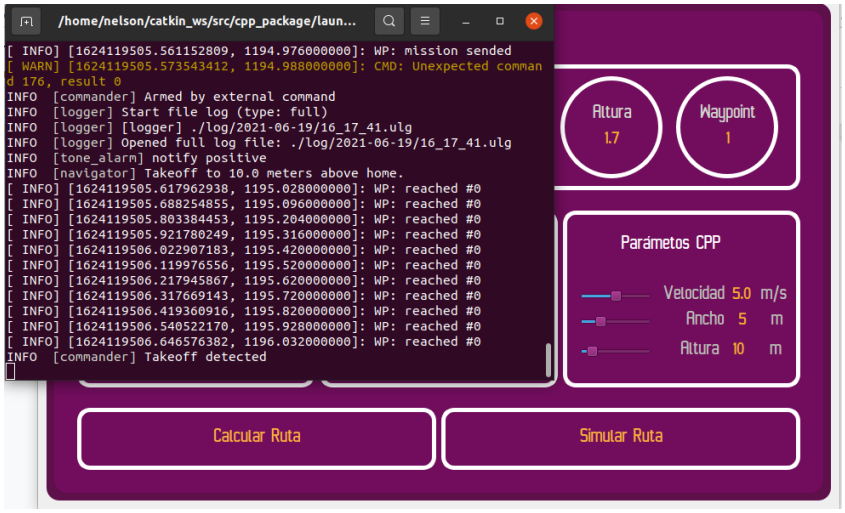


Figura 43. Inicio de la simulación de la ruta de cobertura

Al presionar el botón "Simular Ruta", se despliegan en la terminal los datos que se muestran en la Figura 43. Se puede ver que, dentro de los mensajes desplegados en la terminal, hay uno que informa del despegue del dron y la altura a la que llegará en dicho despegue. Este mensaje confirma el inicio de la simulación.

8. Visualizar el recorrido del VANT.

Figura 44. Visualización de la ruta de cobertura calculada y ruta cubierta por el VANT

Como se observa en la Figura 44, una vez se está realizando la simulación, el software QGroundControl muestra el recorrido que ha realizado el dron usando líneas rojas. Las líneas amarillas corresponden a la ruta calculada que aún está por recorrer.