Proyecto Fin de Carrera
Ingeniería Electrónica, Robótica y Mecatrónica

Métodos de ayuda a la navegación en exteriores para robot SUMMIT en entorno ROS

Autor: Diego José Bermúdez Salguero

Tutores: Miguel Ángel Ridao Carlini y Ramón Andrés García Rodríguez

Dpto. Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2022

Proyecto Fin de Carrera
Ingeniería Electrónica, Robótica y Mecatrónica

Métodos de ayuda a la navegación en exteriores para robot SUMMIT en entorno ROS

Autor:
Diego José Bermúdez Salguero

Tutores:
Miguel Ángel Ridao Carlini
Catedrático de Universidad

Ramón Andrés García Rodríguez
Profesor de Universidad

Dpto. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2022

Proyecto Fin de Carrera: Métodos de ayuda a la navegación en exteriores para robot SUMMIT en entorno ROS

Autor: Diego José Bermúdez Salguero

Tutor: Miguel Ángel Ridao Carlini

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2022

El Secretario del Tribunal

A mi familia

A mis amigos

Agradecimientos

Con la finalización de este trabajo, se cierra una de las etapas más importantes de mi vida. Ha sido un largo y arduo camino, pero a su vez, estos cuatro años han sido realmente especiales. He conocido a muchas personas que me han marcado y sé que me acompañarán por siempre. Por eso debo agradecer tanto a ellos como a mi familia el apoyo brindado durante este tiempo.

También debo dar las gracias a mis tutores Miguel Ángel Ridao y Ramón Andrés García por la ayuda a lo largo del desarrollo de este proyecto y además a los miembros del laboratorio por colaborar con el funcionamiento del robot.

Por otro lado, quiero agradecer a los desarrolladores de ROS que he consultado y se han prestado a resolver las dudas que me surgían.

Por último, debo mencionar a mis compañeros de piso: Fernando, Manu y Jesús. Gracias a ellos todos estos años han sido mucho más llevaderos y aun teniendo un mal día, llegabas a casa y acababas llorando de la risa mientras divagábamos sobre nuestro día.

Diego José Bermúdez Salguero

Alumno de la Escuela Técnica Superior de Ingeniería

Sevilla, 2022

RESUMEN

El grupo de investigación de Automática y Robótica Industrial del departamento de Ingeniería de Sistemas y Automática de la Escuela Técnica Superior de Ingeniería forma parte del proyecto europeo OCONTSOLAR, que tiene como objetivo desarrollar nuevos métodos de control para la optimización del funcionamiento de las plantas de energía solar.

Para la obtención de los diferentes datos de los sensores para el control, se necesita de un robot con la capacidad de navegar autónomamente. El robot elegido para ello es el SUMMIT-XL, creado por Robotnik. Esta funcionalidad ya ha sido desarrollada y se tratará de mejorarla mediante la implementación de un sistema GPS, para ayudar a la localización del robot y reducir el error en la estimación de la posición.

Además, la gran mayoría de plantas solares presentan un terreno irregular, pudiendo haber agujeros en el suelo y pendientes. Para solucionar este problema, se otorga al robot de un sistema de detección de agujeros y rampas haciendo uso de la cámara presente en este.

Abstract

The Industrial Automation and Robotics research group of the Systems and Automation Engineering Department of Seville University is part of the European project OCONTSOLAR, which main target is to develop new control methods to optimize solar power plants functioning.

To obtain the different data from the sensors for the control, it is needed a robot able to navigate autonomously. SUMMIT-XL, created by Robotnik, is the robot chosen for this activity. This feature has already been developed and the objective of this Project is to upgrade it by a GPS system deployment that will help to localize the robot, minimizing the position error.

Besides, most of solar power plants have an uneven terrain where there might be holes. To solve this issue, a holes and slopes detection system is provided to the robot making use of its camera.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

El desarrollo de la tecnología es el principal motor del avance de la humanidad. Esta comprende un abanico de diferentes campos, siendo dos de ellos la robótica y la automatización. El perfeccionamiento y la innovación en ellos provoca un mejor rendimiento al realizar diferentes tareas y que cada vez se requiera menor presencia humana en los procesos de producción, además de mejorar la calidad de vida a nivel doméstico.

La robótica es una rama de la ingeniería donde se integran varias disciplinas tecnológicas, tales como la electrónica, la mecánica, la informática, etc. Su objetivo es el diseño de autómatas o robots que realicen diferentes tareas. Estos se pueden clasificar según su ubicación, distinguiendo si permanece fija o en cambio tienen la capacidad de moverse por el entorno (móvil).

Dentro de la robótica fija, se encuentran principalmente dos tipos de robots:

- **Robots articulados.** Son los más comunes en entornos industriales. Presentan articulaciones rotatorias que reciben el nombre de ejes. La adición de ejes provoca un mayor número de grados de libertad y, por lo tanto, mayores capacidades de movimiento.
- **Cobots.** La principal funcionalidad de este robot es cooperar con los humanos para mejorar las labores de producción. Suelen emplearse para las tareas más repetitivas y manuales.

Por otro lado, respecto a la robótica móvil, hay múltiples tipos, entre los que podemos distinguir:

- **AMR.** Se trata de un robot móvil autónomo que se encarga de realizar el transporte de material de un punto a otro en un sistema de producción.
- **AGV.** Circulan autónomamente por rutas establecidas previamente y facilita el transporte de materiales.
- **Robots submarinos.** Están diseñados para llevar a cabo tareas bajo el agua, entre las que se incluyen cartografía, inspección y mantenimiento.
- **Robots aéreos o UAV.** Son aeronaves que realizan tareas a cierta altura y cuyo manejo puede ser por control remoto o autónomo. No llevan piloto a bordo, de ahí sus siglas (Unmanned Aerial Vehicle).
- **Humanoides.** Como su propio nombre indica, tratan de imitar el cuerpo humano. Pueden ser completos, es decir, que consten de cabeza, tronco y las cuatro extremidades o que solo presenten parte del cuerpo.
- **Zoomórficos.** Basan su estructura y movilidad en algún animal existente. Pueden ser caminadores o no caminadores. Estos robots suelen ser bastante eficientes enérgicamente.

1.1 Marco contextual y estado del arte

Este trabajo se trata de una continuación sobre las investigaciones previas sobre creación de mapas y navegación autónoma con el robot SUMMIT-XL. El sistema de generación de mapas fue diseñado por Manuel Mora Nieto en su Trabajo de Fin de Máster utilizando la información que el robot recibe mediante el lidar 3D y la combinación de las cámaras RGBD. Partiendo de esto, Manuel Serrano Rodríguez realizó su Trabajo de Fin de Máster sobre la navegación del robot en entornos tanto explorados como sin explorar.

La finalidad de todos estos avances realizados es la colaboración con el proyecto OCONTSOLAR *(Optimal Control of Thermal Solar Energy Systems)*, donde el grupo de investigación de Robótica y Automática Industrial de la Universidad de Sevilla toma parte. Este proyecto consiste en desarrollar nuevos métodos de control donde se utilicen sensores sobre vehículos no tripulados aéreos (drones) o terrestres (UGV). Estos se implementarán sobre plantas de energía solar para optimizar su producción mediante estimaciones y

predicciones. Además, los resultados obtenidos de la investigación podrán ser extrapolados a otros campos como la agricultura, administración de energía en edificios, microgrids, etc.

Figura 1-1. Proyecto OCONTSOLAR

La localización es un área muy trabajada para el manejo de robots móviles. Se puede realizar de diversas formas además de por GPS como se ha elegido en este trabajo. Entre ellas se encuentran 802.11, RFID, bluetooth, por infrarrojos, por ultrasonidos, Wi-Max, Zing-Bee, UWB y visión artificial.

En el caso del GPS, al estar ampliamente extendido hay muchos proyectos y artículos realizados durante estos últimos años entre los que se incluyen:

- **Localización GPS aplicada a robots móviles en campos de golf.** Este artículo se centra principalmente en robots que pueden encontrarse en campos de golf, tales como recogedores de pelotas, cortadoras de césped, alisadoras de búnker, etc.

  En el caso de las recogedoras de bolas, se hace uso de una cámara con la que mediante un algoritmo de visión artificial trata de hallar las regiones con una mayor concentración de pelotas. Gracias a la localización por GPS y la información de los diferentes sensores se implementa un sistema de navegación basado en un control Pure Pursuit. Este trabajo fue desarrollado en la Universidad de Almería y para realizar las pruebas se hizo uso del robot móvil Fitorobot.

Figura 1-2. Fitorobot

- **Técnica GPS para medir contaminación con robot submarino.** En la Facultad de Ciencias de la Computación de Puebla en México se diseñó un robot submarino capaz de realizar análisis del agua en tiempo real debido a la gran contaminación presente en la zona. El robot consta con sensores que ayudan a detectar espacios donde se filtra el agua. Por otro lado, tienen sensores de tensión para medir el estado de la batería y un GPS para llevar a cabo la localización.

Figura 1-3. Robot submarino

- **Diseño de un sistema robótico para mantenimiento de campos sembrados.** Para la elaboración de este proyecto se emplearon varios robots móviles, cada uno dotados de GPS, que almacenaban su posición en un Sistema de Información Geográfico mediante una red ethernet inalámbrica. Gracias a esto el enjambre de robots pueden localizar y navegar alrededor del campo.
- **Vehículo de detección de rutas mediante GPS orientado a regadíos por ambiente a escala de laboratorio.** El fin de este trabajo diseñado en una universidad de Imbabura (Ecuador) era aplicar la tecnología GPS a un pequeño robot dotado de sensores para que determinara el camino más corto para que los sistemas de riego otorgaran agua a las zonas donde se requería. El robot empleado fue el FarMi4.

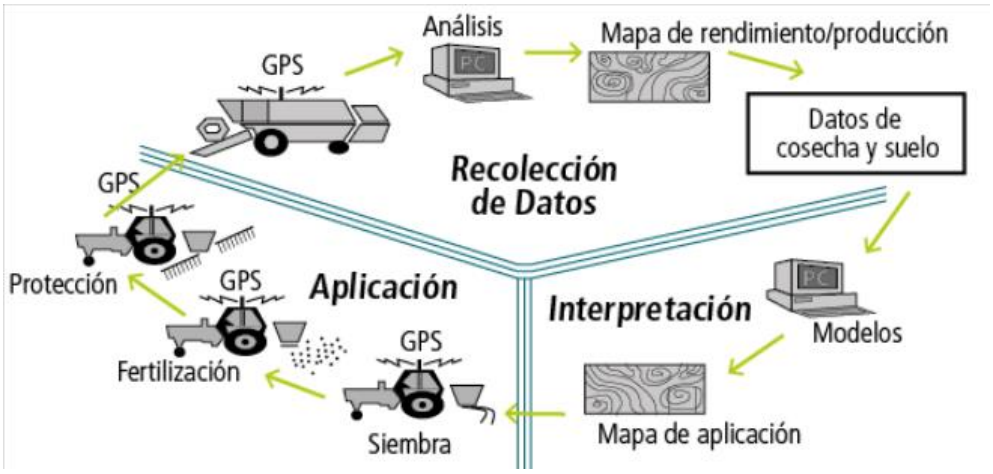


Figura 1-4. Esquema de funcionamiento de GPS orientado a regadíos

- **Localización de materiales radiactivos mediante un robot con GPS.** Esta investigación se realizó por la necesidad de protección de los técnicos y así poder mantenerse a distancias seguras de las fuentes de radiación.

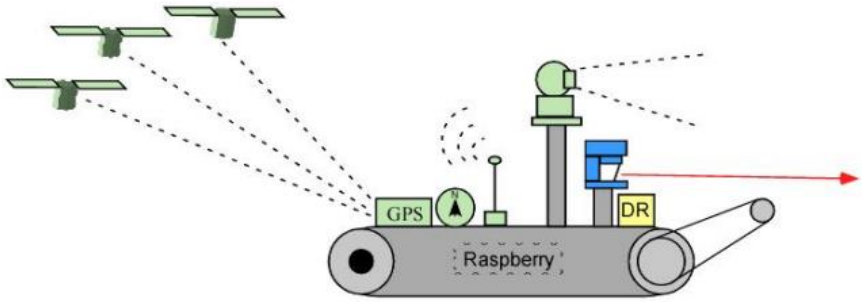


Figura 1-5. Composición del sistema de localización de cuerpos radiactivos

1.2 Objetivos del trabajo

El objetivo de este trabajo es la mejora del sistema de navegación autónoma mediante la implementación de un sistema GPS, que ayude a la localización del robot. Además, se ha otorgado al SUMMIT-XL de una técnica de detección de agujeros y otra de pendientes.

Con respecto a la localización mediante GPS, se tratará de encontrar la manera donde se produzca un menor error al usarlo. También, se realizarán pruebas en casos donde el funcionamiento del GPS empeore, tales como interiores y zonas al lado de edificios y como afecta esto a la localización.

Por otro lado, para la detección de agujeros en el suelo, haciendo uso de alguna de las nubes de puntos que proporcionan los sensores del robot (LIDAR o cámara RGBD). Cuando dentro de la zona seleccionada de la nube de puntos haya menos puntos de cierto umbral, significará que en esa zona hay un agujero.

Además, se agregará un sistema de detección de pendientes que, dado un ángulo de inclinación, determine si la pendiente es lo suficientemente inclinada como para que el robot no sea capaz de subirla.

Otro gran móvil de este trabajo es la integración de todo lo diseñado, ya que se parte de unas funcionalidades ya diseñadas previamente y no se deben presentar incompatibilidades con las nuevas características que se le proporcionan al robot.

1.3 Metodología empleada

Para el desarrollo de este trabajo, se han seguido los siguientes pasos:

- Instalación de Ubuntu 16.04 y ROS Kinetic.
- Comprobación del funcionamiento de los nodos propios del Summit-XL.
- Descarga e integración de los paquetes y funciones necesarios para el desarrollo de los objetivos determinados.
- Comprobación y corrección de su funcionamiento.
- Pruebas de campo con el robot real.

En el capítulo presente a continuación se describe el robot empleado, sus componentes y su configuración cinemática. Además, como se hace uso del sistema GPS, se explica el funcionamiento de un GPS genérico y del diferencial y se especifica el sensor que será empleado en este trabajo junto con la IMU, ya que se hará uso de ella para la transformación de coordenadas GPS.

En el tercer capítulo, se explica el software empleado, donde se incluye una descripción del firmware ROS, el simulador Gazebo y el visualizador RVIZ.

En el cuarto capítulo, se enumeran y reseñan los paquetes empleados para el total funcionamiento del sistema, tanto la locomoción general del robot, como la localización por GPS y detección de agujeros.

En el quinto capítulo, se detallan los diferentes ensayos realizados, tanto en el entorno de simulación como las pruebas con el robot real.

Por último, en el sexto capítulo se llevará a cabo un análisis en general del trabajo y se elaborarán unas conclusiones.

También, para apoyar una futura instalación del sistema desarrollado, se incorpora un anexo detallando los pasos seguidos para que todo funcione correctamente.

2 DESCRIPCIÓN DEL HARDWARE

En el presente capítulo se realizará una descripción física de los componentes empleados en el trabajo. Principalmente partimos del robot SUMMIT-XL, de la empresa Robotnik, que es un robot móvil con capacidad para operar tanto en interiores como en exteriores. Además, se describirá el sensor GPS utilizado y su funcionamiento.

2.1 SUMMIT-XL

El SUMMIT-XL es una plataforma robótica móvil y robusta diseñada como UGV (*unmanned ground vehicle*). Es fabricado por la empresa internacional con sede en Valencia, Robotnik, especializada en productos de robótica móvil. Este robot es muy versátil y resistente, por lo que es utilizable tanto al aire libre como en interiores y posee un gran número de aplicaciones entre las que se incluyen el I+D, militar, entrar en áreas peligrosas, vigilancia y monitorización remota.

Figura 2-1. Robot SUMMIT-XL

Tiene unas dimensiones de 720 x 614 x 416 mm, además de un peso de 65 kg. Entre sus especificaciones se encuentran:

- Rango de temperatura de 0 a 50 ºC.
- Velocidad máxima de 3 m/s.
- Capacidad de carga de 65 kg.
- Máxima pendiente de 80 %.
- Autonomía de 10 horas de movimiento continuo.
- Sistema de tracción definido por cuatro ruedas.
- Motores de tracción compuestos por cuatro servomotores de 500 W cada uno sin escobillas.

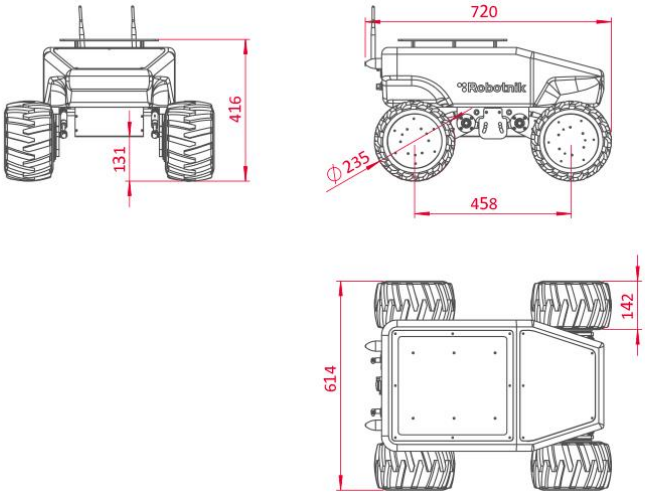


Figura 2-2. Plano de las dimensiones del SUMMIT-XL

2.1.1 Configuración cinemática

El SUMMIT-XL consta de dos configuraciones cinemáticas posibles: skid-steering u omnidireccional. Para la configuración skid-steering deberemos emplear las ruedas de goma y son recomendadas para cuando el robot se encuentre en un interior.

Figura 2-3. Ruedas de Summit-XL

En este trabajo se empleará una configuración skid-steering, por lo que se han elegido las ruedas de goma. Esta configuración tiene una utilización muy extendida, sobre todo en vehículos como tanques y excavadoras, comúnmente conocidos como vehículos oruga. La dirección deseada se alcanza mediante una velocidad o dirección diferente a cada lado del vehículo, siendo importante que sean iguales en un mismo costado del vehículo y provocando un deslizamiento de las ruedas en el suelo.

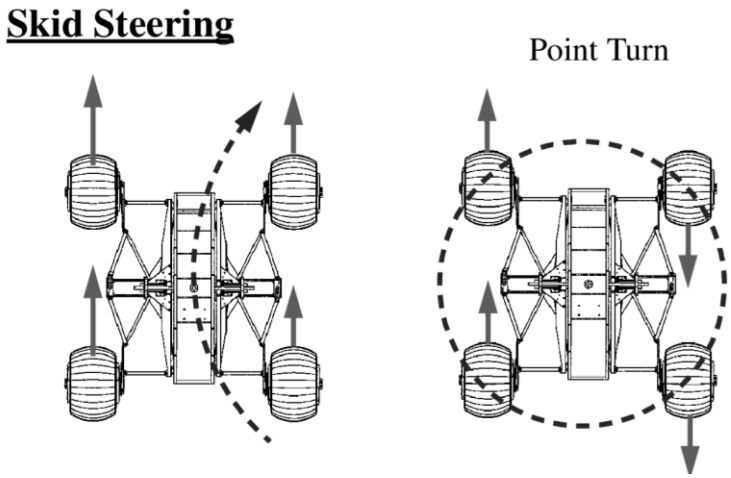


Figura 2-3. Esquema del funcionamiento de la configuración skid-steering

Este deslizamiento tiene una desventaja importante apreciable en la odometría. La odometría es la estimación de la posición de un vehículo con ruedas basándose en la información sobre la rotación de estas. Partiendo de esta definición es fácilmente deducible que el derrape de las ruedas empeorará la información obtenida mediante la odometría, ya que las ruedas patinando no demuestran el movimiento real del robot. Este inconveniente en nuestro caso no será demasiado perjudicial, ya que nuestro sistema consta de bastantes sensores y sistemas de detección.

Esta configuración tiene como ventaja una mayor tracción que viene dada por las varias ruedas motrices a cada lado del robot. Además, no tiene efecto de rueda giratoria, propio de la configuración diferencial. En esta, comúnmente, además de dos ruedas hay una rueda de apoyo que recibe el nombre de rueda giratoria o rueda caster, cuyo fin es otorgar de más estabilidad al robot. En la configuración skid-steering se prescinde de esta rueda debido a que hay más ruedas fijas a cada costado.

2.1.2 Componentes del robot

El SUMMIT-XL presente en el laboratorio contiene los componentes, sensores y accesorios mencionados a continuación:

- Un procesador embebido Intel Baytrail J1900 en el ordenador de a bordo con el sistema operativo Ubuntu 16.04 instalado, además de ROS *Kinetic Kame*.
- Un router inalámbrico RUT240 que permite establecer una conexión WI-FI 802.11n con una tasa de transferencia de datos de 300 Mbps.
- Una batería de litio-ferrofosfato o batería LiFePO4 recargable que dota de alimentación al robot con un voltaje de operación de 24 V.
- Un sistema de medida inercial (IMU) que consta de un acelerómetro 3D, un giroscopio 3D y un magnetómetro 3D en el chasis y cuyo modelo es VN-100T-CR de VectorNav.
- Un sistema RTK-GPS, cuyo modelo es el C94-M8P de u-blox. Consta de estación, batería y trípode.
- Una conectividad interna dada por USB, RS232 y GPIO.
- Una conectividad externa dada por USB, RJ45, 12VDC y batería.
- Cámara RGB-D Orbbec Astra montada en la parte delantera.
- Lidar 3D CE30-C Benewake sobre la cubierta.

Para este trabajo se tomará información dada por varios sensores que se encuentran en el robot. Entre ellos se encuentran la cámara RGB-D Orbbec Astra, el lidar 3D CE30-C Benewake, el modelo de GPS C94-M8P y una IMU VN-100T-CR. Se detallará el funcionamiento del GPS, el sistema de medida inercial y la cámara, ya que son los que más se enfocan en la tarea encomendada en este trabajo y el lidar toma más relevancia en las partes investigadas previamente del proyecto referentes a la detección y evitación de obstáculos.

2.2 Sistema RTK-GPS

2.2.1 Definición

El significado de las siglas RTK es Real Time Kinematic (posicionamiento cinemático en tiempo real). Es una tecnología basada en el sistema GNSS (*Global Navigation Satellite Systems*) y es usada para perfeccionar la precisión de la medida GPS. Los GPS tradicionales pueden determinar la posición un objeto con error de unos pocos metros (comúnmente entre dos y cuatro), en cambio, con el uso de RTK, este error se reduce hasta el error de centímetros.

El funcionamiento de un sistema GPS se basa en una técnica que recibe el nombre de trilateración. No se debe confundir con la triangulación, que sirve para medir ángulos y no distancias como es este caso. Los dispositivos GPS leen e interpretan las señales enviadas y para calcular la ubicación deben obtener la de al menos cuatro satélites, siendo lo más general que reciban de seis o más.

Con esta señal sabemos la distancia, pero no el ángulo, por lo que se forma una circunferencia con radio igual a esta distancia centrado en la posición del satélite, lo que indicaría que el dispositivo GPS se encontraría en algún lugar de esa circunferencia. Ocurriría exactamente lo mismo con el resto de los satélites disponibles por lo que el punto de intersección de las circunferencias se correspondería con la ubicación del dispositivo.

Pero, además, debemos tener en cuenta que el planeta no es un plano, sino que es una esfera, por lo que las circunferencias mencionadas anteriormente se corresponderían en realidad con esferas huecas, cuya intersección sería la posición del receptor GPS.

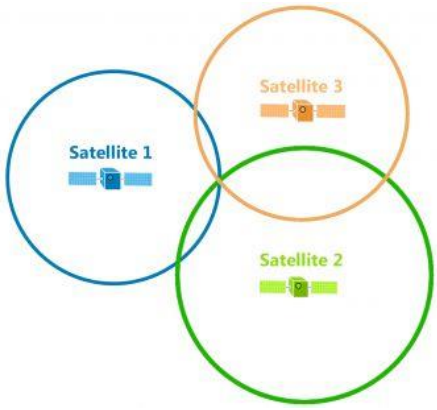


Figura 2-4. Trilateración en 2D

Figura 2-5. Trilateración en 3D

Sin embargo, esta técnica es sensible a varios errores que pueden afectar a la estimación de la posición que se desea conocer. Entre estos orígenes de errores se incluyen:

- **Dilución geométrica (GDOP) y de posición de precisión (PDOP).** Se trata de la disposición relativa de los satélites. Si estos se encuentran muy cerca entre sí reducen la calidad de la ubicación.

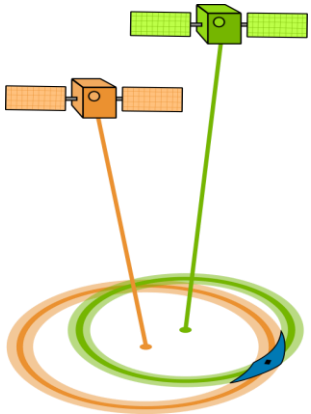


Figura 2-6. Efecto de un GDOP pobre

- **Efecto ionosférico.** La ionosfera es un medio dispersivo, es decir, la velocidad de propagación de una onda electromagnética depende de la frecuencia, por lo que el índice de refracción tiene efectos diferentes en la fase y en el código. Para eliminarlo, se lleva a cabo una combinación lineal de dos frecuencias que resultan en la conocida libre ionosfera.
- **Efecto troposférico.** Al contrario que en el caso anterior, la troposfera es un medio no dispersivo, por lo tanto, el índice de refracción es independiente de la frecuencia, dependiendo solo de los parámetros meteorológicos (humedad, temperatura y presión) y la longitud del recorrido.
- **Efectos multitrayecto.** Este efecto se produce cuando el dispositivo GPS se encuentra en un entorno con estructuras alrededor como montañas o edificios. Estos hacen que la señal rebote e interfiera con la que ha llegado de forma directa al receptor.
- **Hora del satélite.** Cualquier diferencia del reloj del emisor y el tiempo real produce una desviación en la ubicación calculada.
- **Lugar del satélite (Efemérides).** Si existe alguna discrepancia entre la posición y velocidades reales del satélite y las emitidas, también provoca un error en la estimación.
- **Disponibilidad selectiva.** Antes del año 2000, el Departamento de Defensa de los Estados Unidos degradó intencionalmente la exactitud del GPS. Mediante esta técnica, se agregaban entre 15 y 100 metros de error horizontalmente y 156 verticalmente. Solo los receptores militares de Estados Unidos y sus aliados podían decodificar este error. Debido al creciente uso del GPS por parte de la población civil, la disponibilidad selectiva fue eliminada.
- **Radiointerferencias.** Puede producirse una degradación en la señal debido a la numerosa cantidad de señales que pueden interferir.

Todo este cúmulo de errores pueden ser minimizados mediante el empleo de un sistema RTK-GPS.

2.2.2 Descripción del RTK-GPS

El sistema RTK, comúnmente conocido como DGPS (*Differential Global Positioning System*) en el ámbito del GPS, es un procedimiento de posicionamiento basado en la modificación de la información y se parte del mismo método de trilateración que en el GPS convencional, pero en este caso, hay dos receptores. La inclusión de este segundo receptor es lo que va a ayudar a reducir el error en la estimación de la posición. Este, recibe el nombre de estación de referencia y se coloca en una posición fija que se conoce con exactitud. Tras recibir la ubicación dada por los satélites, esta se compara con la posición ya conocida. Mediante esta comparación, se evalúan las diferencias entre ambas posiciones y se establecen una serie de correcciones para convertir la posición obtenida por la información de los satélites en la real. Estas correcciones son enviadas al dispositivo GPS cuya posición se quiere conocer. Estas correcciones son diferenciales de fases y para su cálculo se emplea la información de la portadora de la señal.

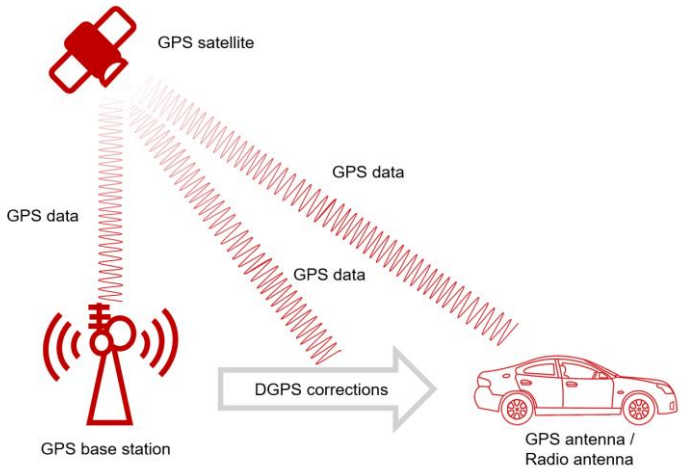


Figura 2-7. Gráfico sobre el funcionamiento de un RTK-GPS

2.2.3 C94-M8P-3

El sensor RTK-GPS disponible en el laboratorio se trata del modelo C94-M8P-3, fabricado y distribuido por la empresa u-blox. Esta placa tiene integrado un módulo NEO-M8P que sirve a su vez de estación base y de rover (receptor). Mediante el flujo de datos de corrección, el sistema es capaz de estimar la posición relativa del rover con una precisión del nivel de centímetros.

Además, incluye un enlace de radio UHF (*Ultra-High Frequency*). En nuestro caso se configurará a 433 MHz para cumplir con los requisitos europeos. Por otro lado, también consta de una serie de pines de conexión.

Las especificaciones del módulo según la hoja de datos del fabricante son los siguientes:

- Dimensiones de 75x55 mm.
- Peso de 35 g.
- Alimentación de 5 V mediante USB o por batería entre 3.7 V y 20 V.
- Temperatura de operación entre -40 ºC y 65 ºC.
- Pines para comunicación UART (3.3 V).
- Un puerto USB para la transmisión de datos GNSS y alimentación.
- Un RS232 para la configuración por radiofrecuencia.

Figura 2-8. Sensor RTK-GPS C94-M8P-3

2.3 Unidad de Medida Inercial (IMU)

2.3.1 Definición

Otro dispositivo indispensable en la elaboración de este trabajo se trata de la unidad de medida inercial presente en el SUMMIT-XL. Esta tiene la capacidad de medir e informar sobre el estado dinámico del robot incluyendo diferentes valores como intensidad del campo magnético, velocidad de rotación y aceleración gracias a magnetómetros, giroscopios y acelerómetros respectivamente. Es común encontrarla en vehículos aéreos y terrestres no tripulados, aeronaves, satélites, etc.

En un sistema de navegación, los datos proporcionados por la IMU son suministrados a una computadora, donde mediante una serie de algoritmos se calcula la posición y la velocidad. Estos algoritmos son capaces de

fusionar las medidas de los sensores integrados, de manera que, si algún dato es erróneo, puede desecharlo.

A continuación, se describirán los tres sensores presentes en la IMU a detalle:

- **Acelerómetros.** Son dispositivos que detectan fuerzas de aceleración, tanto estáticas como dinámicas. Además de medir las diferencias de velocidad, también cuantifica vibraciones y movimientos rotacionales. Hay diferentes tipos de acelerómetros entre los que se incluyen mecánicos, piezoeléctricos, capacitivos, piezoresistivos y térmicos.
- **Giroscopios.** Se tratan de instrumentos capaces detectar rotaciones y medir velocidades angulares. Hay diferentes tipos: mecánicos, MEMS capacitivos, y ópticos.

  Los mecánicos constan de un disco pesado rotando a gran velocidad que tiende a mantener la orientación del plano de giro gracias a su momento angular, siendo capaz de oponerse a pares que traten de desviarlo

  Los MEMS capacitivos consisten en dos masas que oscilan en oposición y se basan en el efecto de Coriolis.

  Los ópticos constan de dos rayos láser que viajan en sentidos opuestos dentro de un circuito cerrado y se basan en el efecto Sagnac. Este efecto demuestra que la diferencia de fase al llegar es proporcional a la velocidad angular.
- **Magnetómetros.** Miden la fuerza del campo magnético al que están sometidos. Existen dos grandes grupos: escalares y vectoriales. Mientras que los escalares solo miden la fuerza del campo, los vectoriales también pueden hallar la dirección.

2.3.2 IMU VN-100T-CR

La unidad de medida inercial que se encuentra en el SUMMIT-XL del laboratorio se trata del modelo VN-100T-CR. Aparte de una IMU, contiene un sistema de referencia de actitud y rumbo. Incluye sensores tales como acelerómetros 3D, giroscopios 3D, magnetómetros 3D, un sensor barométrico y un procesador de 32 bits.

Entre las especificaciones del sistema se encuentran:

	Acelerómetro	Giroscopio	Magnetómetro	Barómetro
Rango	± 16 g	±2000 º/s	±2.5 Gauss	10 a 1200 mbar
Varianza de Allan	< 0.04 mg	< 10º/hr (5-7º/hr tip.)	-	-
Densidad de ruido	0.14 mg/√Hz	0.0035 º / s / √Hz	140 µGauss/√Hz	-
Ancho de banda	260 Hz	256 Hz	200 Hz	200 Hz
Sensibilidad del eje	±0.05 º	< 0.05 º	±0.05 º	-

Tabla 2-1. Especificaciones de VN-100T-CR

Figura 2-9. VN-100T-CR

2.4 Cámara Orbbec Astra

El modelo Orbbec Astra es una cámara de gran calidad y fiabilidad. Es una cámara de tipo RGBD, es decir, proporciona tanto datos de profundidad (D) como de color (RGB) en tiempo real. Para captar la información de profundidad se hace uso de un sensor ToF y para el color, un sensor RGB. El funcionamiento de un sensor ToF (Time of Flight), como indica sus propias siglas, se basa en el tiempo de vuelo de la luz. Calcula la distancia mediante lo que tarda el haz de luz generado por el emisor de la cámara en llegar al obstáculo situado en frente y rebotar para ser recibido por el receptor. La luz generada es infrarroja y el sensor que la recibe es IR.

Figura 2-9. Cámara Orbbec Astra

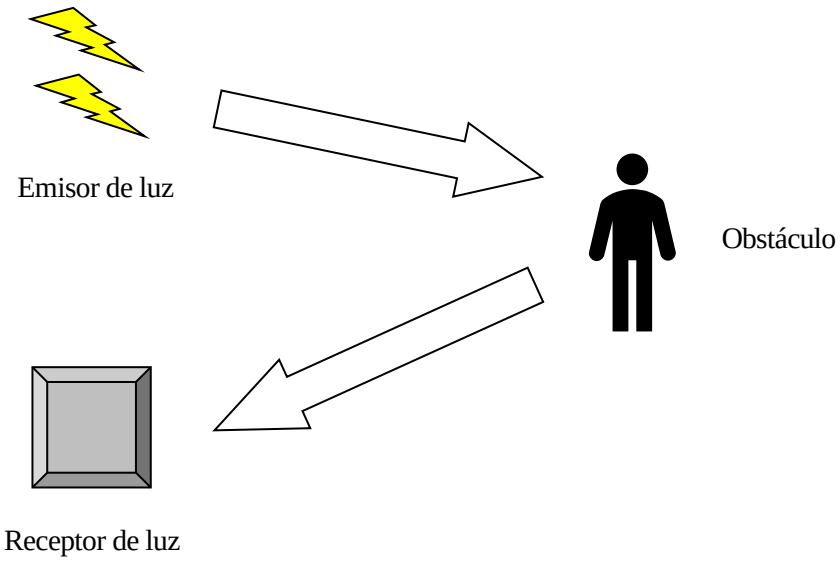


Figura 2-10. Funcionamiento de técnica ToF

Las especificaciones de este modelo son las siguientes:

- Campo de visión de 60º en horizontal, 49.5º en vertical y 73º de diagonal.
- Resolución de 640 x 480 a 30 FPS tanto para la imagen de profundidad como para la imagen de color.
- Tamaño de 165mm x 30mm x 40mm.
- Alimentación y transferencia de datos mediante USB 2.0.
- Precisión entre 1 y 3 mm para distancias en torno a 1 m.
- Sistemas operativos compatibles: Android, Linux y Windows.

Para su uso en ROS, y en concreto, en el visualizador RVIZ, se hará uso de la nube de puntos producida y será la base del funcionamiento de la detección de agujeros y pendientes. El lidar 3D presente en el robot también proporciona una nube de puntos, pero de menor densidad y abarca un menor rango de alturas que la cámara, por lo que se prefirió el uso de esta última.

3 DESCRIPCIÓN DEL SOFTWARE

Durante este capítulo se describirán todos los software empleados en el desarrollo del proyecto, mencionando sus principales características y funcionalidades, además de los conceptos y términos necesarios para su comprensión.

3.1 ROS

ROS (Robot Operating System) se trata de un middleware para el desarrollo e investigación de tecnologías relacionadas con la robótica. Fue desarrollado en 2007 por el Laboratorio de Inteligencia Artificial de Stanford. Propuso un gran avance en la innovación en el ámbito de la robótica, proporcionando una serie de herramientas para la creación o mejoras de proyectos.

Al ser tan altamente empleado, tiene una comunidad muy amplia, por lo que hay muchas aplicaciones y funcionalidades ya implementadas disponibles para su descarga en repositorios como Github y Gitlab. Además, hay diversos foros donde los usuarios resuelven problemas y plantean soluciones a errores que pueden ser de ayuda.

Figura 3-1. Logo de ROS

Cabe destacar que ROS no se trata de un sistema operativo. Podría definirse como un meta-sistema operativo ya que se ejecuta sobre otro sistema operativo, que en nuestro caso será Ubuntu 16.04 de Linux. ROS proporciona funciones como transmisión de datos entre diferentes hardwares, manejo de errores y planificación.

El funcionamiento de ROS se basa en una serie de nodos que son las unidades base dedicándose cada uno a una tarea por separado y siendo localizables entre sí gracias al Máster. Los nodos intercambian información mediante el empleo de *topics*, pudiendo tanto publicar mensajes como suscribirse a ellos, pero siendo inviable que realice ambas cosas a la vez. Esta interacción entre nodos también se puede llevar a cabo mediante servicios. Estos tienen un mecanismo similar al cliente/servidor. Un nodo registra el servicio y otro nodo puede obtener los datos solicitándolo. Al contrario que con los *topics*, en este caso la comunicación es bidireccional. Aquello que se publica en los *topics* o se recibe de ellos, se denominan mensajes. Estos son datos con una estructura determinada.

Una vez diseñado el sistema requerido en ROS, puede observarse un diagrama de la interconexión de los nodos y los *topics* mediante la introducción del comando *rqt_graph*.

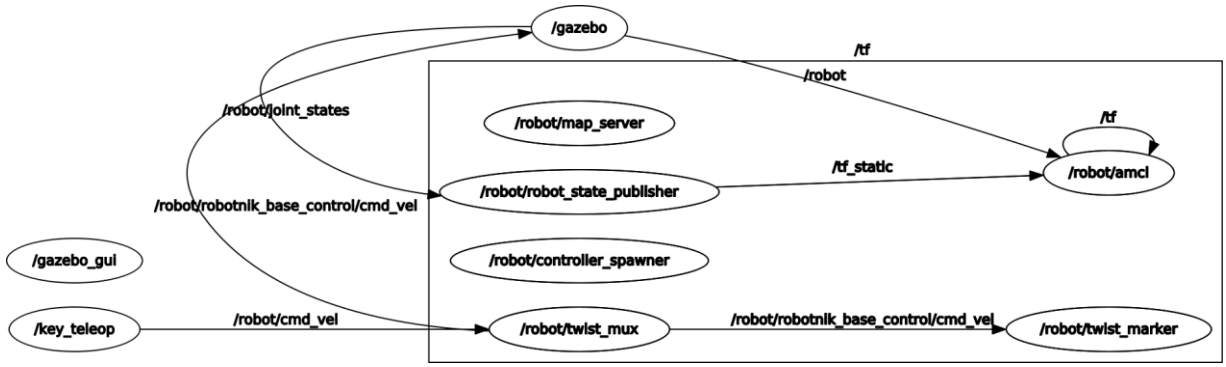


Figura 3-2. Rqt_graph

La versión de ROS seleccionada para el trabajo es ROS *Kinetic Kame*, que es su décima versión, ya que es la presente en el ordenador de a bordo del robot.

3.2 Gazebo

Gazebo es un simulador 3D de código abierto muy extendido en el campo de la robótica. Las físicas vienen dadas por defecto por el motor ODE, aunque este puede ser sustituido por otro. ODE está especializado en simular físicas de colisiones, vehículos, paisajes, etc. Por otro lado, el renderizado de gráficos es realizado por la interfaz de programación de aplicaciones OpenGL, incluyendo sombras, luminosidad y texturas de alta fidelidad.

Figura 3-3. Logo del simulador Gazebo

Cabe destacar que un apartado muy interesante que se puede simular es el ruido de los sensores. Gazebo está conformado principalmente por seis componentes:

- **Servidor.** Es el encargado de llevar a cabo un análisis de los archivos de la escena que se está tratando de simular.
- **Cliente.** Su objetivo es renderizar el ambiente gráfico de la simulación.
- **Archivos .world.** Estos archivos contienen la información necesaria para la simulación del entorno entre los que se incluyen iluminación, entorno, robot, etc.
- **Archivos .sdf.** Almacenan los modelos tridimensionales, sin tener que ser estos codificados cada vez que se requiera su uso.
- **Variables de entorno.** Su función es ofrecer soporte con la localización de archivos y comunicaciones entre cliente y servidor,
- **Plugins.** Otorgan funcionalidad a los modelos.

Gazebo es utilizado principalmente en unión con ROS. La versión de Gazebo seleccionada para la realización de simulaciones en este proyecto es la 8.0.

3.3 RViz

Es una herramienta de visualización 3D para proyectos de ROS. Es muy útil porque establece una fiel representación de la información detectada por los diferentes sensores tales como LIDAR, cámaras, IMU, etc.

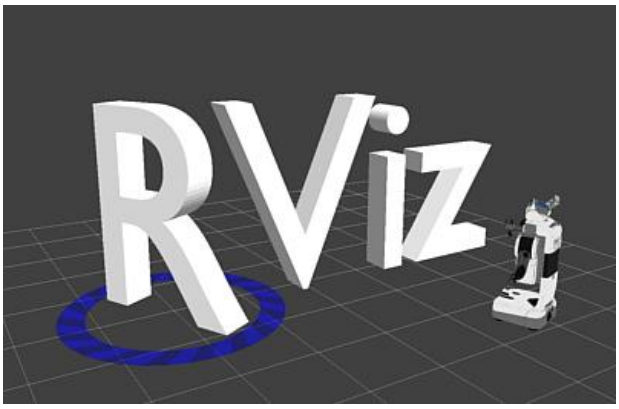


Figura 3-4. Logo de RViz

Al iniciar la simulación, aparecerá una interfaz similar a la presente en la siguiente imagen:

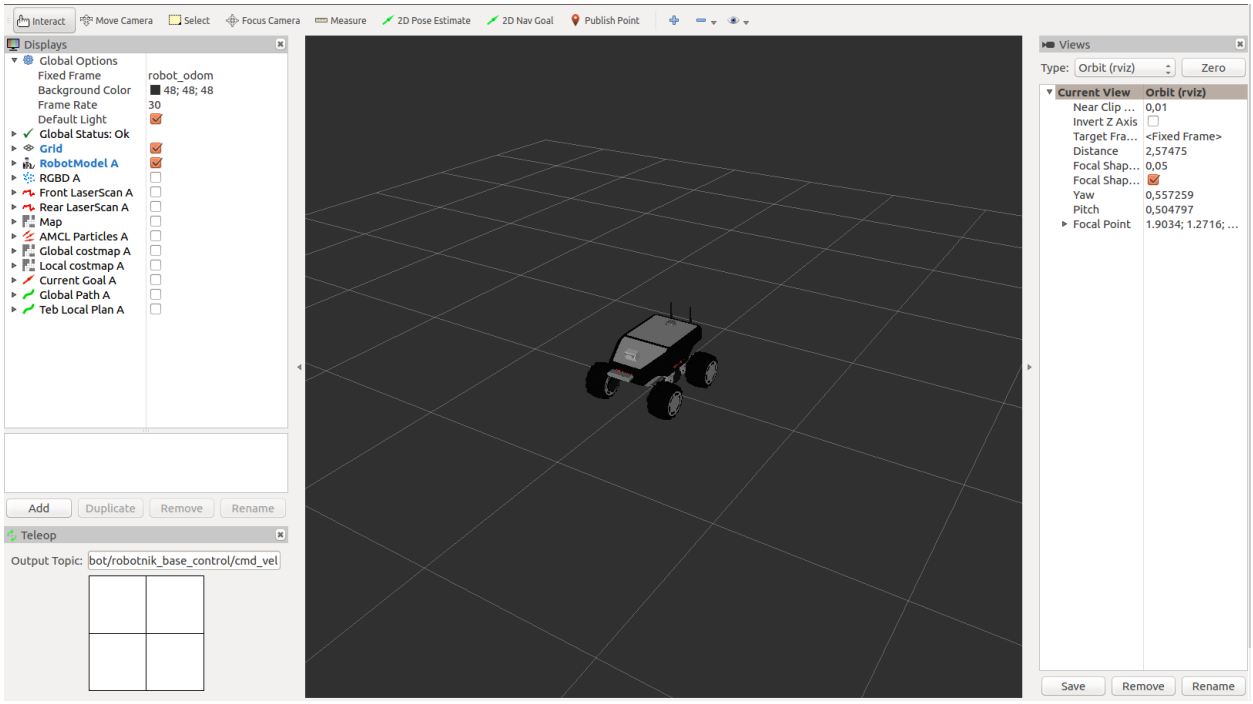


Figura 3-5. Interfaz de RViz

A la izquierda de la pantalla, se tiene un menú donde se puede realizar la selección de datos de sensores y funciones que se quieren monitorizar.

Si se quiere mostrar algo que no aparezca en dicho menú, debemos seleccionar la opción *Add* abajo a la izquierda y aparecerá la siguiente pestaña:

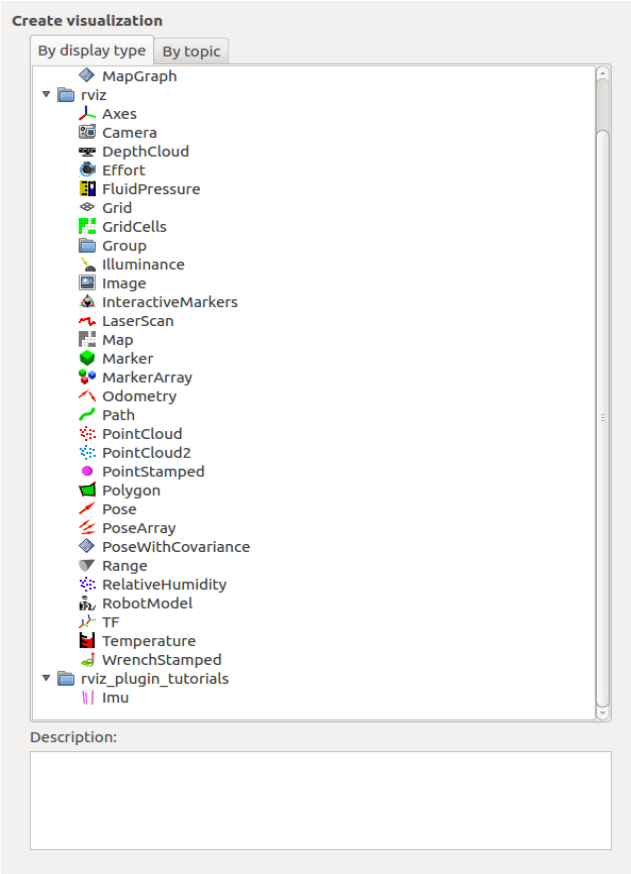


Figura 3-6. Menú de herramientas de visualización

Los tipos de visualización con mayor relevancia en el trabajo son:

- **Axes**. Muestra un sistema de coordenadas formado por tres ejes. El rojo coincide con el eje X, el verde, con el Y y el azul, con el Z.
- **Camera**. Crea una ventana donde se visualiza lo observado por la cámara.
- **PointCloud2.** Enseña una nube de puntos recibida por un mensaje de tipo sensor_msgs/PointCloud2, que es el que envía tanto el LIDAR como la cámara RGBD presentes en el Summit-XL.
- **Map.** Despliega el mapa proporcionado por el topic indicado sobre el plano del suelo.
- **Odometry.** Muestra las diferentes posiciones dadas por la odometría que se reciben a lo largo del tiempo.
- **Path.** Muestra el camino generado por los nodos de navegación que debe seguir el robot.
- **TF.** Expone el árbol de ejes de transformación.

3.4 U-Center

U-center se trata de un software desarrollado por u-blox para realizar una evaluación de la localización GPS que otorga el sensor conectado al PC. Hay dos versiones principales: u-center y u-center 2. El segundo se emplea para sensores de la propia u-blox de décima o mayor generación mientras que el primero para versiones anteriores.

Además, nos permite grabar las diferentes posiciones recibidas e incluso una vez tener el documento con estos datos, convertirlo en un archivo apto para importarlo en Google Earth y se nos muestre el camino realizado sobre el mapa real.

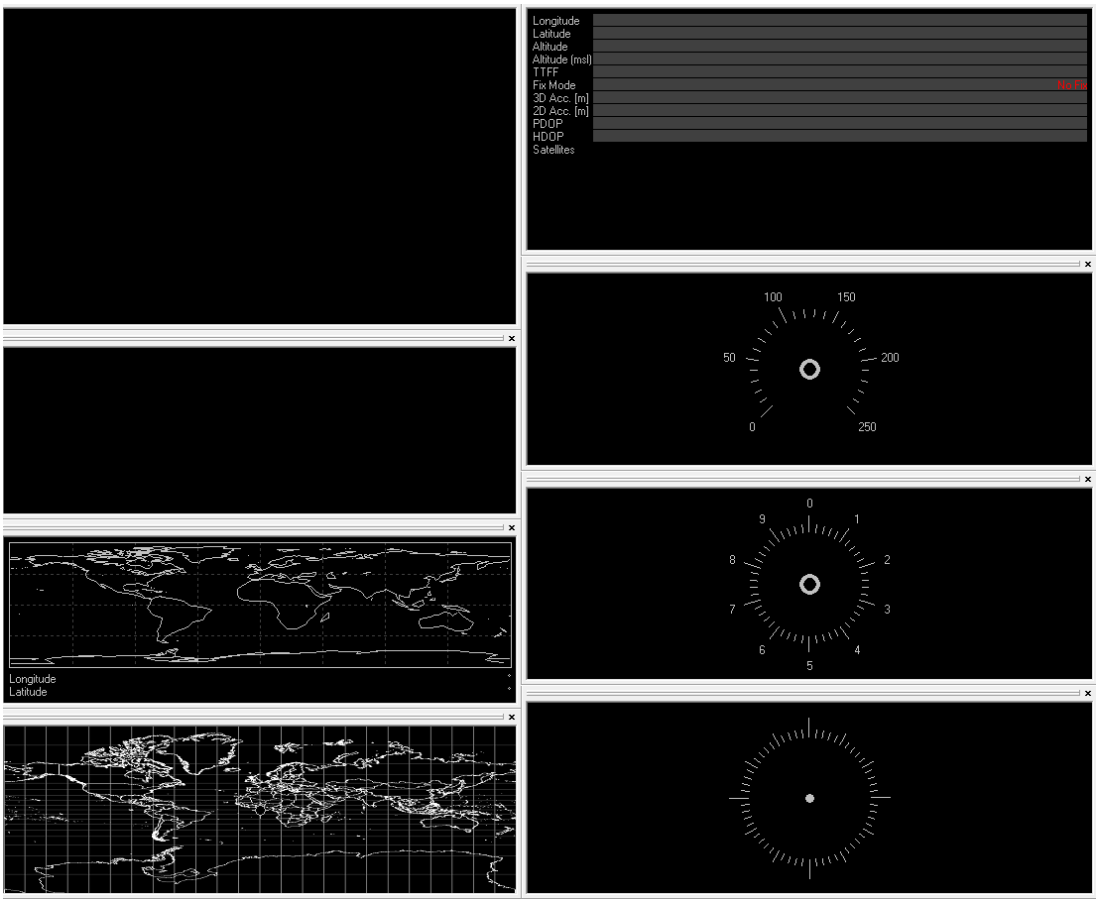


Figura 3-7. Panel de visualización de u-center

En este panel se puede observar diversas representaciones entre las que se encuentran: mapas con la posición de los satélites y el sensor conectado, una brújula, un reloj, aceleraciones, longitud, latitud, altitud, etc.

4 PAQUETES EMPLEADOS

En este capítulo se va a describir el desarrollo del trabajo para que cumpla con la función requerida, recogiendo tanto la base sobre la que se establece como la explicación de todos los paquetes empleados y los parámetros de los que constan.

4.1 Paquetes de ROS

Los paquetes de ROS constituyen la base del funcionamiento de ROS, ya que mediante ellos se pueden implementar diversas funcionalidades. En los paquetes se incluyen nodos, archivos de lanzamiento, archivos de configuración, un CMakeLists, etc. Estos suelen ser creados por usuarios, o bien, por desarrolladores de las empresas creadoras de robots o sensores. Como los paquetes normalmente son creados para unas ciertas condiciones y modelos específicos, muy probablemente al hacer uso de ellos se deban modificar ciertas partes de los códigos o algunos parámetros.

4.1.1 Stacks del SUMMIT-XL

Para el manejo del robot SUMMIT-XL, en el repositorio de GitHub se encuentran diferentes paquetes compartidos por la propia compañía que lo comercializa. Entre ellos se encuentran:

- **Summit_xl_common.** Proporciona descripciones URDF del robot y además de otros modelos como el Summit XL Steel y el Summit XL HL. Contiene los siguientes paquetes:
    - Summit_xl_control. Permite lanzar los controladores necesarios para el control del Summit XL.
    - Summit_xl_description. Incluye la descripción del Summit XL, archivos de lanzamiento, modelos .urdf, etc.
    - Summit_xl_localization. Contiene archivos de lanzamiento que permiten usar el filtro de Kalman extendido que ayudarán a llevar a cabo la localización del robot.
    - Summit_xl_navigation. Permite ejecutar los algoritmos del AMCL y navegación SLAM.
    - Summit_xl_pad. Posibilita el manejo del robot mediante diferentes tipos de controles entre los que se incluye el mando DualShock 4 de PS4.
    - Summit_xl_perception. Contiene nodos con labores relacionadas a los sistemas de percepción.
    - Summit_xl_robot_local_control. Incluye lo necesario para ejecutar las librerías relacionadas al control local del robot.
- **Summit_xl_sim.** Incorpora los paquetes necesarios para efectuar la simulación del Summit-XL.
- **Robotnik_Sensors.** Incluyen los modelos de diversos sensores.
- **Robotnik_msgs.** Define los mensajes y servicios utilizados por los paquetes desarrollados por Robotnik.

4.1.2 Paquete u-blox

Como el trabajo ha sido realizado en la versión *Kinetic* de ROS, el paquete de sensores descargado es el referente a la rama de esta versión en GitHub. Sin embargo, el archivo .xacro encargado del funcionamiento

del sensor GPS de u-blox fue desarrollado posteriormente para Melodic, que es posterior a la empleada. Como simplemente implementa un sensor, fue descargado de esta rama y añadido al directorio donde se encuentran el resto de los sensores. Además, para su modelo de simulación, se requiere del archivo .stl correspondiente a la antena que se encuentra también en la misma rama.

Para desarrollar el comportamiento del GPS, el archivo *ublox.urdf.xacro* recurre al archivo libhector_gazebo_ros_gps.so. Este se encuentra dentro de la librería de plugins para gazebo creada por Team Hector. En ella se incluyen una IMU, un sensor del campo magnético terrestre, un sonar, un control para un robot con configuración diferencial de seis ruedas y el GPS.

Con lo que respecta al plugin del GPS, este publica dos tipos de mensajes:

- sensor_msgs/NavSatFix. Proporciona la altitud, la longitud y la latitud del robot en coordenadas WGS84. WGS84 se trata de un sistema de coordenadas geodésico que define cualquier punto del planeta mediante tres coordenadas (x, y, z). El origen de este sistema se describe en el archivo .xacro.
- geometry_msgs/Vector3Stamped. Otorga el vector de velocidad GNSS en coordenadas NWU (north, west, up).

4.2 Configuración de los nodos referentes a la localización GPS

4.2.1 Robot_localization_odom

Mediante este archivo .launch se ejecuta el nodo ekf_localization_node. Este, aplica el filtro de Kalman extendido para la localización del robot, que proporciona una estimación de esta, partiendo de la odometría y las medidas de los sensores para sistemas no lineales. Consta de dos fases: predicción y corrección. En la etapa de predicción, a partir del estado anterior del sistema se estima el estado actual. En la de corrección, se lleva a cabo una rectificación mediante las medidas otorgadas por los sensores.

Partimos de que un sistema dinámico no lineal queda descrito mediante la siguiente expresión:

$$x_k = g_k(x_{k-1}, u_k) + \varepsilon_k$$

donde x_k es el estado del sistema, g_k, el vector de transición y ε_k, el ruido que viene denotado por R.

Por otro lado, se tiene la ecuación de medidas:

$$z_k = h_k(x_k) + \delta_k$$

donde h_k indica la relación dada entre el vector de estado y las variables medidas en el sistema. En el caso en que tanto vector de estado como variables medidas coincidan, h_k tendrá un valor igual a x_k. δ_k se corresponde con el error y su matriz de covarianza asociada es Q.

El filtro se basa en las siguientes ecuaciones, donde se aplica el jacobiano a las dos expresiones vistas anteriormente:

$$G_k = \frac{\partial g_k(x_{k-1}, u_k)}{\partial x_{k-1}} = \begin{bmatrix} 1 & 0 & \frac{v_k}{w_k}(\cos(\theta_{k-1} + w_k\Delta_t) - \cos(\theta_{k-1}) \\ 0 & 1 & \frac{v_k}{w_k}(\sin(\theta_{k-1} + w_k\Delta_t) - \sin(\theta_{k-1}) \\ 0 & 0 & 1 \end{bmatrix}$$

$$H_k = \frac{\partial h_k(x_k)}{\partial x_k} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R = Q = \begin{bmatrix} \sigma^2{}_x & 0 & 0 \\ 0 & \sigma^2{}_y & 0 \\ 0 & 0 & \sigma^2{}_z \end{bmatrix}$$

Una vez descritas estas matrices podemos determinar las dos etapas del filtro. La etapa de predicción está compuesta por las siguientes ecuaciones:

$$\hat{x}_k^- = g(x_{k-1}, u_k) + w_k$$

$$P_k^- = G_k P_{k-1} G_k^T + R$$

Donde $\hat{x}_k$ se corresponde con la media y P_k, con la matriz de covarianza del error.

Por otro lado, las ecuaciones de la fase de corrección son:

$$K_k = P_k^- H_k^T (H_k P_k^- H_k^T + Q)^{-1}$$

$$\hat{x}_k = \hat{x}_k^- + K_k (z_k - h(\hat{x}_k^-))$$

$$P_k = (I - K_k H_k) P_k^-$$

Partiendo de las salidas de la fase de predicción, se trata de hallar la ganancia de corrección del sistema, K_k.

Este filtro descrito, se implementa mediante el ejecutable del nodo ekf_localization_node, como se explicó anteriormente. Para su correcto funcionamiento, hay que definir varios parámetros:

Tabla 4–1 Parámetros del nodo ekf_localization_node

Parámetros	Valor	Descripción
Frecuency	45 Hz	Frecuencia a la que se produce cada estimación de estado.
Sensor_timeout	0.1	Tiempo máximo de espera para recibir un nuevo valor de los sensores. En caso, de no recibir nada, solo se llevará a cabo la fase de predicción.
Two_d_mode	True	Mediante la activación de esta variable booleana, cancelamos los valores recibidos por la imu relativos al eje z, balanceo y cabeceo.
Odom_frame	$(arg prefix)odom	Marco de coordenadas de la odometría.
Base_link_frame	$(arg prefix)base_footprint	Marco de coordenadas de la base del robot.
World_frame	$(arg prefix)odom	Marco de coordenadas del mundo, que coincide con la odometría, ya que es lo conveniente en el caso de que solo se empleen odometría e IMU.
odom0	/robot/robotnik_base_control/odom	Configura el parámetro con los valores recibidos por el topic referente a la odometría.
Imu0	/imu/data	Configura el parámetro con los valores recibidos por el topic referente a la IMU.

Odom0_config	Elementos correspondientes a la posición en *x* y en *y* y a sus derivadas a *true* y lo demás a *false*	Definimos los valores proporcionados por la odometría que queremos utilizar en la fase de corrección del filtro.
Imu0_config	Elementos correspondientes a la guiñada y su derivada a *true* y lo demás a *false*	Definimos los valores proporcionados por la IMU que queremos utilizar en la fase de corrección del filtro.
odom0_differential	False	Se declara si los valores de pose de la odometría deben integrarse de forma diferencial.
imu0_differential	False	Se declara si los valores de pose de la IMU deben integrarse de forma diferencial.
odom0_relative	False	En caso de estar activado, los valores de odometría recibidos se fusionan con la primera medida.
imu0_relative	False	En caso de estar activado, los valores recibidos de la IMU se fusionan con la primera medida.

Tabla 4–2 Topics utilizados en el nodo ekf_localization_node

Topic	Mensaje	Descripción
/robot/robotnik_base_control/odom	45 Hz	Recibe los valores de odometría proporcionados por los encoders de las ruedas del robot.
/imu/data	sensor_msgs/Imu	Recibe los valores proporcionados por la IMU.
/odometry/filtered	Nav_msgs/odometry	Publica el valor de odometría tras aplicar el filtro.

4.2.2 Nav_sat_transform

Con este .launch se pone en ejecución el nodo navsat_transform_node, el cual toma los valores proporcionados por el topic de salida /odometry/filtered del nodo que ejecuta el Filtro de Kalman Extendido y los datos que otorgue el GPS. A su vez, produce un mensaje de odometría de mayor calidad para la localización del robot.

Tabla 4–3 Parámetros utilizados en el nodo navsat_transform_node

Nombre	Valor	Descripción
magnetic_declination_radians	0	Mediante este valor se determina el grado de inclinación magnética.
yaw_offset	0	Este parámetro se utiliza para agregar un desplazamiento al valor de yaw, en el caso de que la IMU, al estar orientada al este no sea 0.
use_odometry_yaw	False	Al estar en false se deniega que se obtenga yaw por la odometría, sino por la IMU.
wait_for_datum	False	Si es verdadero, se esperará obtener datos del parámetro de referencia y el servicio set_datum
zero_altitude	True	En el caso de que valga True, el valor en Z producido por este nodo en el mensaje de la odometría será nulo.
broadcast_utm_transform	True	Si se define como verdadero, enviará la transformación entre la cuadrícula UTM y el marco de los datos entrada de la odometría.
publish_filtered_gps	True	Al ser verdadero, se transformará la posición del marco del mundo a coordenadas GPS.

Tabla 4–4 *Topics* utilizados en el nodo navsat_transform_node

Topic	Mensaje	Descripción
/imu/data	sensor_msgs/Imu	Recibe los valores proporcionados por la IMU.
/gps/fix	sensor_msgs/NavSatFix	Recibe los valores de posición en base a coordenadas GPS del robot.
/odometry/filtered	Nav_msgs/odometry	Obtiene la posición del robot dada por el Filtro de Kalman Extendido explicado anteriormente.

/odometry/gps	Se publican la posición del robot basada en GPS en mensaje odométrico aplicada al marco de coordenadas del robot
/gps/filtered	Se publica opcionalmente la posición en GPS del robot en el marco de coordenadas del mundo

4.3 Configuración de los nodos referentes a la detección de agujeros

Para la detección de obstáculos a un nivel por debajo del suelo, es decir, agujeros en el terreno, se ha optado por el uso del paquete iri_point_cloud_hole_detection.

Se recibe una nube de puntos procedente de la cámara del robot, y dependiendo de si el número de puntos se encuentra por encima o por debajo de cierto umbral establecido, se considera zona transitable u obstáculo.

Tabla 4–5 Parámetros utilizados en el nodo iri_point_cloud_hole_detection

Nombre	Valor	Descripción
hole_min_p	50	Umbral mínimo de puntos para que una región pueda ser considerada un agujero.
num_cells	3	Número de zonas dentro de la región detectada.
box_y	1.6	Tamaño de la región de detección de agujeros en el eje *y* con respecto al marco de coordenadas base_link (en metros).
box_x_ini	-0.5	Posición inicial de la región de detección de agujeros en el eje y con respecto al marco de coordenadas base_link.
box_x_end	0.5	Posición final de la región de detección de agujeros en el eje y con respecto al marco de coordenadas base_link (en metros).
box_z_ini	-0.1	Altura inicial de la región de detección de agujeros en el eje y con respecto a base_link.
box_z_end	0.1	Altura final de la región de detección de agujeros en el eje y con respecto al marco de coordenadas base_link (en metros).

Tabla 4–6 *Topics* utilizados en el nodo iri_point_cloud_hole_detection

Topic	Mensaje	Descripción
/robot/front_rgbd_camera/depth/points	sensor_msgs::PointCloud2	*Topic* por el que se recibe la nube de puntos. En este caso será el otorgado por la cámara.
/ana/point_cloud_hole_detection/hole_obs_cloud_out	sensor_msgs::PointCloud2	Contiene la misma nube de puntos que a la entrada, pero con las regiones de interés en diferente color
/ana/point_cloud_hole_detection/hole_zone_cloud_out	sensor_msgs::PointCloud2	Nube de puntos donde se señalan las zonas donde no se supera el mínimo de puntos requerido.

4.4 Configuración de los nodos referentes a la detección de pendientes

Para la detección de pendientes escalables por el robot, es decir, por debajo de cierto grado de inclinación se ha elegido el paquete iri_obstacle_detection_normals, de los mismos desarrolladores que el de detección de agujeros. Su funcionamiento se basa en la nube de puntos de la cámara al igual que en el caso anterior. Según la información que le llegue en el eje Z, distinguiremos cuando una pendiente es transitable o no.

Tabla 4–7 Parámetros utilizados en el nodo iri_obstacle_detection_normals

Nombre	Valor	Descripción
filter_min_dist	0.28	Altura a partir de la cual se considera que la nube está interceptando un obstáculo.
normal_KSearch	30	Número de vecinos necesarios para estimar una superficie.
max_inc	0.3	Inclinación máxima especificada para que una pendiente se considere escalable por el robot.

Tabla 4–8 *Topics* utilizados en el nodo iri_obstacle_detection_normals

Topic	Mensaje	Descripción
/robot/front_rgbd_camera/depth/points	sensor_msgs::PointCloud2	Topic por el que se recibe la nube de puntos otorgada por la cámara RGBD.
normal_KSearch	sensor_msgs::PointCloud2	Número de vecinos necesarios para estimar una superficie
max_inc	sensor_msgs::PointCloud2	Inclinación máxima especificada para que una pendiente se considere escalable por el robot.

4.5 Creación de mapa propio para pruebas

Inicialmente, se pensó en diseñar un mapa en Gazebo para realizar pruebas de movimiento y principalmente para ayudar con la visualización de los sensores en un principio y si su funcionamiento era correcto.

Con gazebo de base solo se pueden crear paredes y obstáculos tales como cubos y esferas. Para realizar un mapa con obstáculos más diversos se optó por descargar los modelos de *Open Robotics* disponibles en su GitHub.

Una vez disponibles estos nuevos modelos en nuestro entorno, creamos el mapa. Para empezar, en el menú de herramientas seleccionamos la opción Building Editor.

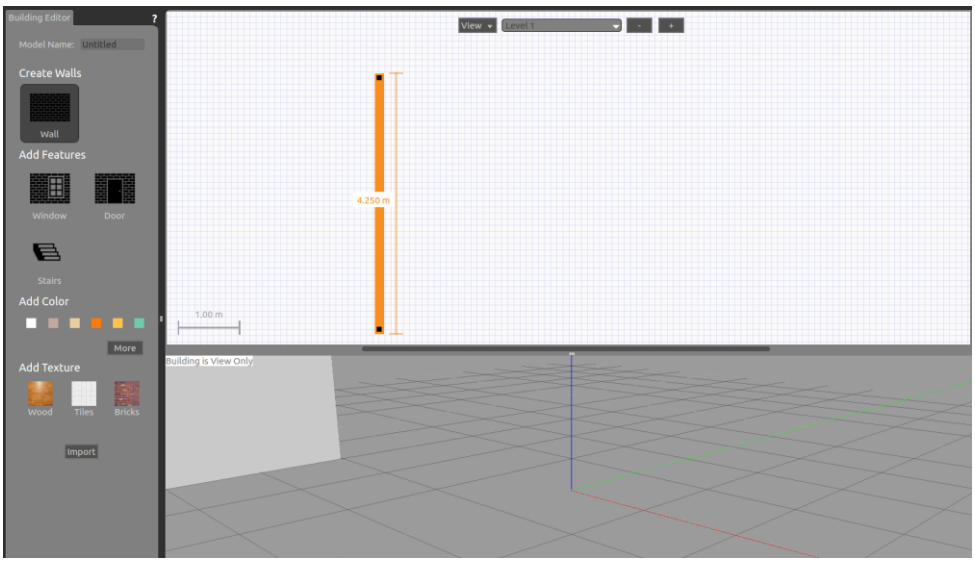


Figura 4-1. Building Editor de Gazebo

En él, elegimos la opción *Wall* y en la cuadrícula de la derecha creamos un espacio de cuatro paredes del tamaño requerido. En este caso se optó por 12 m. Además, se puede seleccionar una textura para la pared. Para este mapa se escogió la de madera.

Una vez añadidas las cuatro paredes, se deben agregar los modelos dentro de ellas. Para ello, se accede al menú *Insert* y se arrastran hacia la cuadrícula del mapa los modelos que se deseen. Con el fin de incorporar entidades diversas y con formas variadas, el mapa final quedó así:

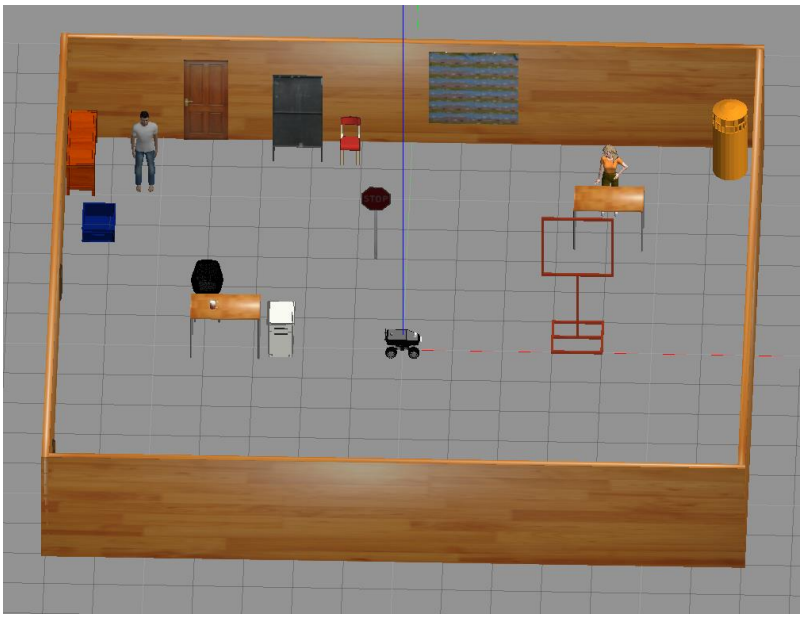


Figura 4-2. Mapa creado

4.6 Launcher para crear modelo del robot

Cuando se realizaron las pruebas con el robot real, inicialmente el Summit-XL no aparecía en la visualización de RVIZ. Los ejes de coordenadas asociados a diferentes elementos del robot (ruedas, odometría, GPS, base, …). Este error se producía ya que al ejecutar RVIZ no tenía capacidad para acceder al modelo *.xacro*. Para solucionar esto, se creó un archivo de lanzamiento que permitía la correcta búsqueda del archivo *Summit_xl_std.urdf.xacro* y se produce un cambio de topic desde */joint_states* a */robot/joint_states*.

5 DESARROLLO DE LOS RESULTADOS

Durante este capítulo se explicarán todos los pasos que se han llevado a cabo para un correcto funcionamiento del sistema, además de las diversas pruebas con el robot. Los experimentos realizados se han efectuado tanto en simulación como con el Summit-XL del laboratorio.

5.1 Perfeccionamiento del funcionamiento del GPS

Para la localización GPS, emplearemos uno de los paquetes presentes en el *stack* del Summit-XL. Se trata de *Summit_xl_localization*. Para la obtención de las medidas GPS inicialmente, se deberán seguir los pasos descritos en el anexo de este propio trabajo donde se añade el sensor GPS de u-blox al conjunto de sensores operativos del robot.

Para llevar a cabo la localización, se hace uso de dos lanzadores: *robot_localization_odom* y *navsat_transform_node*. Mediante el primer *launcher* ejecutamos el nodo que aplica el Filtro de Kalman Extendido explicado previamente en el capítulo 4. Mediante la activación de este nodo obtenemos la posición estimada basada en la odometría. Este valor se le otorga al nodo *navsat_transform_node*, que se encarga de producir un nuevo mensaje de odometría, pero con una mejora llevada a cabo gracias al apoyo de IMU y GPS. Además, al ver los *topics* a los que se publican ambos nodos, no coinciden con los que está publicando el sistema general. Para ello se debe realizar un *remapeo* de dichos *topics* a los que está suscrito o publica estos dos nodos.

Primeramente, se probará su funcionamiento en un mapa vacío:

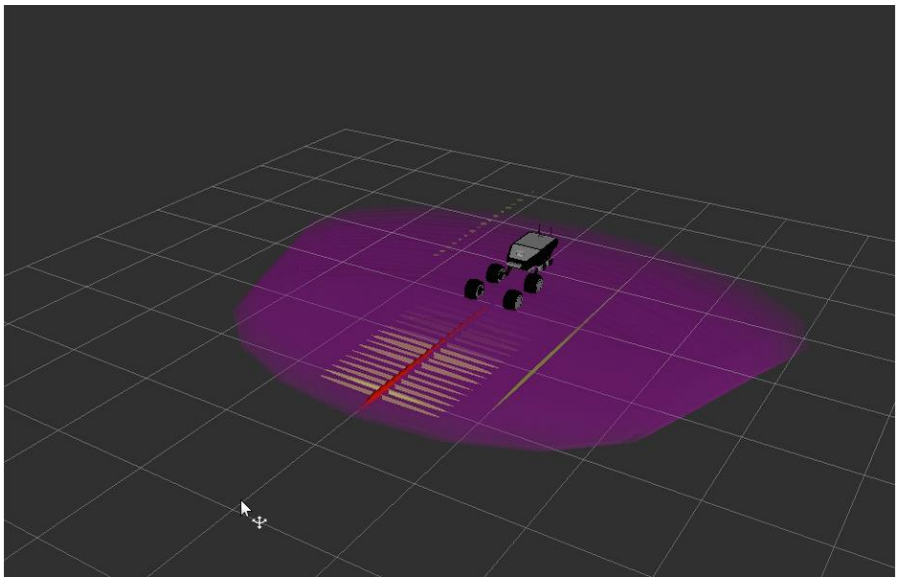


Figura 5-1. Primera prueba de localización

Como se aprecia en la imagen, la localización no funciona correctamente ya que las estimaciones de posición son extremadamente alejadas de la posición real. Se seleccionó una trayectoria recta hacia delante, pero las estimaciones de odometría se representan como una línea recta hacia atrás y el robot se teletransporta a lo largo de todo el mapa. El primer cambio realizado para una mejora del resultado es desactivar la opción de configuración diferencial para la odometría y activar la relativa.

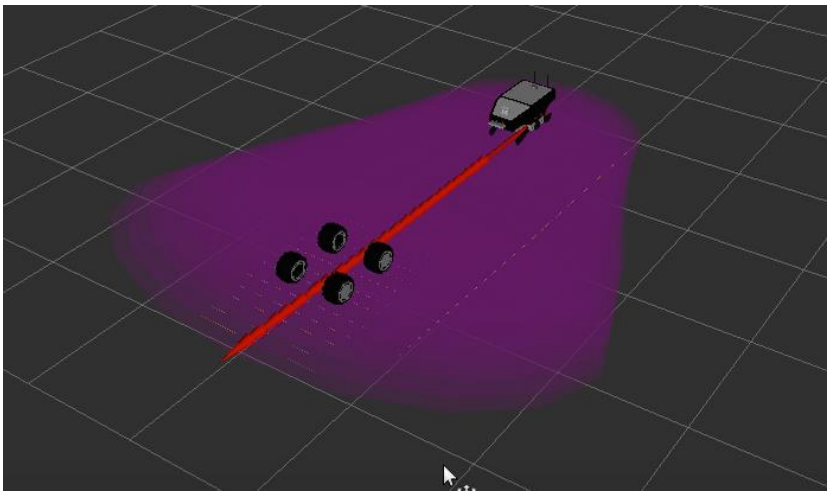


Figura 5-2. Segunda prueba de localización

Tras esto, el robot sigue sin ser localizado con exactitud. Al analizar el nodo *ekf_localization_node*, vemos que está suscrito a unos *topics* donde no se está publicando nada, porque estos *topics* no coinciden con los que están publicando los sensores. Esto se debe a que al poner en marcha la simulación mediante el lanzador *Summit_xl_complete.launch* se le añade el prefijo */robot*. Para enviar todo lo recibido en este *topic* al que lee *ekf_localization_node* se recurre a la función *remap*. La estructura de uso de esta función es la siguiente:

```
<remap from="topic_original" to="topic_nuevo"/>
```

Al hacer este cambio, vemos que las medidas, aunque con algo de error, ya que el robot oscila continuamente alrededor de la posición real, son bastante mejores que las anteriores:

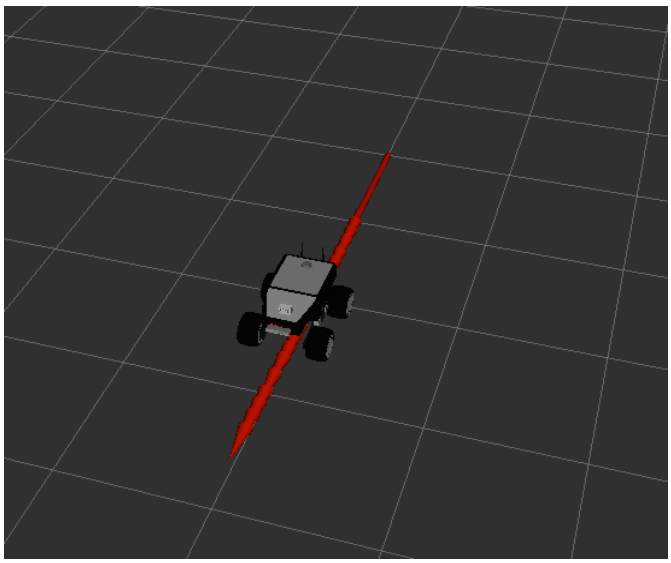


Figura 5-3. Tercera prueba de localización

Ahora, el problema es ajustar la medida para que no se desplace alrededor de la posición real, sino que se mantenga en ella. Para solucionar este problema se probaron diversas cosas:

- Cambiar los parámetros que se obtenían de odometría e IMU.
- Añadir una matriz de covarianza de ruido, denominada con la letra Q en la explicación del filtro del Kalman extendido del capítulo anterior. Este es un parámetro que es bastante difícil de ajustar, así que

inicialmente se optó por utilizar la presente en otro archivo de lanzamiento de localización por odometría.

- Activar la odometría y/o IMU relativas o diferenciales.

Después de probar toda esta serie de cambios y diferentes combinaciones de estos se llegó a una configuración con la que el robot se ajusta con exactitud a la posición donde debe estar. En esta se descarta cualquier valor procedente de la IMU poniendo en *false* todos los parámetros de la matriz de configuración (X, Y, Z, roll pitch, yaw, $\dot{X}, \dot{Y}, \dot{Z}, \dot{roll}, \dot{pitch}, \dot{yaw}, \ddot{X}, \ddot{Y}, \ddot{Z}$), excepto los referentes a la guiñada (yaw y $\dot{yaw}$). En cuanto a la configuración de la odometría se activa la recepción de X, Y y sus derivadas. Además de esto, es importante que se mantengan desactivadas la configuración relativa y diferencial tanto para la IMU como para la odometría.

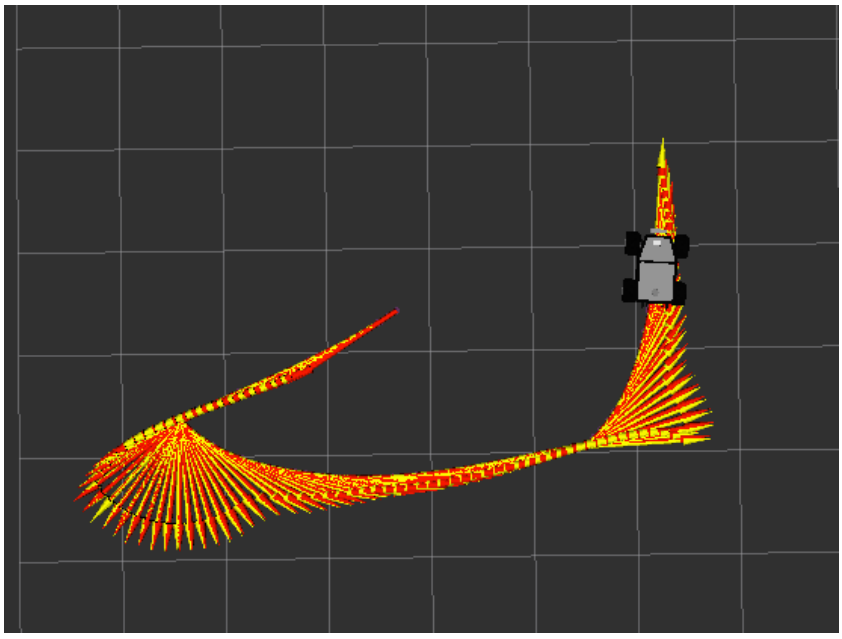


Figura 5-4. Comparación entre odometría corregida por GPS y posición del robot

5.2 Perfeccionamiento del funcionamiento de la detección de agujeros

Tras descargar el paquete *iri_point_cloud_hole_detection* y añadirlo a nuestro espacio de trabajo, se ejecutó el archivo de lanzamiento *node.launch*. En primera instancia, en el visualizador RVIZ, vemos lo siguiente:

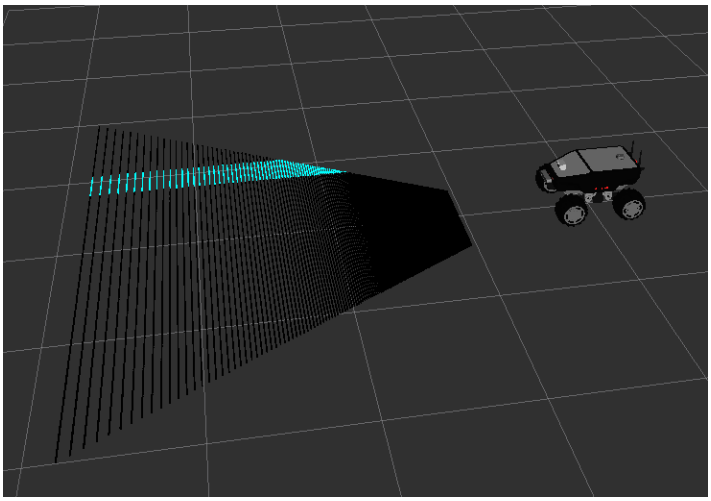


Figura 5-5. Primera prueba de detección de agujeros

Observamos que la nube de puntos ya procesada por el nodo de detección de agujeros se ve entera negra, mientras que lo correcto sería que hubiera una zona con diferentes tonos de azul, que es la sección en la cual se lleva a cabo la detección. Además, la nube de puntos artificial que debe aparecer cuando la zona de detección coincide con un área a menor altura que el suelo se muestra casi siempre. La aparición de esta se debe a que, al no mostrarse la demarcación azul, los puntos de la zona son lógicamente menores que el umbral, lo que propicia la aparición de la nube de puntos artificial. Pero por otra parte vemos que se muestra a la derecha del robot, mientras que en el repositorio nos muestran que se tendría que ver en la parte frontal del robot.

Luego, se prueba que sucede al interponer un obstáculo en la nube de puntos. Para ello se posiciona el robot frente a una rampa.

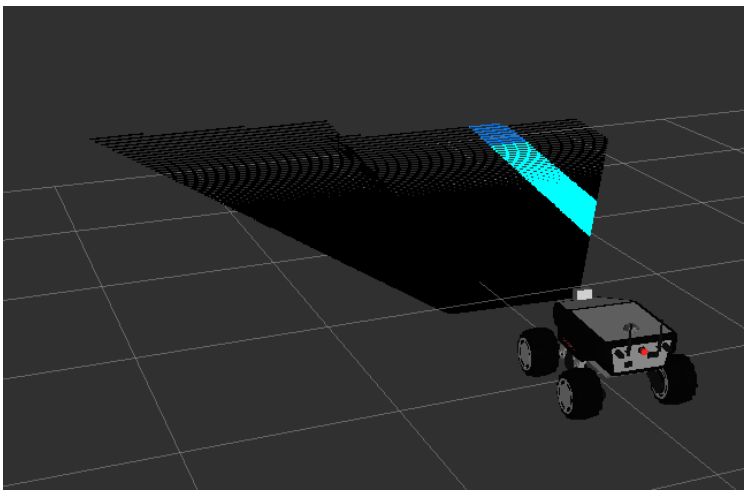


Figura 5-6. Segunda prueba de detección de agujeros

Al interceptar la nube de puntos el obstáculo, se aprecia que en la parte más lejana aparece la zona azul. Como el área parece aparecer en el límite más lejano se probó a reducir considerablemente el parámetro que especifica la frontera en el eje X de la zona de detección.

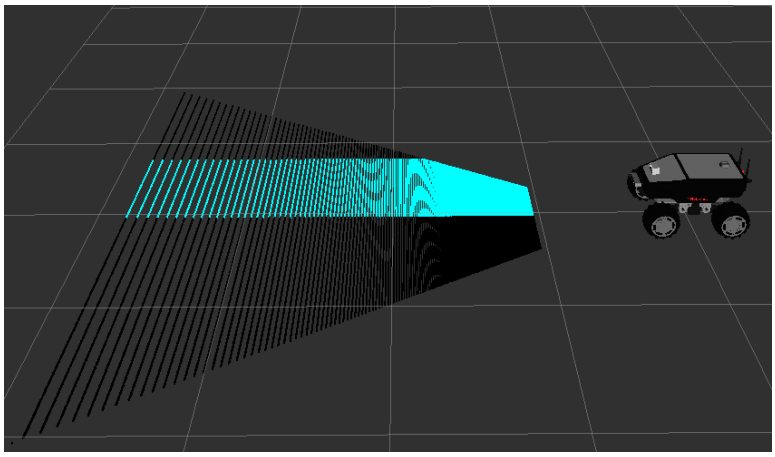


Figura 5-7. Tercera prueba de detección de agujeros

Al simular de nuevo, el área de detección ha quedado desplazado en el eje Y, por lo que el principal problema puede deberse a un intercambio de los ejes X e Y. Para comprobarlo, se cambia en el código la parte donde se delimita el área en Y, se altera la variable cloud_in.points.y por cloud_in.points.x en la condición.

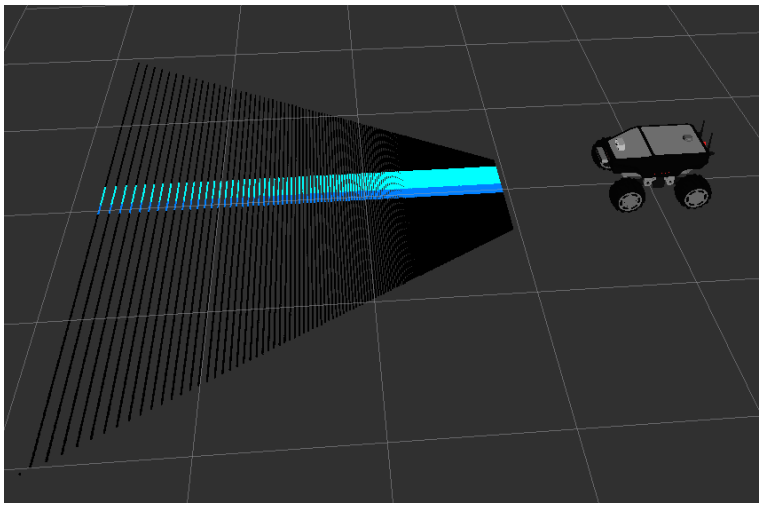


Figura 5-8. Modificación del algoritmo de detección de agujeros

Una vez realizado el cambio, ya se representa la región de detección, aunque esta abarca toda la nube de puntos, en vez de un rectángulo dentro de esta. A continuación, se hacen experimentos cambiando los valores límites tanto en X como en Y y ver cómo afecta esto al tamaño y forma del sector.

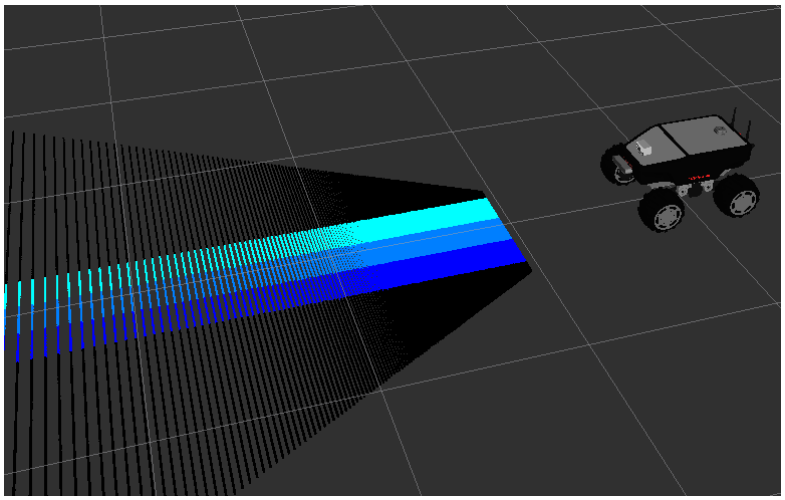


Figura 5-9. Modificación de los valores del algoritmo de detección de agujeros

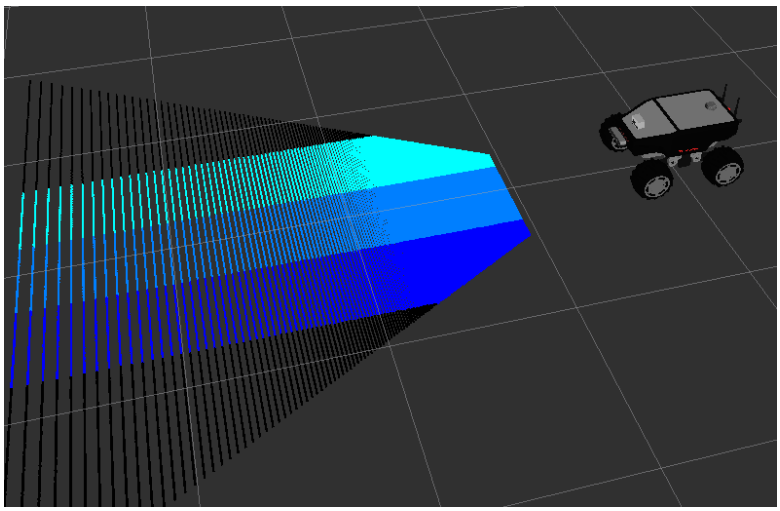


Figura 5-10. Modificación de los valores del algoritmo de detección de agujeros

En las imágenes se percibe que los cambios en ambas variables solo afectan al eje X (del sistema de coordenadas del propio algoritmo de detección de agujeros). Los cambios en los límites en Y impactan en el ancho de la región, mientras que los cambios en X realizan un desplazamiento en este propio eje. Es lógico que los cambios en Y alteren el ancho de la zona, porque el umbral está calculado con este propio valor en el código original. Sin embargo, también debería afectar a la longitud de esta, ya que forma parte de una de las condiciones para su creación.

Con el fin de comprobar la causa de la inacción de los cambios en Y con respecto a la longitud, se realiza una impresión por línea de comandos de los valores de *cloud_in.points.y* a lo largo del tiempo:

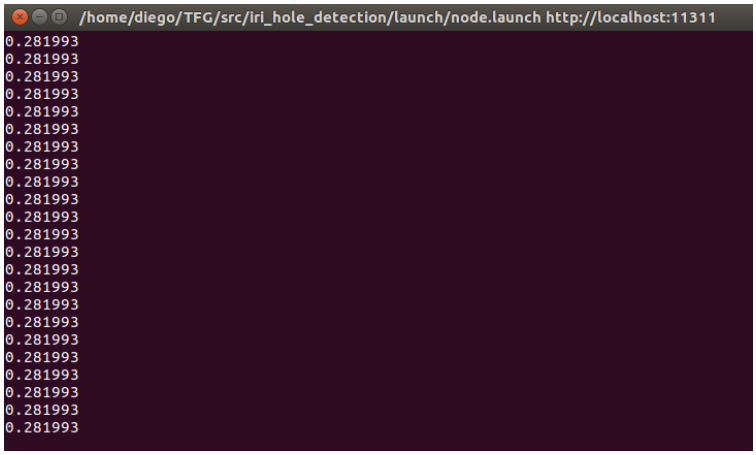


Figura 5-11. Terminal imprimiendo la componente y de la nube de puntos

El problema se da en que *cloud_in.points.y* permanece constante a lo largo del tiempo. Debido a esto se deduce que hay otro intercambio de ejes, en este caso entre el Y y el Z, ya que, si la nube de puntos está entera sobre el plano del suelo, este último es el que debe permanecer constante.

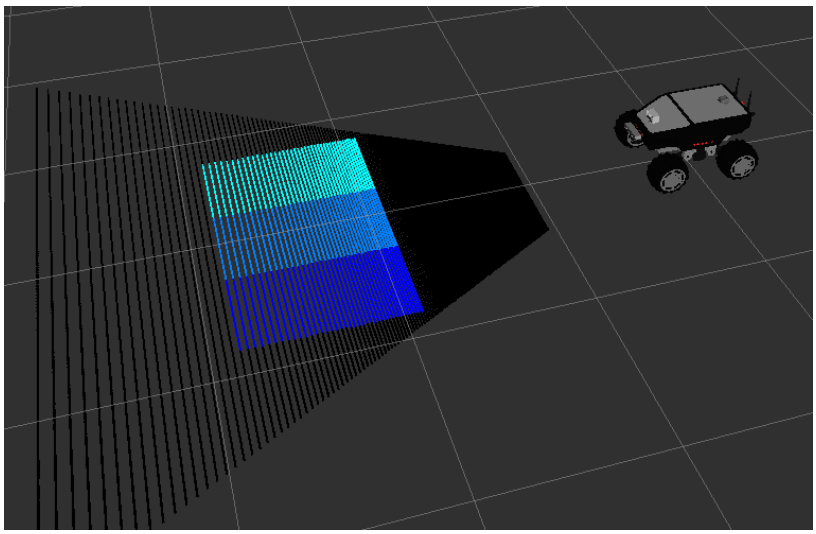


Figura 5-12. Sección de detección mostrada correctamente

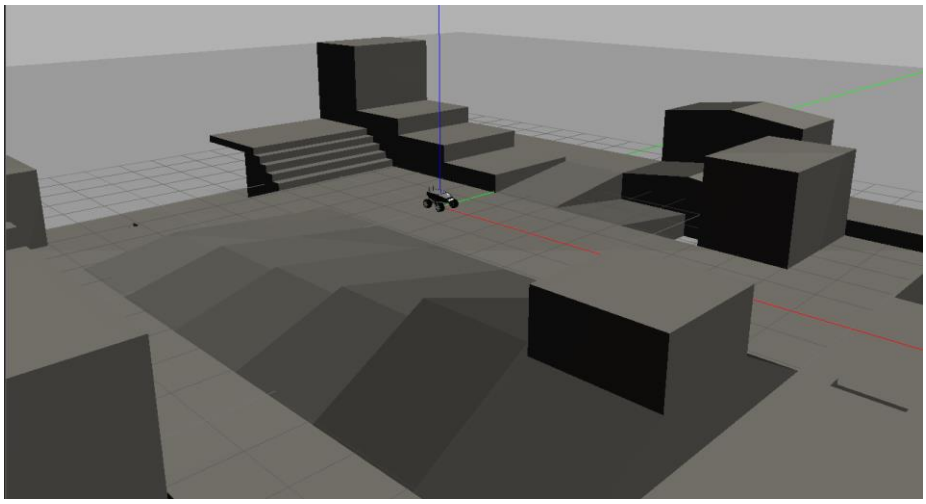


Figura 5-13. Mapa *Ramps*

Al realizar el cambio de *cloud_in.points.y* por *cloud_in.points.z*, la sección ya aparece correctamente y tamaños especificados. Sin embargo, al posicionar el robot frente a un agujero del mapa *Ramps*, la nube de puntos artificial sigue sin aparecer al frente, sino que se muestra a un lado.

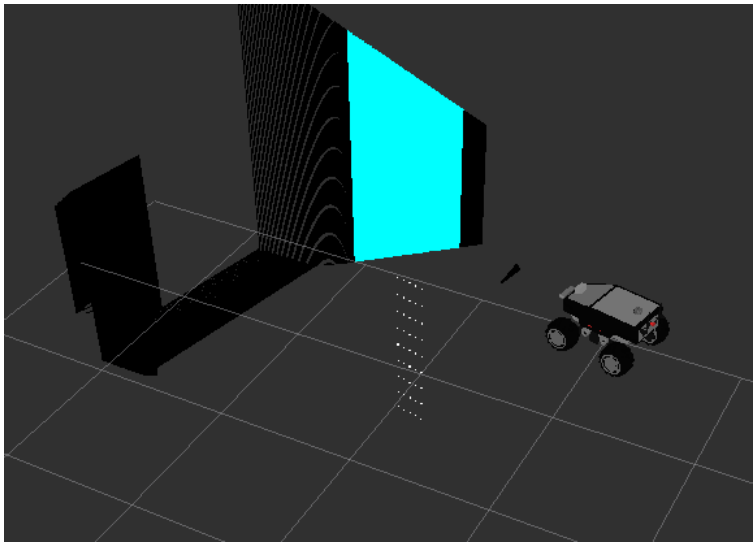


Figura 5-14. Nube de puntos artificial mal posicionada

Teniendo en cuenta las variaciones producidas en los ejes, se realizará un cambio en la función donde se provoca la aparición de esta nube, ajustándola al nuevo sistema de referencia al igual que se realizó con el área de detección previamente.

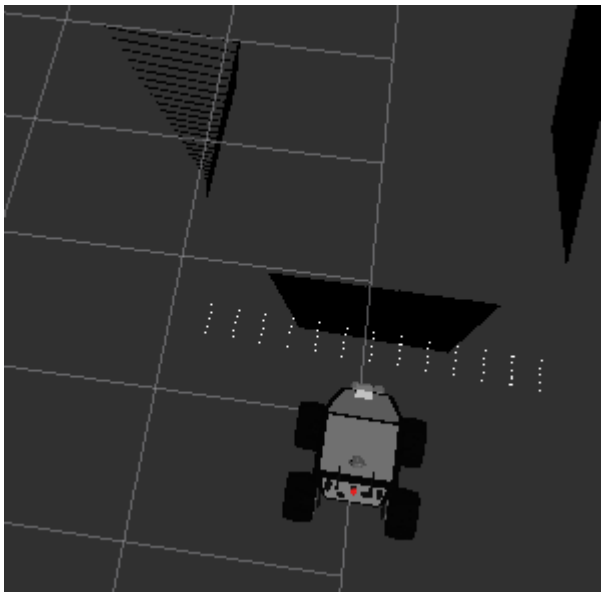


Figura 5-15. Nube de puntos artificial correctamente posicionada

Tras los nuevos cambios, ya se presenta de forma correcta, abarcando el ancho que tiene el agujero situado frente al robot.

5.3 Perfeccionamiento del funcionamiento de la detección de pendientes

Una vez clonado el repositorio del paquete de detección de pendientes en nuestro espacio de trabajo lo ejecutamos mediante el lanzador *node.launch*. Al igual que en el caso anterior hay que insertar el *topic* donde está siendo publicado la nube de puntos proporcionada por la cámara. Mediante este *launcher* se llama al ejecutable del nodo de nombre *iri_obstacle_detection_normals*. Una vez lanzado en el mapa *Ramps*, al igual que el paquete anterior, obtenemos lo siguiente en RVIZ:

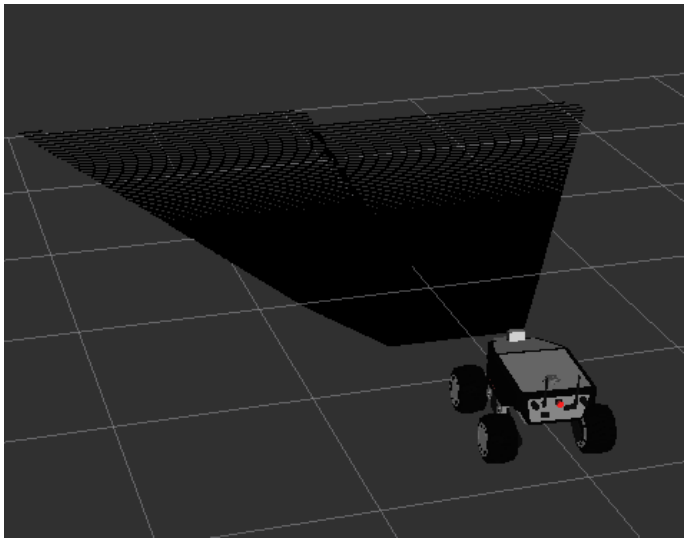


Figura 5-16. Nube de puntos vacía

Vemos que, al igual que en el apartado anterior, la nube de puntos sale íntegra, es decir, sin procesarla de la manera requerida. Al tratarse de un nodo diseñado por los mismos desarrolladores que el anterior, se

presupone que se dará el mismo cambio de ejes que en el caso previo.

Para solucionar este problema, inicialmente se cambia la primera condición donde se determina la distancia a partir de la cual se detecta la rampa, por lo tanto se cambia de *cloud_out.points.x* por *cloud_out.points.z* obteniéndose ahora la nube de puntos de esta manera:

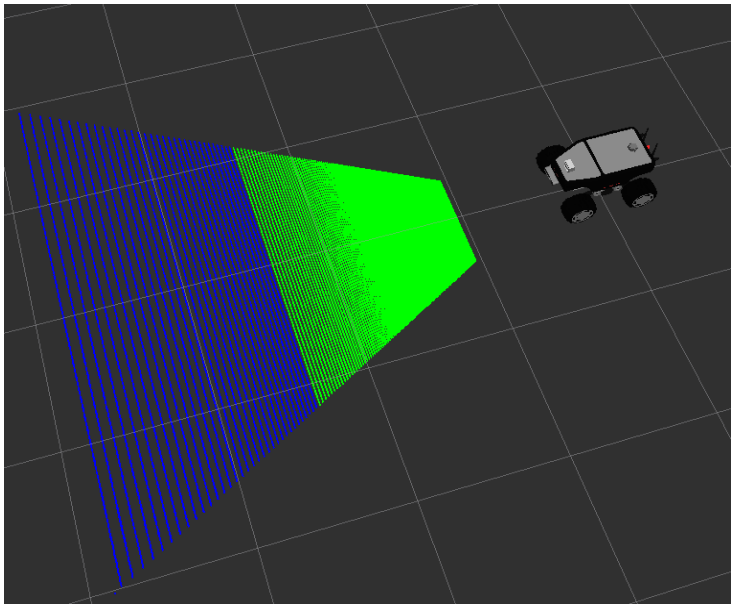


Figura 5-17. Prueba tras cambio de ejes.

En la imagen anterior, no se está enfocando ningún obstáculo, por lo que no debería verse en color verde ninguna zona de la nube de puntos, porque este color es exclusivo de las pendientes escalables. Esto se debe a un desajuste de la mínima distancia, ya que esta se debe realizar con respecto al eje vertical y está con respecto al eje frontal del robot. Hay que considerar que este valor con respecto al eje *base_link* sin ningún obstáculo delante será alrededor de 0.28 y decreciente cuanta más altura tenga por lo que si hay un obstáculo, será menor que la mínima distancia, fijada en 0.28.

Para especificar si la inclinación de la pendiente es escalable o no, se ha optado por realizarlo mediante el cálculo del seno del ángulo de máxima inclinación y compararlo con la variable *cloud_out.points.normal_z*.

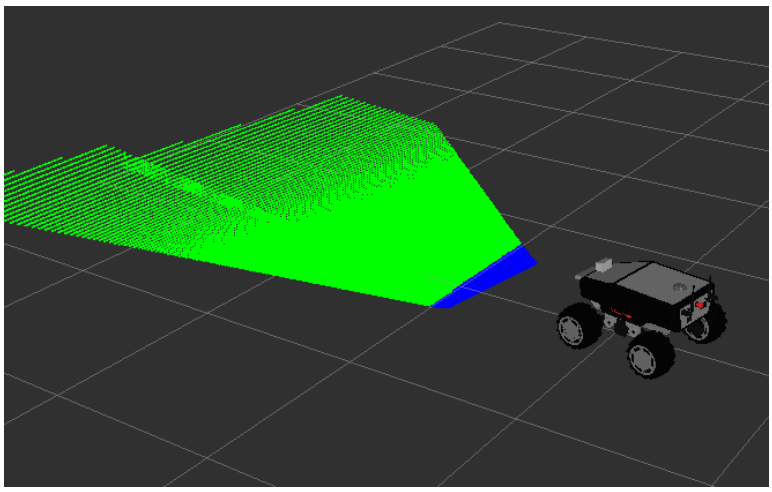


Figura 5-18. Summit-XL detectando una pendiente escalable

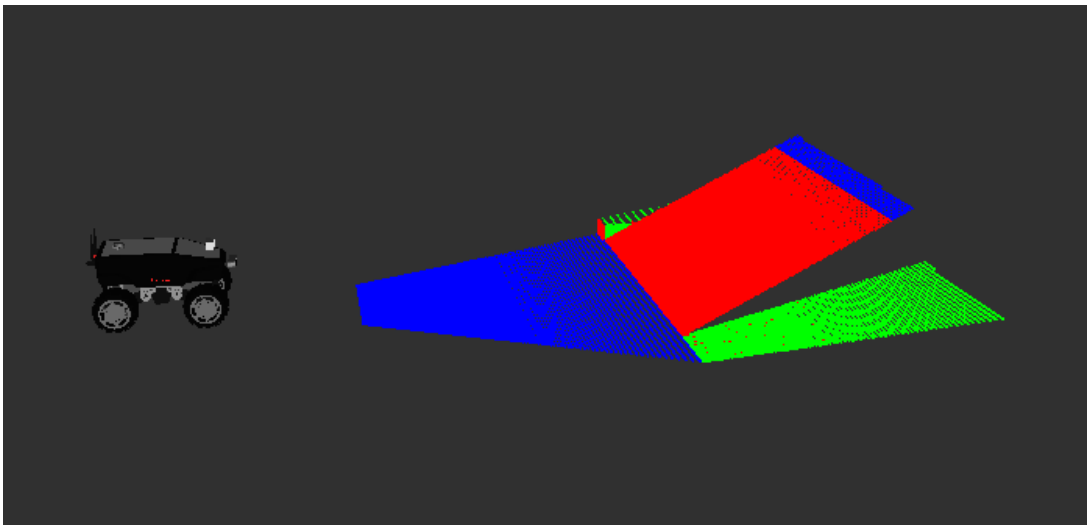


Figura 5-19. Summit-XL detectando una pendiente no escalable junto a otra que sí lo es

5.4 Pruebas con el robot real

Además de haber realizado las pruebas en el simulador Gazebo, en los laboratorios de la ETSI está presente el Summit-XL real. Para comprobar cómo funciona todo lo probado con anterioridad en la realidad nos conectamos al robot y realizamos varias pruebas.

5.4.1 Detección de pendientes

En cuanto a la detección de pendientes, inicialmente se probó colocando cajas dentro del laboratorio. En la siguiente imagen podemos ver la situación antes de colocar las cajas formando la pendiente, encontrándose el robot enfrentado a unas tablas de maderas, donde ya vemos que como la pendiente es de 90º.

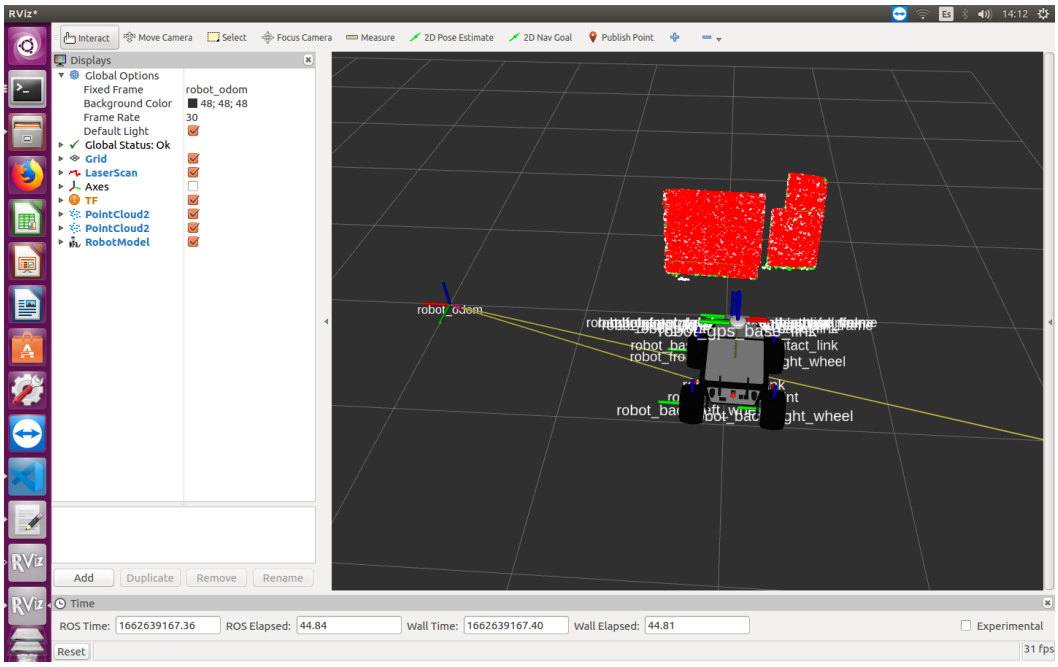


Figura 5-20. Experimento de pendientes en interior sin cajas en RVIZ.

A continuación, se muestra la situación una vez colocadas las cajas en forma de pendiente. En primera instancia, se estructuró una pendiente bastante pronunciada y se puede observar como la nube de puntos en la zona donde se encuentra la caja es de una tonalidad roja.

Figura 5-21. Experimento de pendientes en interior con caja muy inclinada.

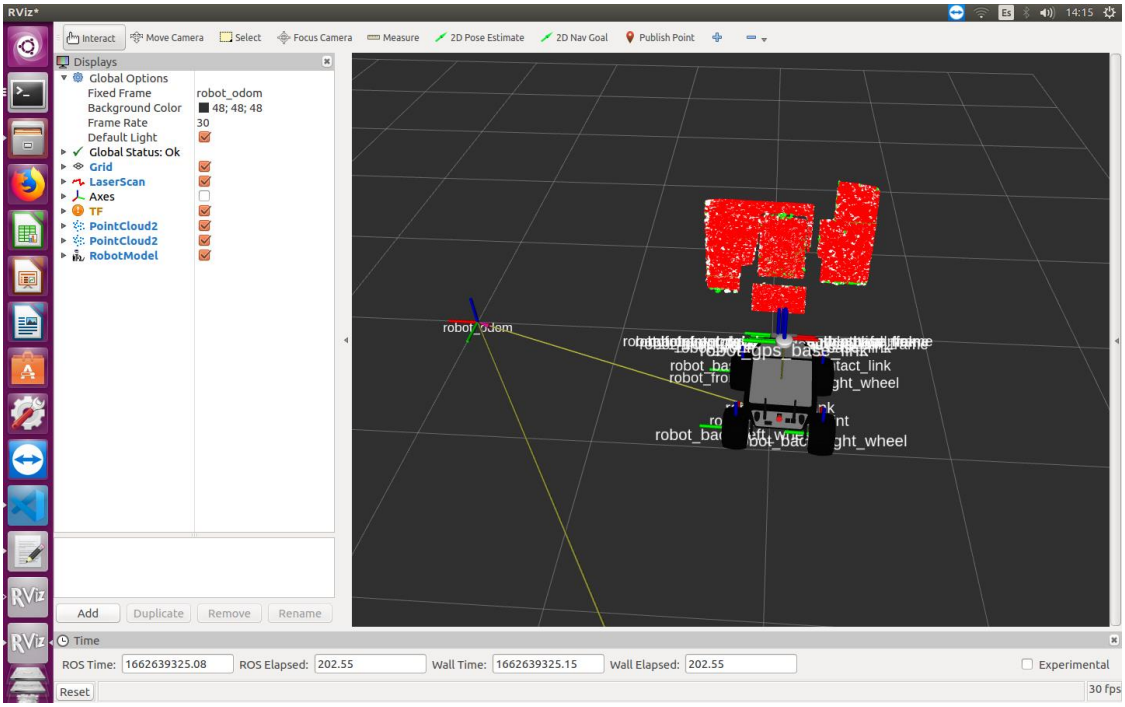


Figura 5-22. Experimento de pendientes en interior con caja muy inclinada en RVIZ.

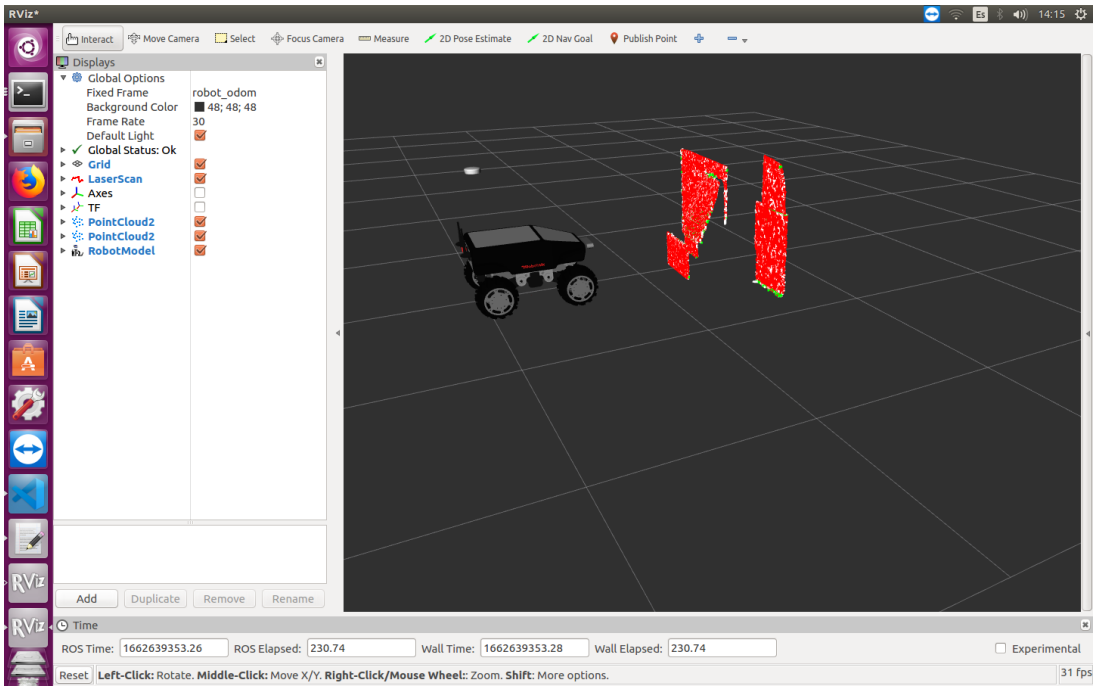


Figura 5-23. Experimento de pendientes en interior con caja muy inclinada en RVIZ.

Tras esto, se varió la inclinación de la caja, de manera que recreara una pendiente con menor grado de inclinación. Ahora en la representación, podemos ver que la zona correspondiente a esta caja frente al robot se marca en verde, puesto que es menos inclinada que el límite establecido.

Figura 5-24. Experimento de pendientes en interior con caja poco inclinada.

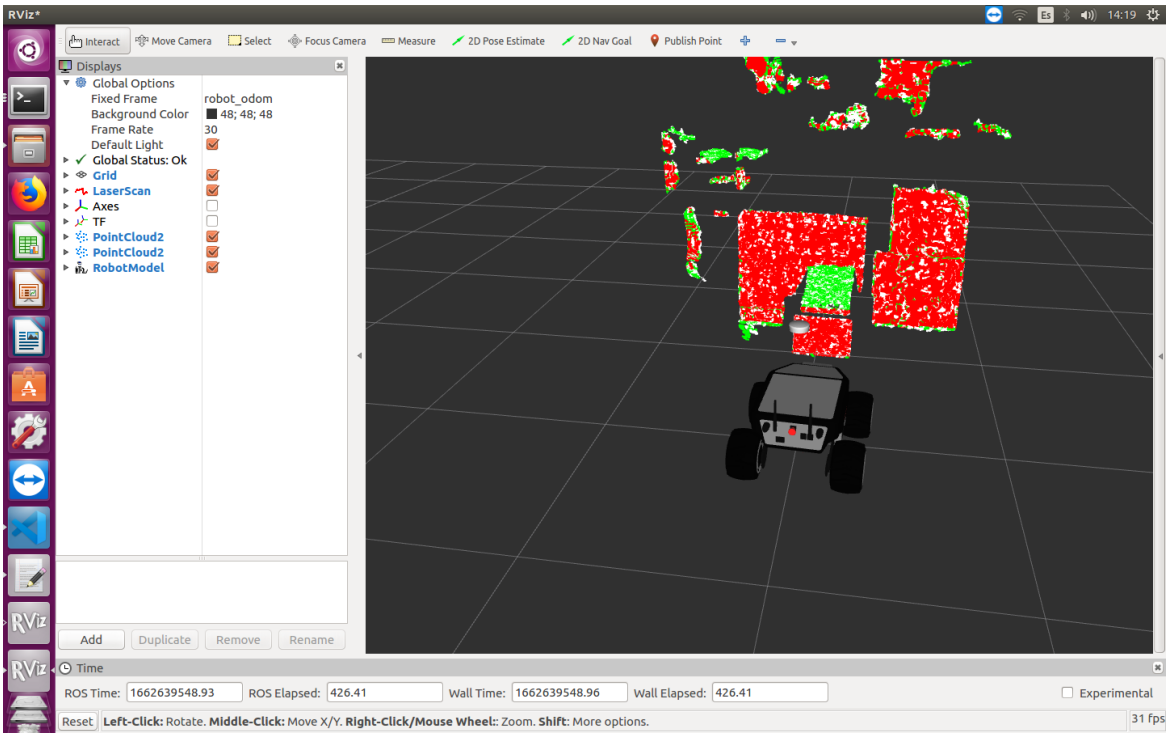


Figura 5-25. Experimento de pendientes en interior con caja muy inclinada en RVIZ.

Después, se probó a realizar otra comparación de pendientes durante las pruebas realizadas en el exterior. Había un problema en cuanto a la movilidad ya que los experimentos se tenían que realizar cerca de la caseta de la pista de baloncesto, ya que, por los motivos explicados en el anexo, se recurrió al uso del NUC del Turtlebot, en lugar de utilizar el portátil personal. Por ello, se utilizaron unas tablas apoyadas en la pared para hacer otra comprobación de pendientes porque la conexión SSH no permitía llevar el robot frente a las rampas de césped debido a la distancia.

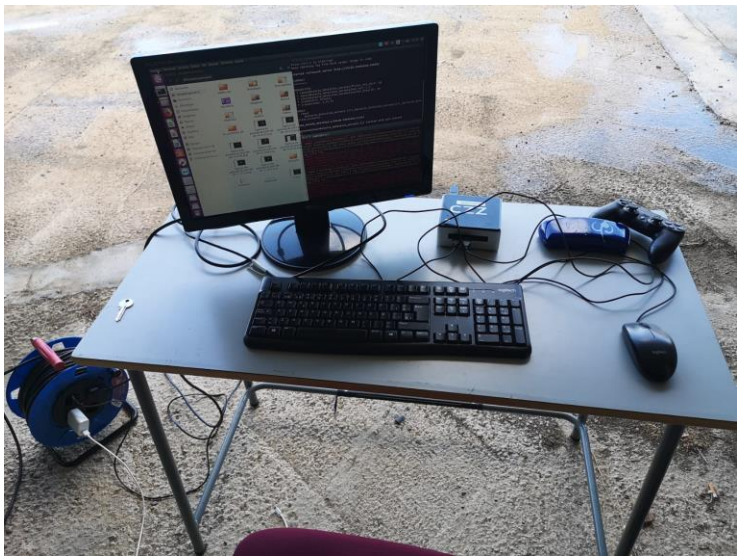
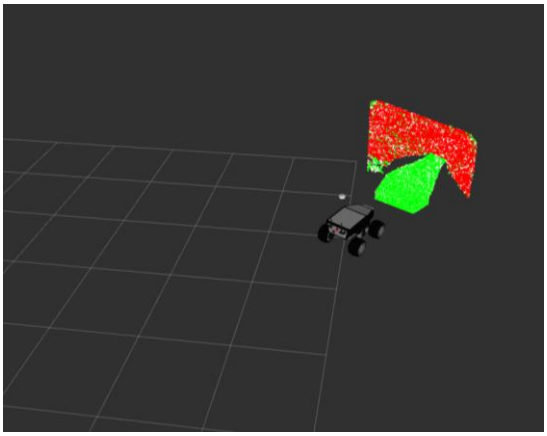
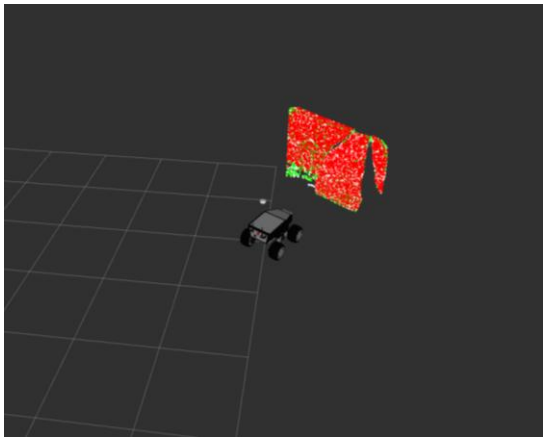


Figura 5-26. Material necesario para el manejo en el exterior.

Figura 5-27. Experimento con tabla en el exterior.





Figuras 5-28 y 5-29. Experimento con tabla en el exterior en RVIZ.

Al igual que en el experimento realizado en el interior del laboratorio, si la inclinación de la pendiente es mayor que el límite, la nube de puntos al colisionar con ella es de color rojo y en el caso contrario, es verde.

A continuación, se hace la prueba de colocar dos tablas con dos inclinaciones diferentes de manera simultánea y comprobar si las distingue:

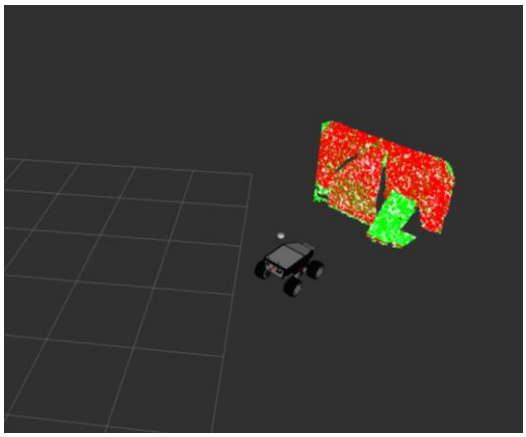


Figura 5-30 y 5-31. Experimento con tablas en diferente inclinación en el exterior.

Observamos que, efectivamente, diferencia las dos, aunque estén juntas. La menos inclinada se marca de verde y la otra, de rojo.

5.4.2 Navegación GPS

En primera instancia, se probó el funcionamiento de la navegación por GPS dentro del laboratorio. Como en este mismo proyecto se ha detectado con anterioridad, en interiores y cerca de edificios, el GPS puede llegar a tener varios metros de error. Esto es apreciable en la siguiente imagen:

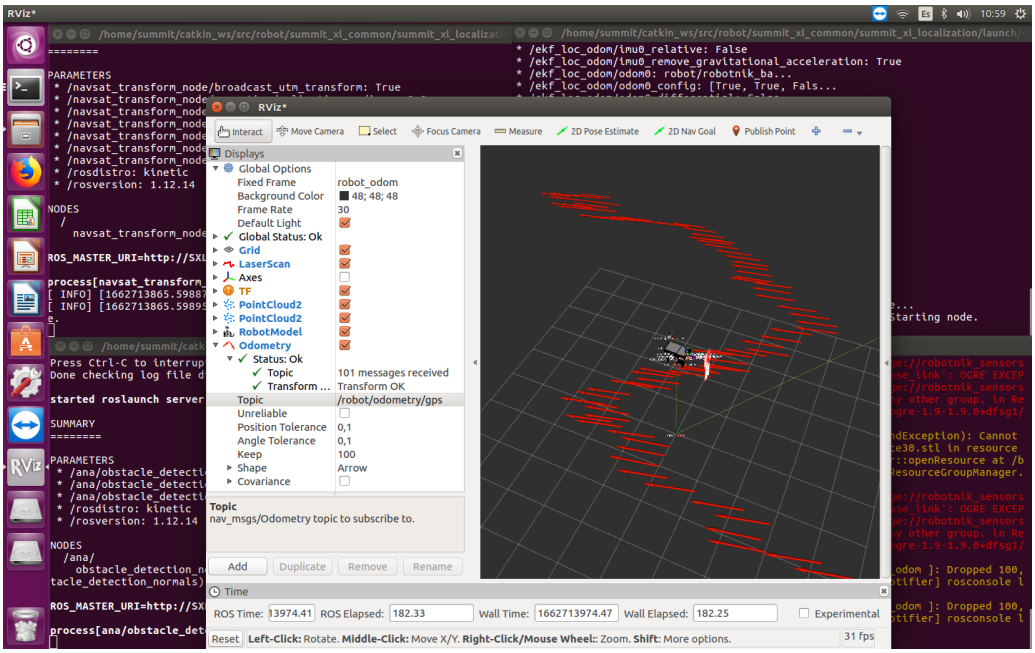


Figura 5-32. Experimento de GPS en interiores.

El modelo del robot oscilaba ligeramente en torno a la posición real, pero las medidas recibidas en el *topic odometry/gps* tienen demasiado error como se puede ver en la escena de RVIZ anterior.

Por ello y para obtener medidas más fieles se optó por llevar el robot al exterior, en concreto a la pista de baloncesto de la escuela.

Figura 5-33. Summit-XL en la pista de baloncesto.

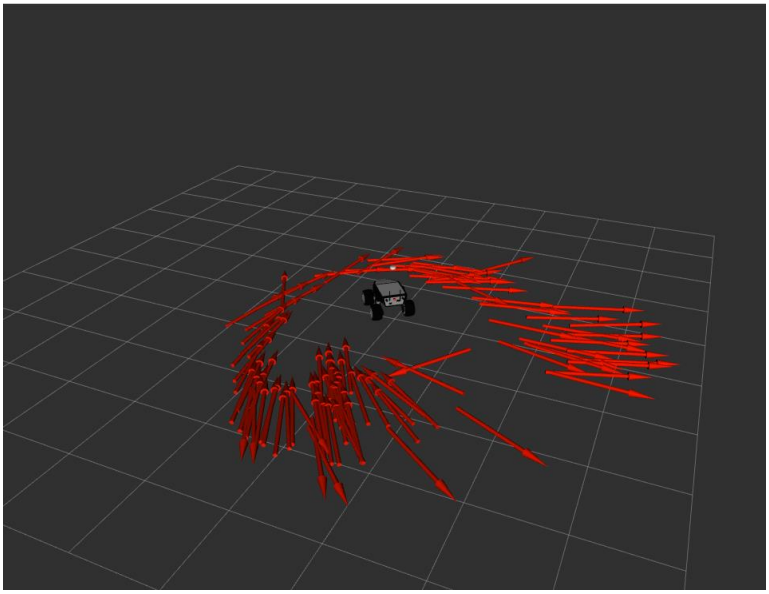


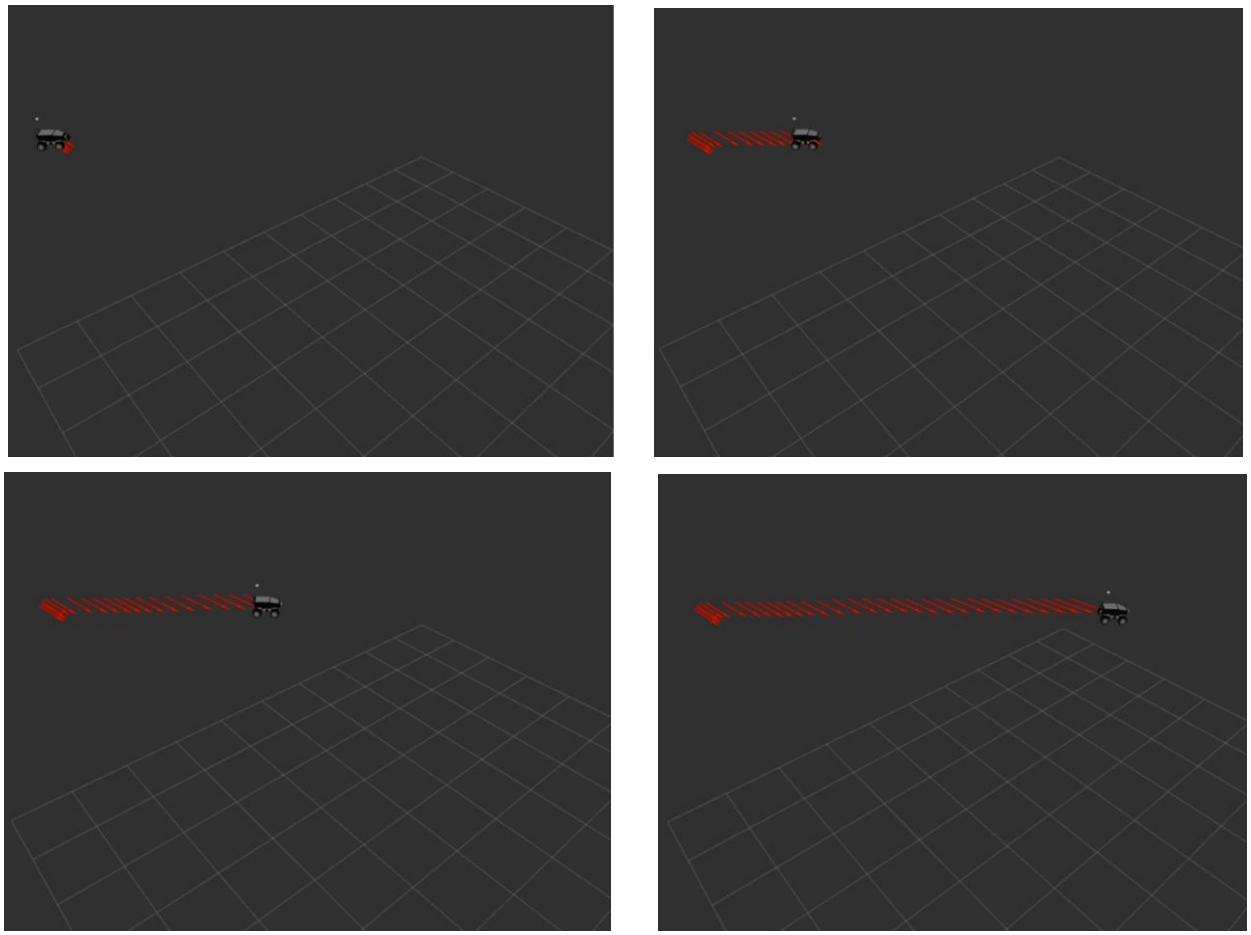
Figura 5-34. Visión en RVIZ de la localización GPS en exterior en zona cubierta.

Se aprecia que las flechas que indican la posición del robot se aproximan a la ubicación real del robot (donde aparece el modelo) con un error dentro de lo esperado según lo explicado con anterioridad, que suele estar en un rango de entre tres y cinco metros, ya que el robot se encontraba debajo del puente detrás de la pista de baloncesto.

A continuación, se procede a mover el Summit, fuera del puente, es decir, al aire libre donde debe mejorar considerablemente la estimación de la posición.

Figura 5-35. Summit-XL alejado del puente.

El recorrido realizado para comprobar la fiabilidad es en forma de L. Se parte de una de las canastas del fondo y se llega a la siguiente para posteriormente girar a la derecha hasta llegar al lateral de la caseta.



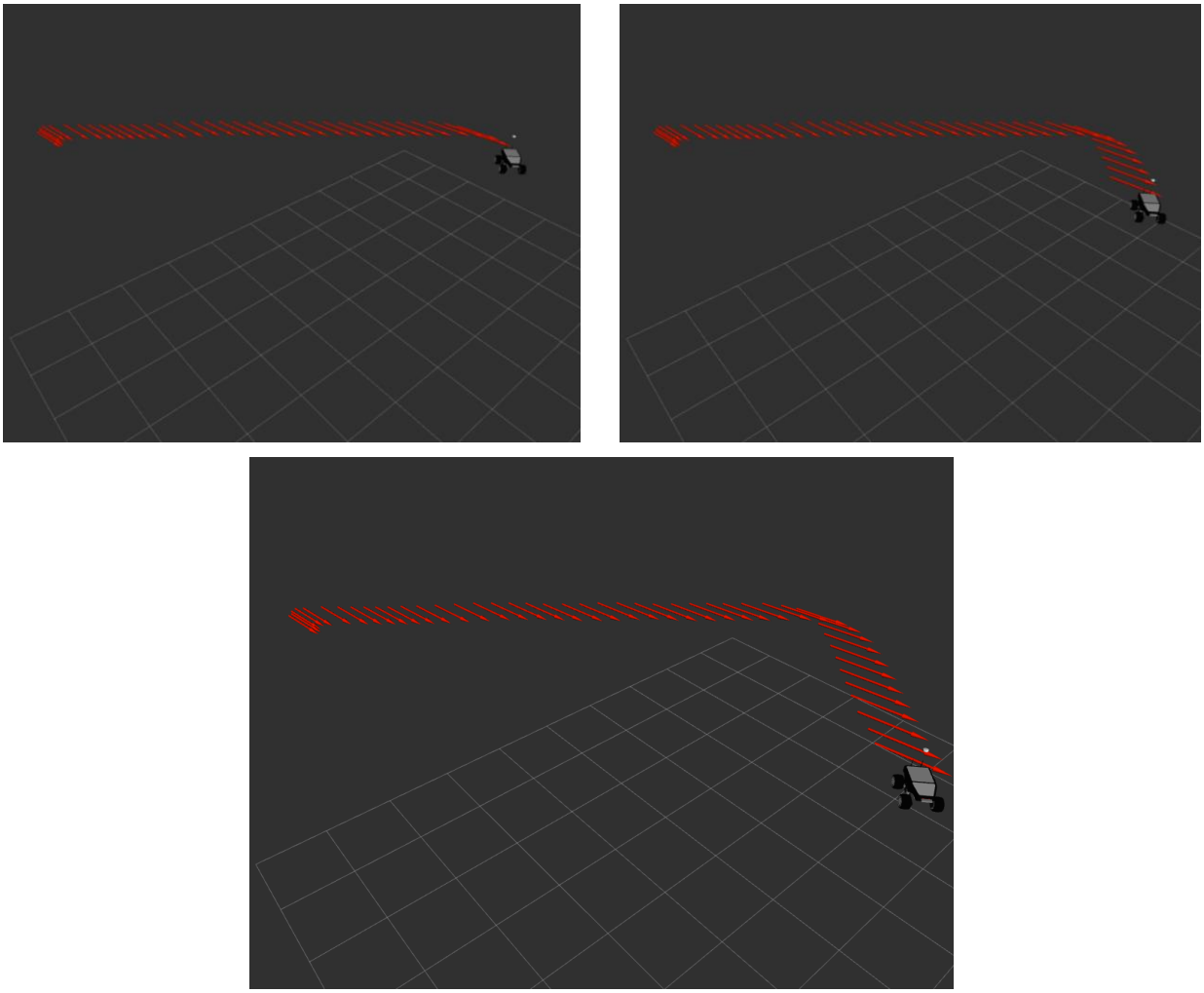


Figura 5-36. Trayectoria en forma de L

Tras alejar el robot de edificaciones que puedan interferir en la medida del GPS se aprecia que la medida es mucho más exacta. En el caso anterior había un error de unos pocos metros, pero al encontrarse al aire libre las marcas que se van señalando de odometría corregida con GPS coinciden prácticamente a la perfección con el camino realizado.

Para la utilización del GPS diferencial se planteó el uso de la antena disponible en el laboratorio, que consta de un trípode ajustado a una caja que contiene un sensor GPS de la marca u-blox cuyo modelo es C099-F9P. Una vez colocada se procedió a conectar la antena al PC y visualizar los datos que concedía en *u-center*.

Figura 5-37. Antena para GPS diferencial.

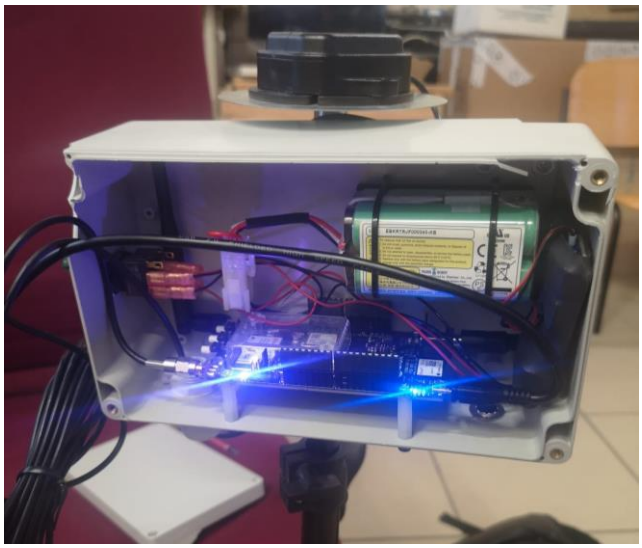


Figura 5-38. Interior de la caja de la antena

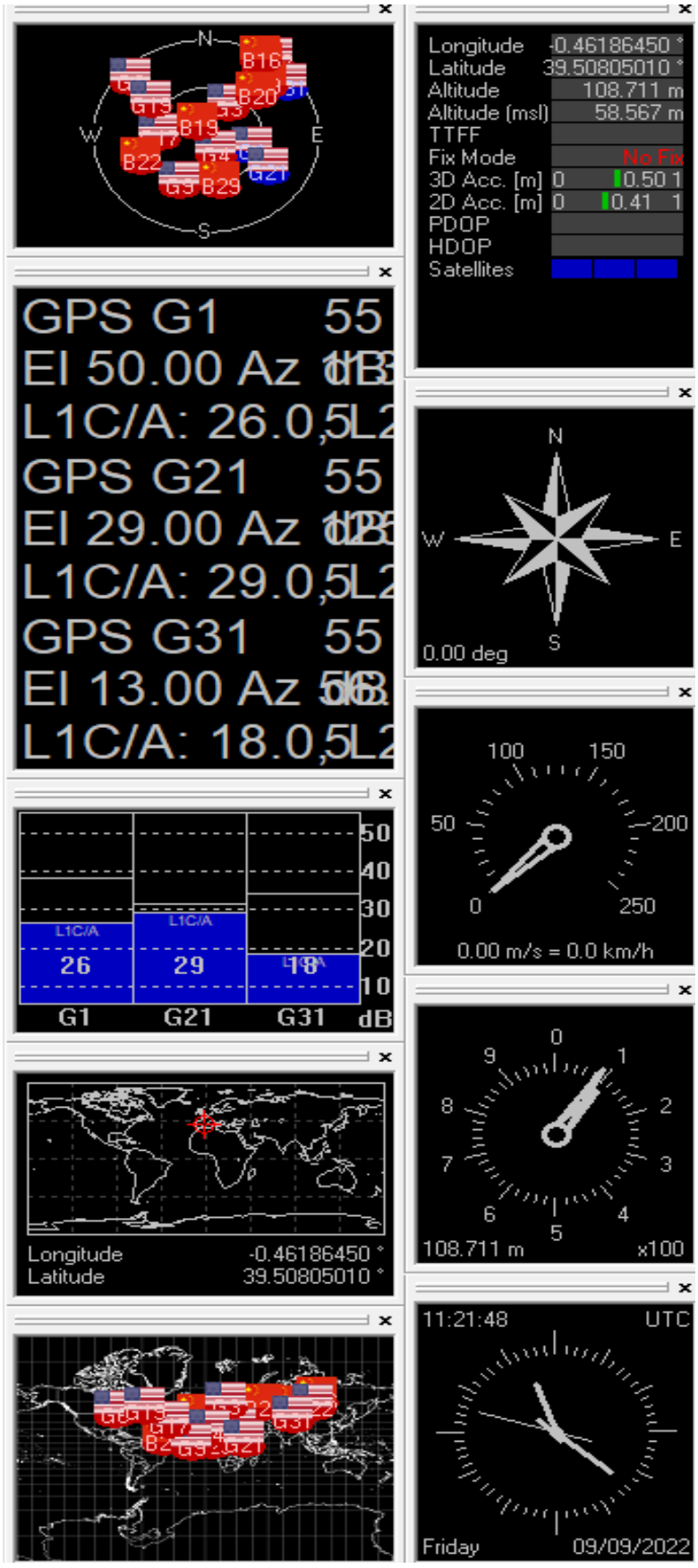


Figura 5-39. Resultados de la conexión de la antena al PC

Si nos fijamos en los resultados podemos percibir que la localización que marca la cruceta en el mapa del mundo parece algo alejada de la posición correcta, ya que la antena se encuentra en Sevilla y marca una localización al este de la península.

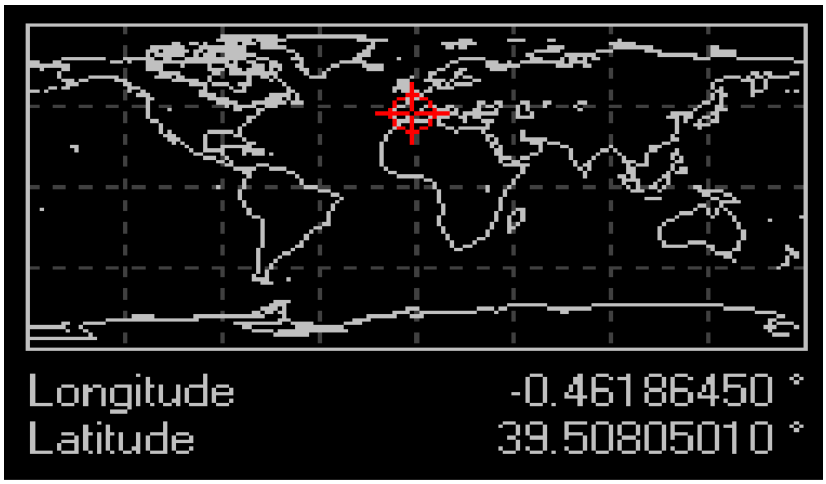


Figura 5-40. Mapa de localización de la antena GPS

Para averiguar la localización exacta señalada, exportamos los datos recibidos a un archivo .kml, que es apto para ser importado en Google Earth y así ver donde está percibiendo la posición.

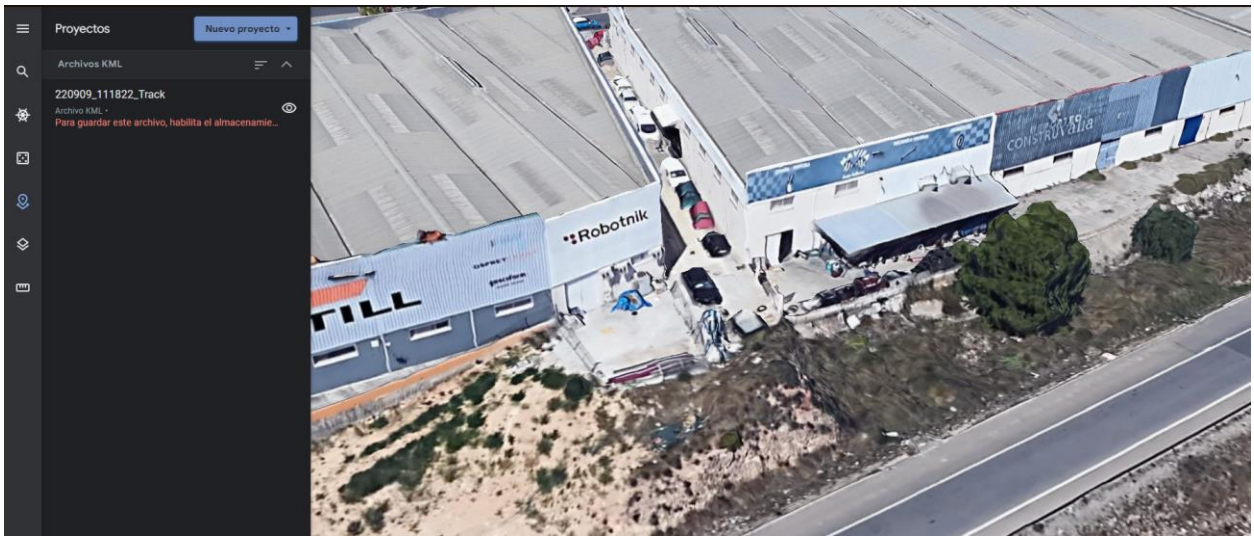


Figura 5-41. Localización en Google Earth de la antena GPS

Una vez importado, se puede apreciar que la localización señalada es la nave industrial de Robotnik. Se planteó como prueba la reconfiguración de la antena, de manera que fuera conectada por Bluetooth o Wi-Fi en lugar de por cable y si así variaba la medida.

Para ello, hay que introducir unos comandos por puerto serie, señalando el puerto correspondiente al módulo ODIN en el terminal y seleccionar 460800 baudios.

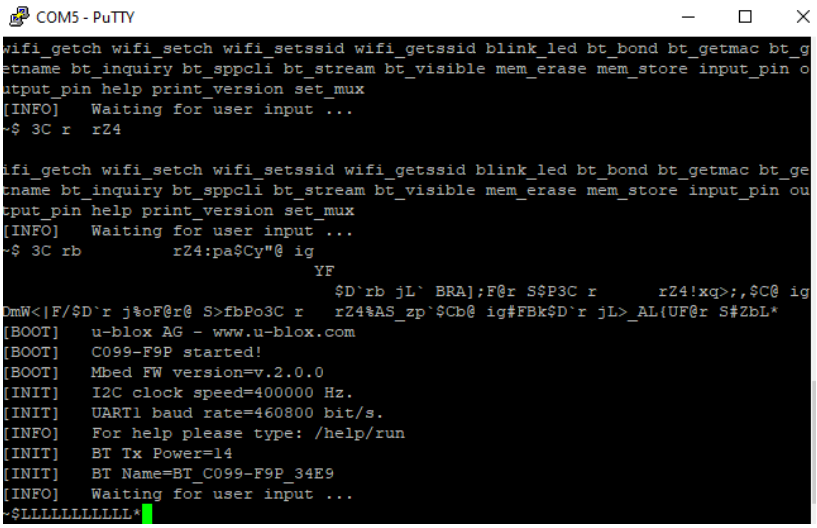


Figura 5-42. Mensaje del terminal tras la conexión de la antena

Debería aparecer un texto y una línea para introducir las ordenes especificadas en el manual de usuario del fabricante, sin embargo, aparecen caracteres erróneos además de dicho texto y no deja introducir comandos.

5.4.3 Detección de agujeros

Por último, se va a comprobar el funcionamiento del algoritmo de detección de agujeros. Estas pruebas se realizaron dentro del laboratorio.

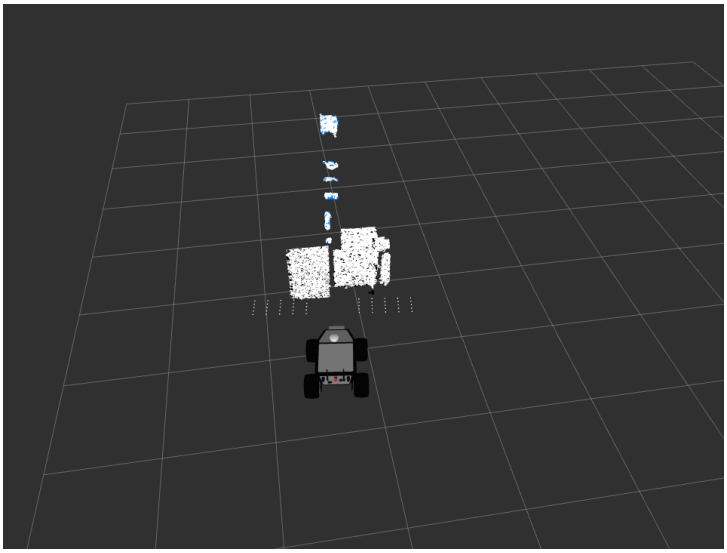


Figura 5-43. Pruebas de detección de agujeros

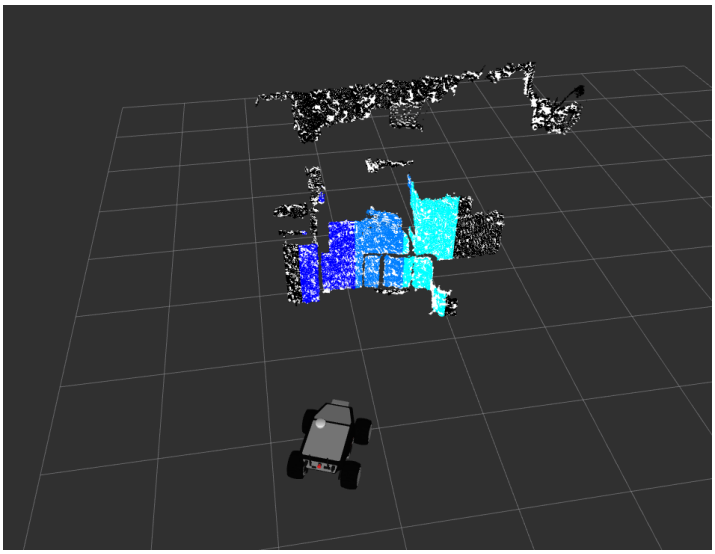


Figura 5-44. Pruebas de detección de agujeros

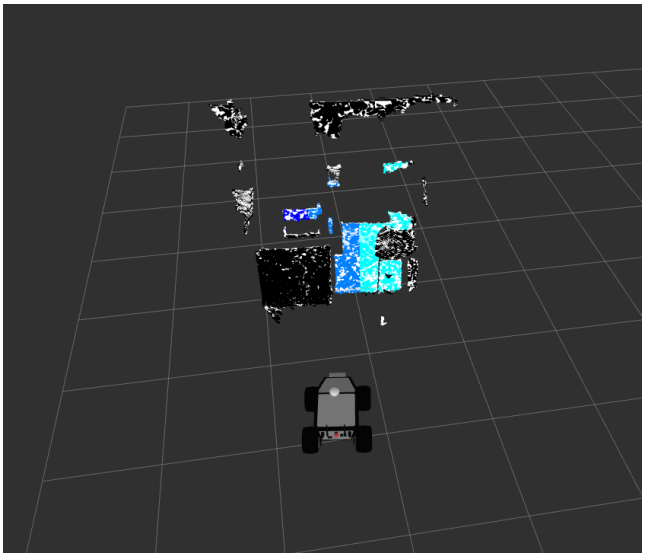


Figura 5-45. Pruebas de detección de agujeros

Vemos que hay un problema. La nube de puntos artificial que aparece cuando se detecta un agujero aparece por defecto. Se puede deducir fácilmente el motivo. En la simulación realizada sin el robot real al visualizar en RVIZ, la nube de puntos aparece entera, aunque ningún obstáculo interfiera en su rango. Con el Summit-XL del laboratorio esto no sucede así. La nube de puntos solo aparece cuando hace contacto con obstáculos, por lo que el algoritmo diseñado no sería aplicable al robot en el que han sido realizado las pruebas.

6 CONCLUSIÓN

Tras todo lo realizado queda completo el sistema propuesto. El robot se localiza haciendo uso de la tecnología GPS y detecta agujeros y pendientes. A lo largo de este capítulo se repasará todo lo aprendido durante el proyecto y unas posibles adiciones que se pueden realizar en un futuro.

6.1 Materias aprendidas

Este trabajo, al realizarse principalmente sobre ROS, se ha debido investigar bastante sobre su uso. Se trata de un entorno de trabajo poco intuitivo y con alta variedad de funcionalidades, por lo que para su uso inicial se partió de los conocimientos adquiridos en la asignatura de *Control y Programación de Robots.* Además, para conocer el funcionamiento y estructura de este firmware, fue de gran apoyo el libro *Robot Operating System for Absolute Beginners*. Este libro está orientado a aprender cómo usar ROS partiendo de lo básico como las definiciones de nodos, *topics*, etc. de cara al uso del Turtlebot.

Figura 6-1. Turtlebot

Una vez adquiridos los conocimientos básicos de ROS, para el funcionamiento básico del Summit-XL se detalla bien en el propio GitHub de Robotnik. Aunque siempre pueden surgir fallos, ya sean de compatibilidad con librerías, errores de instalación o dudas de uso. Para solucionar estos errores suele ser de ayuda acudir a foros de preguntas como por ejemplo *Ros Answers* y *Stack Overflow*. Estas comunidades son muy activas y suelen responder dudas en un breve periodo de tiempo y con eficacia. Además de hacer las propias preguntas en el foro también se puede investigar si a alguien le ha ocurrido el mismo error y este ha sido resuelto.

6.2 Mejoras posibles al trabajo realizado

Tras todas las funcionalidades implementadas tanto en este trabajo como en los anteriormente desarrolladas, aún quedan diversos objetivos que se pueden plantear:

- Intentar solucionar el problema de la antena GPS o adquirir una diferente para crear la estación de GPS diferencial y reducir considerablemente el error de la estimación de la posición hasta el rango de centímetros
- Arreglar la orientación de salida del nodo *navsat_transform_node*, ya que en el mensaje de odometría que publica, esta se mantiene fija cuando lógicamente debería ir variando de acuerdo al movimiento del robot.

REFERENCIAS

[1] M. Mora, «Integración de sistema láser y visión en la navegación de un robot móvil» Universidad de Sevilla, Sevilla, Trabajo de Fin de Máster 2020.

[2] M. Serrano, «Desarrollo de algoritmos para evitación de obstáculos para robot móvil SUMMIT» Universidad de Sevilla, Sevilla, Trabajo de Fin de Máster 2022.

[3] Ubuntu Releases, «Ubuntu 16.04.7 LTS (Xenial Xerus)».
Online: https://releases.ubuntu.com/16.04.7/

[4] ROS, «Ubuntu install of ROS Kinetic».
Online: http://wiki.ros.org/kinetic/Installation/Ubuntu

[5] ROS, Tom Moore, «navsat_transform_node».
Online: http://docs.ros.org/en/jade/api/robot_localization/html/navsat_transform_node.html

[6] ROS, Tom Moore, «State Estimation Nodes».
Online: http://docs.ros.org/en/melodic/api/robot_localization/html/state_estimation_nodes.html

[7] Robotnik, «Summit-XL User Guide Manual - V.2.0,» User Guide Manual, 2020.

[8] ROS, «ROS documentation».
Online: http://wiki.ros.org/kinetic/Installation/Ubuntu

[9] GitHub, «GitHub».
Online: https://github.com

[10] ROS, «ROS Answers».
Online: https://answers.ros.org/questions/

[11] Sergi Hernández, «iri_point_cloud_hole_detection».
Online:
https://gitlab.iri.upc.edu/labrobotica/ros/navigation/3d_navigation/iri_point_cloud_hole_detection

[12] Alejandro López Gestoso, «iri_base_algorithm».
Online: https://gitlab.iri.upc.edu/labrobotica/ros/iri_core/iri_base_algorithm

[13] Sergi Hernández, «iri_obstacle_detection_normals».
Online:
https://gitlab.iri.upc.edu/labrobotica/ros/navigation/3d_navigation/iri_obstacle_detection_normals

[14] Stack Overflow, «Stack Overflow Questions».
Online: https://stackoverflow.com/questions

[15] The Construct, Sitio web oficial.
Online: https://www.theconstructsim.com

[16] Google, «Google Earth».
Online: https://www.google.com/intl/es/earth/

[17] Robotnik, «RobotnikAutomation - Github».
Online: https://github.com/RobotnikAutomation

[18] Wikipedia, Sitio web oficial.
Online: https://es.wikipedia.org

[19] Keith Kotay, «Robo-Rats Locomotion: Skid-steer Drive».
Online: https://groups.csail.mit.edu/drl/courses/cs54-2001s/skidsteer.html

[20] Oficina de Coordinación de Posicionamiento, Navegación, y Cronometría por Satélite, «GPS.gov».
Online: https://www.gps.gov/systems/gps/spanish.php

[21] John Kyes, «Gestión de flotas».
Online: https://www.geotab.com/es/blog/que-es-gps/

[22] Revista de Robots, «Vehículos AGV».
Online: https://revistaderobots.com/robots-y-robotica/robot-agv-aiv-los-vehiculos-de-guiado-automatico-inteligentes/

[23] Cade Cobots, «¿Qué es un cobot y cuáles son sus ventajas?».
Online: https://cadecobots.com/que-es-un-cobot/

[24] Intel, «Tipos de robots».
Online: https://www.intel.es/content/www/es/es/robotics/types-and-applications.html

[25] ArcGeek, «¿Cómo funcionan los dispositivo GPS?».
Online: https://acolita.com/como-funcionan-los-dispositivos-gps-trilateracion-vs-triangulacion/

[26] Natalia Garrido-Villén, «Errores atmosféricos en GNSS».
Online: https://nagarvil.webs.upv.es/errores-atmosfericos-gnss-gps/

[27] Xataka Ciencia, «¿Cómo funciona el GPS?».

Online: https://www.xatakaciencia.com/sabias-que/como-funciona-el-gps-y-iii

[28] Topo Servis, «Conoce el receptor RTK GPS y sus funciones».

Online: https://toposervis.com/conoce-el-receptor-rtk-y-sus-funciones/

[29] José Luis R., «Cómo funciona un acelerómetro».

Online: https://como-funciona.co/un-acelerometro/

[30] Digital Key, «IMU para obtener una ubicación precisa».

Online: https://www.digikey.es/es/articles/imus-for-precise-location-part-2-how-to-use-imu-software-for-greater-precision

[31] Hardware libre, «¿Qué es ROS?».

Online: https://www.hwlibre.com/ros/

[32] R. Suárez, «Robot Operating System (ROS)».

Online: https://www.aer-automation.com/wp-content/uploads/2022/03/ROS_articuloAER.pdf

[33] Dalia Patiño, «Prototipo de robot submarino para medir contaminación».

Online: http://www.cienciamx.com/index.php/tecnologia/robotica/25088-prototipo-robot-submarino-contaminacion

[34] F. Parrón, «Sistema de localización para robots móviles aplicado a campos de golf».

Online: https://docplayer.es/16242967-Sistema-de-localizacion-para-robots-moviles-basado-en-gps-aplicacion-a-campos-de-golf.html

[35] Luis Antonio García, «Sistema robótico para la exploración de campos sembrados».

Online: https://www.redalyc.org/pdf/304/30420469006.pdf

[36] U-blox, «Guía de usuario de U-Center».

Online: https://content.u-blox.com/sites/default/files/u-center_Userguide_UBX-13005250.pdf

[37] U-blox, Sitio web oficial.

Online: https://www.u-blox.com/en

[38] Robotnik, «Robot móvil Summit-XL».

Online: https://robotnik.eu/es/productos/robots-moviles/summit-xl-es/

[39] Oracle, «Virtual-Box».

Online: https://www.virtualbox.org

[40] Lentin Joseph, «Robot Operating System (ROS) for Absolute Beginners,».

[41] Antonio Sala, «Predicción en Sistemas Dinámicos no lineales».
Online: http://personales.upv.es/asala/DocenciaOnline/material/FiltrKalmanExt.pdf

[41] Diego Bermúdez, «Repositorio de los paquetes empleados».
Online: https://github.com/RaccoonAD/TFG---DIEGO.git

GLOSARIO

OCONTSOLAR: Optimal Control of Thermal Solar Energy System. Proyecto para el que se realiza este trabajo.

GPS: Global Positioning System. Sistema que permite localizar cualquier objeto.

LIDAR: Light Detection and Ranging o Laser Imaging Detection and Ranging. Dispositivo que permite estimar la distancia mediante la emisión de un haz láser.

Autómata: Robot programable y controlable de carácter industrial.

Robot: Máquina programa capaz de realizar diferentes tareas y operaciones.

Ubuntu: Distribución de Linux basada en Debian.

Linux: Sistema Operativo de código abierto tipo UNIX.

ROS: Robot Operating System. Firmware muy utilizado en robótica.

Gazebo: Simulador de robótica en tres dimensiones.

RVIZ: Visualizador empleado en robótica donde se muestra información de los sensores.

PC: Personal Computer.

TFG: Trabajo de Fin de Grado.

TFM: Trabajo de Fin de Máster.

Orbbec: Fabricante de la cámara presente en el Summit.

Wi-Fi: Tecnología de conexión inalámbrica entre dispositivos.

RGBD: Tipo de cámara que proporciona información de imagen y profundidad.

Microgrid: Red eléctrica distribuida con diversas fuentes de energía y cargas.

RFID: Sistema de almacenamiento de datos mediante el uso de radiofrecuencias.

Bluetooth: Red inalámbrica personal de corto alcance.

Kinetic: Versión de ROS utilizada en Ubuntu 16.04.

IMU: Unidad de Medida Inercial

Cinemática: Parte de la mecánica que estudia el movimiento de los objetos.

Firmware; Software capaz de controlar el hardware a bajo nivel.

Sensor; Dispositivo capaz de medir una magnitud.

I+D; Investigación y desarrollo.

UGV; Unmmaned ground vehicle. Vehículo terrestre no tripulado.

Odometría: Estimación de la posición mediante el movimiento realizado por las ruedas.

Tracción: Esfuerzo producido por dos fuerzas de sentido contrario.

Router: Dispositivo capaz de dirigir las comunicaciones de una red.

Trípode: Utensilio de tres patas para estabilizar o apoyar un objeto.

Benewake: Empresa fabricante del Lidar 3D del robot.

RTK: Real-Time Kinematic.

Trilateración: Método para determinar posiciones basado en geometría triangular.

Efemérides: Conjunto de valores a partir de los cuales se obtiene la posición de un cuerpo astronómico.

UHF: Frecuencia muy alta (Ultra High Frequency) que abarca el rango de 300 MHz a 3 GHz.

Piezoresistivo: Sensores basados en la propiedad de algunos semiconductores de variar la resistividad al ser afectador por un esfuerzo mecánico.

MEMS: Tecnología de desarrollo de dispositivos microscópicos.

ToF: Time of Flight.

OpenGL: API que permite a ciertos programas reproducir gráficos en dos y tres dimensiones.

Latitud: Distancia en grados con respecto al ecuador.

Longitud: Distancia en grados con respecto al meridiano de Greenwich

Altitud: Elevación sobre el nivel del mar.

Stack: Conjunto de paquetes de ROS.

Paquete de ROS: Directorio que agrega ciertas funcionalidades a disposición del espacio de trabajo.

Varianza de Allan: Es una medida de estabilidad de la frecuencia en relojes, osciladores y amplificadores.

ANEXO: MANUAL DE USUARIO

A.1 Preparación del Sistema Operativo Ubuntu 16.04

Para el uso de ROS, se recurre a la distribución de Linux, Ubuntu. Como la versión instalada en el propio Summit XL es la 16.04, esta ha sido la elegida para el desarrollo del trabajo. Para la instalación de este sistema operativo se puede optar por una partición del disco o hacer uso de una máquina virtual. En concreto para este trabajo se ha escogido la segunda opción. El software utilizado es VirtualBox.

Los pasos que se deben seguir para la ejecución de Ubuntu 16.04 en VirtualBox son los siguientes:

1. Descargar Oracle VM VirtualBox 6.1 de la página oficial cuyo enlace es [39].
2. Descargar la imagen de escritorio para 32 bits o 64 bits según del ordenador del que se disponga desde [3].
3. Añadir Ubuntu 16.04 a la lista de máquinas virtuales:
    a. Seleccionar *Nueva* en la barra de herramientas.
    b. Darle un nombre a la máquina virtual, seleccionar el tipo de sistema operativo y su versión y seleccionar la carpeta donde queremos que se guarden los archivos relacionados con ella.

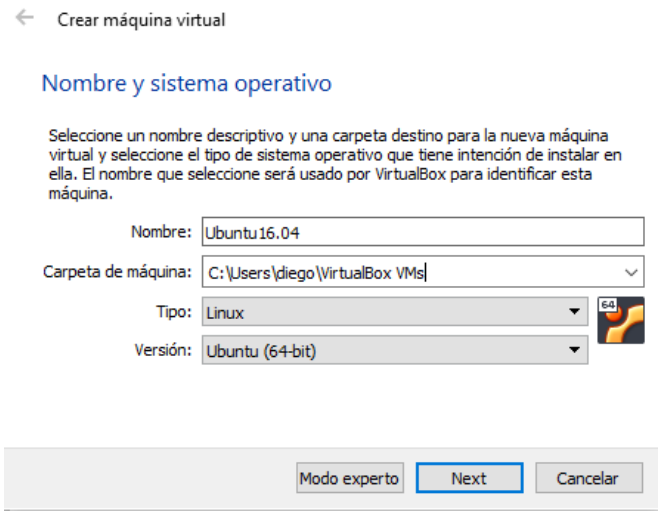


Figura A-1. Crear nombre y tipo de máquina Virtual

    c. Seleccionar el número de MB de memoria RAM para asignarlos durante la ejecución de la máquina. Un valor a partir del que tiene un funcionamiento aceptable son 5 GB.
    d. Crear un disco duro virtual o usar uno ya existente. En el caso de crear uno nuevo, es preferible definir que su memoria se reserve dinámicamente.

4. Al ejecutar la máquina virtual se deberá seleccionar la imagen de Ubuntu 16.04 que hemos descargado previamente.
5. Configurar Ubuntu 16.04:
    a. Seleccionar el idioma deseado y la opción *Instalar*

Figura A-2. Menú de instalación

b. En la siguiente página, no marcar la descarga automática de actualización, ya que queremos permanecer con la versión 16.04.

c. Marcar la opción *Borrar disco e instalar Ubuntu.*

d. Escoger la ubicación desde donde se instale.

e. Elegir idioma del teclado.

f. Registrar un nombre de usuario y una contraseña.

A.2 Instalación de ROS

Para la instalación de ROS, nos pondremos en contacto mediante comandos con packages.ros.org. La serie de comandos que deben introducirse son los siguientes:

```
• sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
• sudo apt install curl # Si no has instalado curl
• curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc | sudo apt-key add -
• sudo apt update
• sudo apt install ros-melodic-desktop-full
• apt search ros-melodic
• echo "source /opt/ros/melodic/setup.bash" >> ~/.bashrc
• source ~/.bashrc
• sudo apt install python-rosdep python-rosinstall python-rosinstall-generator python-wstool build-essential
```

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

A.3 Instalación de paquetes

A.3.1 Summit-XL

A continuación, se deberán descargar los paquetes correspondientes al funcionamiento básico del Summit-XL. Para ello, se accederá al repositorio en GitHub de Robotnik, los propios fabricantes del robot. Una vez en él descargaremos los siguientes paquetes:

- Summit_xl_common
- Summit_xl_sim
- Robotnik_msgs
- Robotnik_sensors

Es importante que cuando los descarguemos estemos en la rama correspondiente a la versión Kinetic de ROS, como se puede observar en la imagen:

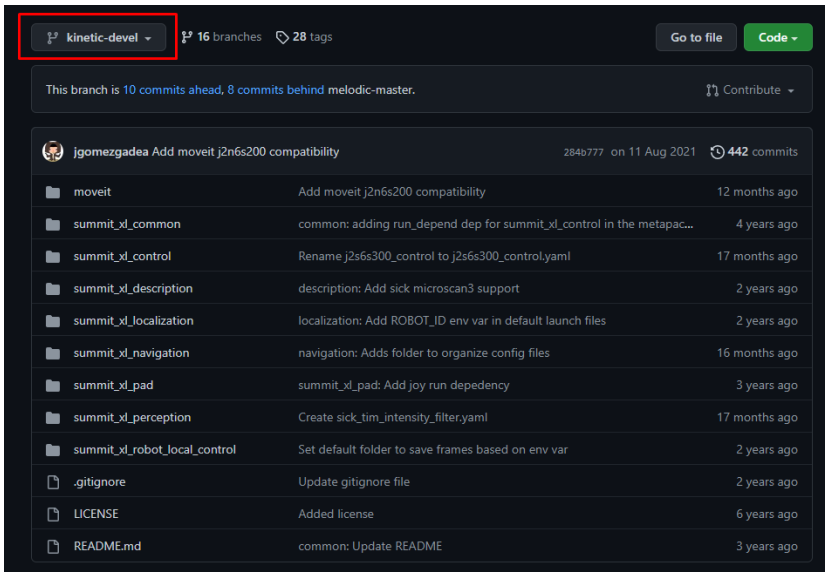


Figura A-3. GitHub de Robotnik

Para la configuración posterior del GPS, debemos acceder a la rama de la versión Melodic de robotnik_sensors, ya que en el paquete disponible en la Kinetic-devel, no incluye el archivo .urdf de ublox. Una vez en ella, descargaremos el siguiente archivo y lo incluiremos el directorio urdf de nuestro paquete de robotnik_sensors:

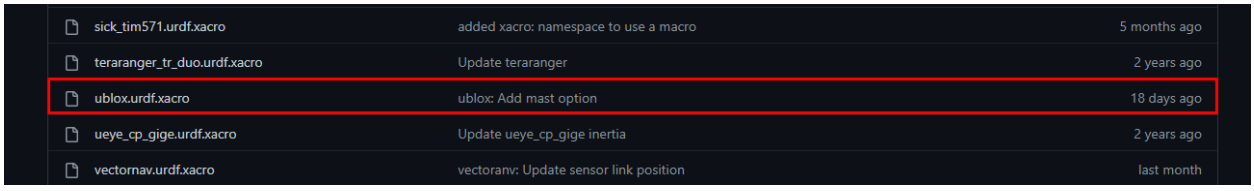


Figura A-4. GitHub del modelo del GPS

Una vez hayamos copiado los paquetes en nuestro *workspace*, lo compilamos mediante estos comandos:

```
• cd ~/ros_workspace
• catkin_make
• source devel/setup.bash
```

Tras esto, al intentar compilar puede ser que el terminal nos comunique que faltan algunas dependencias, entre las que se pueden incluir:

- Joint_state_controller
- Ackerman_msgs
- Ackerman_vehicle
- Dynamixel_sdk

Para instalar estas dependencias debemos introducir el siguiente comando en el terminal, sustituyendo los guiones bajos (_) por guiones normales (-).

```
• sudo apt-get install ros-kinetic-<nombre-del-paquete>
```

Para comprobar si se han instalado correctamente, volvemos a ejecutar catkin_make. En caso de que nos de un nuevo error al faltar otra dependencia repetimos el comando anterior y volvemos a comprobar.

A.3.2 Iri_point_cloud_hole_detection

Para la detección de agujeros en el suelo emplearemos el paquete *Iri_point_cloud_hole_detection*. Este puede descargarse del repositorio de GitLab abierto por su creador. Para que pueda compilar previamente deberemos haber instalado *iri_base_algorithm* e *iriutils*. Basta con clonar los repositorios de las dependencias y el paquete principal en nuestro workspace y realizar la compilación.

```
• git clone
  https://gitlab.iri.upc.edu/labrobotica/ros/iri_core/iri_base_algorithm.
  git
• git clone
  https://gitlab.iri.upc.edu/labrobotica/algorithms/iriutils.git
• git clone
  https://gitlab.iri.upc.edu/labrobotica/ros/navigation/3d_navigation/iri
  _point_cloud_hole_detection.git
• catkin_make
```

A.3.3 Iri_obstacle_detection_normals

```
• git clone
  https://gitlab.iri.upc.edu/labrobotica/ros/navigation/3d_navigation/iri_ob
  stacle_detection_normals.git
```

```
catkin_make
```

A.4 Conexión del ordenador

Para llevar a cabo la conexión con el ordenador de a bordo del Summit XL, primero debemos obtener el nombre del usuario y la dirección IP de nuestro PC. Si se está utilizando una máquina virtual:

- El nombre de usuario o hostname se corresponderá con el nombre elegido al instalar Ubuntu seguido de -Virtualbox, si es este el software que se ha utilizado.
- La dirección IP tendrá un formato similar a 127.0.X.X.

Para obtenerlos con certeza, debemos introducir las siguientes órdenes por un terminal:

```
hostname
hostname -I
```

Ahora, debemos encender el Summit-XL girando el interruptor presente en la parte trasera de este a ON y conectarnos a su red WiFi. Esta red tiene de nombre **SXL00-190926AA** y su clave es **R0b0tn1K**.

Una vez conectados debemos seguir los siguientes pasos:

1. Agregar el hostname del robot al archivo *hosts* dentro de la carpeta */etc*.

```
cd /etc
sudo nano hosts
```

Al ejecutar estos dos comandos, se abrirá el fichero hosts en la propia terminal. Debemos añadir al principio el hostname del robot y su IP:

```
192.168.0.200  summit
```

Tras esto, lo guardamos y lo cerramos.

2. En otro terminal, nos debemos conectar con el ordenador del robot:

```
export ROS_MASTER_URI=http://summit:11311
ssh summit@summit
```

Si la conexión de ha realizado correctamente, se debe sustituir el nombre de usuario en la terminal por summit@summit. Tras esto compilamos:

```
cd catkin_ws
source devel/setup.bash
```

3. Al igual que en el paso 1, editar el archivo */etc/hosts*, pero ahora el del ordenador del robot.

```
cd /etc
sudo nano hosts
```

Al abrirlo, incluimos nuestra dirección IP y el hostname.

A.5 Adición de paquetes al robot real

A continuación, se procede a explicar los pasos seguidos para implementar los paquetes en el Summit-XL real. Si los paquetes a utilizar constan de pocos archivos puede aprovecharse la conexión mediante ssh puede realizarse la copia mediante comandos. Primeramente, se deberá conectar al Summit-XL y crear en el directorio *catkin_ws/src* carpetas con los mismos nombres que las presentes en el paquete original mediante la orden mkdir. Una vez seleccionado el directorio elegido, para copiar los archivos se ejecuta el siguiente comando:

- `scp archivo_para_copiar summit@summit:~catkin_ws/src/directorio_creado`

Este proceso se irá repitiendo hasta que el paquete quede con la misma estructura exactamente que en el ordenador original.

En el caso de este proyecto, los paquetes tienen numerosos archivos para ser implementados de esta manera. Para que no sea tan tedioso se optó por abrir el robot y conectar el ordenador de a bordo a un monitor mediante el puerto HDMI y un ratón y un teclado por USB, además de un pen drive. Como el robot solo tiene dos puertos USB se conectó un HUB USB para disponer de más puertos y poderlo usar todo sin problemas.

Figura A-5. Interior de Summit-XL

Una vez conectado el pen drive se copian los paquetes en el directorio catkin_ws y compilamos:

- `catkin build`
- `source devel/setup.bash`

Es importante realizar mediante la compilación con catkin build, ya que a diferencia del workspace en el ordenador personal, inicialmente no se hizo con *catkin_make*.

Para los paquetes empleados de detección de pendientes y agujeros al compilar daba errores de permisos al acceder al fichero de configuración dentro de la carpeta cfg. Para solucionarlos, debemos darle permisos de ejecución mediante el comando desde la carpeta cfg:

- `chmod +x archivo.py`

A.6 Simulación en el robot real

Para llevar a cabo la simulación en RVIZ, inicialmente se planteó el uso de la comunicación SSH, pero la visualización en el ordenador es demasiado lenta y tediosa. Por ello se optó por ejecutar la simulación directamente en el ordenador en vez de acceder remotamente al del Summit-XL. Los comandos que se deben introducir en el primer terminal son los siguientes

```
• export ROS_MASTER_URI=http://summit:11311
• ssh summit@summit
• export ROS_IP=192.168.0.200
```

Por otro lado, en otro terminal, introducir:

```
• export ROS_MASTER_URI=http://summit:11311
• export ROS_IP=<Dirección IP del ordenador principal>
• rosrun rviz rviz
```

Esto no debería dar problema si se está haciendo uso de una partición del disco. Pero este trabajo se realizó sobre una máquina virtual en VirtualBox. En esta, al efectuarse una conexión, solo se tiene una IP local que no sirve para esta labor que queremos llevar a cabo. Se necesita una IP con formato 192.168.XX.XX y no 127.0.X.X. Una posible solución sería modificar la configuración de red de la máquina y cambiar el adaptador de red de tipo NAT a un adaptador puente. Esto sirve para conexiones por cable y en algunas ocasiones para conexiones WiFi. Sin embargo, la conexión inalámbrica con el robot no entra dentro de esta excepción, por lo que para solucionar este problema se optó por utilizar el NUC asociado al TurtleBot que tenía varias versiones de Ubuntu, incluida la 16.04.

Figura A-6. NUC

El sistema operativo tenía asociada una contraseña desconocida por lo que no se podía editar el archivo hosts, que es un paso crucial para el funcionamiento de la simulación. Para cambiar la contraseña se siguieron los siguientes pasos:

1. Reiniciar Ubuntu.
2. En el menú de selección de sistema operativo, elegir *Opciones Avanzadas para Ubuntu* de la versión requerida.
3. Cuando aparezca el siguiente menú, seleccionar *root – Consola de superusuario.*
4. Activar las opciones de lectura y escritura mediante la siguiente orden:

```
mount -rw -o remount /
```

5. Tras esto ya podemos cambiar la contraseña insertando este comando:

```
passwd
```

Una vez solucionado este problema, ya podemos editar el archivo *Hosts* y ejecutar la simulación sin problema.

Para lanzar los diferentes archivos de lanzamiento que se necesitan debemos acceder al directorio del workspace dentro del robot y compilar.

- `cd ~/catkin_ws`
- `catkin build`
- `source devel/setup.bash`

Una vez compilado el workspace ya podemos ejecutar los nodos sin problema:

```
roslaunch summit_xl_sim_bringup summit_xl_complete.launch
```

- Nodo de navegación GPS:

```
roslaunch summit_xl_common robot_localization_odom.launch
roslaunch summit_xl_common rnavsat_transform_node.launch
```

- Nodo de detección de agujeros:

```
roslaunch iri_point_cloud_hole_detection node.launch
```

- Nodo de detección de pendientes:

```
roslaunch iri_obstacle_detection_normals node.launch
```