Universidad de Alcalá
Escuela Politécnica Superior

GRADO EN INGENIERÍA ELECTRÓNICA Y AUTOMÁTICA INDUSTRIAL

Trabajo Fin de Grado

Robótica cooperativa para exploración planetaria

ESCUELA POLITECNICA SUPERIOR

Autor: Adrián Sánchez Chaves

Tutor/es: Pablo Muñoz Martínez

2022

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

GRADO EN INGENIERÍA ELECTRÓNICA Y AUTOMÁTICA INDUSTRIAL

Trabajo Fin de Grado

Robótica cooperativa para exploración planetaria

Autor: Adrián Sánchez Chaves

Tutor/es: Pablo Muñoz Martínez

TRIBUNAL:

Presidente: Antonio José Vicente Rodríguez

Vocal 1º: Agustín Martínez Hellín

Vocal 2º: Pablo Muñoz Martínez

FECHA: 13/07/2022

Índice:

RESUMEN 9
ABSTRACT 10
RESUMEN EXTENDIDO 11
1. INTRODUCCIÓN 15
1.1. OBJETIVOS 16
1.2. ESTRUCTURA Y CONTENIDOS 16
2. ESTADO DEL ARTE 18
2.1. ROS2 18
2.2. PX4 AUTOPILOT 21
2.3. DTMs 22
2.4. PATH PLANNING 23
3. DESARROLLO 25
3.1. ROBOTS Y SU ENTORNO 25
3.1.1. SIMULACIÓN DE MARTE 25
3.1.2. HUSKY ROVER 28
3.1.3. IRIS UAV 34
3.2. INTERFAZ DE CONTROL 40
3.3. INTEGRACIÓN 42
3.3.1. CONEXIÓN DRON-ROVER 42
3.3.2. CONEXIÓN ROVER-INTERFAZ 43
4. EXPERIMENTACIÓN 47
4.1. EXPERIMENTO 1 47
4.2. EXPERIMENTO 2 53
4.3. EXPERIMENTO 3 58
5. CONCLUSIONES Y TRABAJOS FUTUROS 60
5.1. CONCLUSIONES 60
5.2. TRABAJOS FUTUROS 61
Anexo I. Presupuesto 62
Anexo II. Manual 63
- Instalación 63
- Uso 64
Anexo III. Diagrama nodo controlador del Husky 67
Anexo IV. Diagrama nodo controlador del UAV 68
BIBLIOGRAFÍA 69

LISTA DE FIGURAS:

Figura 1: Dibujo de las partes de un brazo robótico 18
Figura 2: Esquema de funcionamiento de ROS2 20
Figura 3: Astrobee en la EEI 20
Figura 4: Funcionamiento de PX4 con un simulador 21
Figura 5: Mapa de conexiones con PX4 22
Figura 6: DTM de Utopia Planitia en Marte 23
Figura 7: Modelo SDF de marte 26
Figura 8: Árbol de ficheros de la carpeta de Marte 27
Figura 9: Simulación de Marte en Gazebo 28
Figura 10: Husky simulado en Marte 28
Figura 11: Dibujo representativo de links y joints 30
Figura 12: Comandos para adquirir la ruta de un Xacro 31
Figura 13: Declaración del nodo en archivo launch.py 32
Figura 14: Esquema de comunicación entre nodos del Husky 33
Figura 15: Triángulo formado por entre posición inicial y meta 34
Figura 16: Dron Iris simulado en Gazebo 35
Figura 17: Ajustes de la amplitud del LIDAR 36
Figura 18: Ajuste del QoS del LIDAR 36
Figura 19: Diagrama de nodos y topics de UAV y LIDAR 37
Figura 20: Esquema de conexión entre ROS2 y PX4 37
Figura 21: Representación del escáner del LIDAR en la altura 39
Figura 22: Aspecto visual de la interfaz gráfica 41
Figura 23: Diagrama de conexión ROVER-DRON 43
Figura 24: Código para crear un mapa nuevo en la interfaz 45
Figura 25: Imagen de la estación de control 47
Figura 26: Escenario del experimento 1 48
Figura 27: Mapa de la interfaz al ser creada tras la conexión 49
Figura 28: Dron después de alcanzar la altura de despegue 49
Figura 29: Visualización del mapa mientras se escanea el terreno 50
Figura 30: Dron escaneando la zona 50
Figura 31: Mapa del experimento 1 51
Figura 32: Ruta seleccionada para experimento 1 52
Figura 33: Posición 1 del experimento 1 52
Figura 34: Posición 2 del experimento 1 53
Figura 35: Posición final experimento 1 53
Figura 36: Perspectiva 1 de Marte en experimento 2 54
Figura 37: Perspectiva 2 de Marte en experimento 2 55
Figura 38: Dron escaneando zona del experimento 2 55
Figura 39: Mapa del experimento 2 56
Figura 40: Posición inicial Husky experimento 2 56
Figura 41: Posición 1 experimento 2 57

Figura 42: Posición 2 experimento 2 58
Figura 43: Posición final experimento 2 58
Figura 44: Marte con obstáculo voluminoso 59
Figura 45: Mapa experimento 3 59
Figura 46: Terminal al ejecutar el agente del puente 65
Figura 47: Diagrama del nodo controlador del Husky 67
Figura 48: Diagrama del nodo controlador del UAV 68

RESUMEN

Gracias a la robótica, hoy en día se puede llevar a cabo una serie de tareas de manera más eficiente y precisa o incluso realizar actividades que un ser humano no puede realizar. Uno de estos ejemplos, es la exploración planetaria. Marte, es el planeta donde nos hemos focalizado para, no solo estudiar el futuro, sino también el pasado. Por ello, se realizará un proyecto en el que se optimizará dicha exploración a través de la comunicación de dos robots, uno aéreo y otro terrestre. Se desarrollará a través de ROS2 y PX4 para simular misiones de exploración cooperativa a través de Gazebo.

Palabras clave: Robótica, Marte, Exploración, Comunicación, ROS2.

ABSTRACT

Thanks to robotics, nowadays we are able to do a list of tasks in a more efficient and accurate way or even do some activities that humans can't do. One of these examples, is the planetary exploration. Mars, is the planet where we have focus for, not only studying the future but also the past. Because of that, it will be made a project in which it will be optimized the mentioned exploration through a communication between two robots, one aerial and the other one over ground. It will be developed with ROS2 and PX4 and being simulated through Gazebo.

Key words: Robotics, Mars, exploration, communication, ROS2.

RESUMEN EXTENDIDO

La robótica, es una rama de la ingeniería cuyo objetivo es, a través del uso de altas tecnologías, conseguir automatizar de forma parcial o total una serie de actividades. Su desarrollo tiene una gran ventaja y es que se puede realizar distintas tareas de manera mucho más precisa que si lo hiciese un ser humano, teniendo solo que llevar un mantenimiento adecuado para que perdure en el tiempo. La robótica no solo hace los trabajos más precisos, sino que también lo hacen más rápido, por lo que se gana en eficiencia tanto a nivel técnico como a nivel temporal.

Otra de sus ventajas es el poder ejecutar algunos trabajos que requieren de mucho esfuerzo o que son peligrosos para un ser humano. Por esta razón, cada vez se están automatizando más las tareas de ciertos sectores como la minería, la construcción o el sector industrial. También tiene otra serie de utilidades como es el ejemplo de la medicina. En esta rama de las ciencias de la salud, se puede sustituir miembros del cuerpo por brazos robóticos u otro tipo de tecnologías para ayudar a mejorar la vida de las personas o facilitar las intervenciones quirúrgicas.

Sin embargo, el apartado más importante para el proyecto es la implementación de dicha tecnología en la exploración. A través de robots, se pueden visitar lugares que antes no se podían visualizar, ya sea por su ubicación o por su difícil acceso. Uno de los inventos más utilizados en este campo es el dron o UAV. El dron, es un vehículo aéreo no tripulado controlado a través de algoritmos de automatización o por RC (Radio Control). La demanda de drones está empezando a aumentar ya que se pueden usar en distintas aplicaciones. Por ejemplo, se utiliza para grabar escenas audiovisuales como las películas que se ven en el cine, o también para otras actividades prácticas como la entrega de paquetes o la agricultura.

En este proyecto, se aplicará lo anteriormente mencionado para poder optimizar la exploración planetaria a través de la comunicación de un UAV y un Rover. La misión cosiste en simular un terreno como si fuera la superficie de Marte ya que, a día de hoy, el ser humano no lo puede explorar de forma presencial. Esto es debido a las largas distancias entre planetas y las malas condiciones para la vida ya que, a pesar de tener una tenue atmósfera, no es como la nuestra por lo que no se podría sobrevivir sin un traje preparado para ello. Para empezar, se deberá preparar el entorno de simulación que, en este caso, se trata de Gazebo. Gazebo es un software de simulación utilizado para la prueba, investigación y desarrollo de proyectos de robótica. Tiene unas propiedades y una configuración muy realista, pudiendo así experimentar de forma segura antes de utilizar el *hardware* en el entorno real. Además, también permite utilizar cualquier tipo de entorno donde probar los robots, incluido el de un planeta como es Marte. Una vez preparado el simulador, se necesitará crear los modelos del UAV y del rover para poder utilizarlos dentro de Gazebo. En este caso, se utilizará el vehículo terrestre de exploración llamado Husky y el vehículo aéreo llamado Iris.

Una vez que esté todo preparado, se hará despegar el dron a una cierta altura para que, a través del uso de un LIDAR, pueda tomar medidas de la distancia al suelo. Esto lo hará según va avanzando por una zona concreta que se quiera explorar. Un LIDAR es un sensor que utiliza un emisor láser para poder medir las distancias. Para ello, va emitiendo una serie de pulsos y calculando el tiempo que tarda en volver a detectarlo, midiendo así el espacio que hay hasta

el objeto contra el que impacta el rayo lanzado. Mientras el dron va barriendo todo el terreno, se irá calculando las distancias y, con la información de la posición del GPS, se calculará una distancia para cada 0,5m de superficie recorrida. Según se obtengan los valores de los resultados, se trasmitirán al rover de forma inmediata. Esto es debido a que será el UGV el que contenga el servidor, el cual se conectará la interfaz de control para la transmisión de los datos. Esto es debido a que es más sencillo tener una conexión directa entre un servidor y un cliente que tener que conectar el cliente a dos servidores distintos. Sobre todo, teniendo en cuenta que el cliente, la interfaz de control que se utilizará, estaría ubicada en nuestro planeta, mientras que los robots estarían a millones de kilómetros, por lo que es una comunicación complicada. Esta comunicación, se haría a través de un protocolo TCP/IP donde a través de *sockets*, se podrá contactar entre servidor y cliente.

Al crear el servidor, se deberá especificar el tamaño del mapa a visualizar para que, cuando detecte la conexión, se preconfigure un mapa base. A partir de ahí, se comenzará a procesar los datos enviados por el rover y, tras ello, se visualizará en la interfaz gráfica el mapa en forma de DTM. Un mapa DTM es un tipo de mapa de elevación digital. Este se caracteriza por mostrarnos la topografía de un entorno concreto, clasificando por coordenadas en función de las distintas alturas.

Después de obtener el territorio a explorar, se podrá marcar manualmente tanto el punto de partida como la coordenada que se quiera alcanzar. La posición del UGV se podrá ir visualizando actualizada en la propia interfaz a través de un triángulo azul. Cuando se tenga el objetivo marcado, se usará el algoritmo 3Dana el cual se encargará de trazar la ruta más segura en función de una pendiente máxima previamente especificada. Una vez obtenidas las coordenadas a seguir, serán enviadas al rover para que vaya ejecutando de forma autónoma un algoritmo de control para llegar a los objetivos, explorando así la zona de una forma optimizada.

El control de los robots se logrará a través de un *software* llamado ROS2. Dicho *software* contiene una serie de herramientas que permiten el desarrollo de la robótica tanto a nivel *software* como *hardware*. En cuanto al dron, no solo se necesitará el uso de ROS2 sino que también tendrá que comunicarse a través de un DDS al *softwar*e PX4. Dicho *software* es utilizado para pilotar y controlar drones en la vida real, pero también tiene la posibilidad de usarlo en herramientas de simulación como Gazebo.

Gracias a este proyecto, se aprenderá a controlar robots aéreos y terrestres de forma totalmente autónoma. En consecuencia, se utilizará un sistema de exploración que ayudará a movernos por entornos desconocidos de otros mundos a mayor velocidad y con mayor seguridad.

GLOSARIO DE ACRÓNIMOS Y ABREVIATURAS

NASA National Aeronautics and Space Administration

TCP Transmission Control Protocol

IP Internet Protocol

DDS Data Distribution Service

TFG Trabajo Fin de Grado

EEI Estación Espacial Internacional

SITL Software In The Loop

API Application Programming Interfaces

UDP User Datagram Protocol

UAV Unmanned Aerial Vehicle

RC Radio Control

LIDAR Laser Imaging Detection and Ranging

GPS Global Positioning System

DTM Digital Terrain Model

UGV Unmanned Ground Vehicle

ROS Robot Operating System

RVIZ ROS Visualization

QGC QGroundControl

DEM Digital Elevation Model

HiRISE High Resolution Imaging Science Experiment

MOLA Mars Orbiter Laser Altimeter

IMU Inertial Measurement Unit

SDF Simulation Description Format

URDF Unified Robot Description Format

QoS Quality of Service

NED North East Down

ENU East North Up

SCTP Stream Control Transmission Protocol

SLAM Simultaneous Localization and Mapping

RTPS Real Time Publish Subscribe

MRO Mars Reconnaissance Orbiter

1. INTRODUCCIÓN

La era espacial comenzó en el año 1957 con el lanzamiento del Sputnik-1 por la Unión Soviética. Este fue el primer satélite del mundo en salir de nuestro planeta. A partir de ese instante, comenzó una carrera espacial la cual trajo importantes descubrimientos y distintos hitos. De los eventos más destacados se encuentra la llegada de Yuri Gagarin al espacio en 1961 o cuando el ser humano llegó a la Luna en 1969.

Después de este momento, el ser humano actualizó sus fronteras y comenzó a centrarse en otros planetas, como Venus o Marte. En 1997, el primer rover de la historia aterrizaba en el planeta rojo. Este fue bautizado como el *Sojourner.* No fue hasta 2004 que se consiguió aterrizar el *Opportunity,* el cual logró sobrevivir hasta 15 años en funcionamiento. Unas semanas antes, se aterrizó en otra parte del planeta el robot *Spirit.* Después, le siguieron el *Curiosity* en 2012 y, por último, el *Perseverance* en 2021. Este último, aparte de ser el más avanzado a nivel tecnológico, incluye un dron llamado *Ingenuity*, el cual ha conseguido realizar el primer vuelo en otro planeta. Aunque inicialmente estuviese programado para hacer 5 vuelos, a día de hoy sigue en activo, habiendo hecho ya hasta un total de 25 vuelos, proporcionando así una serie de imágenes que ayudarán a entender mejor el planeta.

Esta misión llevada a cabo por la NASA ha servido de inspiración para la realización de este proyecto. En concreto, se utilizará tanto un dron como un rover, que cooperarán para explorar el planeta. Primero, se hará despegar al dron a una altura determinada, pudiendo así barrer con su movimiento un total de 2,5m^2 de superficie. Después, el dron se moverá de forma autónoma para poder recorrer un espacio determinado. Mientras lo hace, se irán recogiendo los datos de la distancia entre el dron y el suelo a través de un sensor llamado LIDAR. Una vez obtenidos, se trasmitirán al rover para que este, pase la información a través de una conexión TCP/IP a una interfaz de control de Java, donde se podrá ver como se crea un mapa DTM en función de las medidas obtenidas. Tras lograrlo, se habilitará la opción de elegir manualmente el destino al que se quiera llegar y así, gracias al algoritmo 3Dana, se conseguirá obtener la ruta más accesible, evitando pendientes excesivas. Posteriormente, se transmitirá dicha información al robot terrestre para que este pueda desplazarse de forma segura.

El proyecto se llevará a cabo en un entorno de simulación llamado Gazebo. Se controlará con un *software* diseñado para la automatización de robots llamado ROS2. Además, se utilizará un DDS como middleware para conectar ROS2 con PX4, un programa utilizado para el control de drones. Todo ello, se ejecutará utilizando el sistema operativo Ubuntu, una distribución de Linux.

1.1. OBJETIVOS

El objetivo principal del TFG es conseguir optimizar la exploración planetaria a través de la comunicación de dos tipos de robots distintos. El primero, el UAV, se encargará de investigar la zona de alrededor y recoger los datos del relieve de la misma. El segundo, llevará incluido el servidor de comunicación con la interfaz de control y se encargará de comunicarse con el dron, al mismo tiempo que con dicha interfaz. Además, recorrerá el planeta con la información recibida, tanto del robot aéreo como del algoritmo 3Dana. Gracias a esta cooperación, un robot podrá avanzar y coger muestras de lo que necesite de forma segura, mientras el otro se encarga de explorar la zona para asegurarse de que el rover pueda llevar a cabo la misión comentada sin peligro alguno.

Para lograr este objetivo, se necesita estructurar el trabajo en una serie de hitos:

- Estudio e instalación de ROS2 y Gazebo como *softwares* que se utilizarán a lo largo del proyecto.
- Obtener los modelos 3D de Marte, el dron y el rover para poder simularlos en el entorno de Gazebo.
- Implementar el código necesario para conseguir mover nuestro UGV de forma totalmente autónoma, en función de unas coordenadas concretas.
- Trabajar en la integración de un servidor dentro del rover para que se comunique a través del protocolo TCP/IP con la interfaz gráfica de control. Con ello, se podrá enviar y recibir mensajes para llevar a cabo la misión.
- Estudio e instalación del software PX4, con su respectivo DDS para la conexión del dron con ROS2 y poder así controlar el vuelo del dron de forma automatizada.
- Realizar conexión entre robots para poder enviar la información a la interfaz de control.
- Adaptar la interfaz de Java para recibir los datos y, en función de la información obtenida, crear un mapa DTM.
- Finalmente, trazar las rutas necesarias con el algoritmo 3Dana desde la interfaz gráfica y enviarlas al rover para que las ejecute. Con ello, se comprobará en varias situaciones el funcionamiento del proyecto completo cambiando el entorno donde se realiza.

1.2. ESTRUCTURA Y CONTENIDOS

La memoria del proyecto a realizar está dividida en 5 diferentes secciones:

1. **Introducción**. Se presentará el trabajo a desarrollar, así como el contexto y los objetivos principales que se pretenderán alcanzar.
2. **Estado del arte**. Sección en la que se explica con mayor profundidad cómo funcionan y que son las distintas técnicas que se utilizarán.
3. **Desarrollo.** Implementación e integración del código necesario que se realizará para la ejecución correcta del proyecto.

4. **Experimentación**. Comprobación practica del trabajo realizado con distintas condiciones y entornos.
5. **Conclusiones y trabajos futuros.** Trabajos futuros relacionados con el proyecto, así como las conclusiones del mismo.

2. ESTADO DEL ARTE

Esta sección está destinada para explicar las principales plataformas y elementos tecnológicos que se utilizarán a lo largo de todo el proyecto. Se comentará qué son, como funcionan y para que se necesitan a la hora de lograr la misión establecida.

2.1. ROS2

ROS2 (Robot Operating System 2) es un *software* de código abierto formado por un conjunto de herramientas, componentes e interfaces, necesarias para simplificar la construcción de distintas aplicaciones robóticas. En este caso, se utilizará la versión de ROS2 Galactic, la última distribución que hay disponible actualmente.

La mayoría de los robots están formados por actuadores, sensores y controladores. Los actuadores son la parte de los robots encargadas de la ejecución, es decir, del movimiento que tengan que realizar. Por ejemplo, en un brazo robótico como el de la Figura 1, los actuadores se encargan de subir, bajar o coger el objeto que tiene delante.

Los sensores son los responsables de captar información de su entorno, lo que equivaldría a los sentidos en el ser humano. Hay de muchos tipos como sensores de humedad, temperatura, presión, luz, sonido, distancia o posición, entre otros. En el caso del proyecto a realizar, se utilizará un LIDAR como sensor principal, el cual irá embarcado en el dron encargado de medir distancias mientras avanza por un terreno. Esto lo hace emitiendo un pulso de rayo láser y calculando el tiempo que tarda en volver a recibirlo. Una vez que se conoce la velocidad y el tiempo, se podrá calcular el espacio de forma trivial.

Por último, están los controladores, el que desempeña la función de ser el cerebro del propio robot. Con ello, se manda las instrucciones a los actuadores y al mismo tiempo, recibe los datos de los sensores.

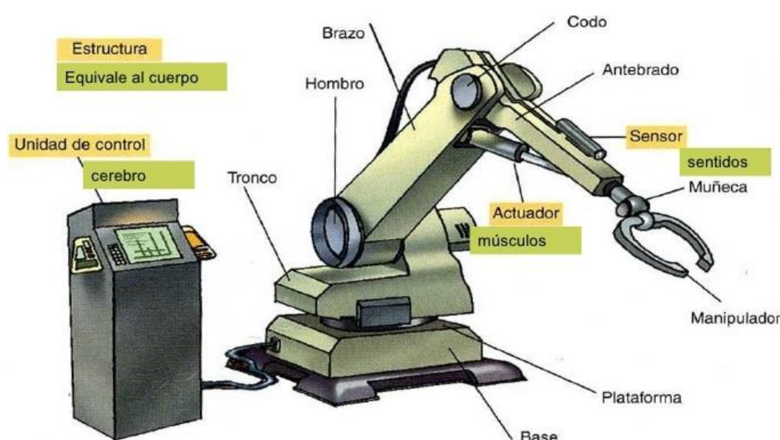


Figura 1: Dibujo de las partes de un brazo robótico

Gracias a ROS y ROS2, la última versión más actualizada del *software*, se podrá construir y configurar estos componentes, además de conectarlos entre sí desde un mismo sistema. De esta manera, se permite una comunicación sencilla, haciendo así más fácil el desarrollo de cualquier robot. Además, la información de la comunicación entre los 3 elementos puede ser almacenada y comprobada para la mejora del rendimiento del robot.

Otra ventaja de ROS2 es que no solo trabaja con *hardware*, sino que también puede simular robots a través de otros *softwares* como Gazebo. También se pueden usar otro tipo de simuladores, como el de la propia herramienta que incluye el paquete de instalación de ROS2 llamado RVIZ. La virtud es que funciona con cualquier componente que tenga un modelo creado a nivel de software, por lo que tiene un gran catálogo de opciones que se pueden utilizar, tanto en sensores como en robots. Esto es debido a que es un *software* de código totalmente libre y gratuito que se nutre de la contribución de sus usuarios. Gracias a esta cooperación, cada vez más usuarios eligen utilizar ROS/ROS2 y con ello, cada vez hay más modelos y ejemplos que poder utilizar. Gracias a su versatilidad, se puede tener el mismo robot que en la vida real, dentro de tu simulador, pudiendo así, probarlos con seguridad antes de utilizarlos de forma física.

En cuanto a su funcionamiento, ROS2 se divide en varias partes interconectadas, como puede verse en el esquema de la Figura 2:

- **Nodos:** Los nodos son la parte principal de ROS2. Consiste en la parte del código que contiene las instrucciones para realizar o procesar la información del robot. Por ejemplo, si se quiere mover un robot, el código que dice a donde ir y como moverse, se ejecuta en un nodo. Normalmente, se utiliza un nodo para cada elemento que se quiera controlar. Teniendo en cuenta esto, la comunicación entre los nodos es fundamental. Esta se hace a través de *topics*, servicios, acciones y parámetros.
- **Topics*:*** Los *topics* son una parte esencial. Su misión es comunicar un nodo con otro a través de mensajes. Para ello, utiliza dos componentes creados en los nodos:
  - **Publicador**: Se configura en el nodo emisor del mensaje. Es el punto de partida del mensaje que contiene el *topic*.
  - **Subscriptor**: Se configura en el nodo receptor del mensaje. Es el punto de llegada del mensaje que contiene el *topic*.
- **Servicios:** Los servicios son otro tipo de comunicación alternativa a los topics. Están formados por dos partes:
  - **Servidor**: El servidor se encarga de proporcionar una información concreta la cual se va actualizando periódicamente, pero solo la publica cuando el cliente se lo solicita.
  - **Cliente**: El cliente solicita al servidor cierta información solo cuando la necesita y el servidor se la proporciona.
- **Parámetros**: Los parámetros son como los ajustes de los nodos. Con ellos, se puede configurar y cambiar en cualquier momento un valor concreto del nodo.

- **Acciones**: Son otro tipo de comunicación entre nodos. Funciona como una mezcla entre los servicios y los *topics* ya que contiene un servidor y un cliente, pero, al mismo tiempo, tiene un *topic* que va publicando la información en forma de *feedback*.

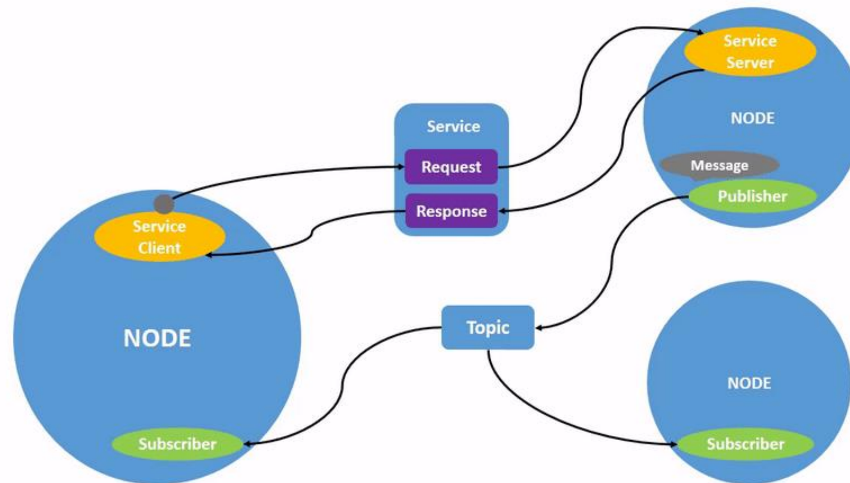


Figura 2: Esquema de funcionamiento de ROS2

En conclusión, ROS2 es una herramienta muy útil para los desarrolladores que se dedican a la robótica, ya que aporta muchas facilidades para controlar y crear nuevos robots, ya sea para su simulación o para su aplicación en la vida real. Además, con la ayuda de la comunidad, ya que es de código abierto, cada vez hay más modelos y más tipos de implementación que permite que se pueda llevar a cabo un mayor número de proyectos.

Debido a lo mencionado anteriormente, cada vez más empresas están utilizando este tipo de *software*, sobre todo en el ámbito espacial. La NASA ya ha utilizado ROS para poder probar algunos de sus robots e incluso utilizarlos en el espacio, como por ejemplo el robot *Astrobee (*véase Figura 3) de la EEI, que funciona a través de ROS. Por ello, en este proyecto se desarrollará la exploración planetaria de los robots a través de dicho *software*.

Figura 3: Astrobee en la EEI

2.2. PX4 AUTOPILOT

PX4 Autopilot es un *software* de código abierto orientado a aeronaves autónomas para manejarlas a través de control remoto. Se usa tanto en personas que quieran trabajar de forma individual como a nivel industrial. Está diseñado para ser utilizado con *hardware* en la vida real, y es uno de los más potentes del mercado. Para poder ser utilizado, necesita conectarse a una estación de control que detecte la posición real del dron a través de la configuración del GPS. Esta aplicación se llama QGroundControl. Sin ella, el dron no podría despegar por lo que se usará en este proyecto para dicho fin.

Aunque se suela utilizar para controlar los vehículos reales, también existe la opción de poder simularlos. Esto es debido a que el control de drones no es tarea fácil y puede llegar a ser peligroso por lo que, a veces, se utiliza herramientas de simulación como Gazebo. Para ello, utiliza el tipo de simulador SITL (Software in the Loop).

Para que PX4 pueda comunicarse con un simulador, necesita utilizar una API conocida como MAVlink. Esta API funciona como puente entre las dos plataformas, permitiendo así la comunicación bidireccional entre ambas. Además, MAVlink define una serie de mensajes para dicha comunicación. PX4 recibe información de sensores (véase Figura 4), GPSs, cámaras y otros, y el simulador recibe instrucciones para ejecutar el control previamente definido de los actuadores.

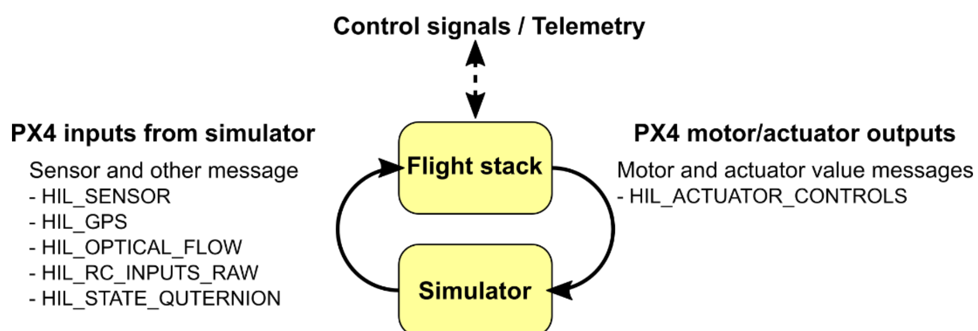


Figura 4: Funcionamiento de PX4 con un simulador

Para que esta y otras conexiones sean posibles, MAVlink utiliza una serie de puertos UDP y TCP a los que PX4 se conecta, dependiendo del elemento al que necesite unirse. Por ejemplo, si se quiere conectar con QGC, la estación de control se conectará a través del puerto UDP 14550. En cuanto a la transmisión de mensajes con el simulador, utiliza una conexión TCP en el puerto 4560 como se muestra en la Figura 5. Asimismo, se debe añadir que en el proyecto a desarrollar no se va a controlar el vuelo del dron a través de PX4, sino que se utilizará ROS2 al igual que con el rover. A esto se le conoce como control externo y necesitará conectarse a PX4 a través del puerto UDP 14540.

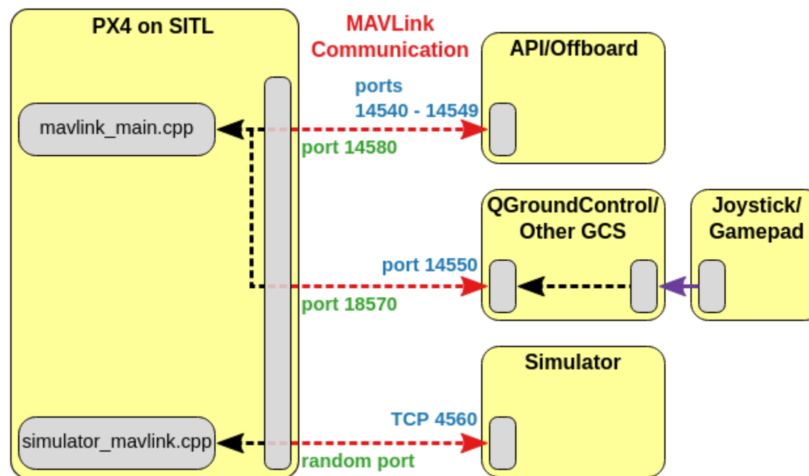


Figura 5: Mapa de conexiones con PX4

2.3. DTMs

Los DTMs (Digital Terrain Model) son mapas topográficos del terreno en formato digital, para que pueda ser manipulado por programas informáticos. Los datos recopilados en los mapas comprenden las distintas alturas del terreno observado y, posteriormente, las almacena para saber cómo varía el nivel de la superficie. A menudo se suele confundir con los mapas DEMs, pero la gran diferencia es que, los DTMs, son una representación de lo que se conoce como el terreno totalmente desnudo. Esto significa que solo tiene en cuenta la superficie, pero no el resto de los elementos como puede ser la vegetación, edificios u otro tipo de objetos elevados que pudiesen influir en las medidas.

En el caso de Marte, los mapas DTMs son realizados por HiRISE, una cámara muy potente que orbita el planeta a bordo de la sonda MRO (Mars Reconnaissance Orbiter) desde el año 2006. Esta cámara de alta resolución es capaz de tomar imágenes de hasta 30cm por píxel. La cámara fue lanzada en una colaboración de la NASA con la Universidad de Arizona y, gracias a ella, existen mapas DTMs de alta definición de la superficie marciana. Esto ayuda a estudiar el terreno y con ello elegir los puntos de aterrizaje para los robots y, en un futuro, incluso para humanos Las imágenes están disponibles públicamente en el repositorio del LPL de la Universidad de Arizona[1].

Para crear estos mapas, se toman dos imágenes diferentes de una misma zona, pero desde distintos ángulos. El problema es que no es un proceso tan sencillo, ya que implica una gran cantidad de tiempo, tanto a nivel personal como computacional. Por esta razón, todavía no se han realizado DTMs de todas las fotos obtenidas. En cuanto a su preparación, consta de varios procesos:

[1] https://www.uahirise.org/dtm/

- **Preparación**: En este proceso, la imagen es editada para eliminar las distorsiones ópticas que puedan haber surgido tras la captura.
- **Triangulación:** Es el proceso más exigente en cuanto a esfuerzo y tiempo. Consiste en registrar las imágenes junto al mapa de elevación MOLA.
- **Creación**: En este proceso se crean los mapas DTMs una vez las imágenes han sido trianguladas.
- **Edición**: Este proceso es muy costoso por lo que no siempre se realiza. Consiste en corregir los errores que se formen tras la creación del mapa.
- **Ortorrectificación**: Una vez se tiene el DTM, se puede crear otro tipo de mapas como por ejemplo el ortorectificado el cual consiste en transformar el mapa de tal manera que parezca que se mira cada pixel desde una posición ortogonal.

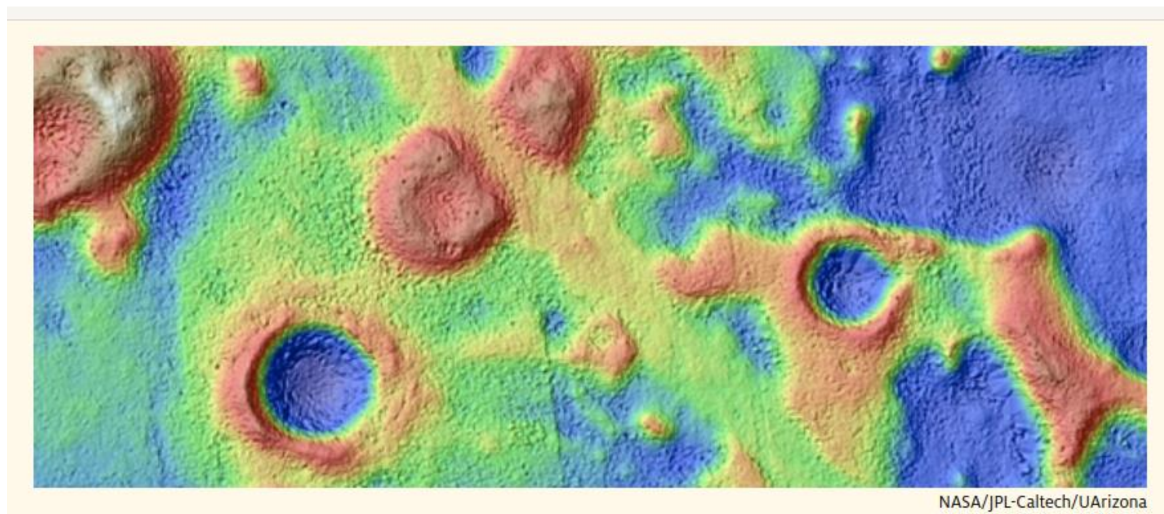


Figura 6: DTM de Utopia Planitia en Marte

2.4. PATH PLANNING

El *Path planning* o planificación de rutas consiste en un algoritmo cuya misión es la creación de una ruta de un vehículo para poder trasladarse desde un punto inicial, a una meta establecida previamente. Este desplazamiento debe ser de la manera más segura posible, evitando así obstáculos u otras condiciones que puedan perjudicarle.

Para trabajar con este tipo de algoritmos se necesita un mapa. Este mapa es dividido en una serie de cuadrículas para utilizar un sistema de coordenadas donde, cada celda, corresponde con un nodo. Estos nodos tienen pueden representar la elevación del terreno, como es nuestro caso, u otras características del mismo (como obstáculos, presencia de piedras, arenas sueltas, etc.). En consecuencia, el algoritmo tratará de moverse de un punto a otro eligiendo solo las celdas libres y evitando zonas que puedan ser consideradas peligrosas por las características del terreno.

Para desarrollar los distintos algoritmos de planificación, se utilizan técnicas heurísticas. Esto consiste en un tipo de algoritmo diseñado para resolver problemas de una manera más rápida

y eficiente que otro tipo de métodos. Esta técnica de optimización es muy utilizada para la planificación de rutas, pero también para otro tipo de aplicaciones como diseño de sistemas electrónicos digitales, simulación y control para el modelado de sistemas o clasificación de datos. Dentro de los algoritmos de búsqueda del movimiento planificado, hay varios tipos que se han desarrollado a lo largo del tiempo, siendo modificados y perfeccionados para aumentar su eficiencia. Sin embargo, los más habituales están basados en el algoritmo A*.

3. DESARROLLO

En esta sección se explicará el desarrollo realizado para la correcta ejecución del proyecto. Para ello, se dividirá en tres partes. La primera, consiste en la explicación de los robots utilizados y cómo funcionan además del entorno donde se probarán. El siguiente apartado, corresponde con la parte de la interfaz gráfica de control. En dicho apartado se comentará cómo opera, qué hace y que papel desempeña en la misión. Además, se comentará el uso del algoritmo 3Dana para su correcto funcionamiento. Por último, se hablará sobre la integración del código a ejecutar, así como la conexión de ambas partes, es decir, como se comunican los robots con la interfaz.

3.1. ROBOTS Y SU ENTORNO

Primero se comentarán los robots elegidos, tanto el dron como el rover, y el entorno donde se simula, el cual será un terreno que imita la superficie de Marte. Además, se explicará el por qué se han elegido y sus características principales.

Como se ha comentado anteriormente, el proyecto se simula a través de un *software* llamado Gazebo. Este simulador consiste en un conjunto de herramientas, librerías y servicios que permiten simular todo tipo de modelos de robótica, para así probarlos en distintos entornos con facilidad antes que crearlos en la vida real.

A la hora de ejecutar una simulación en Gazebo, hay dos tipos de archivos:

- **Modelos:** Son archivos del tipo *model.sdf* o *".urdf"* escritos en lenguaje XML. Cualquier tipo de robot u objeto que se quiera añadir será creado en base a este formato de archivo y bajo este dominio. Ya sean robots, sensores o incluso los entornos. Un modelo ".sdf" puede contener otros dentro para hacer sistemas más complejos.
- **Mundos:** Son archivos del tipo *world_name.world* también escritos en XML. Utilizan el mismo formato que el sdf, pero añadiendo modelos que simulen el entorno. Suele ser utilizado con ROS2 para que al crear un archivo *".launch",* pueda abrirse la aplicación de Gazebo con el entorno preconfigurado.

3.1.1. SIMULACIÓN DE MARTE

Para conseguir simular Marte, se utilizará un archivo escrito en XML para que Gazebo pueda reconocerlo. Para ello, primero hay escoger el terreno que se quiere modelar. En este caso, se ha decidido utilizar un modelo 3D basado en una ubicación real del planeta. Dicho modelo se obtiene a partir de la página de la base de datos del instrumento HiRISE comentada anteriormente. El modelo seleccionado corresponde con el cráter Gale, conocido por ser el

lugar donde aterrizó el rover *Curiosity.* Cabe destacar que se ha escogido no solo por ser una ubicación conocida, sino porque tiene distintos cambios de relieve lo cual ayudará a la hora de hacer distintas pruebas durante la experimentación.

Para lograr introducirlo, se necesita obtener el modelo en formato *".dae"* para poder incluirlo dentro de un archivo *model.sdf.* Para conseguirlo, se necesita de una aplicación externa ya que, al descargar el modelo, está en formato *".stl".* En este caso, se utilizará el *sofware* Blender usado para trabajar con modelos 3D. Una vez se importe el archivo original, se podrá exportar con el formato *".dae"* deseado.

Después, se procede a crear el mencionado archivo *model.sdf,* mostrada en la Figura 7. Para ello, se sabe que el lenguaje XML funciona por etiquetas, por lo que hay que crear una estructura definida con etiquetas unas dentro de otras. El modelo debe estar definido como un *link* ya que es la manera de indicar que forma parte de la estructura principal. En el caso de Marte, solo hay un *link* ya que todo el terreno es solo un componente. Dentro del *link* se tiene que definir la geometría tanto visual como la parte de colisión la cual debería ser las mismas para que a la hora de chocar con otros objetos, puedan interactuar o, en este caso, poder funcionar como suelo y que los robots se posen encima.

```
1  <?xml version="1.0" ?>
2  <sdf version="1.7">
3      <model name="mars">
4          <pose>0 0 0 0 0 0</pose>
5          <static>true</static>
6          <link name="body">
7              <visual name="visual">
8                  <geometry>
9                      <mesh><uri>model://mars/media/curiosity_landing.dae</uri></mesh>
10                 </geometry>
11                 <material>
12                     <script>
13                         <uri>model://mars/media/scripts</uri>
14                         <uri>model://mars/media/textures</uri>
15                         <name>RepeatedMars_texture/Diffuse</name>
16                     </script>
17                 </material>
18             </visual>
19             <collision name="collision">
20                 <geometry>
21                     <mesh><uri>model://mars/media/curiosity_landing.dae</uri></mesh>
22                 </geometry>
23             </collision>
24         </link>
25     </model>
26 </sdf>
27
```

Figura 7: Modelo SDF de marte

Para que se pueda utilizar, se crea una carpeta dentro de uno de los paquetes creados de ROS2, organizándolo de la manera que se puede visualizar en la Figura 9.

Para poder darle una estética realista y que no esté en blanco y negro, se procederá a descargar la textura oficial de Marte a través de una web proporcionada por la NASA llamada Mars Trek. Para ello, hay que seleccionar una zona y solicitar un fichero *".obj"* el cual viene integrado con la textura correspondiente en formato *png.* Después, se incluye en la carpeta "media". Para que pueda verse a través de todo el terreno, se crea un código para repetir el material a lo largo de todo el modelo.

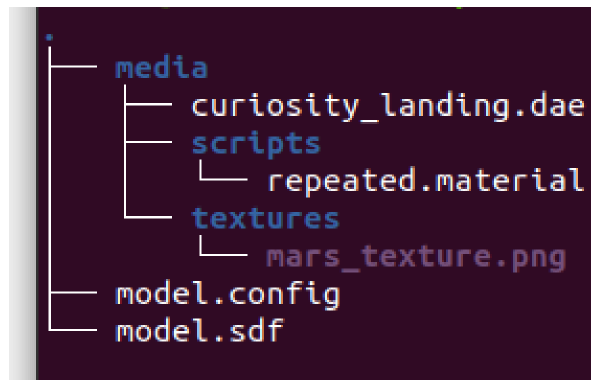


Figura 8: Árbol de ficheros de la carpeta de Marte

Por último, es importante recordar que para que el modelo sea reconocido por Gazebo, también debe incluir un archivo en la misma ubicación que el *sdf* que se llama *model.config,* el cual contiene solo información del nombre del modelo y datos como la descripción o el autor.

Para poder introducirlo en Gazebo manualmente, hay que colocar la capeta del modelo en la ubicación ~/.gazebo/models, donde podrá incluirse desde la casilla de insertar.

Al principio, se decidió añadir el modelo a un fichero *".world"* con el nombre de *mars.world.* Para esto, solo había que hacer un XML con la etiqueta <world> e incluir un modelo del Sol para que dé luz al entorno, y el propio modelo de Marte previamente configurado. Sin embargo, al añadirlo posteriormente junto al dron, el UAV daba problemas con el terreno ya que, al lanzarlos juntos, detectaba una serie de problemas que argumentaban que no era seguro despegar. Esto se debe a que PX4 es un *software* diseñado para tratar con drones en la vida real, por lo que podría ser peligroso no hacerlo en las condiciones idóneas. Por lo tanto, PX4 tiene una parte del programa dedicado a comprobar el entorno y si la situación es la adecuada. Estas comprobaciones se llaman "*Preflight checks".* Aunque se modificó el código para intentar deshabilitarlo, siguió dando errores por lo que se decidió insertar manualmente después de hacer aparecer el dron.

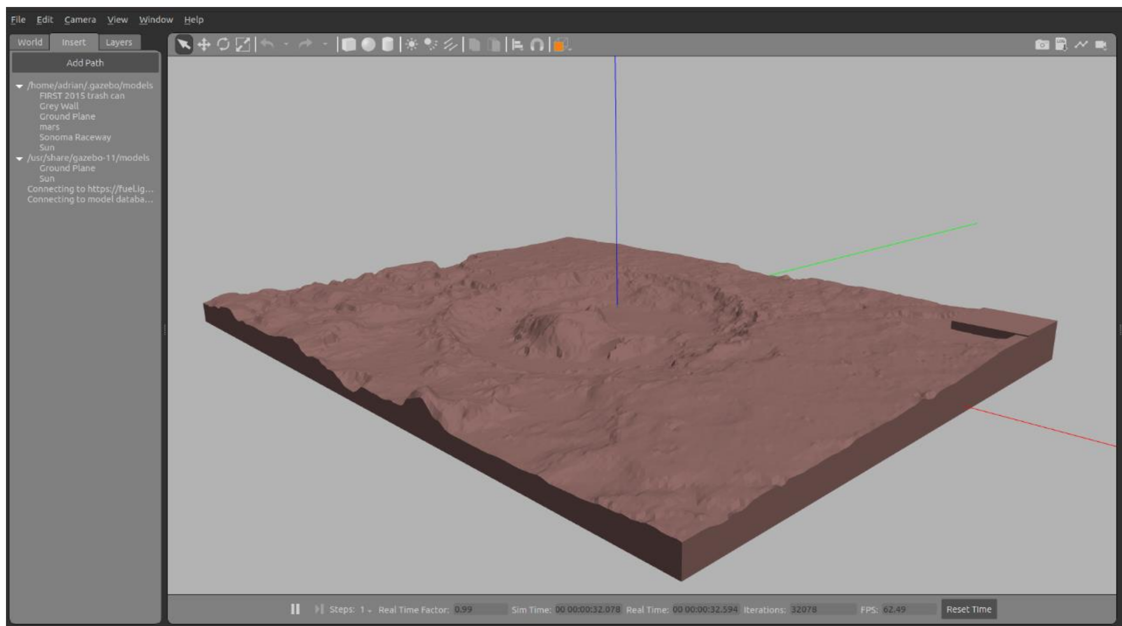


Figura 9: Simulación de Marte en Gazebo

3.1.2. HUSKY ROVER

En el caso del vehículo terrestre, se ha escogido el modelo Husky diseñado por Clearpath Robotics. Este robot es de tamaño medio e incluye la parte superior contiene una especie de base lo que hace muy sencillo el poder acoplar distintos accesorios. Está diseñado para incluir desde LIDARs, cámaras u otros sensores hasta GPSs o IMUs. Además, tiene una estructura muy compacta sin apenas partes móviles, lo que le convierte en un rover robusto y fiable. Otra propiedad por la que se ha elegido es por el tipo de ruedas, ya que ha sido diseñado para todo tipo de terrenos, especialmente para zonas rocosas. Aparte de estas facetas, se le añade que fue uno de los primeros UGVs en ser compatibles con ROS.

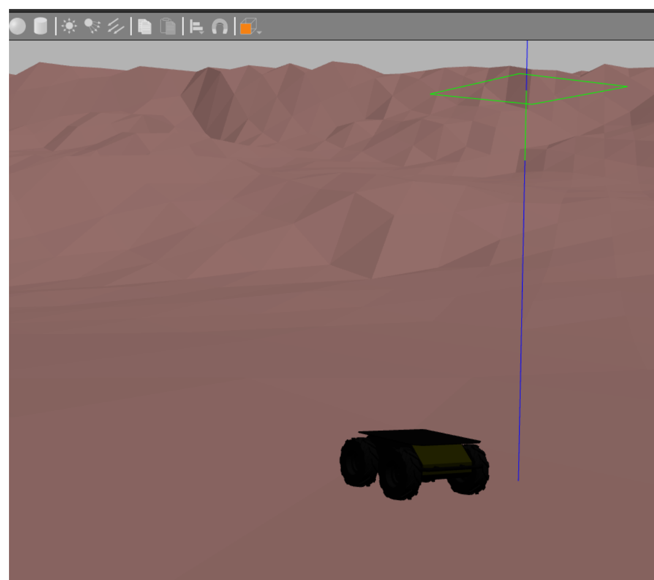


Figura 10: Husky simulado en Marte

3.1.2.1. DISEÑO

Debido a lo explicado anteriormente, Husky es la opción seleccionada para este proyecto, otorgando así la posibilidad de controlarlo a través de ROS2 y simularlo en Gazebo. Para obtenerlo, se clonará el repositorio Github mencionado en la bibliografía en nuestro ordenador. Dentro de todos los directorios que contiene el repositorio, solo se necesitará la carpeta llamada “husky_description”, ya que solo se quiere tener la parte estética. Esto es debido a que la parte de control será creada a través de un nodo. Dentro de la carpeta, se pueden encontrar dos directorios. Uno de ellos se llama “*meshes*”, el cual contiene las texturas y las formas del rover para hacerlo más realista. Además, a través de los archivos que definen los modelos, solo se pueden incluir formas geométricas simples, por lo que para formas irregulares como la de este vehículo, se tiene que descargar los *meshes.* El otro directorio, se llama “URDF”. En él se incluye los archivos *urdf.xacro* del robot, sus ruedas y sus accesorios. En el caso de este proyecto, no se necesita añadir ningún suplemento por lo que solo hay que editar el código suprimiendo las partes en las que se define la cámara o el arco para accesorios.

URDF

Anteriormente se ha mencionado los modelos SDF pero no los URDF. URDF es un tipo de archivo muy parecido al SDF, la diferencia es que los SDF están diseñados para funcionar en Gazebo, pero los URDF no, ya que son la manera que tiene ROS2 de definir sus robots. Sin embargo, se puede llegar a insertar si se utiliza una serie de etiquetas como por ejemplo <gazebo>, que sirve para poder incluir los elementos que tienen un propósito concreto que cumplir en la simulación. Esto permite que se puedan definir las partes de SDF que no están en URDF, como por ejemplo para incluir los *plugins* necesarios. Cabe añadir que también es necesario la inclusión de la etiqueta <inertia> en los *links* para realizar el movimiento. En el caso del Husky, se crea un *link* específico para declarar la “*inertia”* mencionada.

Por lo demás funciona bastante similar y se constituye principalmente de 3 partes:

- **Links:** Los *links* están diseñados para definir cada parte del robot. Como se aprecia en la Figura 12, son los elementos que no producen un movimiento por sí mismo, sino que representan una estructura fija. Su homólogo en el cuerpo humano serían los huesos que se encuentran entre las distintas articulaciones. Además, hay un link que se establece como punto de referencia normalmente llamado *base_link.*
- **Joints:** Los *joints* están diseñados para definir las uniones de cada uno de los *links.* Es una parte muy importante ya que hace que la suma de todos los links forme una sola estructura. Asimismo, tienen la función de mover los *links. S*e pueden definir de varios tipos como por ejemplo rotativos, como sería el ejemplo de los ejes de las ruedas de un rover o traslacionales. También se pueden poner en formato fijo para solo unir *links.* Su homólogo en el cuerpo humano serían las articulaciones.

Dentro de los estos elementos se utilizan diferentes parámetros, diferenciando dos partes principales, como se ha comentado en el apartado de Marte. Esas partes son la visual y la de colisión. En la visual se define la geometría de la estructura que está siendo definida al igual que se hace en la colisión para que interactúen con otros objetos de manera realista. También se puede configurar con <origin> la ubicación de cada uno y la orientación que tienen. A esto se le puede añadir otro tipo de etiquetas para mejorar el diseño como materiales, colores, texturas o las inercias para sus movimientos.

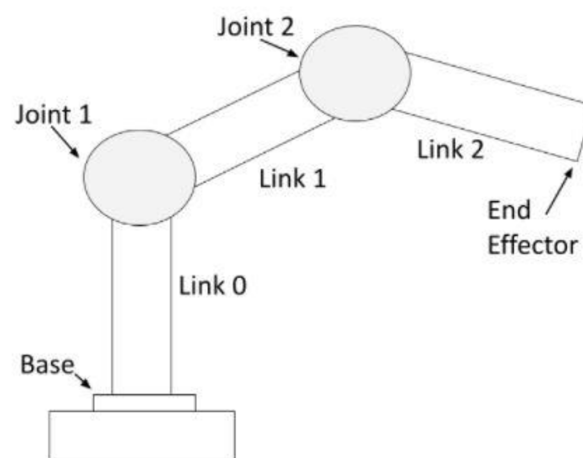


Figura 11: Dibujo representativo de links y joints

- **Plugins:** Estos sirven para poder controlar el movimiento del robot, así como recibir datos de salida de sensores, cámaras, GPSs u otros incluidos. Al ser específicos para su funcionamiento dentro de Gazebo, necesitarán la etiqueta comentada previamente de <gazebo>. Hay tres tipos de *plugins,* los de modelos, como el que será utilizado para controlar el Husky, los de sensores, para recoger la información que aportan, y los visuales.

XACRO

Cabe destacar, que el diseño del Husky utilizado tiene un dominio terminado en “xacro”. Esto es utilizado para agilizar el proceso de creación de los documentos, simplificando el código de manera notoria a través de funciones y variables. Otra característica es el uso de macros, que consiste en definir previamente un conjunto de características, para posteriormente poder mencionar el nombre de la macro sin necesidad de poner todas las etiquetas. Esto ayuda a la hora de escribir el código ya que se reduce en gran cantidad el número de líneas utilizadas y con ello, queda mucho menos cargado de contenido.

3.1.2.2. LANZAMIENTO EN GAZEBO

Para hacer aparecer al Husky en Gazebo, se usará una herramienta proporcionada por ROS2 llamada “ros2 launch”. Con ella, se puede lanzar un nodo definido previamente. Tanto el

archivo "launch" como el nodo, están implementados con el lenguaje de programación Python.

NODO

En este caso, se utilizará un servicio como método de comunicación con Gazebo. Este concepto fue previamente definido en su sección correspondiente. Para utilizar este servicio, se necesita importar la librería correspondiente, llamada "gazebo_msgs.srv". El resto del código se define en la función *main.* Como partes más relevantes, cabe destacar que el nodo debe contener siempre los siguientes comandos:

- rclpy.init(): Se encarga de inicializar la librería rclpy.
- node = rclpy.create_node("entity_spawner"): Se menciona el nombre del nodo y lo crea como ejecutable.
- rclpy.spin(node): Ejecuta el nodo creado.
- rclpy.shutdown(): Se encarga de cerrar el nodo una vez haya terminado.

Una vez ejecutado el nodo, es decir, después del comando "spin", se crea el cliente que se conecta al servidor con el comando:

client = node.create_client(SpawnEntity, "/spawn_entity")

Otras consideraciones a tener en cuenta son, que para poder introducir un documento definido con xacro, se necesita instalar e importar su correspondiente librería. Además, se tienen que utilizar los dos últimos comandos de la figura 13 para poder definir el llamado "path" del URDF del Husky, que se corresponde con la ruta del archivo.

```
24    # Get the file path for the robot model
25    file_path = os.path.join(
26        get_package_share_directory("mars_spawner"), "models",
27        "husky_description", "urdf", "husky.urdf.xacro")
28    robot_description_config=xacro.process_file(file_path)
29    robot_desc=robot_description_config.toxml()
30
```

Figura 12: Comandos para adquirir la ruta de un Xacro

LAUNCH

Como se ha comentado en el apartado anterior, se ha creado una conexión servidor-cliente donde el servidor, pertenece al propio simulador de Gazebo, mientras el cliente corresponde con el nodo. Por lo tanto, se ejecutará el nodo a través del archivo *launch* para así, solicitar el objetivo al servidor, el cual responderá haciendo aparecer el robot en el simulador. Para

configurar el código, se debe indicar la ruta de los archivos utilizados en el paquete aparte de la declaración del nodo (Figura 14).

```
31
32      #GAZEBO_MODEL_PATH has to be correctly set for Gazebo to be able to find the model
33      #spawn_entity = Node(package='gazebo_ros', node_executable='spawn_entity.py',
34      #                     arguments=['-entity', 'demo', 'x', 'y', 'z'],
35      #                     output='screen')
36      spawn_entity = Node(package='mars_spawner', executable='spawn_robot',
37                          arguments=['Husky', 'husky', '0', '-1', '7.210091'],
38                          output='screen')
```

Figura 13: Declaración del nodo en archivo launch.py

También se puede añadir argumentos y configurarlos en el nodo para ubicarlo en una posición concreta en el eje de coordenadas.

3.1.2.3. CONTROL

COMUNICACIÓN

Una vez entendido el funcionamiento del diseño y todos sus elementos, se comentará como se ha implementado la parte que controla el movimiento del rover. Para alcanzar dicho objetivo, se creará un nodo con ROS2 programado en C++.

En primer lugar, se debe añadir un *plugin* al URDF del Husky para poder controlar el giro de las ruedas como si de un motor se tratara. En un principio se añadió el controlador solo para un par de ruedas, pero al hacer la prueba, al girar el robot se quedaba estancado. Debido a ese problema, se modificaron dos características. Primero se actualizó el valor de la constante de rozamiento en la dirección perpendicular al robot, reduciéndola a 0.5, la mitad del valor inicial. El movimiento de rotación mejoró, pero seguía sin ser lo suficientemente efectivo. Dado que no es un vehículo omnidireccional, el giro sobre el eje Z no es tarea fácil. Para solventar el problema, se creó otro *plugin* idéntico al anterior, pero con el otro par de ruedas, otorgando así una tracción total a las 4 ruedas.

Al dar nombre al *plugin,* hay que incluir un apartado llamado “filename” en el cual llama a un fichero en la nube de gazebo que habilitará su uso. En el caso del proyecto, se utilizó el fichero “*libgazebo_ros_diff_drive.so.”.* Adicionalmente, se añade una serie de características para su correcto funcionamiento, como por ejemplo la distancia entre las ruedas derecha e izquierda, el diámetro de las mismas, o datos sobre su rendimiento. Sin embargo, la parte más importante es la declaración de los *topics*. En este caso, para poder controlar el movimiento del robot, se utilizarán 2 *topics* distintos. Uno será de salida, donde el nodo publicará la odometría del Husky hacia el nodo controlador desarrollado. Este se llama /husky/odom. El otro *topic* será de entrada, ya que recibirá las instrucciones desde el nodo controlador para ejecutar el movimiento. Se ha nombrado como /husky/cmd_vel.

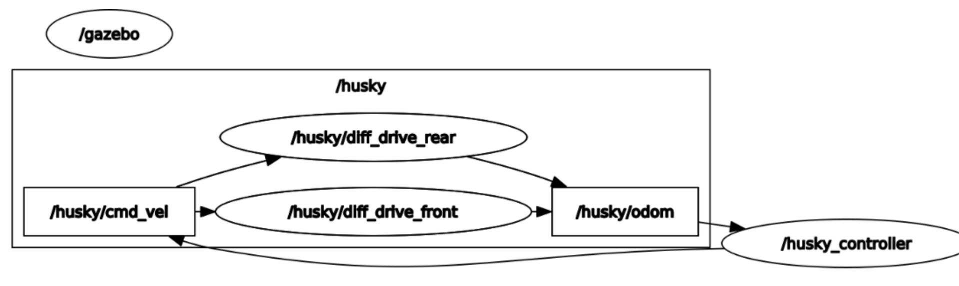


Figura 14: Esquema de comunicación entre nodos del Husky

Se puede ver el comportamiento del sistema a través del grafo de la Figura 15. Los nombres inscritos en las elipses corresponden con los nodos, tanto como el controlador, como los dos ejes del Husky. Los rectángulos son los *topics* previamente descritos.

PROGRAMACIÓN

Para poder llevar a cabo la conexión deseada, se debe implementar un código que haga que el nodo mande y reciba las instrucciones pertinentes. Todo ello se realizará en el archivo ubicado en el directorio “src”, llamado controller.cpp. En él, se creará una clase para poder ejecutar el algoritmo deseado.

Primero, se escribe en la función *main()* los comandos indispensables. Estos son la inicialización de la librería rclcpp, crear y ejecutar el nodo nombrando a la clase (llamada Husky_controller) desde el comando “spin”, y, por último, cerrar el nodo.

Después, se debe configurar la clase que es donde se ejecutará el programa principal. Para empezar, se crea la subscripción y la publicación a los *topics* correspondientes dentro del constructor. El subscriptor en concreto va ligado a una función llamada “odom_callback()”, la cual se activará cada vez que el nodo reciba un mensaje a través del *topic* de la odometría. Esta función desempeña el papel de guardar, en sus respectivas variables, el valor de la posición actual, tanto en el eje X como en el eje Y, y del valor de rotación respecto al eje Z conocido como *yaw*. Cabe destacar que, para obtener el valor de *yaw*, se tiene que llamar a otra función llamada “euler_from_quaternion()” ya que, para dar su orientación, el robot proporciona un grupo de 4 variables, por lo que se necesita transformar a un grupo de 3 para obtener el valor real.

Una vez obtenidos estos valores, son enviados a una función llamada “send_instructions()” la cual será la principal del programa. En ella, a través de la posición obtenida previamente, se ejecutará el movimiento del rover.

El movimiento del UGV se realizará con la ayuda de 2 tipos de movimientos, uno de rotación, y otro de translación, en otras palabras, el robot girará o avanzará en función de la orientación que tenga respecto a la meta asignada. El objetivo será trasmitido a través de las coordenadas X e Y, las cuales recibirá desde la interfaz de control que se explicará más adelante. Para definir las variables, se utilizará la herramienta aportada por Gazebo, perteneciente a

geometry_msgs, llamada Point(). Con ella, se puede acceder a los valores de la posición que se deseen, por lo que se guardará en la variable "goal.x" y "goal.y". Respecto a las órdenes de avanzar y rotar, se utilizará la herramienta de Twist(), guardańdola en la variable "inst". Por lo tanto, las instrucciones se enviarán a través de "inst.linear.x" e "inst.angular.z".

Dicho esto, lo primero que se necesita es girar el robot hasta alcanzar la orientación adecuada para, una vez que el robot esté mirando al objetivo, pueda avanzar hasta él. Para lograrlo, se tiene que dibujar un triángulo rectángulo entre la posición inicial y el objetivo final. En la Figura 16 se puede apreciar una representación de lo que se plantea. Después, solo se tiene que calcular los lados de dicho triángulo restándole al valor objetivo en el eje X menos el valor actual en ese mismo eje.

Para el otro cateto, se opera de la misma manera, pero con los valores del eje Y. Una vez que se obtienen los catetos, solo hay que hacer el arco tangente para obtener el ángulo que falta para estar orientado hacia la meta.

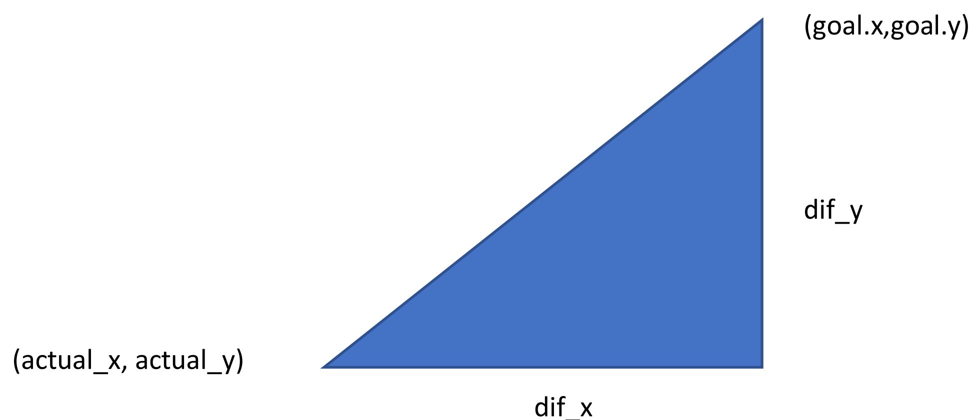


Figura 15: Triángulo formado por entre posición inicial y meta

Para entender mejor el funcionamiento del nodo, puede apreciarse un diagrama explicativo en el Anexo III.

3.1.3. IRIS UAV

Para el caso del dron que sobrevolará la zona, se ha escogido el vehículo llamado Iris. Es un tipo de UAV comúnmente conocido y desarrollado por 3D Robotics. Este modelo es un quadricóptero diseñado para funcionar de forma autónoma o a través de radio control. Además, se le pueden añadir una serie de accesorios lo cual le otorga una utilidad bastante práctica para el sector. En este caso, se utilizará dicho dron con un LIDAR incorporado apuntando hacia el suelo para calcular las distancias, como se muestra en la Figura 16.

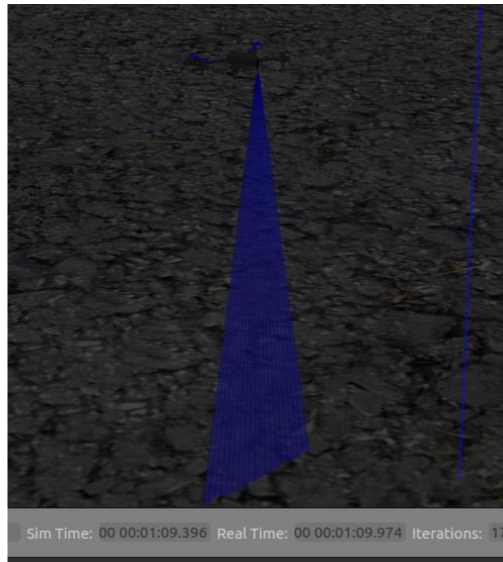


Figura 16: Dron Iris simulado en Gazebo

3.1.3.1. DISEÑO

Para la obtención de los diseños, se consiguieron a través del repositorio de PX4 que se tiene que instalar para su uso. Están escritos directamente en formato SDF para poder simularlos en Gazebo. En concreto, se obtuvo un modelo llamado "Iris_rplidar" el cual contenía un LIDAR y un RPLIDAR, uno en vertical y otro en horizontal respectivamente, y una cámara. Por lo tanto, se modificó el documento para que solo contuviese el LIDAR vertical.

Además, dentro de las especificaciones del sensor de distancia, se modificó el archivo para cambiar el ángulo de visión del propio sensor, y para añadirle un *plugin* cuya función consiste en publicar los datos al nodo controlador.

En cuanto a la parte del código que se ha modificado, corresponde con la Figura 18. Como al definir su posición ya se ha orientado hacia abajo, se pone la dirección de los rayos en posición horizontal. En cuanto a la amplitud, se desea tener un ángulo de 20 grados, por lo que se usará una resolución de 20 muestras para así tener una muestra por grado. Dada su orientación, el ángulo del rayo perpendicular al suelo estaría en 180 grados. Por ello, los valores de mínimo y máximo tendrán que ser de 180-10 grados y 180+10 grados. Para terminar, se debe poner el resultado en radianes.

```
31        <ray>
32          <scan>
33            <horizontal>
34              <samples>20</samples>
35              <resolution>1</resolution>
36              <min_angle>2.967</min_angle> <!--pi-10º-
37              <max_angle>3.316</max_angle> <!--pi+10º-
38            </horizontal>
39          </scan>
```

Figura 17: Ajustes de la amplitud del LIDAR

3.1.3.2. CONTROL

COMUNICACIÓN

Como se mencionó en el apartado anterior, también se debe añadir un *plugin* para que, al igual que el Husky, pueda trasmitir datos a través de un *topic* conectado al nodo controlador. En este caso, se usa la librería “libgazebo_ros_ray_sensor.so”, y se configura para que publique la información con el tipo de mensaje “sensor_msgs” en el *topic* “scan”. Adicionalmente, hay que configurar un parámetro llamado “qos” y forzarlo a que sea de tipo “reliable”. Esto es debido a que, al subscribirse a este *topic*, el tipo de “qos” que tiene la subscripción es de “reliable” pero, si en el publicador no se configura nada, por defecto es de tipo “best effort” lo cual es incompatible con el anterior. Por ello, se debe poner ambos al mismo nivel.

```
52        <plugin name="scan" filename="libgazebo_ros_ray_sensor.so"
53          <ros>
54            <remapping>~/out:=scan</remapping>
55            <qos>
56              <topic name="/scan">
57                <publisher>
58                  <reliability>reliable</reliability>
59                </publisher>
60              </topic>
61            </qos>
62          </ros>
63          <output_type>sensor_msgs/LaserScan</output_type>
64          <frame_name>link</frame_name>
65        </plugin>
```

Figura 18: Ajuste del QoS del LIDAR

Respecto al nodo del controlador, también se tiene que crear una subscripción a un nodo llamado “tymesync_node”. Esto es debido a que se va a controlar un nodo de PX4 perteneciente al dron, con un nodo de una aplicación externa, en este caso ROS2. En

consecuencia, ambos nodos deberán estar sincronizados para su correcto funcionamiento. Por lo tanto, se crea la subscripción al *topic* que los comunica llamado "/fmu/timesync/out".

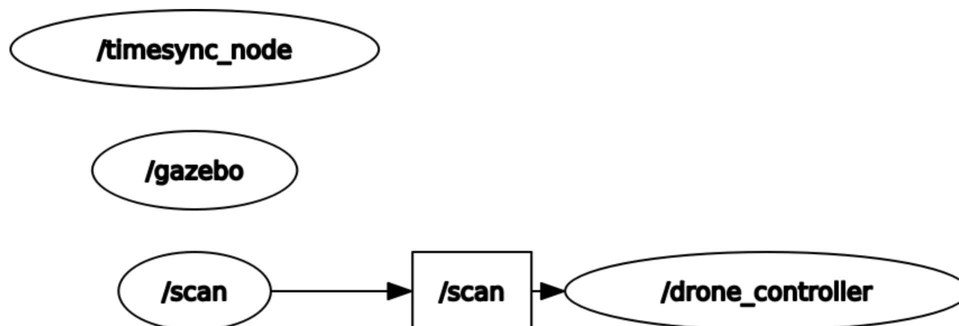


Figura 19: Diagrama de nodos y topics de UAV y LIDAR

Como previamente se ha comentado, para poder controlar el dron a través de ROS2, utilizando publicadores y subscriptores, se tiene que hacer de una manera específica. Para ello, se debe realizar una instalación previa que garantice la conexión entre ROS2 y PX4. Para conseguir dicha finalidad, se utilizará un DDS/RTPS.

DDS (*Data Distribution Service)*, es un *middleware* utilizado en sistemas de tiempo real del tipo "publish/subscribe". En el caso del trabajo desarrollado, se utilizará el "eProsima Fast RTPS". Para usarlo con PX4 se le llama "microRTPS bridge". Su función es operar como puente entre ROS2 y PX4, pudiendo así comunicarse entre nodos sin ningún problema a tiempo real. Para su correcto funcionamiento, se debe crear tanto un agente como un cliente. Uno se encontrará ubicado en PX4 (cliente), y otro se enlazará a ROS2 (agente). Tras conectarse ente ambos, se habilita la transmisión de mensajes de forma bidireccional. Esto permite que se pueda subscribir y publicar datos en los *topics* de PX4 (uORB topics) como si del mismo *software* se tratara. En la Figura 21 se puede apreciar un esquema del funcionamiento completo.

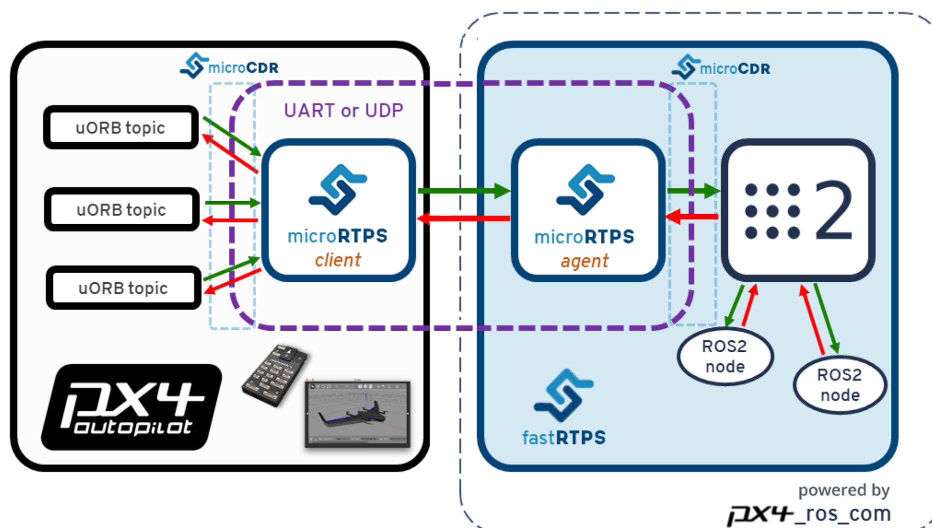


Figura 20: Esquema de conexión entre ROS2 y PX4

Para que todo lo mencionado funcione, se tiene que instalar dos paquetes que proporciona PX4 con el fin de poder crear los nodos dentro del mismo y poder así compilarlos juntos. Los paquetes son:

- **Px4_msgs:** Contiene la definición de los menajes PX4-ROS2. Al compilar el paquete, se crean las compatibilidades necesarias para ROS2 y para crear el agente del microRTPS.
- **Px4_ros_com:** Contiene las plantillas del agente para crear los publicadores y subscriptores. En este paquete es donde se escriben los nodos para compilarlos y ejecutarlos.

Una vez instalado y configurado el puente, ya se puede crear los publicadores y subscriptores desde ROS2. Para el adecuado control del movimiento del dron, se necesitará crear publicadores en el *topic* "fmu/offboard_control_mode/in", que sirve para elegir que parte se habilita para el control del movimiento (ya sea controlar la velocidad, la posición...). Luego está "fmu/trajectory_setpoint/in", para indicar las metas a alcanzar a la hora de automatizar el movimiento. Por último, "fmu/vehicle_command/in" para poder enviar comandos como, por ejemplo, armar el dron para que se prepare para poder despegar, o cambiar el modo de vuelo para darle el control a ROS2. Una condición a cumplir es que, siempre que se quiera mandar una instrucción al dron por el *topic* “setpoint”, tiene que ir siempre emparejado con el de *offboard_control_mode* para poder activarse de forma correcta.

Además de los subscriptores definidos previamente, también se debe crear otro para leer la posición del GPS del dron dentro del simulador. Este se conecta al *topic* “fmu/vehicle_odometry/out”.

PROGRAMACIÓN

El código del dron sigue una estructura similar a la del Husky. El nodo principal se ejecutará en una clase creada llamada “DroneController”. Para crearlo, hay que escribir los comandos ya mencionados en los anteriores nodos dentro de la función *main()*.

Una vez iniciada la clase, se encuentran las funciones asociadas a sus respectivos *topics* que juegan el papel llamado “callback”, donde se activarán cuando reciban un mensaje. El *callback* asociado al GPS, tiene como misión la adquisición de la posición del dron dentro de Gazebo, para así irse actualizando de forma constante, guardándola en unas variables para cuando se necesite.

Un detalle a tener en cuenta es que la posición se publica en el sistema de coordenadas NED, por lo que para pasarlo al sistema tradicional que usa Gazebo y ROS2 (ENU), se tendrá que intercambiar el valor de “x” con el valor de “y”. Adicionalmente, la posición en el eje Z se tendría que cambiar de signo. Además, hay que ser conscientes de que, para PX4, el origen de coordenadas está en el punto (1,1,0), ya que es el lugar donde aparece al lanzar la simulación. Para compensarlo, se tendrá que sumar 1 a los valores obtenidos en “x” y en “y”.

El otro *callback* corresponde al LIDAR. En esta función, será donde se obtengan los datos de las distancias, y se pasen a medidas de altura. Se ha decidido barrer 2,5m de terreno a la vez. Esto es posible gracias a la amplitud previamente definida en el LIDAR, donde se estableció una apertura de 20 grados. Al tener los datos de la distancia en el suelo a barrer y del ángulo proporcionado, se puede formar un triángulo isósceles donde, al dividirlo a la mitad, se obtienen dos triángulos rectángulos. Esto permite aplicar las fórmulas trigonométricas. De esta manera, se puede extraer el valor de la altura necesaria a la que tiene que estar el dron para abarcar la longitud mencionada. Al tener el cateto opuesto y el ángulo, se obtiene la altura con:

$$h = \frac{1.25}{tg(10)} = 7.089$$

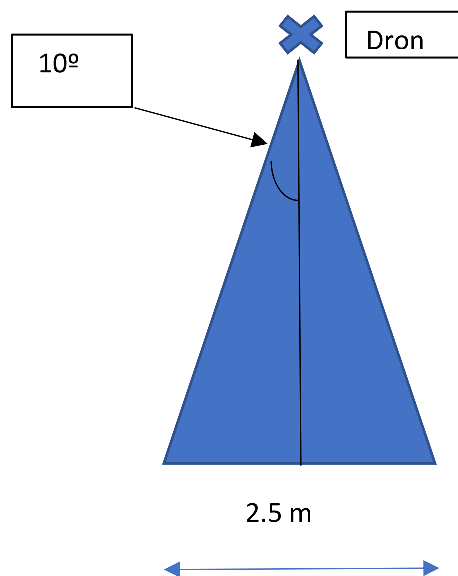


Figura 21: Representación del escáner del LIDAR en la altura

Una vez se ha calculado la altura adecuada, hay que tener en cuenta las medidas que se obtienen. Como previamente se estableció una resolución de 20 muestras, una por cada grado, se puede deducir que cada 0.5m se obtendrán 4 medidas. En consecuencia, con la ayuda de bucles, se irá recogiendo los valores de 4 en 4 para calcular la media y, posteriormente, los datos se almacenarán en una variable. Hay que tomar en consideración que todos los rayos parten del mismo punto, por lo que la distancia medida en los extremos no es la real que habría en esa posición. Sin embargo, si se calcula por trigonometría el valor real, difiere del obtenido en poco menos de 0.1m. Como el sistema está diseñado para diferenciar alturas mayores, este margen de error será despreciado.

Después de haber obtenido los valores para cada 0.5 metros, se realiza la conversión a la altitud. Hay que contar con que se está midiendo la distancia al suelo, pero lo que realmente se quiere es la altura, por lo que, a menos distancia, se tendrá una mayor altura. Por este motivo, se ha decidido poner nuestro sistema de referencia en la altura 0 según el eje de coordenadas del propio Gazebo. Así todos los valores serán positivos. Para la conversión, se

guarda la altura al inicio, es decir, antes de despegar, y se le sumará el resultado de la diferencia entre la altura del dron y el valor obtenido en el LIDAR.

Otra sección del programa es un *timer* que opera cada 200ms. Su misión consiste en activar la función que publica la posición a seguir, así como recoger los datos de la altura y la posición del GPS. Primero, se debe ejecutar el despegue, es decir, coger la altura que se ha establecido con los cálculos, además de colocarse en la posición correcta de inicio, que es (0.1,25). Esta posición se debe a que los rayos abarcan una superficie de 2,5 metros por lo que el dron se adaptará a la mitad. A partir de este momento, se activará el *flag* de avance por lo que cada 0.2 segundos, el dron avanzará 0.5 metros a lo largo del eje X y recogerá la información de la altura y la posición. De esta manera, se obtendrá un valor de altura para cada 0,5 m^2.

Una vez que el dron llegue a una posición prestablecida en el eje X, se moverá 2,5 metros en el eje Y, y volverá hacia un valor de x igual a 0 para barrer una nueva zona. De este modo actuará continuamente hasta que consiga tomar medidas de toda la superficie.

Para un mejor entendimiento del funcionamiento del nodo controlador del dron, se ha introducido un diagrama en el Anexo IV.

3.2. INTERFAZ DE CONTROL

Se ha comentado en los apartados anteriores, toda la información respecto a los robots utilizados y su funcionamiento. Sin embargo, hay otra parte del proyecto que es esencial para la realización del mismo. Se trata de la interfaz de control. Como se explicó al principio de la memoria, el proyecto está pensado para simular una situación de exploración en otro planeta como es Marte. Aunque los robots tengan su propia autonomía de movimiento, se necesita un controlador en Tierra que pueda comunicarse con ellos, ya sea para recoger información o para entregarla. En este caso, su función no se reducirá únicamente a la comunicación, si no que servirá como interfaz gráfica para la persona que controle la misión.

Esta interfaz está programada en Java y proporciona un aspecto visual simple de entender. Al iniciarla, a la izquierda fabrica un mapa de 8x8 con unos datos aleatorios ya que, a cada coordenada, le corresponde un valor de altura. En este caso, cuanto más alto sea el valor de la altura, más oscura estará representada la celda. Además, hay que mencionar que la interfaz ya estaba creada y solo ha sido necesario adaptar ciertos componentes para la comunicación con los robots como se comentará más adelante.

Para situar la explicación, se tiene como referencia la Figura 23. En la parte derecha, se puede observar una serie de configuraciones. Primero, se puede observar una casilla para seleccionar el número del puerto del robot a conectar y su dirección IP. Esta configuración se comentará con más profundidad en el siguiente apartado. Debajo de esta casilla, se encuentran unos huecos que se pueden rellenar para la configuración. Se incluye las coordenadas tanto iniciales como finales, y la máxima inclinación admitida. Para establecer

las posiciones de partida y las de llegada, también se puede hacer a través del ratón, haciendo *click* izquierdo para la inicial, y *click* derecho para la final.

En la parte inferior, una vez se han establecido las coordenadas, se puede pulsar el botón "SEARCH ROUTE" para así activar el algoritmo 3Dana y posteriormente, enviarlo al robot al pulsar "EXECUTE ROUTE". También existe la opción de mandar un comando directo de movimiento al rover a través de las casillas "Rotate" y "Move forward".

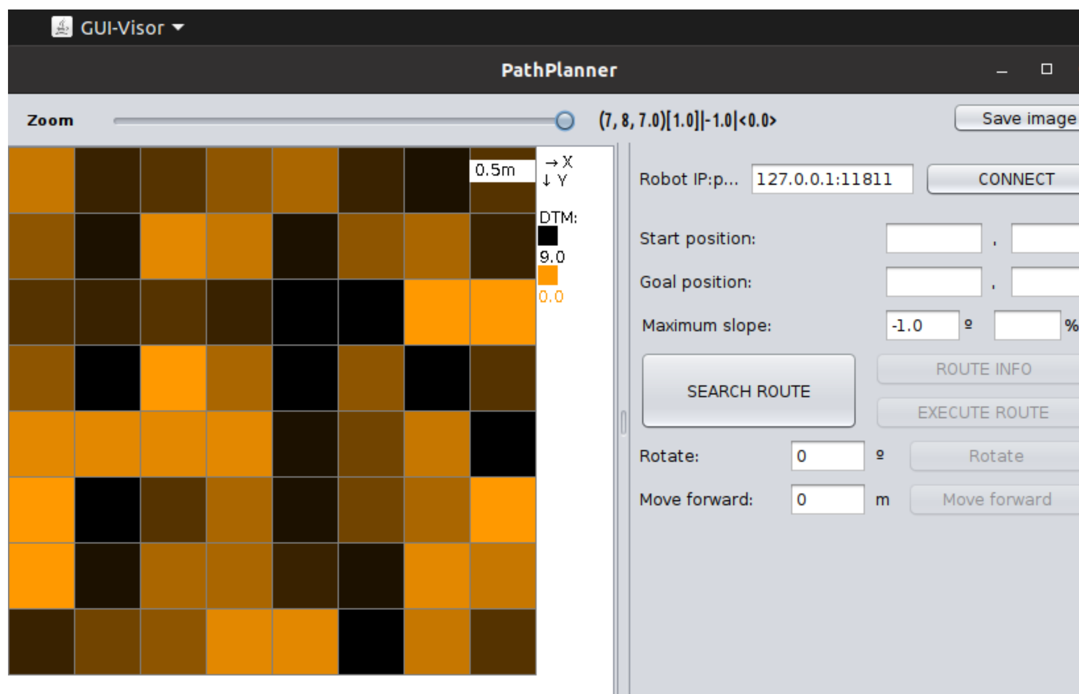


Figura 22: Aspecto visual de la interfaz gráfica

3Dana es un algoritmo encargado de establecer el camino más eficiente entre el punto de partida y el de llegada. Para ello, analizará las casillas a su alrededor para poder elegir en función de una pendiente máxima, que celdas debe recorrer el robot. Con ello, evitará obstáculos o desniveles que puedan ser peligrosos para el robot o que precisen de un uso mayor de energía y tiempo. Este algoritmo es del tipo *path planning* lo cual se comentó en el estado del arte. Para encontrar el camino, hace una búsqueda heurística sobre mapas como el DTM utilizado.

Además de tener en cuenta la pendiente y el coste de cada celda, también tiene en cuenta las posibles variaciones en la orientación del robot, pudiendo así evitar ciertos caminos en los que se necesite unas variaciones excesivas consiguiendo rutas más seguras y eficientes energéticamente.

3.3. INTEGRACIÓN

En este apartado, se hablará sobre la implementación del trabajo en su conjunto, es decir, se analizará como se ha conseguido integrar los elementos mencionados (Iris, Husky e interfaz) para su correcta comunicación y funcionamiento.

3.3.1. CONEXIÓN DRON-ROVER

Como se ha explicado en el apartado del UAV, es el dron el que se encarga de sobrevolar una zona para posteriormente analizar el relieve del terreno y poder formar el mapa DTM. Aunque la interfaz espere el envío de estos datos, en vez de trasmitirlos directamente, se transmitirán al UGV. Esto es debido a que el UGV va a tener que estar conectado igualmente a la interfaz ya que es desde esta desde donde recibirá las coordenadas para su movimiento. Aparte de esto, se sabe que la distancia interplanetaria es muy grande por lo que la comunicación tarda en ejecutarse. Por lo tanto, es más eficiente tener un solo robot conectado y suministrando toda la información.

Para que ambos vehículos se comuniquen, se aprovechará que ambos se controlan a través de nodos programados en ROS2. Por ello, la transferencia de datos se basará en el tipo de comunicación más simple descrito en la parte teórica del *software*. Esta se corresponde con los *topics*. De acuerdo con lo mencionado, se creará un publicador en el nodo del dron con un *topic* llamado “map_data”. Los datos serán enviados al obtener los valores de posición y altura, justo antes de avanzar los metros correspondientes. El publicador será de tipo *String* para mandar los 3 valores juntos en una sola cadena de texto.

Como al enviar los datos de uno en uno la interfaz no recibía la totalidad de la información, se decidió mandar los datos de 5 en 5, correspondiente a las 5 casillas de 0.5 metros cada una, ya que eran analizadas con el LIDAR al mismo tiempo. De esa manera, después de que el rover enviase los datos, recibía todos a la perfección.

En cuanto al Husky, se tiene que crear un subscriptor enlazado al *topic* “map_data”. A su vez, esta subscripción debe inicializar un *callback* que se activará cada vez que el dron publique los mensajes. En esta función, se almacena la cadena en una variable y se envía a la interfaz de control.

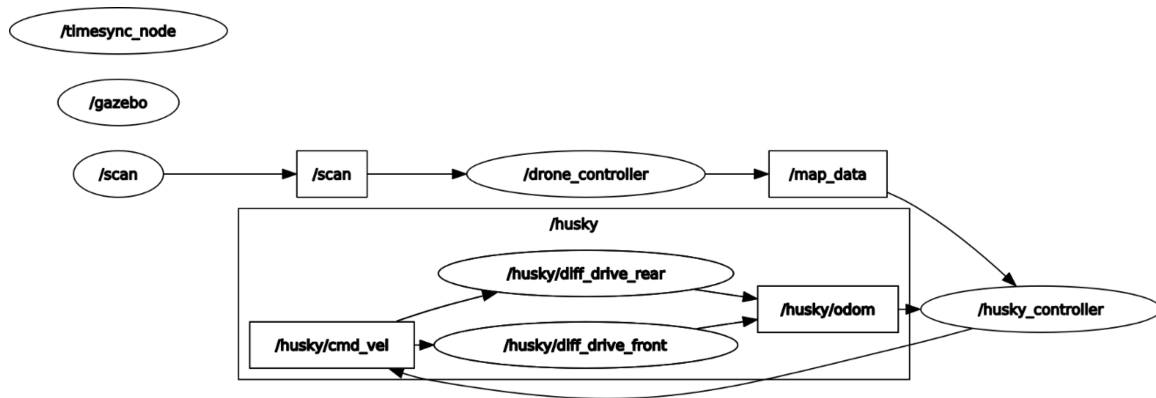


Figura 23: Diagrama de conexión ROVER-DRON

3.3.2. CONEXIÓN ROVER-INTERFAZ

La conexión entre la interfaz de control y el UGV es la fuente principal de intercambio de datos. Esto es debido a que es el único robot conectado directamente con dicha interfaz, por lo que no solo se encargará de recibir los datos del movimiento que tiene que ejecutar, sino que tiene que enviar tanto los datos de su posición como los obtenidos por el dron, que son los que proporcionan la información para realizar el DTM. Esto sigue la dinámica del rover marciano Perseverance con el dron Ingenuity: el dron se comunica a través de enlaces de baja energía y alcance con el rover, y es este el que se comunica con antenas de alta ganancia con las estaciones de tierra o con los orbitadores en Marte que actúan como estaciones de retransmisión.

Para efectuarla, se recurrirá al tipo de conexión TCP/IP. Este tipo de conexión es un protocolo de control de transmisión de datos como UDP o SCTP. Su objetivo es lograr crear diferentes conexiones dentro de una red de datos. Una de sus características es la transmisión ordenada y fiable de los datos, ya que la información llega en el mismo orden en el que se enviaron originalmente. Además, se pueden utilizar una serie de puertos para diferenciar distintas conexiones. En el caso del proyecto, se usa el puerto 11811 para conectar con el rover de ROS2.

Este tipo de conexión es muy utilizado ya que es la base de la comunicación vía Internet, como por ejemplo a través de los navegadores. Además, tiene una dirección IP asociada por la que cada red se identifica. En el caso del uso de ROS2, se utiliza la IP 127.0.0.1. Por lo tanto, la conexión de ambos robots se define por el número que se visualiza en la Figura 23 que es 127.0.0.1:11811.

Para completar la conexión, se necesita de un cliente y un servidor. En este caso, el cliente será la interfaz gráfica y el servidor se encontrará en el nodo del Husky. Una vez creados ambos elementos, se necesita crear otro tipo de archivo más. Estos archivos se llaman *sockets*

y están encargados de habilitar la comunicación cuando se produce la conexión, permitiendo así la transmisión de datos.

3.3.2.1. HUSKY

Para implementar este tipo de conexión, se tiene que incluir un servidor en el nodo del Husky. Para ello, se utilizará la posibilidad de recibir argumentos desde el *main().* De esta manera, el usuario pueda introducir desde el terminal que puerto desea utilizar. Adicionalmente, se le añade los archivos proporcionados con el *socket* del servidor, así como con el archivo "RobotServer", el cual está incluido con la interfaz de control.

Una vez preparados los respectivos archivos, solo se tendrá que introducir en el código la parte que recibe y envía los datos. Para la recepción de los mensajes, se utiliza el comando "server.receive(message)", además de la herramienta "strstr" de c++ para poder comparar la información que contiene, y así poder decidir si se actúa de una manera u de otra a través de bucles *if().* Además, se separan los mensajes en distintos *tokens* con el comando "strtok", el cual divide las palabras utilizando un carácter concreto como separador. Para su correcto funcionamiento, se debe estipular un patrón de mensaje, en este caso, se escribirá una palabra que describa la acción que se quiere ejecutar, seguido de ":". A partir de ahí, se introduce el contenido separado por una coma.

De este modo, si se recibe la palabra "ROTATE" se ejecutará una rotación con los grados especificados, la palabra "FORWARD", hará que el robot avance los metros deseados. Si se recibe la instrucción "Finish", significará que el dron ha terminado de barrer el terreno y el mapa ha sido creado. Esto activará un *flag* que desbloqueará la posibilidad de que el rover se mueva además de empezar a enviar su posición.

Otro posible mensaje es el de "client", el cual se envía nada más hacer la conexión servidor-cliente. Este apartado sirve para poder mandar un mensaje a la interfaz del tamaño del mapa escogido. Este tamaño se especifica a través del terminal al igual que el puerto, almacenándose en una variable. Al recibir la palabra "cliente", el valor de la variable es enviado a la interfaz.

Por último, está el mensaje "GOAL" el cual va seguido de las coordenadas pertenecientes a la ruta escogida por el algoritmo 3Dana. Como no interesa recibir todas las coordenadas al mismo tiempo, se creará el código adecuado para que solo se reciba una nueva cuando se llega al objetivo. Esto se consigue enviando un mensaje a la interfaz que empieza con la frase "You have reached the goal", seguido de las coordenadas a las que ha llegado. Cuando la interfaz recibe el mensaje, inmediatamente envía la siguiente posición.

Para realizar el envío de mensajes, se utiliza el comando "server.send(message)", pero antes, hay que utilizar la herramienta "sprintf" para asignar el contenido. Aparte de lo mencionado, también se tiene que enviar la información aportada por el dron. Este mensaje se envía nada más recibir los datos por el *topic* comentado, empezando por la palabra "Measure" seguido de una coma y los valores de las 5 casillas escaneadas.

Por último, se debe enviar la posición del Rover cada segundo. De esta manera, se actualizará en la interfaz visualizándose a través de un triángulo azul en el mapa. Para lograr dicho objetivo, se creará un hilo que se ejecute de forma paralela para enviar el mensaje. Se enviará la palabra "POS", seguido de las coordenadas actuales. Para hacer que se ejecute cada segundo, se usa el comando "sleep(1)".

3.3.2.2. INTERFAZ DE CONTROL

En la interfaz gráfica, se utiliza un método parecido al del Husky, pero en Java. El método consiste en dividir el mensaje recibido en tokens separados por una coma, ya que todos los mensajes enviados desde ROS2 se han configurado con una coma como separación. La gestión de la recepción de los mensajes se ejecuta en la función "handleReceivedData()". Después, la primera palabra o frase que precede a la coma, se guardará en una variable llamada "compare". Posteriormente se utiliza la función "compare.equals" para comparar las cadenas de texto y clasificar en bucles distintos según el mensaje recibido.

Si se recibe la cadena "New map of", se procederá a cambiar el mapa inicial. Esto es debido a que, al abrir la interfaz, se abre un mapa aleatorio como ejemplo, por lo que, al conectar con el servidor, se actualiza el mapa con las dimensiones que contenga el mensaje. El código que pinta un nuevo mapa se puede apreciar en la Figura 25.

```
if(compare.equals("New map of"))  //todo
{
    int map_size_x = Integer.parseInt(strtok.nextToken());
    int map_size_y = Integer.parseInt(strtok.nextToken());
    map = new Map(map_size_x, map_size_y, 0, 1, false, true);
    parent.lienzo.changeMap(map);
    parent.lienzo.visu = Lienzo.ALTITUDE;
    parent.lienzo.repaint();
}
```

Figura 24: Código para crear un mapa nuevo en la interfaz

Otro mensaje que se puede recibir es la posición. Tras recibir el mensaje "POS", solo se tiene que adquirir los valores de "x" y de "y" para mandarlos a actualizar a través de la función "updateRobotPosition(x,y);". Esto actualizará la posición del triángulo azul visible en el mapa.

Si el mensaje recibido corresponde con la palabra "Measure", entonces se debe seleccionar los valores "x", "y" y "z" de cada una de las cinco casillas y posteriormente actualizar el valor en el mapa. Esto se hace con el comando "map.get_node(x_map1, y_map1).setZ(z_map1);", donde el número al final de las variables cambia entre 1 y 5. Tras la realización de este bucle, se verá en el mapa como cambian de color las casillas correspondientes.

Como se mencionó en el apartado del Husky, se recibirá un mensaje donde se informará de que se ha llegado a la meta. Al recibir ese mensaje, se activa un *flag* para que en la función "buttonExecuteActionPerformed()", se pueda ejecutar el envío de la siguiente coordenada.

Por último, si el dron ha terminado de escanear la totalidad del terreno, se envía un mensaje de “finish” al rover, el cual lo envía directamente a la interfaz. Si la interfaz lo recibe, entonces envía un aviso al rover la cual servirá para activar el *flag* de movimiento del UGV previamente explicado. Además, actualiza el mapa para que se cree la escala adecuada de colores, siendo los más oscuros los valores de mayor altura y los más claros los de menor. Esto se hace al final ya que requiere de un cálculo matemático que es mejor realizarlo una vez.

4. EXPERIMENTACIÓN

En este apartado, se procederá a la prueba experimental de lo desarrollado anteriormente, para así comprobar su correcto funcionamiento. Dichas pruebas serán realizadas en entornos distintos para comprobar cómo se crean distintos mapas ajustándose al relieve escaneado. Además, se ejecutará la ruta establecida por 3Dana para verificar como el robot la sigue y evita las zonas menos seguras, ya sea porque contienen obstáculos o una gran pendiente. Primero, se realizará sobre un mundo plano poniendo obstáculos para una mejor visualización de cómo funciona el conjunto del proyecto. Más tarde, se ejecutará en un entorno simulado de Marte.

4.1. EXPERIMENTO 1

En este primer experimento, se utilizará un mundo con una superficie plana para que se pueda visualizar mejor el funcionamiento de los robots en su conjunto. Para que el dron detecte algo de altura, se introducirán unos obstáculos en forma de cubos para así ver como el mapa creado por el dron se asemeja al simulado.

Como con todas las simulaciones que se ejecuten con el dron, primero hay que preparar las configuraciones correspondientes para su funcionamiento. Para empezar, se necesitará lanzar la aplicación de control que se conectará al dron. Esta se llama “QGroundControl” y se utiliza para conectarse por GPS al dron en la vida real. Para ello, utiliza una ubicación real para poder controlarlo desde la misma aplicación. Como en este caso se está en un entorno simulado en otro planeta, no se necesita una ubicación exacta, solo que esté activa para realizar la conexión. Por ello, aparecerá una localización aleatoria en pantalla.

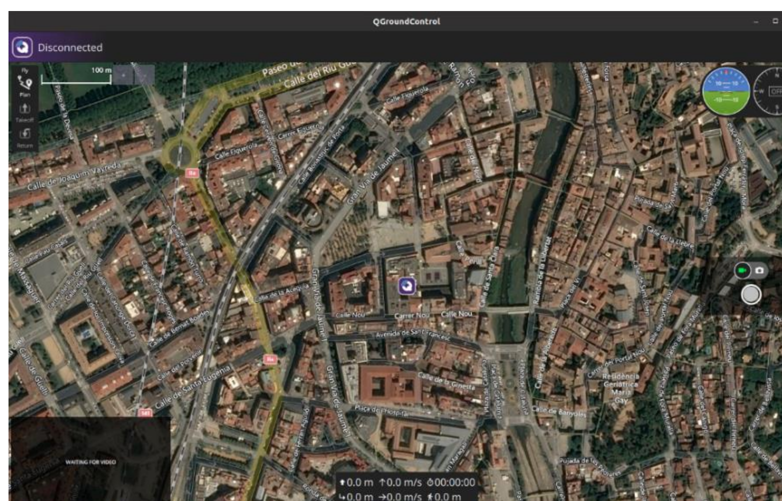


Figura 25: Imagen de la estación de control

Una vez lanzada la unidad de control, se abre un nuevo terminal y, a través del comando “make” en la raíz del directorio de PX4, se ejecuta gazebo junto al dron en un mundo por defecto totalmente plano.

Después se crea el agente del puente entre ROS2 y PX4, se lanza el archivo que hace aparecer al Husky y tras ello ya se tendría preparado a los robots. A continuación, se procede a colocar unos cubos predeterminados de Gazebo en medio de la zona a escanear. Para los experimentos esta zona será de 10 m^2.

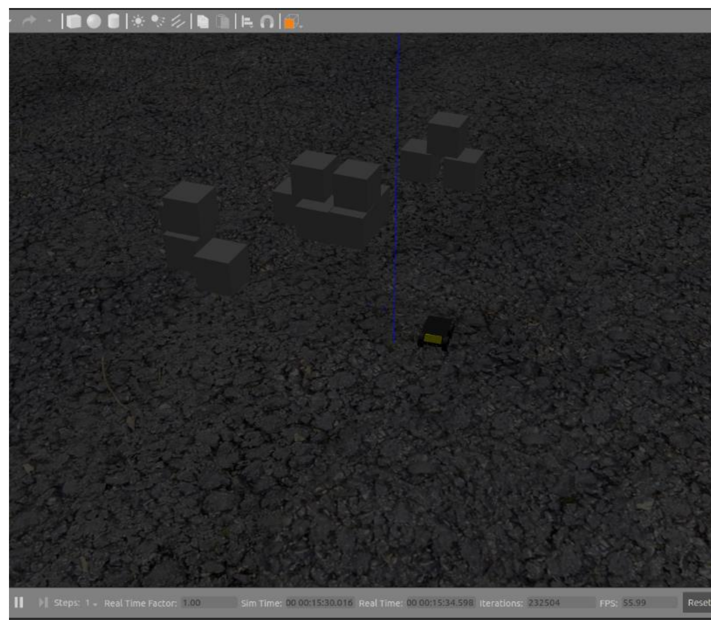


Figura 26: Escenario del experimento 1

A continuación, se debe abrir el servidor en un puerto concreto, en este caso en el puerto 11811. Será el mismo puerto para todos los experimentos. Una vez abierto, se procede a conectar el cliente a través de la interfaz de control. Cuando se conecte, tras pasar un segundo, el mapa se actualizará a un lienzo nuevo uniforme, borrando el aleatorio que se crea al inicio por defecto.

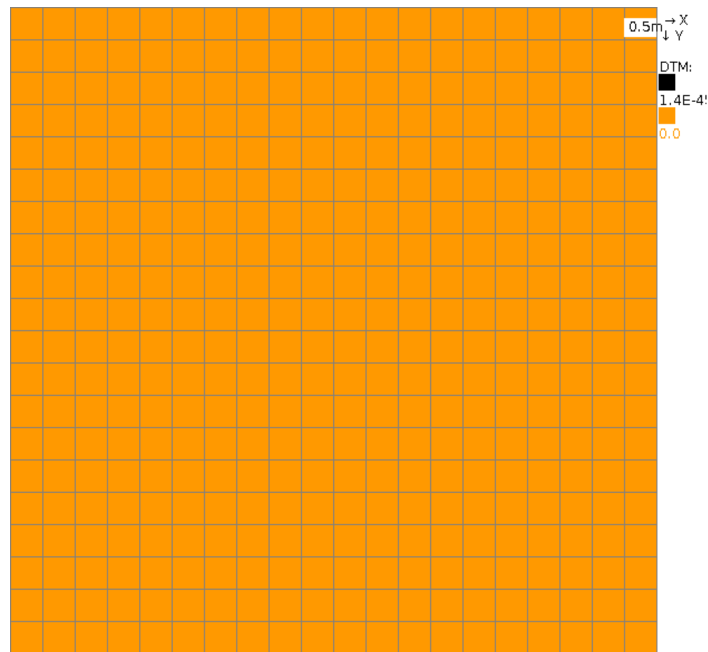


Figura 27: Mapa de la interfaz al ser creada tras la conexión

Una vez esté todo preparado, se activa en un terminal el nodo del dron para que proceda a ejecutar su código. Primero, cogerá la altura previamente definida para poder escanear la zona. Una vez que obtenga dicha altura, comenzará a avanzar y tomará los datos correspondientes.

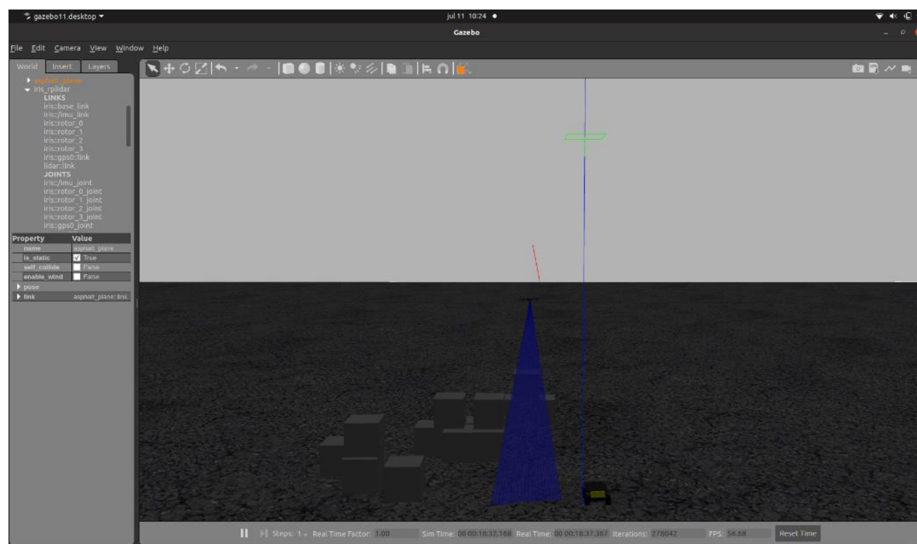


Figura 28: Dron después de alcanzar la altura de despegue

Según avanza, se puede apreciar en la Figura 30 como en la interfaz van desapareciendo los cuadrados marrones y se sustituyen por otros negros o amarillos. Esto es debido a que el cálculo de las celdas se aplica al final del barrido y no mientras el dron está en vuelo.

Cabe destacar que algunas casillas se ven en color amarillo. Esto significa que tienen un valor negativo de altura. Esto se debe a que, al avanzar el dron, se inclina un poco hacia delante, por lo que la distancia medida es ligeramente mayor. Como la superficie está a una altura de 0.1m, a veces saca una medida 0.1 metros mayor de la real, por lo que sale un valor negativo. Igualmente funciona perfectamente al pintar el mapa final. Además, es algo que no ocurrirá cuando se experimente en Marte ya que todas las alturas estarán por encima de 0.

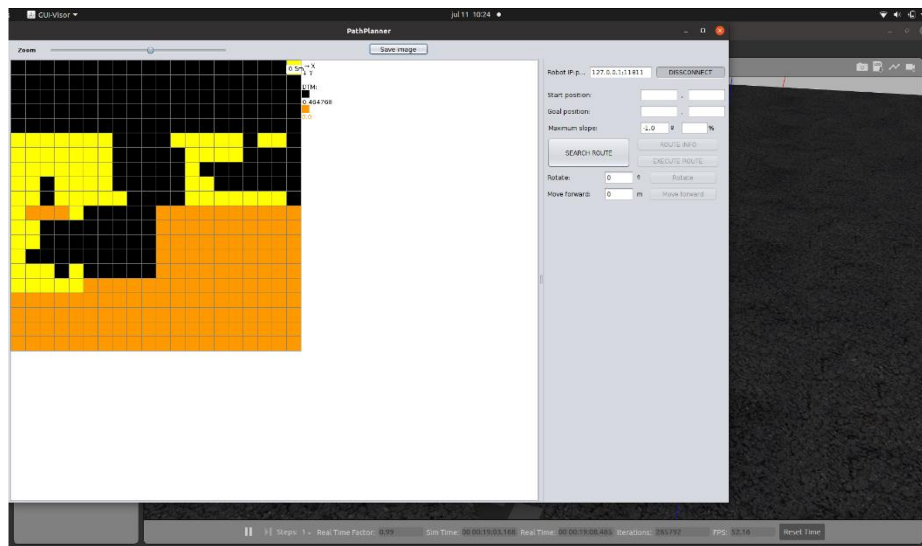


Figura 29: Visualización del mapa mientras se escanea el terreno

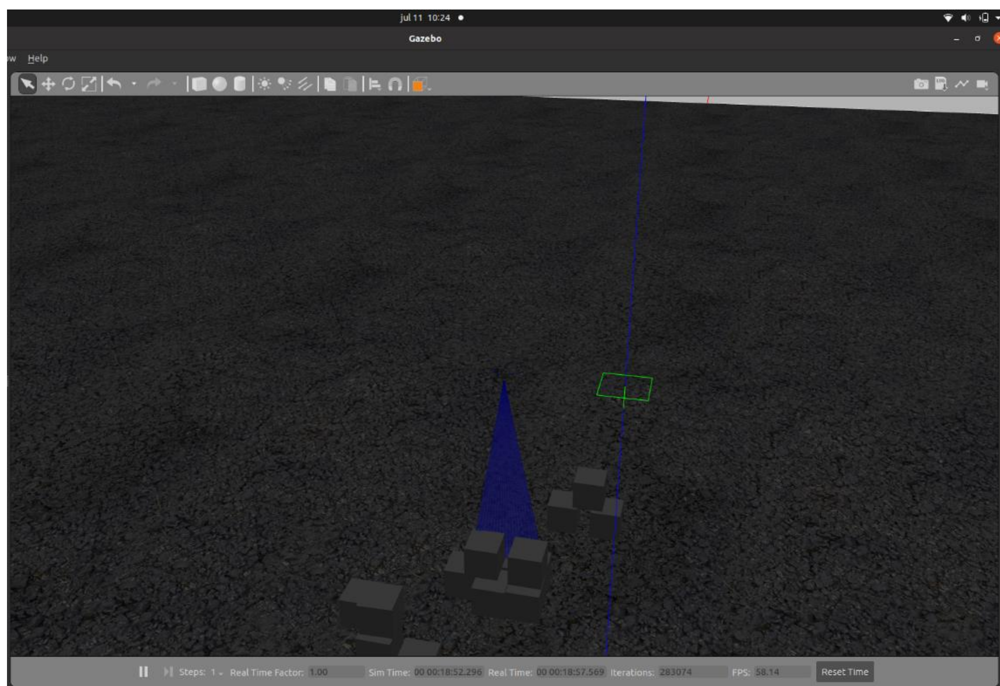


Figura 30: Dron escaneando la zona

Para realizar esta operación, el dron tarda aproximadamente 20 segundos en hacer el recorrido en una superficie de $10m^2$. Tras terminar de escanear toda la zona, el mapa se actualiza quedando como en la Figura 32. Se puede apreciar que se ha hecho correctamente ya que identifica los bloques centrales además de los cubos puestos en las esquinas como se aprecia en la Figura 27.

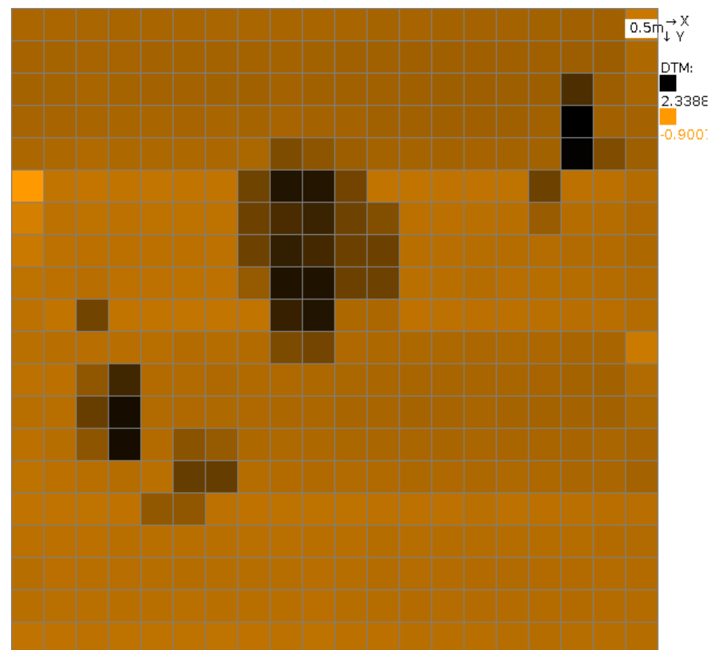


Figura 31: Mapa del experimento 1

El siguiente paso a seguir es el de trazar una ruta. En este caso, se ubica el Husky en la coordenado (5,5). Como cada celda es de un total de 0.5m, en Gazebo se tendrá que modificar la posición a la mitad, es decir, en las coordenadas (2.5,2.5). Como objetivo se establecerá las coordenadas (12,10) por lo que serán las (6,5) en Gazebo. En este caso, se ha decidido establecer un valor de pendiente máximo de 8 grados. Se ha elegido esta meta para ver como rodea la parte central de los cubos para llegar a dicho objetivo. Al dar al botón de búsqueda de la ruta, se ejecuta el algoritmo 3Dana, formando un camino seguro para que el rover llegue a su objetivo como se puede ver en la Figura 33.

Figura 32: Ruta seleccionada para experimento 1

En las siguientes figuras (de la 34 a la 36), se puede apreciar el avance del robot en Gazebo y como va actualizándose en la interfaz. De esta manera se ve como rodea los obstáculos debido a su altura hasta llegar a la posición final. En hacer el recorrido tarda aproximadamente 55 segundos ya que pierde tiempo rotando hasta que encuentra la posición adecuada para seguir avanzando. Además, en cuanto hay una desviación en la trayectoria, tiene que rectificar hasta volver a colocarse en el camino.

Figura 33: Posición 1 del experimento 1

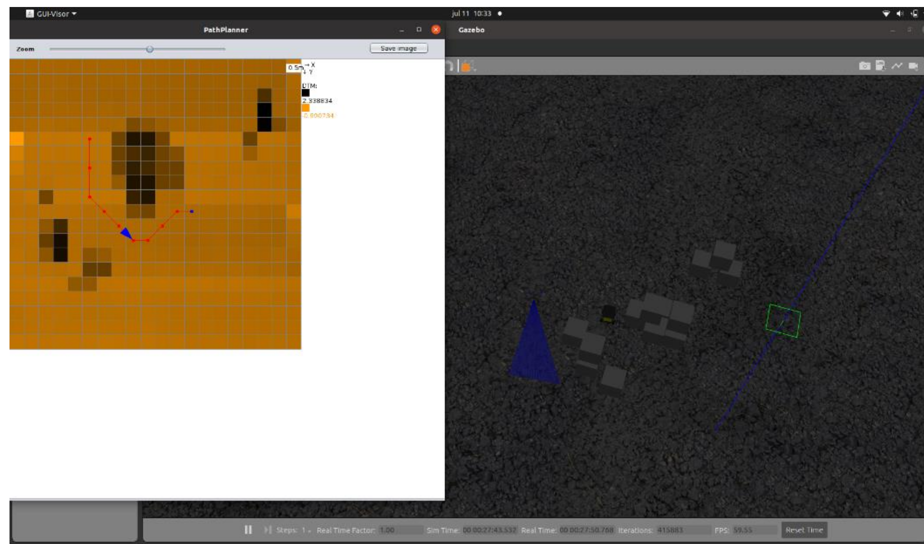


Figura 34: Posición 2 del experimento 1

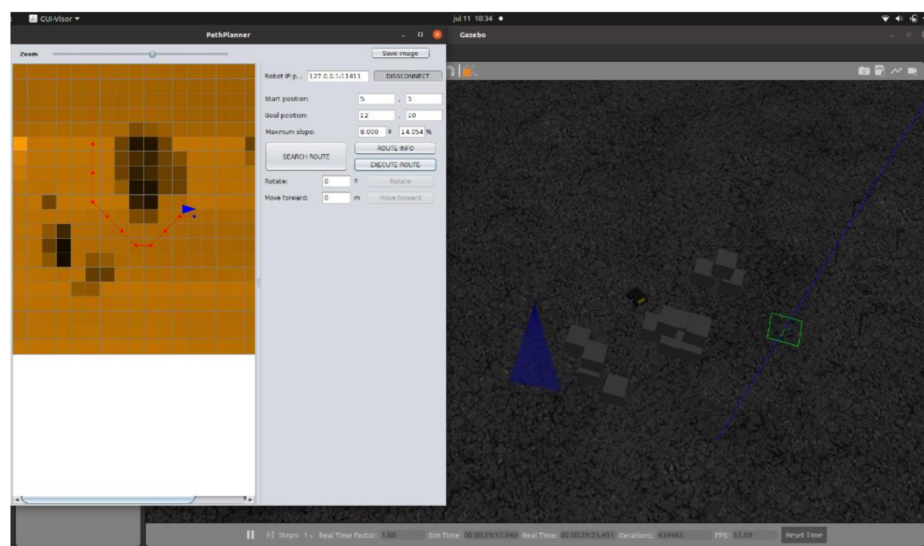


Figura 35: Posición final experimento 1

4.2. EXPERIMENTO 2

En este experimento se probará el proyecto en una superficie simulada de Marte como se definió en su respectiva sección. Para ello, habrá que proceder igual que en el experimento anterior, pero con la diferencia de que, esta vez, se tendrá que introducir marte de forma manual como se explicó.

Para poder posar el dron en la superficie, se tiene que hacer volar a suficiente altura para poder introducir Marte de forma manual por debajo suya. Después, se producirá a ejecutar el comando “commander land” (disponible a través de PX4) para hacerle aterrizar y que actualice su posición inicial de despegue. Esto es debido a que, si se cambia la posición del dron de forma manual, el LIDAR se bloquea y deja de medir las distancias correctamente. Esto sucede porque los rayos se quedan atascados como si hubiera siempre un obstáculo delante suya. Esto se debe a algún tipo de error del *software* de Gazebo, por lo que se procederá de la manera descrita.

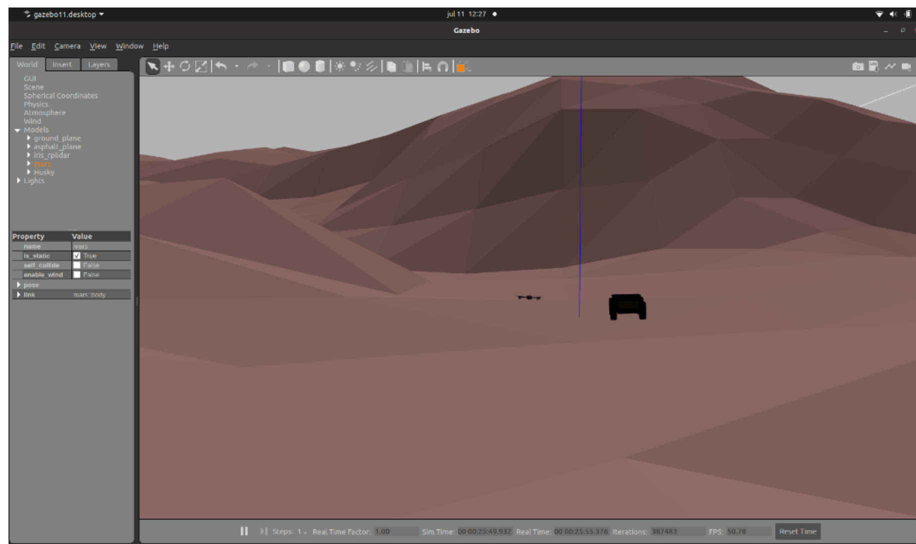


Figura 36: Perspectiva 1 de Marte en experimento 2

Una vez se tienen los robots posicionados y la interfaz conectada al rover, solo hay que ejecutar el nodo del dron para que despegue y tome las medidas de la zona. En este caso, se ha elegido una parte del mapa de Marte donde se puede observar una pequeña montaña a la izquierda de los robots. Se puede apreciar a la izquierda de la Figura 37 y detrás del Husky en la Figura 38.

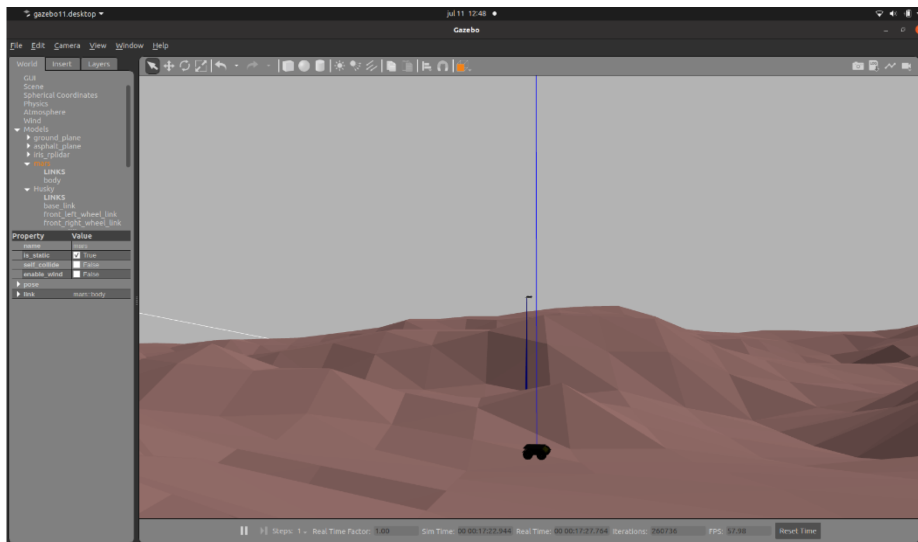


Figura 37: Perspectiva 2 de Marte en experimento 2

Al ejecutar el nodo, el dron escanea la zona al igual que en el experimento de la superficie plana. Como en este caso se está analizando un relieve, el mapa final tendrá un degradado de color en el aspecto, ya que no es una colina que tenga cambios bruscos de altura.

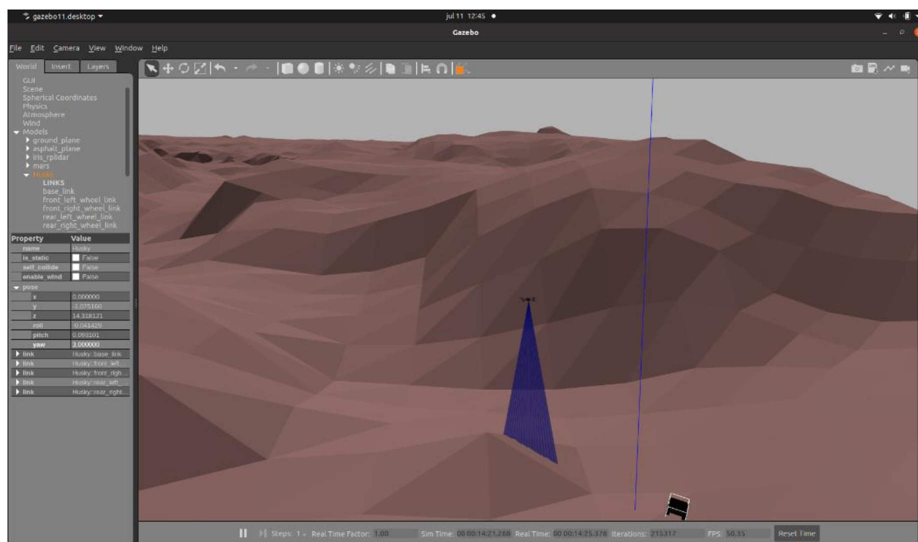


Figura 38: Dron escaneando zona del experimento 2

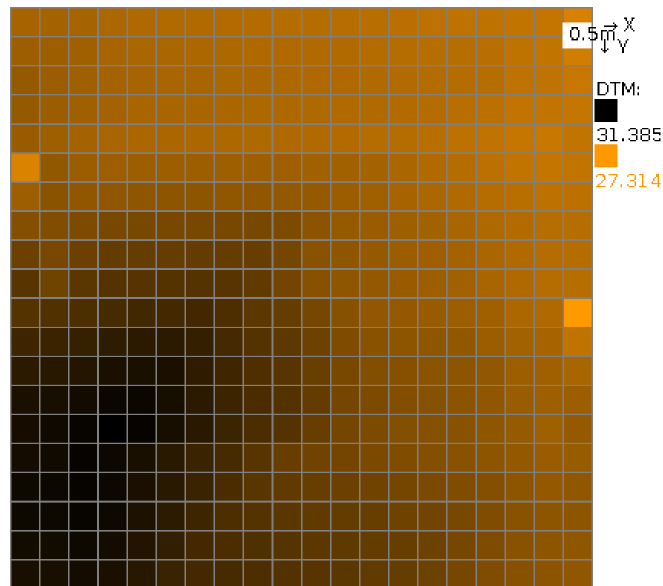


Figura 39: Mapa del experimento 2

Se puede observar en la Figura 40 que las alturas recogidas tienen unos valores que oscilan entre 27.3m y 31.3m respecto al origen de coordenadas, habiendo una diferencia de 4 metros entre la parte más baja y la más alta.

A continuación, a través de la interfaz de control, se marcan las coordenadas, tanto de origen como de llegada, para buscar la mejor ruta que esquive la colina descrita. Una vez obtenido el camino más adecuado, se envía la ruta al rover para que este la ejecute.

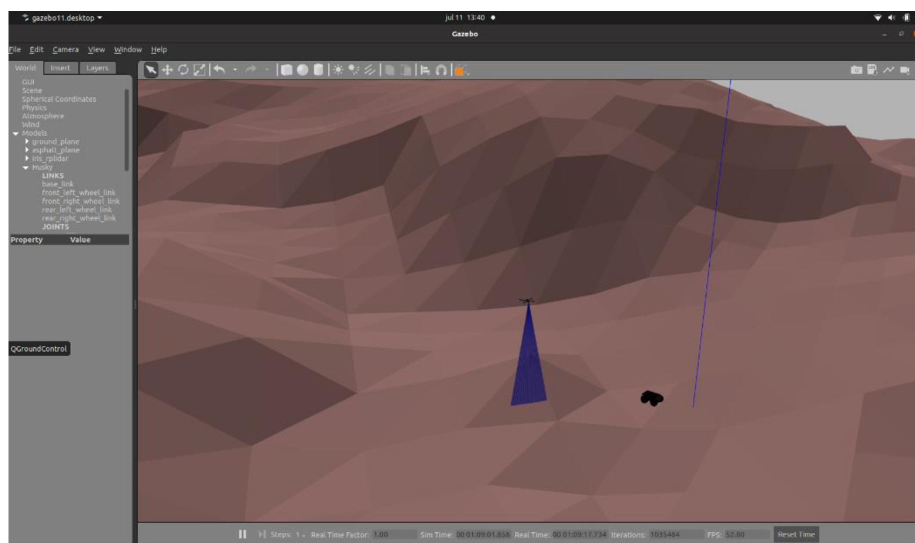


Figura 40: Posición inicial Husky experimento 2

En este caso, se ha elegido la casilla (2,4) como punto de partida y la (13,10) como llegada. Para el cálculo de la pendiente, se ha utilizado un valor de 4 grados como máximo, ya que, al

ser una superficie con una mayor diferencia de alturas, si se le pone un valor superior podrá atascarse en alguna pendiente.

Entre la Figura 42 y la Figura 44, se puede apreciar como el robot va avanzando para esquivar por completo la colina y llegar a su destino. En este caso, se tarda aproximadamente 1 minuto 15 segundos en realizar la ruta. Este incremento se debe a que la superficie es mucho más irregular que en el anterior experimento. Por lo tanto, al robot le cuesta mucho más hacer los giros y se queda algo más atascado. También se incluye el problema de que, aunque no está subido en la colina, no está en una posición totalmente plana, por lo que a veces puede llegar a deslizar y salirse de la posición.

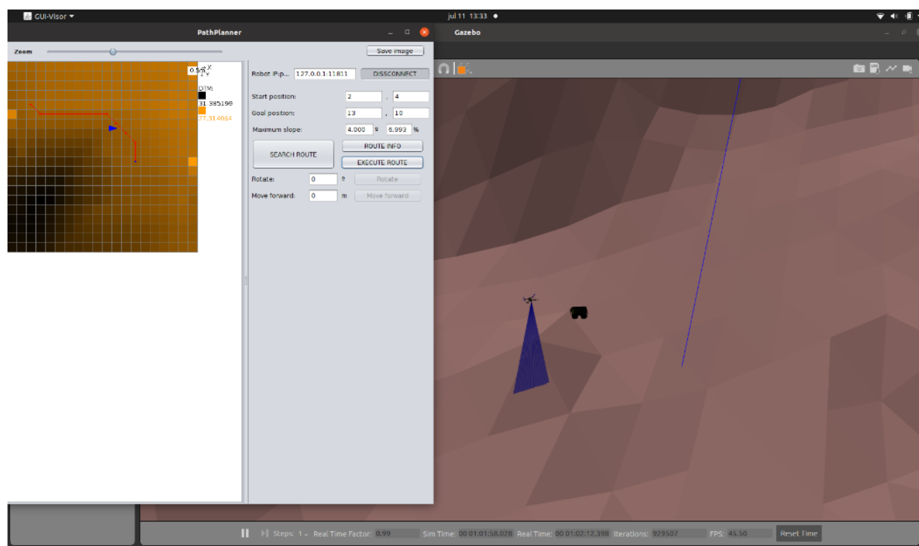


Figura 41: Posición 1 experimento 2

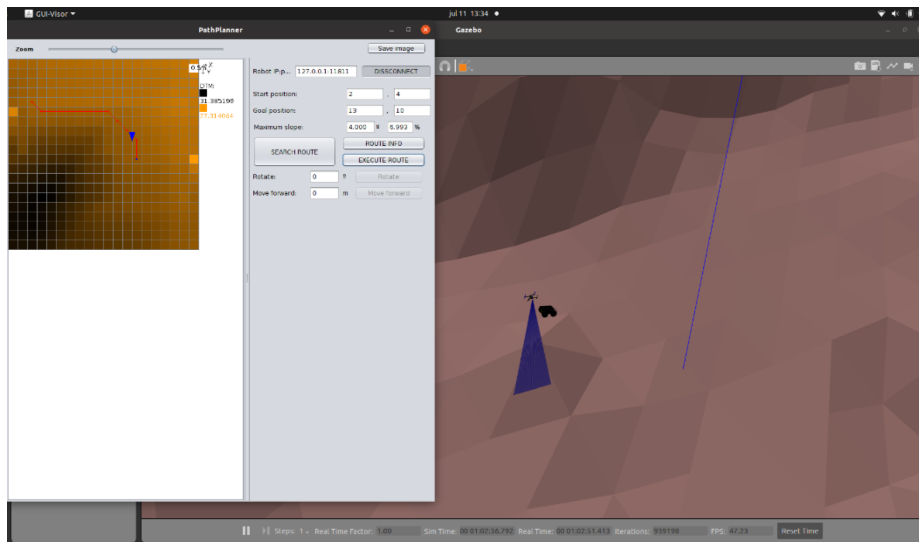


Figura 42: Posición 2 experimento 2

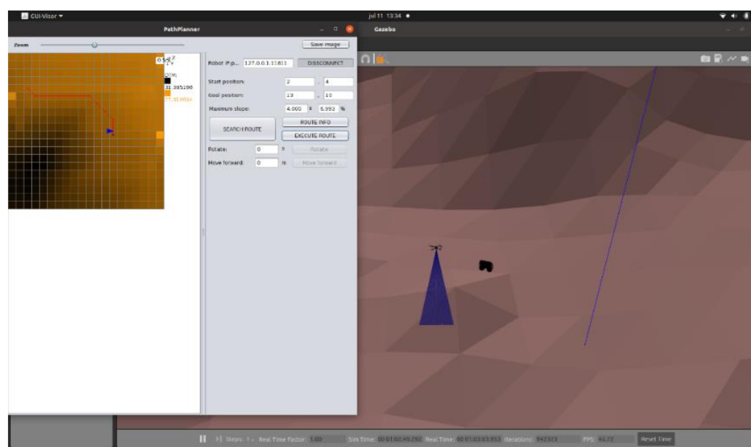


Figura 43: Posición final experimento 2

4.3. EXPERIMENTO 3

En este último experimento, se decidió mostrar el funcionamiento del dron a la hora de mapear una superficie superior. Para ello se introdujo a través del terminal, un valor de 40 en vez de 20, para hacer un mapa 40x40, el doble de tamaño que los experimentos anteriores.

De esta manera, la exploración puede realizarse a grandes escalas por si los obstáculos son de mayor tamaño como en la Figura X.

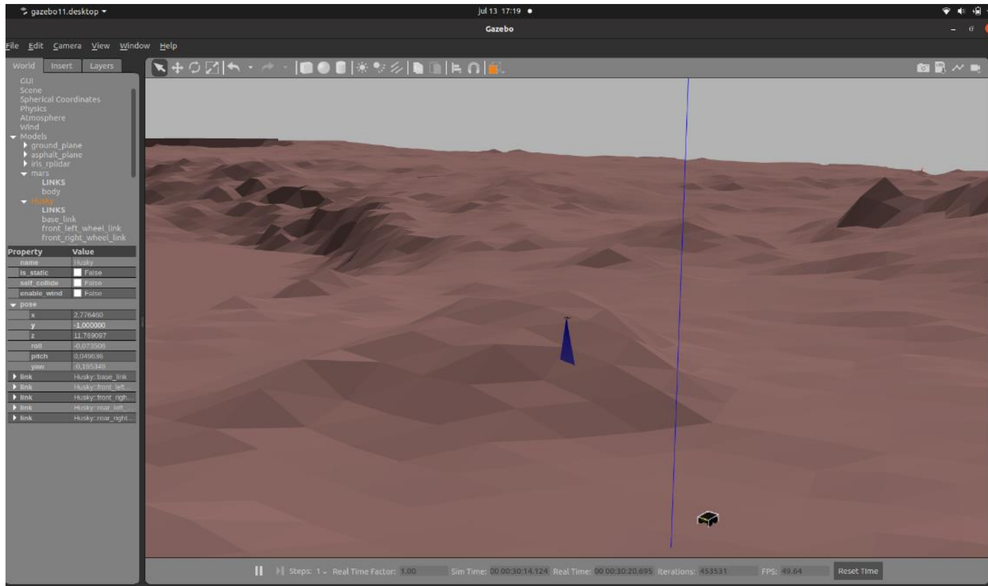


Figura 44: Marte con obstáculo voluminoso

Para realizar este barrido, al ser una superficie mayor, se ha tardado un total de 3 minutos y 30 segundos. El mapa resultante se puede visualizar en la Figura 45.

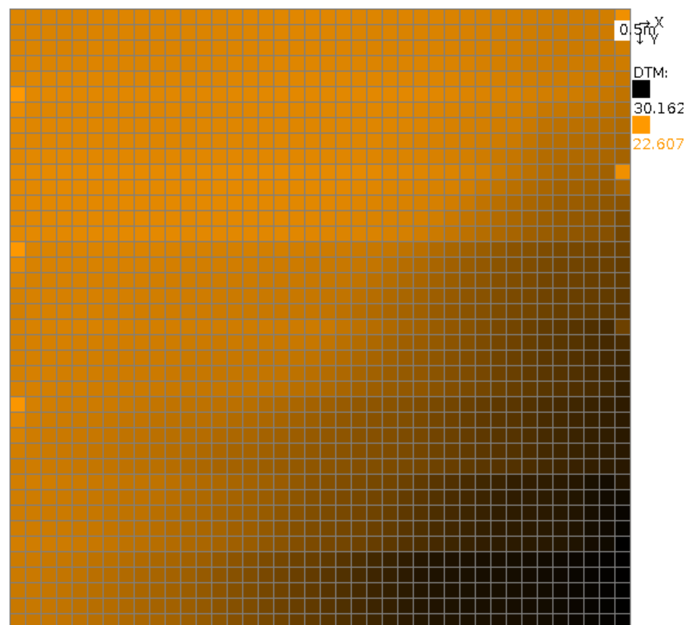


Figura 45: Mapa experimento 3

5. CONCLUSIONES Y TRABAJOS FUTUROS

En esta última sección, se comentará las conclusiones obtenidas tras la realización del proyecto elaborado. Se analizará todo lo desarrollado y se valorará el cumplimiento de los objetivos. También se hablará sobre distintas posibilidades a la hora de mejorar la exploración planetaria.

5.1. CONCLUSIONES

El espacio es un mundo que deja muchas cuestiones sin resolver, convirtiéndose así en uno de los objetos de estudio más interesantes. En las últimas décadas, la investigación en este sector ha ido en aumento, ya que los científicos e ingenieros quieren encontrar las respuestas que ayuden a entender cómo funciona el universo.

Para alcanzar esta meta, hay que abrir varías vías de investigación que puedan aportar la información necesaria para seguir aprendiendo progresivamente. Una de estas vías es la exploración planetaria, la cual no solo ayuda a entender mejor otros planetas, sino que brinda la posibilidad de tener un futuro. Además, ayuda con otro campo de interés como es encontrar otra forma de vida y descubrir que no estamos solos en el universo.

Para dicha exploración, se necesita de un factor imprescindible, la robótica. Gracias a los robots, se ha conseguido avanzar en el entendimiento de otros planetas. El caso más importante ha sido Marte. Después del envío de hasta 5 rovers, el conocimiento que se ha adquirido ha sido incalculable.

Por ello, se ha decidido realizar este proyecto el cual está basado en la exploración de Marte a través de la robótica. Para la optimización de la misma, se ha implementado dos robots que funcionan en cooperación. Uno reconoce el terreno para localizar posibles peligros mientras otro recorre el camino prestablecido gracias a la información del primero.

Para lograrlo, primero se tuvo que hacer un estudio del *software* ROS2, lo cual llevó bastante tiempo para poder comprenderlo en su totalidad y poder así trabajar con él. Además, se tuvo que aprender a utilizar los robots a través de Gazebo. Gracias a este estudio se logró la implementación de un código que pudiese automatizar los movimientos de los robots.

También se ha tenido que estudiar y aprender a controlar un dron en una herramienta de simulación, en este caso, controlado con ROS2 y PX4. Adicionalmente, se ha tenido que descubrir el funcionamiento de una interfaz de control, así como un algoritmo diseñado llamado 3Dana y su comunicación con el resto de los robots. Finalmente, se tuvo que convertir los datos en un mapa DTM para que visualmente se pudiera ver el relieve del terreno escaneado.

Por lo tanto, se ha conseguido aprender a través del estudio como utilizar una serie de *softwares* que permitieran controlar un robot aéreo y otro terrestre. Gracias a ello, se ha conseguido implementar un trabajo con el que poder explorar Marte de una forma más eficiente. Además, se ha conseguido controlar de forma automatizada dos robots y consecuentemente, la correcta cooperación entre ambos. Por ello se concluye que se han logrado los objetivos fijados.

5.2. TRABAJOS FUTUROS

A pesar de que se ha desarrollado un proyecto muy completo que permite optimizar la exploración planetaria, se puede mejorar en un futuro con la adición de las siguientes características:

- Incluir una cámara que trasmita imágenes del terreno explorado desde el UAV. De esta manera, no solo se tendría los datos de las variaciones de altura, sino que también se sabría qué lo causa.
- Incorporar un segundo LIDAR en el UAV. Este se encargaría de detectar obstáculos contra los que pudiese colisionar como por ejemplo una montaña. De esta manera, podría localizarlos y cambiar el rumbo.
- Añadir un LIDAR en el rover para la detección de obstáculos. Esto podría ser útil ya que lo mismo hay una roca en el camino que dificulte el movimiento y no ha sido apreciada por el dron. De esta manera, el rover podría esquivarla y seguir su ruta tras rodearla.
- Utilizar SLAM para mapear el terreno desde el rover. En consecuencia, se obtendría un mapa de la zona desde otra perspectiva para poder así aprender más del entorno.
- Mejorar el algoritmo de movimiento del UGV para reducir el tiempo en realizar las rutas.
- Realizar las pruebas con un rover diferente que permita realizar los giros de manera más eficiente.
- Incluir un brazo robótico que recoja muestras al llegar a los objetivos marcados.

Anexo I. Presupuesto

En este apartado, se comentará sobre los costes totales que supone la realización del proyecto. En ellos se incluirá los gastos aportados por distintos factores como la mano de obra, el *hardware* utilizado u otras cuestiones.

En el apartado de *software* no se aplicará ningún gasto adicional. Esto es debido a que todo lo utilizado en este campo, ha sido de código libre, por lo que su uso es totalmente gratuito. Esto se aplica tanto a ROS2 como a Gazebo o PX4.

En cuanto a la mano de obra, se tiene en cuenta un salario medio anual para un ingeniero electrónico de 28804€ anuales, por lo que, a la hora, se aproxima a un valor de 14,77€.

Respecto al *hardware*, solo se ha utilizado un ordenador portátil con las características apropiadas. El coste asciende a un total de 1169,59€. También se debe añadir el gasto generado a causa del uso de Internet. Al mes se estima un valor de 55 euros, sumando un total de 220€.

Otro aspecto a tener en cuenta es el gasto energético de electricidad a lo largo de los 4 meses trabajados. Se aproxima que el gasto destinado al proyecto es de 12 euros mensuales, lo que hace un total de 48€.

A todo esto, se le añadirá un margen de beneficio el cual se ha prestablecido en el 25%. De esta manera, el presupuesto total queda ilustrado en la tabla 1.

GASTO	**CANTIDAD**	**COSTE POR UNIDAD (€)**	**TOTAL (€)**
Software	3 unidades	0	0
Ordenador portátil	1 unidad	1169,59	1169,59
Ingeniero electrónico	412 horas	14,77	6085,24
Internet	4 meses	55	220
Electricidad	4 meses	12	48
Coste total sin beneficio			7522,83
Beneficio del proyecto		25% del total	1880,71
Presupuesto total			9403,54

Tabla 1: Presupuesto del proyecto

Anexo II. Manual

En esta sección se comentará los comandos necesarios para la realización del proyecto. Incluirá tanto los comandos de instalación como los necesarios para poder preparar la simulación.

- **Instalación** Para poder utilizar los respectivos *softwares,* se requiere de una previa instalación.
  - **ROS2**:

    Primero hay que autorizar antes la llave GPG de ROS2:

    $ sudo apt update && sudo apt install curl gnupg lsb-release
    sudo curl -sSL
    https://raw.githubusercontent.com/ros/rosdistro/master/ros.key -o /usr/share/keyrings/ros-archive-keyring.gpg

    Después se añade el repositorio apt:

    $ echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/ros-archive-keyring.gpg] http://packages.ros.org/ros2/ubuntu

    $ (source /etc/os-release && echo $UBUNTU_CODENAME) main" | sudo tee /etc/apt/sources.list.d/ros2.list > /dev/null

    Por último, se instala el paquete de ROS2 Galactic:

    $ sudo apt install ros-galactic-desktop

    Cabe destacar que, para poder utilizar ROS2, se tiene que cargar el entorno en cada terminal que se vaya a utilizar. Por ello, se puede añadir en el archivo ".bashrc" el comando correspondiente para que automáticamente se active en la terminal al abrirla. Dicho comando es:

    source /opt/ros/galactic/setup.bash

  - **Gazebo**:

    Primero se prepara el ordenador para aceptar el *software:*

    $ sudo sh -c 'echo "deb http://packages.osrfoundation.org/gazebo/ubuntu-stable `lsb_release -cs` main" > /etc/apt/sources.list.d/gazebo-stable.list'

    Luego se añade la llave de configuración:

    $ wget https://packages.osrfoundation.org/gazebo.key -O - | sudo apt-key add -

Por último, se actualiza y se instala Gazebo en su versión 11:

$ sudo apt-get update

$ sudo apt-get install gazebo11

Cabe destacar que, para utilizar ROS2 con Gazebo, se deben instalar los siguientes paquetes adicionales:

$ sudo apt install ros-galactic-gazebo-ros-pkgs

- **PX4:**

Para instalar el *software* se necesita clonar el siguiente repositorio:

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

A la hora de conectarlo con ROS2, otro repositorio de ser clonado y se tiene que aplicar una serie de comandos para instalar el DDS comentado:

**$ git clone --recursive https://github.com/eProsima/Fast-DDS-Gen.git -b v1.0.4 ~/Fast-RTPS-Gen **
**&& cd ~/Fast-RTPS-Gen \ && gradle assemble **
&& sudo env "PATH=$PATH" gradle install

Después, se crea un directorio de trabajo donde instalar los paquetes de PX4-ROS:

$ mkdir -p ~/px4_ros_com_ros2/src

A continuación, se instalan los paquetes:

$ git clone https://github.com/PX4/px4_ros_com.git ~/px4_ros_com_ros2/src/px4_ros_com

$ git clone https://github.com/PX4/px4_msgs.git ~/px4_ros_com_ros2/src/px4_msgs

Por último, se construye el directorio de trabajo para su uso:

$ cd ~/px4_ros_com_ros2/src/px4_ros_com/scripts

$ source build_ros2_workspace.bash

- **Interfaz:**

Para la interfaz se ha instalado un repositorio creado por el tutor del proyecto, Pablo Muñoz, para su posterior modificación:

$ git clone https://github.com/munozp/PathPlannerGUI

- Uso Para poner en marcha el proyecto, se deberá proceder de la siguiente manera:

En el primer terminal, se deberá ejecutar la estación de control:

$ ./QGroundControl.AppImage

Posteriormente, se abrirá otro terminal donde se ejecutará Gazebo junto al dron Iris escogido. En el comando se debe especificar primero el simulador y luego el modelo:

$ make px4_sitl_rtps gazebo_iris_rplidar

En el siguiente terminal, se abrirá el agente del DDS que permite habilita el puente entre ROS2 y PX4:

$ micrortps_agent -t UDP

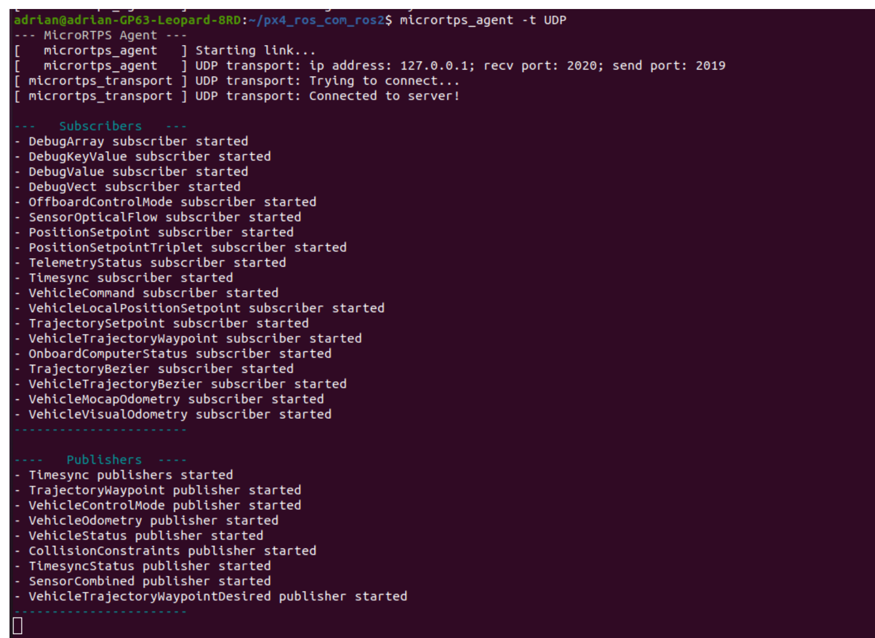


Figura 46: Terminal al ejecutar el agente del puente

Una vez preparado el simulador y el dron, se lanza el Husky a Gazebo a través del nodo creado en ROS2. Primero se pone el nombre del paquete y después el nombre del archivo. Antes de lanzar el robot, se debe cargar el entorno:

$. install/setup.bash

$ ros2 launch mars_spawner mars_bot_world.launch.py

A continuación, se abre la interfaz de control situándose en su correspondiente carpeta:

$ java -jar "PathPlannerGUI.jar"

En este momento se ejecuta el nodo del Husky especificando puerto y tamaño del mapa para posteriormente conectarlo a la interfaz gráfica:

$. install/setup.bash

$ ros2 run mars_controller husky_controller 11811 20

Por último, se ejecuta el nodo del dron que empezará con la operación:

$ ros2 run px4_ros_com drone_controller

Anexo III. Diagrama nodo controlador del Husky

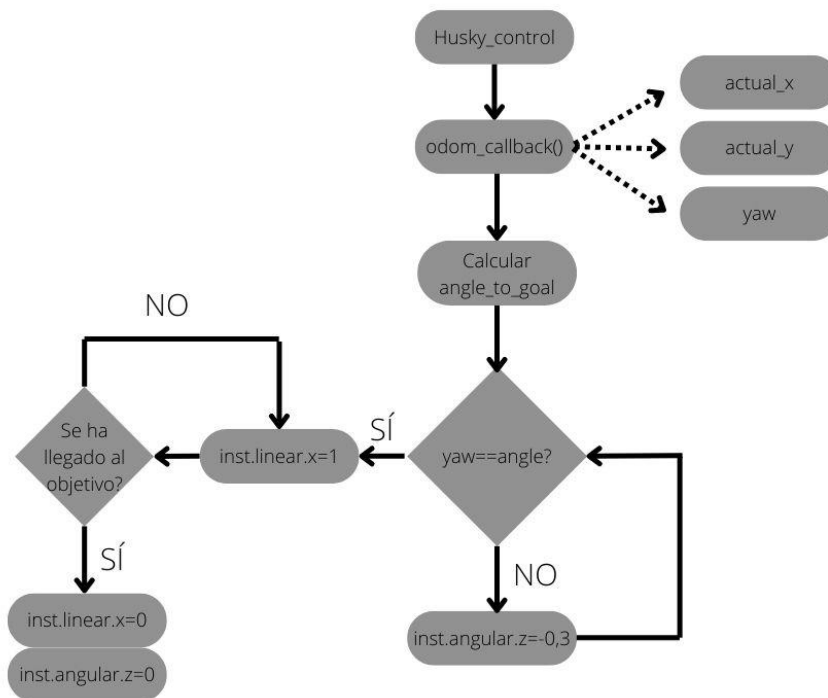


Figura 47: Diagrama del nodo controlador del Husky

Anexo IV. Diagrama nodo controlador del UAV

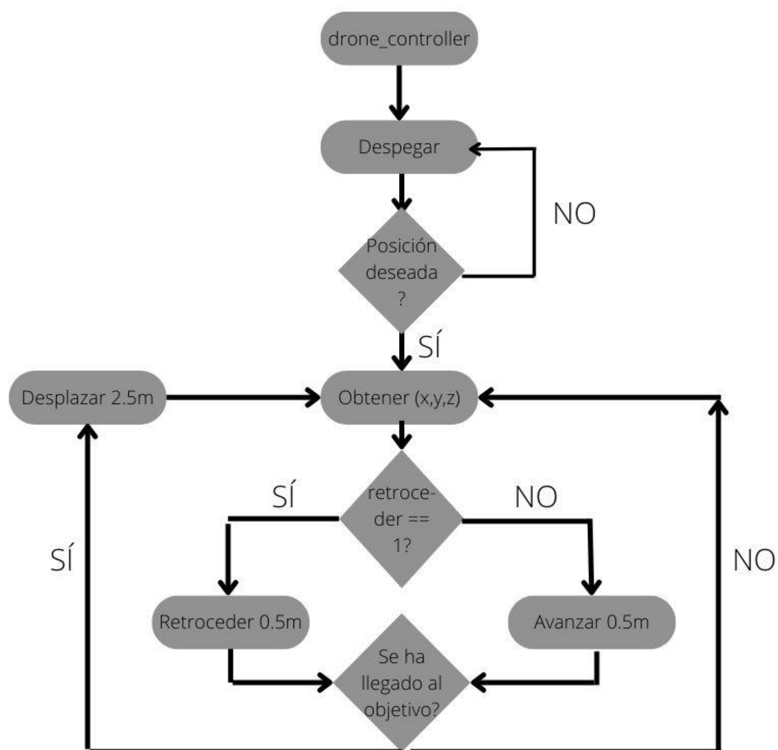


Figura 48: Diagrama del nodo controlador del UAV

BIBLIOGRAFÍA

[1] *3Dana: A path planning algorithm for surface robotics. Muñoz, P.; R-Moreno, M. D. and Castaño, B. Engineering Applications of Artificial Intelligence, vol. 60, pp. 175-192. 2017.*

[2] *University of Arizona, Overview of digital terrain models (DTM).* Disponible en: https://www.uahirise.org/dtm/about.php

[3] *NASA, Mars helicopter tech demo*. Disponible en: https://mars.nasa.gov/technology/helicopter/

[4] *Blender, Download.* Disponible en: https://www.blender.org/download/

[5] *NASA 3D Resources, Curiosity Landing Site (QR).* Disponible en: https://nasa3d.arc.nasa.gov/detail/CuriosityQR

[6] *Jet Propulsion Laboratory, California Institute of Technology, MarsTrek.* Disponible en: https://trek.nasa.gov/mars/#v=0.1&x=0&y=0&z=1&p=urn%3Aogc%3Adef%3Acrs%3AEPSG%3A%3A104905&d=&locale=&b=mars&e=-269.999994963537%2C-132.53906002767377%2C269.999994963537%2C132.53906002767377&sfz=&w=

[7] *Kevin DeMarco, How to create terrain for Gazebo simulation with Blender 2.9.* Disponible en: https://www.youtube.com/watch?v=GNbH8Pf7nGk

[8] *Automatic Addison, How to Load a World File into Gazebo – ROS 2.* Disponible en: https://automaticaddison.com/how-to-load-a-world-file-into-gazebo-ros-2/

[9] *Galvez-Serna, J.; Felipe, L. and Vanegas, F. A Probabilistic based UAV Mission Planning and Navigation for Planetary Exploration.* Disponible en: https://www.researchgate.net/publication/343206746_A_Probabilistic_based_UAV_Mission_Planning_and_Navigation_for_Planetary_Exploration

[10] *Automatic Addison, How to Simulate a Robot Using Gazebo and ROS 2.* Disponible en: https://automaticaddison.com/how-to-simulate-a-robot-using-gazebo-and-ros-2/

[11] *The Construct, Spawning multiple robots in Gazebo with ROS2.* Disponible en: https://www.youtube.com/watch?v=OMwK8u7bap0

[12] *The Construct, How to use XACRO files with Gazebo in ROS2.* Disponible en: https://www.theconstructsim.com/how-to-use-xacro-in-ros2-gazebo/

[13] *Husky, galactic-devel.* Disponible en: https://github.com/husky/husky/tree/galactic-devel

[14] *The Construct, How to Move a Robot to a Certain Point Using Twist.* Disponible en: https://www.theconstructsim.com/ros-qa-053-how-to-move-a-robot-to-a-certain-point-using-twist/

[15] *The Construct, Exploring ROS2 using wheeled Robot - #3 - Moving the Robot.* Disponible en: https://www.youtube.com/watch?v=SinvFQ9Vobg

[16] *ROS2 Documentation: Galactic, Writing a simple publisher and subscriber (C++).* Disponible en: https://docs.ros.org/en/galactic/Tutorials/Beginner-Client-Libraries/Writing-A-Simple-Cpp-Publisher-And-Subscriber.html

[17] *ROS2 Documentation: Galactic, Using parameters in a class (C++)*. Disponible en: https://docs.ros.org/en/galactic/Tutorials/Beginner-Client-Libraries/Using-Parameters-In-A-Class-CPP.html

[18] *Riquelme, S., Como programar socket TCP/IP Cliente-Servidor en C++ Parte 2 implementación.* Disponible en: https://www.youtube.com/watch?v=0I56pVOSK0E

[19] *Auterion, ROS World 2020: Getting started with ROS 2 and PX4.* Disponible en: https://www.youtube.com/watch?v=qhLATrkA_Gw

[20] *PX4-Autopilot.* Disponible en: https://github.com/PX4/PX4-Autopilot

[21] *PX4 User Guide, Building PX4 Software.* Disponible en: https://docs.px4.io/main/en/dev_setup/building_px4.html

[22] *PX4 User Guide, ROS 2 User Guide (PX4-ROS 2 Bridge).* Disponible en: https://docs.px4.io/main/en/ros/ros2_comm.html

[23] *PX4 User Guide, Fast DDS Installation.* Disponible en: https://docs.px4.io/main/en/dev_setup/fast-dds-installation.html

[24] *Andrade, R. [ROS2] QOS profile options and subscribing problem #187.*Disponible en: https://github.com/ros-visualization/rqt/issues/187

[25] *ROS2 Documentation: Rolling, QoS compabilities.* Disponible en: https://docs.ros.org/en/rolling/Concepts/About-Quality-of-Service-Settings.html

[26] *Automatic Addison, Set Up LIDAR for a Simulated Mobile Robot in ROS 2.* Disponible en: https://automaticaddison.com/set-up-lidar-for-a-simulated-mobile-robot-in-ros-2/

[27] *The Construct, Exploring ROS2 with wheeled robot – #2 – How to subscribe to ROS2 laser scan topic.* Disponible en: https://www.theconstructsim.com/exploring-ros2-with-wheeled-robot-2-how-to-subscribe-to-ros2-laser-scan-topic/

[28] *Muyinteresante, ¿Cuántos rovers hemos enviado a Marte?.* Disponible en: https://www.muyinteresante.es/ciencia/articulo/cuantos-rovers-hemos-enviado-a-marte-871596138669

[29] *MIT Technology Review, La NASA se suma a la moda del software de código abierto.* Disponible en: https://www.technologyreview.es/s/13218/la-nasa-se-suma-la-moda-del-software-de-codigo-abierto

[30] *European Environment Agency, digital terrain model.* Disponible en: https://www.eea.europa.eu/help/glossary/eea-glossary/digital-terrain-model

[31] *Redeszone, Qué es un socket TCP o UDP y qué diferencias hay con los puertos.* Disponible en: https://www.redeszone.net/tutoriales/configuracion-puertos/que-es-socket-tcp-udp-diferencias-puertos/

[32] *ROS.* Disponible en: https://www.ros.org/

[33] *ROS2 Documentation: Galactic, Tutorials*. Disponible en: https://docs.ros.org/en/galactic/Tutorials.html

[34] *PX4 User Guide, ROS 2 Offboard Control Example.* Disponible en: https://docs.px4.io/main/en/ros/ros2_offboard_control.html

[35] *PX4 User Guide, Simulation.* Disponible en: https://docs.px4.io/v1.12/en/simulation/

[36] *PX4 User Guide, Gazebo Simulation.* Disponible en: https://docs.px4.io/main/en/simulation/gazebo.html

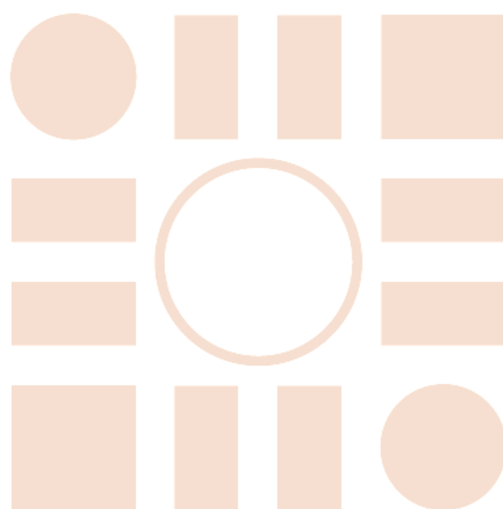
[37] *PX4 User Guide, vehicle_odometry (UORB message).* Disponible en: https://docs.px4.io/main/en/msg_docs/vehicle_odometry.html#vehicle-odometry-uorb-message

[38] *Gazebo.* Disponible en: https://gazebosim.org/home

[39] *Gazebo, Gazebo Tutorials.* Disponible en: https://classic.gazebosim.org/tutorials

Universidad de Alcalá

Escuela Politécnica Superior



ESCUELA POLITECNICA SUPERIOR

Universidad de Alcalá