Universidad Zaragoza
1542

Trabajo Fin de Grado

Navegación autónoma de un robot sobre terreno irregular usando mapas de elevación

Navigation of an autonomous robot on irregular terrain using elevation maps

Autor

Pilar Vidal Aguilar

Director

Luís Montano Gella

Grado de Ingeniería Electrónica y Automática
Escuela de Ingeniería y Arquitectura

2019

RESUMEN

A través de este proyecto se busca la navegación autónoma de un robot sobre un terreno donde hay obstáculos u otros tipos de impedimentos para esa navegación. Se pretende identificar los obstáculos de tal forma que el robot sepa si son transitables o no. Estos obstáculos son principalmente rampas, zanjas o escalones que dependiendo de las características del robot serán o no atravesables o tomados como obstáculos no transitables. El objetivo, además de que el robot pueda planificar la trayectoria y navegar por el entorno, es el registro de todos los obstáculos encontrados con sus características, para que luego otro robot pueda planificar su navegación en base a ese mapa.

Para la realización del proyecto se utilizará el entorno de trabajo para el desarrollo de *software* para robots conocido como ROS (*Robot Operating System*) y el simulador a utilizar será Gazebo. Mediante estas herramientas, se conseguirá poner a prueba el *software* implementado para el robot mediante la definición de sus parámetros característicos.

El robot donde se va a implementar el *software* es el Pioneer 3-AT de tracción diferencial, capaz de moverse en una variedad de terrenos (suelo interior, arenoso, asfalto, barro, etc) y con el sensor láser para la navegación.

Se utilizará un mapa de elevación obtenido por un sensor de haces de láser (lidar) que más tarde será procesado por el robot para la obtención del mapa de transitabilidad.

El sensor a utilizar para la detección de parámetros es el *Velodyne HDL-32E*, un sensor pequeño, ligero y resistente que emite hasta 32 haces de rayos láser que barren un ángulo vertical de hasta 40°. Será el encargado de la obtención del mapa 3D del terreno y los obstáculos.

Con el mapa de elevación del terreno obtenido por *Velodyne HDL-32E*, se obtendrá un mapa de costes para la navegación, que dependerá de los parámetros del robot para así obtener un mapa de obstáculos que comprenderán desde objetos que se encuentren en el escenario como rampas con una pendiente mayor a la máxima que puede subir el robot o escalones de altura mayor a la atravesable.

El mapa obtenido servirá tanto como para la navegación autónoma del mismo robot como para informar a otros robots de los objetos que se encuentran en el entorno de tal forma que, dependiendo de las características del nuevo robot, este sea capaz de etiquetarlos como obstáculos o zonas transitables para él.

ÍNDICE

ÍNDICE DE FIGURAS

1. INTRODUCCIÓN

1.1. CONTEXTO

La robótica es un campo de la tecnología que se encuentra en auge desde los últimos años. Combina diferentes disciplinas, desde la mecánica hasta la inteligencia artificial. Al pensar en ello, mucha gente puede relacionarlo con la ciencia-ficción, sin embargo, se encuentra presente en muchas más aplicaciones de las que se puede imaginar.

La automatización de procesos es un ejemplo muy extendido en el sector industrial. Se busca la reducción de la intervención humana utilizando las tecnologías para el control de los procesos y máquinas. Hasta ahora, principalmente se han utilizado para tareas repetitivas. Sin embargo, el objetivo es conseguir que estos dispositivos tengan la capacidad de llevar a cabo las mismas tareas que realiza el ser humano de forma autónoma.

Los vehículos autónomos requieren tener la capacidad de reconocer el terreno que les rodea de tal forma que la navegación sobre el mismo se realice de forma segura y eficiente.

Se ha investigado en diferentes ocasiones la obtención de un mapa de forma global, donde se cuenta con una localización absoluta exacta. Estos métodos dependen de condiciones de lo que les rodea como luz o señales externas como GPS.

Sin embargo, en este caso se busca la posibilidad de llevar a cabo la navegación autónoma en lugares donde las condiciones no sean las idóneas, como podría ser un túnel subterráneo, donde se realice una localización mediante odometría o basada en localización mediante lidar o una combinación de ambos.

El mapa que se obtendrá en este caso será local donde se mostrará lo que rodea al robot y se irá moviendo con él para luego poder crear un mapa global a través de la fusión de todos.

Es este trabajo se tratará la navegación autónoma de robots en diferentes escenarios, que presentan distintos obstáculos que deberán salvar. Los impedimentos encontrados serán o no obstáculos dependiendo de las capacidades del robot. Por lo tanto, aunque el mapa inicial de la zona a recorrer sea el mismo para todos robots, el mapa conocido como de transitabilidad sobre el cual el robot navegará dependerá del robot.

Se va a seguir con la línea de investigación del grupo de Robótica del Instituto de Investigación de Ingeniería de Aragón (I3A) de la Universidad de Zaragoza.

Este proyecto va en la línea del DARPA Challenge que comienza en 2019, el *"Subterranean Challenge"* o *"SubT Challenge"* que busca dar a conocer la importancia de los ambientes subterráneos, incluyendo túneles, cuevas y zonas urbanas bajo tierra. En el *Challenge* se comprobará la autonomía, la percepción, la comunicación y la movilidad en redes bajo tierra en condiciones impredecibles. Aquí se va a realizar unos experimentos semejantes a la primera fase del *Challenge,* la parte de simulación utilizando ROS y Gazebo.

1.2. OBJETIVOS

En este proyecto se van a tratar los algoritmos de navegación sobre terrenos irregulares, teniendo en cuenta las capacidades de cada robot, que en este caso se trata del Pioneer 3-AT, para etiquetar los obstáculos, ubicarlos en el mapa global y construir mapas de costes para la navegación a través de los mapas de transitabilidad.

Se busca que, a través de la exploración autónoma del terreno por parte de un robot, se consiga crear un mapa que describa todos los obstáculos encontrados como rampas, agujeros, escalones y objetos, entre otros. Una vez obtenido ese mapa, se puede trasladar a otro robot que, habiendo parametrizado todas sus capacidades, planee su navegación sobre el mismo terreno, que puede o no seguir la misma trayectoria que el robot que realizó el mapa.

1.3. TAREAS Y METODOLOGÍA

Las tareas que se han realizado en este trabajo han sido las siguientes:

1. Instalación de la herramienta ROS *Melodic* junto con el sistema operativo Ubuntu 18.04 y realización de cursos en la misma, tratando sus puntos más generales.
2. Familiarización con los entornos de simulación (Gazebo) y visualización (Rviz).
3. Creación de un mapa y posterior localización en el mismo usando el sensor *Velodyne HDL-32E*
4. Creación de mapas de elevación del terreno y aplicación de filtros (vectores normales, inclinaciones, escalones).
5. Obtención del mapa de costes, donde se etiquetan las zonas transitables y los obstáculos.
6. Navegación a partir de la planificación anterior.
7. Realización de diferentes experimentos con diferentes escenarios y parámetros relativos a la posición del sensor.

8. Posibilidad de guardar el mapa de elevación para ser posteriormente utilizado por el mismo robot u otro distinta.
9. Documentación y conclusiones de los experimentos.

La metodología que se va a utilizar es la siguiente:

- **Creación del mapa de elevación.** Para ello, se va a usar el sensor lidar *Velodyne HDL-32E* que, a través de un conjunto de haces láser, devuelve una nube de puntos al rebotar contra los objetos. De esta forma, conociendo su posición respecto a la del robot, se puede crear el mapa de elevación del terreno mediante el paquete *Elevation Mapping.* Este mapa va a tener forma de *grid map* o mapa de celdas, donde cada casilla tiene información de su elevación.

- **Creación del mapa de inclinación.** A partir del mapa de elevación obtenido anteriormente y posterior aplicación del filtro de vectores normales en *z,* se obtiene el valor de inclinación de cada casilla. Este mapa también tiene forma de *grid map.*

- **Etiquetado de obstáculos para crear el mapa de costes.** A partir de los mapas obtenidos anteriormente, se recorren ambos y, dependiendo de qué valores toman la elevación e inclinación teniendo en cuenta los valores críticos, se obtiene el mapa de costes, que tiene formato de *costmap (Occupancy Grid)* sobre el que se pueden aplicar algoritmos de navegación autónoma.

- **Navegación sobre el mapa de costes.** Una vez obtenido el mapa, se ajustan los parámetros del nodo ROS *move_base,* encargado de la navegación. Algunos de los parámetros más importantes a dar valor son los relativos a velocidades lineales, angulares y aceleraciones.

La localización en este caso se hace a través del simulador Gazebo. Se considera localización perfecta, desacoplando los problemas de localización y navegación, para centrarnos en el etiquetado de mapas y en la navegación.

1.4. TRABAJO PREVIO

Partimos de un punto donde el robot es capaz de navegar de forma autónoma gracias a un sensor escáner láser embarcado de un haz que, al hacerlo incidir sobre un objeto, lo marca como obstáculo. Sin embargo, este láser es horizontal y está colocado a una determinada altura. Por lo tanto, si el obstáculo de una altura menor a la altura que está colocado el láser, éste no lo verá. Entonces no lo etiquetará como obstáculo sino como zona transitable. El robot intentará navegar por esa zona y, con una probabilidad alta, se desestabilizará produciéndose una caída.

El uso de este tipo de escáner láser necesita que se realice una integración de medidas durante el movimiento para poder en cuenta inclinaciones o escalones. Por ello, se va a implementar un tipo de navegación con un sensor con varios haces láser, que barren un ángulo grande, para poder hacer un mapa 3D más detallado de obstáculos y superficie sobre la que se navega.

1.5. ESTRUCTURA DEL PROYECTO

La memoria del proyecto está estructurada de la siguiente manera:

Capítulo 1. Introducción. Aquí se puede encontrar una breve presentación del trabajo, así como sus objetivos principales y la metodología utilizada para llevarlo a cabo.

Capítulo 2. Árbol de transformaciones. Localización de sistemas de referencia.

Capítulo 3. Toma de datos mediante lidar. Se explica el sensor utilizado para la toma de datos, tanto sus características como el procesado de los datos que recoge.

Capítulo 4. Obtención del mapa de elevación. Explicación de cómo se crea el mapa de elevación a partir de los datos del sensor. Se hablará de los paquetes de ROS utilizados para obtener ese mapa de elevación.

Capítulo 5. Obtención del mapa de costes y salvaguarda de mapa. Se explica cómo se lleva a cabo el procesamiento del mapa de elevación y los filtros aplicados para la obtención del mapa de costes y posterior navegación. También se habla del guardado de mapas.

Capítulo 6. Nodos de ROS. Se exponen todos los nodos de ROS mediante un mapa de nodos y *topics* utilizados, así como una breve explicación de cada nodo.

Capítulo 7. Evaluación experimental. Se presentan los experimentos más relevantes, así como las conclusiones de cada uno.

Capítulo 8. Conclusiones. Se realiza una vista global del trabajo, teniendo en cuenta desde dónde se partía y a dónde se ha llegado. Se expondrán también los problemas encontrados durante la realización del proyecto y posibles mejoras y se hablará de nuevas posibles investigaciones en el tema.

2. ÁRBOL DE TRANSFORMACIONES

Para la realización y comprensión del trabajo de investigación, es necesario tener en mente cómo es el árbol de transformaciones entre sistemas de referencia. Esto ayuda en la localización del robot y sus sensores, entre otros.

En la imagen se muestra cuál es ese árbol de transformaciones en este caso:

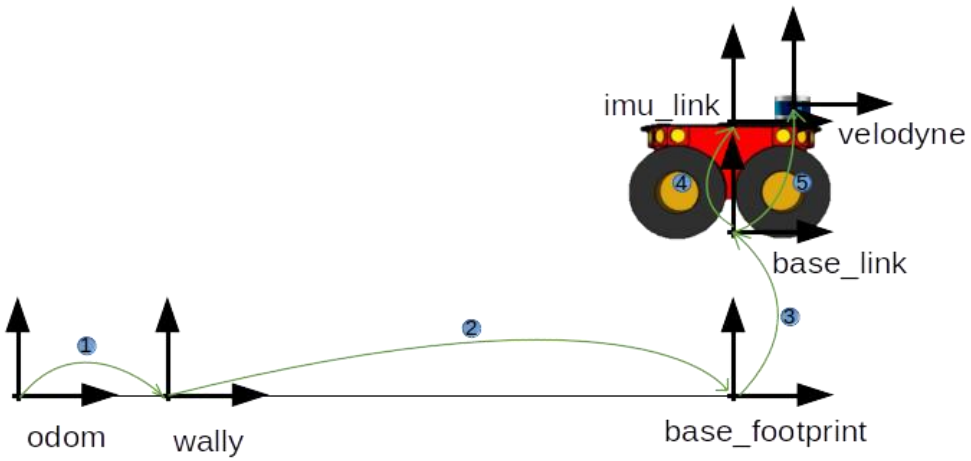

Figura 1. Referencias y transformaciones

1. **Transformación *odom-wally.*** Representa la posición inicial del robot *(wally*) respecto de la absoluta *(odom).*
2. **Transformación *wally-base_footprint.*** Representa la posición del robot proyectada sobre el plano horizontal *(base_footprint)* respecto de la posición inicial del mismo *(wally).*
3. **Transformación *base_footprint-base_link.*** Representa la posición de la base del robot *(base_link)* respecto a su proyección sobre el suelo *(base_footprint).*
4. **Transformación *base_link-imu_link*.** Representa la posición del sensor inercial IMU *(imu_link)* respecto de la base del robot *(base_link).* Ésta es necesaria porque, como se explicará más adelante, se utilizará un sensor inercial IMU para la detección de la inclinación del robot en superficies no planas.
5. **Transformación *base_link-velodyne*.** Representa la posición del sensor lidar *(velodyne)* respecto de la base del robot *(base_link).*

3. TOMA DE DATOS MEDIANTE LIDAR

El sensor utilizado para la navegación es un lidar (***L**ight **I**maging **D**etection **a**nd **R**anging*). Se trata de un dispositivo que determina, a través de un haz láser pulsado, la distancia desde un emisor láser a un objeto o superficie.

Para determinar la distancia que existe al objeto, se mide el tiempo de retraso entre la emisión del pulso y la señal reflejada.

Se utiliza en aplicaciones diversos campos como topografía, geología, sismología, física de la atmósfera, control de calidad en procesos industriales, entre otras. Sin embargo, una de las aplicaciones más novedosas es la conducción autónoma de vehículos, o en nuestro caso, de robots. En esta aplicación se puede utilizar tanto para la creación del mapa como para la localización sobre un mapa ya existente.

Este sensor es similar al SONAR (***So**und **N**avigation **a**nd **R**anging*) y al RADAR (***Ra**dio **D**irection **a**nd **R**anging*). La principal diferencia es que mientras el lidar utiliza haces de luz, el sonar utiliza de ondas sonoras y el radar utiliza ondas de radio.

El lidar nació en los años 60 tras la llegada del láser. Durante la misión del Apollo 15 en 1971, los astronautas consiguieron un mapa de la superficie de la luna, que sirvió de impulso del lidar. Durante el final del siglo XX hasta la actualidad, ha sido uno de los sensores más utilizados en Robótica, conjuntamente con los sistemas de visión. Alrededor de los 2000, el lidar se comenzó a utilizar en coches, haciéndose famoso por su instalación en el coche Stanley, construido por la Universidad de Stanford para el Grand DARPA Challenge de 2005.

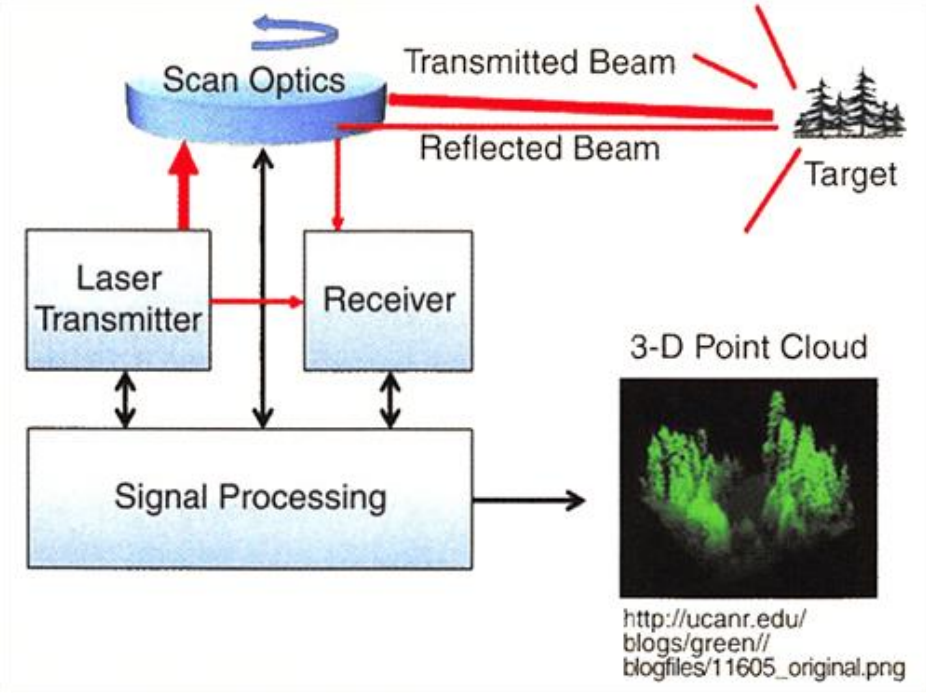

Figura 2. Funcionamiento del lidar

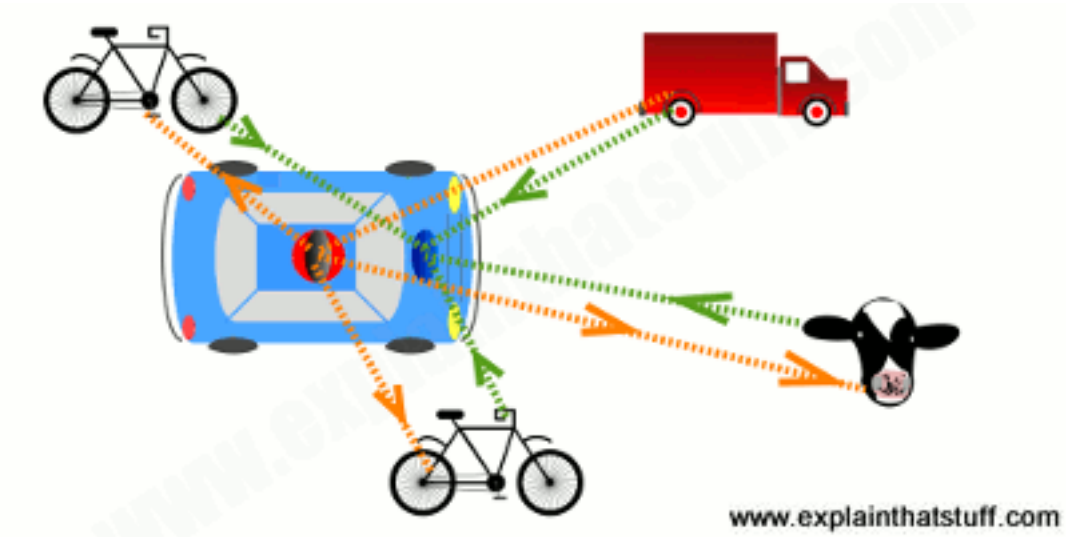

Figura 3. Funcionamiento del lidar en coches autónomos. Haz emisor en naranja, haz receptor en verde. Su forma de uso es la misma que en nuestro caso

Los pasos para el cálculo de la distancia son los siguientes:

1. Emitir un pulso láser sobre una superficie
2. Captura del láser reflejado de nuevo a la fuente del pulso lidar con sensores
3. Medida del tiempo que ha transcurrido entre los pasos anteriores
4. Cálculo de la distancia mediante la siguiente ecuación:

$$Distancia = \frac{Velocidad\ de\ la\ luz * Tiempo}{2}$$

En este caso, se utilizará el sensor *Velodyne HDL-32E*, que crea imágenes 3D usando 32 pares de láser para escanear el escenario.

Con nuestro sensor *Velodyne HDL-32E*, conseguimos crear el mapa de elevación. Devuelve una nube de puntos que barren los 360° en el campo horizontal y unos 40°en el vertical, por lo que, colocado estratégicamente en el robot, conseguimos ver tanto objetos hasta una determinada altura como rugosidades en el terreno alrededor del robot.

Un ejemplo del mapa que se obtiene es el siguiente:

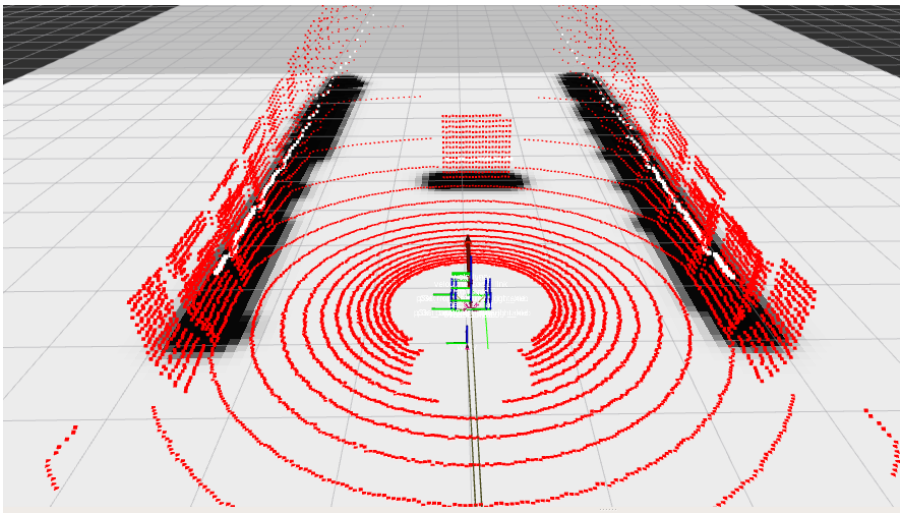

Figura 4. Nube de puntos de devuelve el lidar navegando en el túnel ante un objeto en forma de de prisma. En blanco, los puntos relativos a la nube de puntos proyectados sobre el plano horizontal relativo al sensor. La parte trasera del robot donde la nube de puntos se corta es debido a que el sensor no está colocado de forma simétrica sobre éste, sino en la parte superior delantera (ver Figura 1), por lo que la visión no es simétrica. En capítulos posteriores se explicará con detalle este hecho.

En ella se puede observar de qué forma ve las paredes del túnel y un objeto con forma prismática en su frente.

Añadir también que, primeramente, durante la toma de contacto con la navegación en ROS, se utilizaba un solo haz láser obtenido del proyectado de la nube de puntos sobre el plano horizontal del sensor. De esta forma, se podía navegar de una forma sencilla evitando los obstáculos sin tener en cuenta el mapa de elevación, sino utilizando directamente el nodo *move_base,* creando la capa de obstáculos. El láser es publicado en el *topic /laser_front/scan* por parte del nodo */pointcloud_to_laserscan.*

Estos datos del sensor lidar se publican en un *topic,* cuyo mensaje tiene forma de *sensor_msgs/PointCloud2.* Aquí, los puntos se almacenan como *blobs* binarios, y se describen datos relativos a dimensiones de la nube de puntos y dimensiones de los puntos y filas de puntos en *bytes.*

A este *topic* es al que se suscribirá el paquete de *Elevation_mapping* para obtener el mapa de elevación.

4. OBTENCIÓN DEL MAPA DE ELEVACIÓN

Cuando se trata de navegar sobre terreno irregular, es necesario tener en cuenta las tres dimensiones del terreno. La forma más utilizada para esta navegación es la obtención de un mapa de elevación del terreno, donde cada coordenada en el plano horizontal se asocia con un valor de altura o elevación.

Para ello, se van a comentar los paquetes que se han usado y su pilar fundamental, el paquete *grid map.*

4.1. GRID MAP (PACKAGE)

Existe una librería que se adapta bien a este caso, y se trata de *grid map*. El mapa se divide en celdas en los ejes x e y mediante una retícula, así cada casilla tiene unos valores de coordenadas *x* e *y*. A su vez, el mapa se divide sobre el eje *z* en capas o *layers*, que se sitúan una encima de la otra y pueden describir la elevación o las normales del terreno.

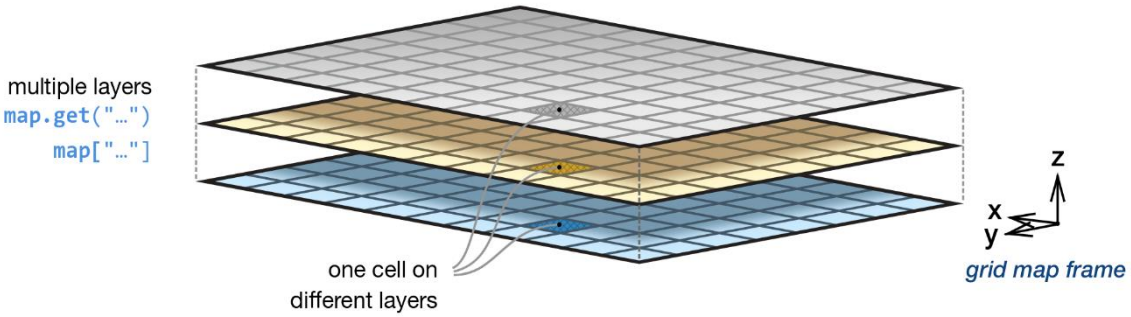

Figura 5. Mapa tipo grid map con sus capas y celdas

Algunas capas que sirven para describir cada celda son las de elevación, vectores normales y *"slope"* o rampa.

Lo primero que se hace es obtener el mapa de elevación obtenido por el paquete *Elevation Mapping* (que se comentará en el 4.2), a partir de la nube de puntos obtenida por el lidar. Posteriormente, hay que suscribirse al *topic* donde se publica el mapa de elevación que se ha obtenido y luego se aplican los filtros antes nombrados, que vienen contenidos en el paquete g*rid map* Aquí explicaremos las más relevantes para nuestro caso:

- **Capa de vectores normales.**

Se obtiene mediante los vectores de valores propios de cada celda y su posterior normalización.

- **Capa de *slope* o rampa.**

Esta capa se obtiene a partir de la capa de vectores normales en z. El ángulo de inclinación se calcula como como α*=acos(Normal_z)* (**Figura 6**) de los vectores normales, referentes a cada celda. Mediante la aplicación del filtro de vectores normales a la superficie que viene incluido en el paquete *grid map* se pueden dar los siguientes casos:

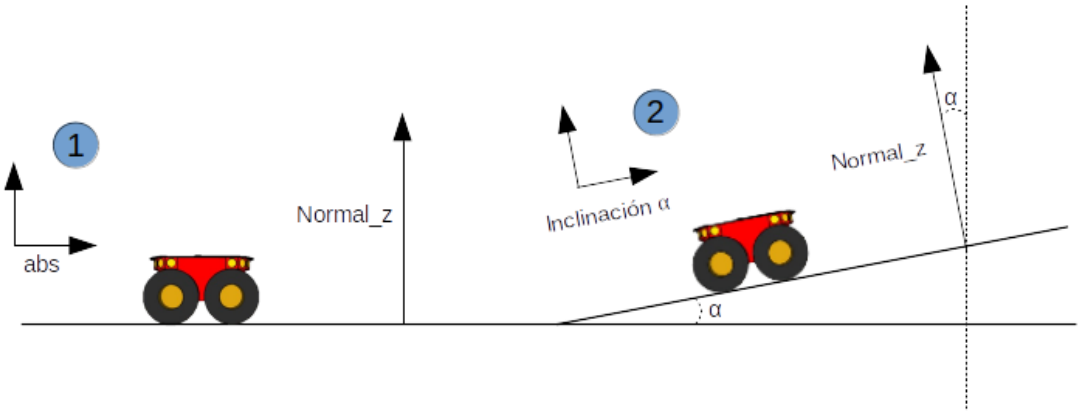

Figura 6. Obtención de la inclinación a partir de la normal en z

1. En este caso, el robot se encuentra atravesando una zona de terreno llano. Aquí, el vector normal a la superficie tiene un ángulo de 90° respecto de la horizontal. Por lo tanto, en este caso, la pendiente es 0°.

2. En este caso, el robot se encuentra atravesando una zona inclinada. Para conocer el valor de la pendiente α, se aplica el arco coseno del vector normal a esa pendiente. El ángulo α es el mismo que forma el vector normal con la vertical.

La información obtenida a partir de la aplicación del filtro de inclinación se publica en un *topic* en forma de *grid_map_msgs/GridMap,* donde se encuentra el mapa con la capa de *slope* y en cada casilla su valor de inclinación.

A partir de ese mapa, se construirá el mapa de costes para su posterior uso en navegación.

4.2. ELEVATION MAPPING (PACKAGE)

El paquete *Robotic-Centric Elevation Mapping* es un paquete de ROS creado para terrenos elevados navegados por un robot móvil. El principio de funcionamiento es el mismo que el del paquete *Grid Map*, que divide el mapa en celdas en la dirección horizontal y capas en la dirección vertical. Este *software* está diseñado para la navegación con robots equipados con una estimación de la posición (odometría e IMU, en nuestro caso) y un sensor de distancia (lidar, en nuestro caso).

El funcionamiento del paquete es el siguiente. Se tienen 4 referencias distintas:

- Referencia inercial I. Solidaria al terreno.
- Referencia del robot B.
- Referencia del sensor S. Unida a la referencia del robot
- Referencia del mapa M.

Conociendo las transformaciones y rotaciones entre ellas y los posibles grados de libertad que existen, cada celda i del mapa viene dada por $P_i = (x_i, y_i, \widehat{h_i})$, donde x_i, y_ison las coordenadas de la casilla i y $\widehat{h_i}$es la elevación estimada de la celda i , aproximada por una distribución normal Gaussiana.

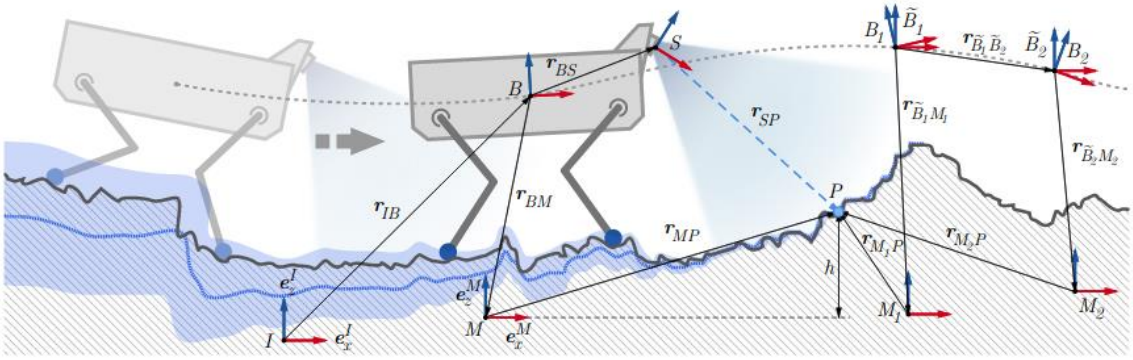

Figura 7. Aquí se muestran los frames utilizados por el paquete. "I" es el frame incercial, que se encuentra fijo al terreno, "B" es el frame relativo al robot, al que tiene fijado el frame "S", referente al sensor. Por último, el frame "M" se trata del referente al mapa.

Este paquete se suscribe al *topic* del *PointCloud* del sensor láser. El mapa de elevación se publica en un *topic* en forma de *grid_map_msgs/GridMap,* donde se da el valor de elevación relativa a cada casilla del mapa. Posteriormente se obtendrá el mapa de inclinaciones.

Tras esta explicación de ambos mapas, en las siguientes figuras se pueden observar un ejemplo de ambos ante una rampa de 5°

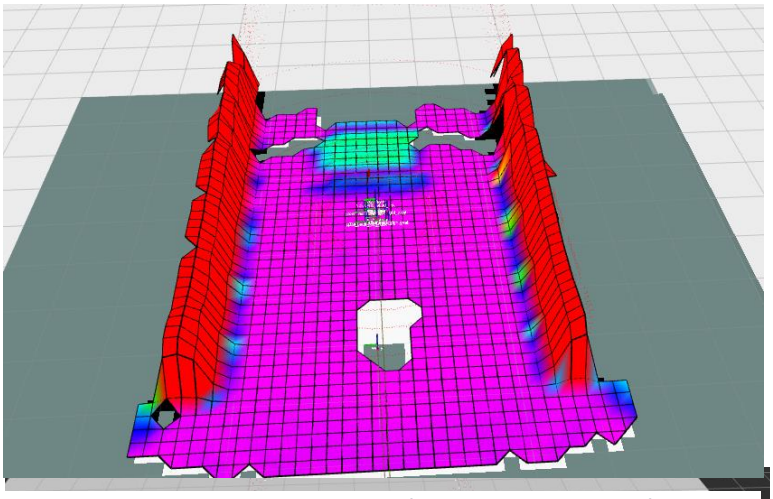

Figura 8. Mapa de elevación de 5º de inclinación

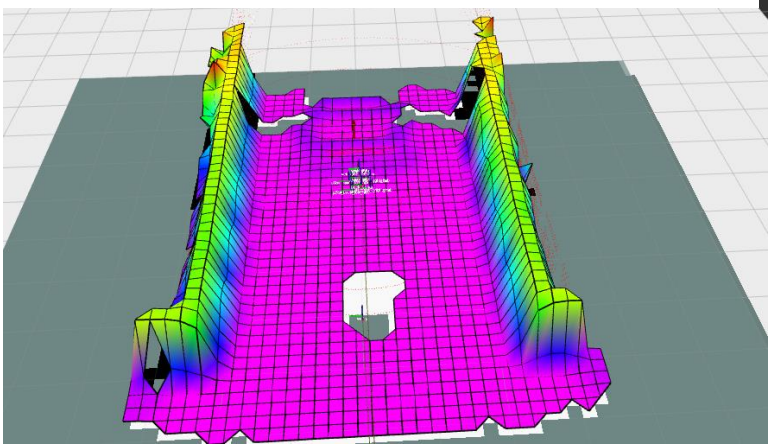

Figura 9. Mapa de inclinación de una rampa de 5° de inclinación

Figura 10. Escena en el simulador Gazebo donde se encuentra la rampa de 5° de inclinación

4.3. TRAVERSABILITY ESTIMATION (PACKAGE)

Este paquete obtiene la traversabilidad de un mapa de elevación (en forma de *Grid Map*) teniendo en cuenta las características de cada robot (inclinaciones y escalones que puede subir o bajar, entre otras). En este mapa cada celda del grid se etiqueta como transitable o no, permitiendo de esta manera a los algoritmos de navegación transitar.

Este paquete valdría para casos en el que el escalón de subida y bajada crítico sea el mismo. Sin embargo, en nuestro caso se utilizará el código que utiliza la capa de elevación para ver escalones de subida y bajada ya que en el caso de escalones de subida, la condición de nuestro robot es muy limitada.

A partir del mapa de elevación obtenido por el paquete *Elevation Mapping* se crea el mapa con la capa de *step.*

Su funcionamiento es el siguiente:

Se realiza un círculo con el radio elegido y se busca la diferencia de alturas. Si el valor encontrado es menor que el valor definido como crítico, la casilla toma un valor mayor de 0 y menor o igual que 1, siendo 1 el terreno llano sin escalones y el valor próximo a 0, valor próximo al escalón crítico. Si el valor del escalón supera al del escalón crítico, la casilla toma el valor de 0.

$$valor\ casilla = 1 - \frac{escalón}{valor\ de\ escalón\ crítico}$$

$$escalón = \ valor\ de\ escalón\ critico - valor\ de\ casilla \cdot valor\ de\ escalón\ crítico$$

Por lo tanto, mediante este mapa, en conjunto con el mapa de inclinaciones obtenido anteriormente, se obtendrá el mapa de costes teniendo en cuenta las capacidades del robot.

Un ejemplo de cómo se vería la capa de *step* es el siguiente:

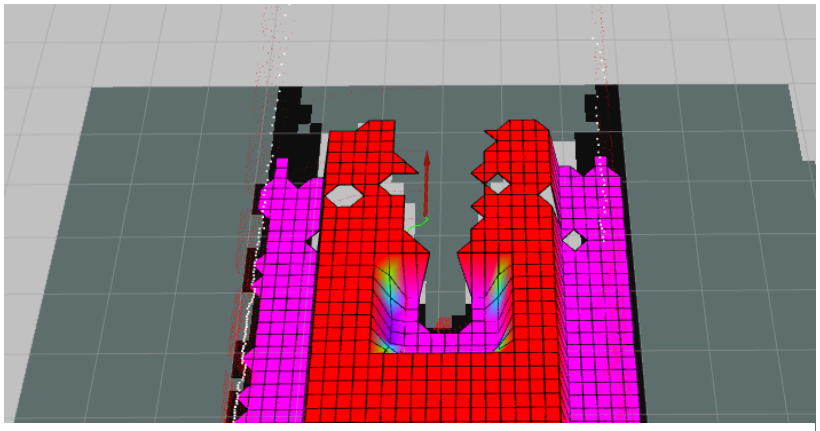

Figura 11. Ejemplo de la capa de step. Las zonas llanas van en rojo ya que toman valor de 1. Las zonas rosas son las relativas a escalones, que toman el valor de 0. En el centro se muestra cómo identifica un cubo de 1m de altura.

4.4. UTILIZACIÓN DEL IMU

Una unidad de medición inercial o IMU *("**I**nertial **M**easurement **U**nit")* es un dispositivo electrónico que sirve para medir la velocidad, orientación y fuerzas gravitacionales, que usa una combinación de acelerómetros, giróscopos, magnetómetros y sensores de presión.

En este caso, se utilizará para conocer la inclinación del robot (giro *pitch*). Es útil en cuanto a su utilización para crear el mapa de costes. Como el sensor lidar es solidario al robot, en el caso de que el robot esté atravesando una rampa y ésta en un punto aumente su inclinación, la inclinación que el robot verá será la diferencia entre su horizontal en ese momento y la nueva inclinación, pero deberá aplicar una potencia de valor relativo a la inclinación absoluta. Por lo tanto, conviene tener un sensor que mida la inclinación del robot en todo momento, para poder sumársela a la inclinación que devolverá el mapa de elevación.

La expresión para obtener la inclinación absoluta del terreno para poder comparar con la inclinación máxima que puede atravesar el robot y así etiquetar como obstáculo o no es la siguiente:

$$inclinación\ absoluta = inclinación\ IMU + inclinación\ mapa\ elevación$$

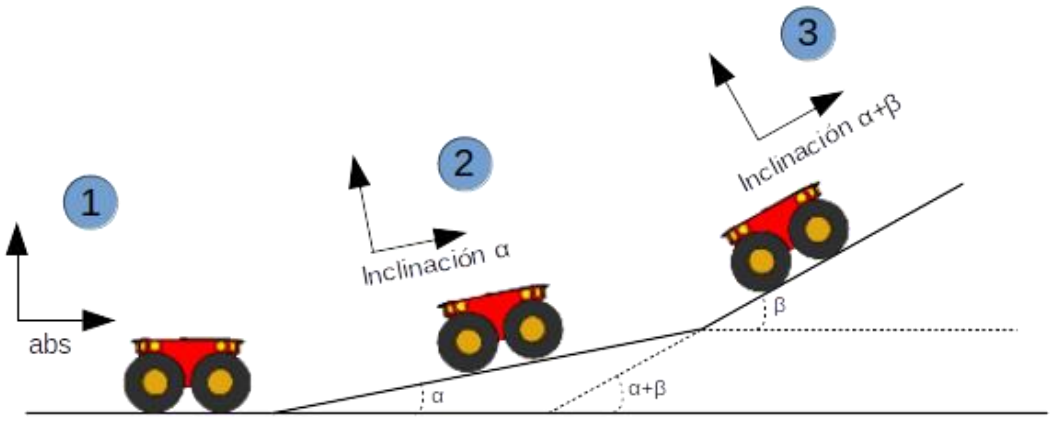

Figura 12. Diferentes inclinaciones del terreno. Importancia del uso del IMU

Centrándonos en el caso de la imagen:

1. Cuando el robot se encuentre en la posición 1 antes de pasar a la 2, la inclinación del IMU será de 0° ya que es terreno llano. El mapa de elevación con el filtro de *slope* mostrará el ángulo α, que es la inclinación del 2. Si α es menor que el ángulo

crítico del robot, éste subirá por la rampa. En el caso contrario, será etiquetado como obstáculo.

2. Cuando se encuentre en la posición 2, justo antes de pasar a la 3, el mapa de elevación con el filtro de *slope* devolverá el ángulo β, sin embargo, ese ángulo es solamente el relativo al robot. Para saber si éste podrá subirlo hay que verlo en ángulos absolutos. Para ello utilizaremos la medida del IMU, que en este caso tomará el valor de α y, aplicando la expresión antes comentada, la inclinación absoluta que hay que comparar con el valor crítico de inclinación que puede subir el robot es α+β.

El sensor IMU publica sus valores en el *topic /imu,* y podemos encontrar valores relativos a orientación, velocidad angular y aceleración líneal. A partir del cuaternio de orientación que devuelve, se obtiene el valor de *pitch,* que es el único valor que se necesita para conocer la inclinación del robot.

Otra forma de conseguir el mismo efecto que con el IMU es almacenando el valor de inclinación de superficie que se va atravesando. Al crear la trayectoria a partir del punto objetivo, si el robot tiene que atravesar una zona de una cierta inclinación, la variable en cuestión tomará el valor de esa inclinación. Si la inclinación aumenta aún más, la variable se incrementará en ese valor. Sin embargo, si el robot vuelve a navegar sobre terreno llano, la variable pasará a tomar el valor 0 de nuevo. Es una forma de llegar al mismo resultado sin tener que utilizar otro sensor.

Sin embargo, en este caso se utilizará el IMU.

5. OBTENCIÓN DEL MAPA DE COSTES Y SALVAGUARDA DEL MAPA

5.1. OBTENCIÓN DEL MAPA DE COSTES

Para la obtención del mapa de costes se usa la capa de elevación y la capa de *slope.* El mapa de costes es el que se usará para planificar la navegación.

Para ello, el nodo (o código) se suscribirá al *topic /grid_map_filter_demo/filtered_map,* que es donde se encuentra el mapa con la capa de *slope* y que tiene formato *gridMap.* También se suscribirá al *topic* del mapa de elevación, que se llama */elevation_mapping/elevation_map.* Por último, se suscribirá también al *topic /imu* para obtener los datos de inclinación.

El nodo publicará sus resultados, en este caso el mapa de costes, en el topic */mapobs,* en forma de *nav_msgs/OccupancyGrid.* Se visualiza utilizando Rviz en ROS.

En el mapa de costes se representan 3 estados:

- Obstáculo (mostrado como negro en Rviz)
- Libre (mostrado como blanco en Rviz)
- Desconocido (mostrado como gris en Rviz)

Para construir el mapa de costes, se va dando valores a las celdas del mapa de costes de la siguiente forma:

- **Obstáculo.** A los objetos etiquetados como no atravesables se les da valor de 100. Este campo engloba también rampas con inclinación mayor que la crítica y escalones mayores que el escalón crítico. Los valores críticos que se han escogido son los siguientes:
    - Rampa de subida. Valor crítico de 8°.
    - Rampa de bajada. Sin valor crítico.
    - Escalón de subida. Valor crítico de 3 cm.
    - Escalón de bajada. Valor crítico de 10 cm.

    En apartados posteriores se explicará por qué se han escogido esos valores.
- **Desconocido.** Las casillas referentes a zonas desconocidas del mapa se les da valor de -1.

- **Zona llana, con rampa o escalón atravesables.** En este caso, el valor que se le asigna a la celda es de 0, ya que se trata de una zona libre para navegar.

La navegación se lleva a cabo mediante el nodo *move_base.* En el apartado de **Anexo VI**, se incluirán los archivos de configuración del mismo. El algoritmo de navegación calculará el camino más corto al objetivo a través de la zona transitable (valor 0 en el mapa de coste).

Debido al ángulo muerto del sensor, de radio de unos 50 cm, las casillas de alrededor del mismo inicialmente no reciben valores de elevación, por lo que el mapa les daría en valor de desconocido. Si esto ocurriese, el robot no podría empezar a navegar. La solución para ello es darles valor de casilla libre solamente en el instante inicial.

En las siguientes figuras se observa lo comentado. El mapa de elevación de la zona inicial del robot no tiene valor, por lo que le correspondería valor de desconocido (-1). Sin embargo, se le asigna el valor de transitable (0) para que el robot pueda moverse.

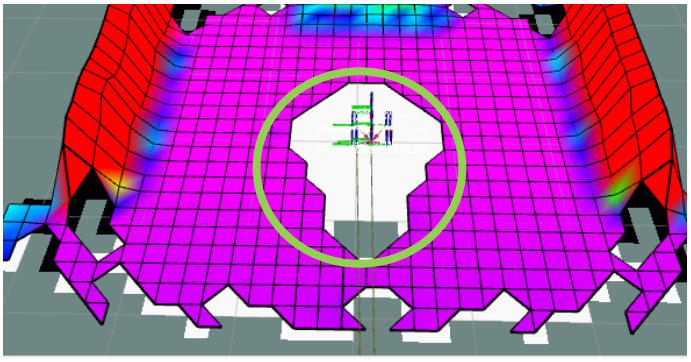

Figura 13. Mapa de elevación de la zona inicial

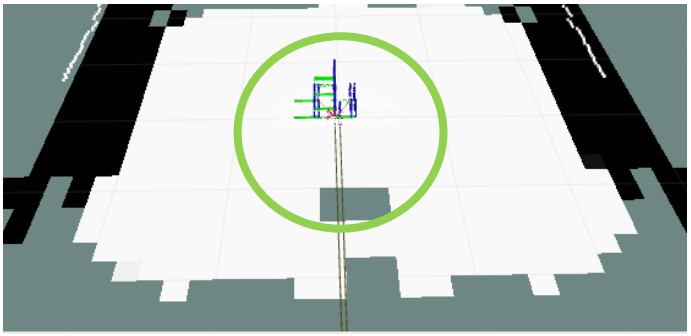

Figura 14. Mapa de costes de la zona inicial

Cabe comentar también la zona gris obtenida en el mapa de costes justo detrás de la zona donde el robot inicia la navegación. Esto ocurre ya que el sensor *Velodyne* no está colocado en el centro de la parte superior del robot sino más cerca de la parte delantera. Por lo tanto, la zona de visión no es simétrica. El robot necesita empezar a moverse para que los haces reconozcan esa zona.

Como el tiempo de obtención del mapa de costes es elevado (6-8 segundos aproximadamente), transcurre un tiempo entre que el robot comienza a ver la zona, se generan los correspondientes mapas de elevación e inclinación y se obtiene el mapa de costes, dándoles el valor real que les corresponde, no el valor de desconocido inicial. Muchas de las capturas están tomadas en ese intervalo de tiempo y por eso las zonas aparecen como desconocidas.

El diagrama de flujo que explica cómo se lleva a cabo la tarea de obtención del mapa de costes completo es el siguiente:

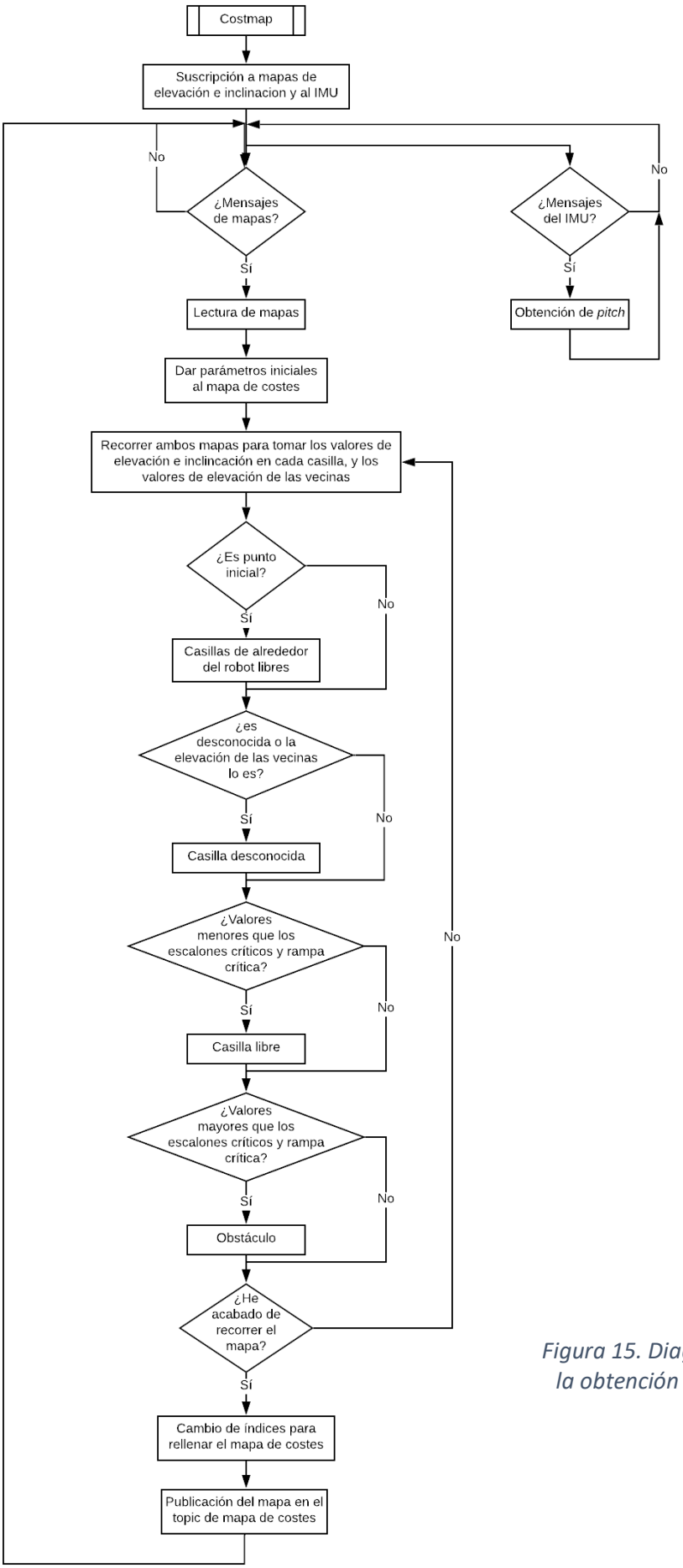

Figura 15. Diagrama de flujos para la obtención del mapa de costes

En el algoritmo se puede ver cómo se ha hecho para diferenciar los escalones de subida de los de bajada. En el momento de recorrer el mapa para ver qué valor toma cada celda de elevación e inclinación, se toma el valor de elevación de las casillas vecinas. Si la casilla en cuestión es mayor que sus vecinas, habrá escalón de bajada. En el caso contrario, el escalón será de subida.

Nótese que la asignación de obstáculos a las casillas se encuentra en último lugar. De esta forma se consigue que, si por cualquier cosa a una casilla se le intenta asignar dos estados diferentes, tiene preferencia la última, que es la de obstáculo. Es una medida de precaución.

Este mapa de costes se configura para navegar sobre él en los ficheros de configuración del nodo ROS *move_base* (**Anexo VI**). Se toma el *costmap* publicado en */mapobs* como una capa *static* del mapa, que representa ese mapa de costes generado por SLAM *(Simultaneous Localization and Mapping)* y será sobre la que se realice la navegación. El mapa local es sobre el cual el robot planifica la navegación para moverse y el global acumula todos los locales para conseguir un mapa competo.

A continuación se muestra una imagen donde se ha impuesto un objetivo al robot y planifica su trayectoria, en verde en la imagen.

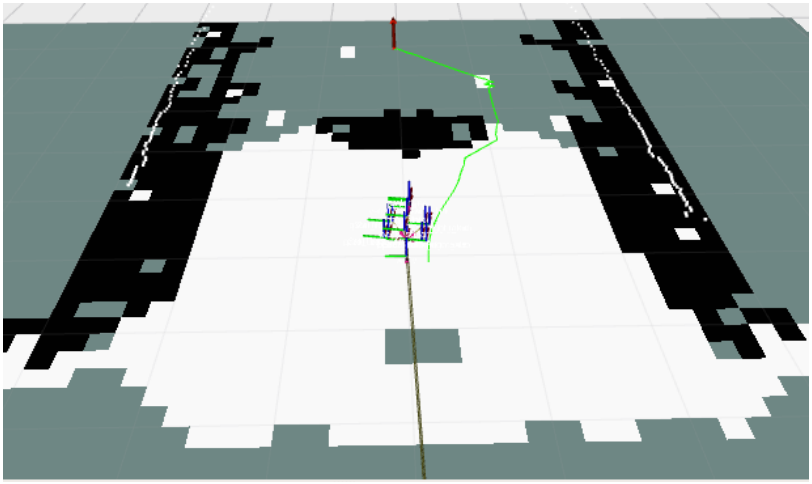

Figura 16. Planificación de la trayectoria sobre el mapa de costes para salvar el obstáculo (en verde)

5.2. PROCESAMIENTO Y SALVAGUARDA DE MAPA

Uno de los objetivos principales en este tipo de aplicaciones es conocer el terreno por el que se navega, ya sean los objetos que se encuentran, los desniveles u otro tipo de obstáculos. Por lo tanto, conviene que quede constancia de todos ellos, colocados sobre el mapa global de ese terreno.

Para ello, en el *package* de ROS de *Elevation_mapping* existe un servicio de ROS (*rosservice)* que se llama "*save_map".* Lo que hace es guardar el G*rid Map* del *topic* que se especifique (en este caso */elevation_mapping/elevation_map)* en el formato *"bag"* en el *path* donde se elija.

Los *"bags"* se crean mediante la herramienta *"rosbag"*, que se suscriben a uno o más *topics* de ROS y almacenan *data* en serie en el archivo.

Se almacena el mapa de elevación del terreno, como ya se ha dicho anteriormente. Por lo tanto, sólo hace falta que un robot navegue una vez por la escena para poder crear dicho mapa.

Posteriormente, el mapa almacenado el *bag* se puede traspasar a otro robot con iguales o distintas características que el que realizó el mapa.

Esta es la gran ventaja de guardar el mapa de elevación y no el mapa de costes. Cada robot tiene parámetros distintos (por ejemplo, los valores de rampa críticos cada robot puede subir o bajar son distintos dependiendo de potencia de motor, peso, estabilidad, etc) por lo que, al haber almacenado el mapa de elevación y conociendo esos parámetros, se puede obtener el mapa de costes para ese robot concreto sobre ese mapa concreto. En el caso de tratarse del mismo robot que recorre la escena dos veces, se podría almacenar solamente el mapa de costes para reducir tiempos de cómputo a la hora de obtener ese mapa de costes a través del mapa de elevación.

6. NODOS DE ROS

6.1. MAPA DE NODOS DE ROS

En el siguiente mapa se muestran los nodos y *topics* del proyecto.

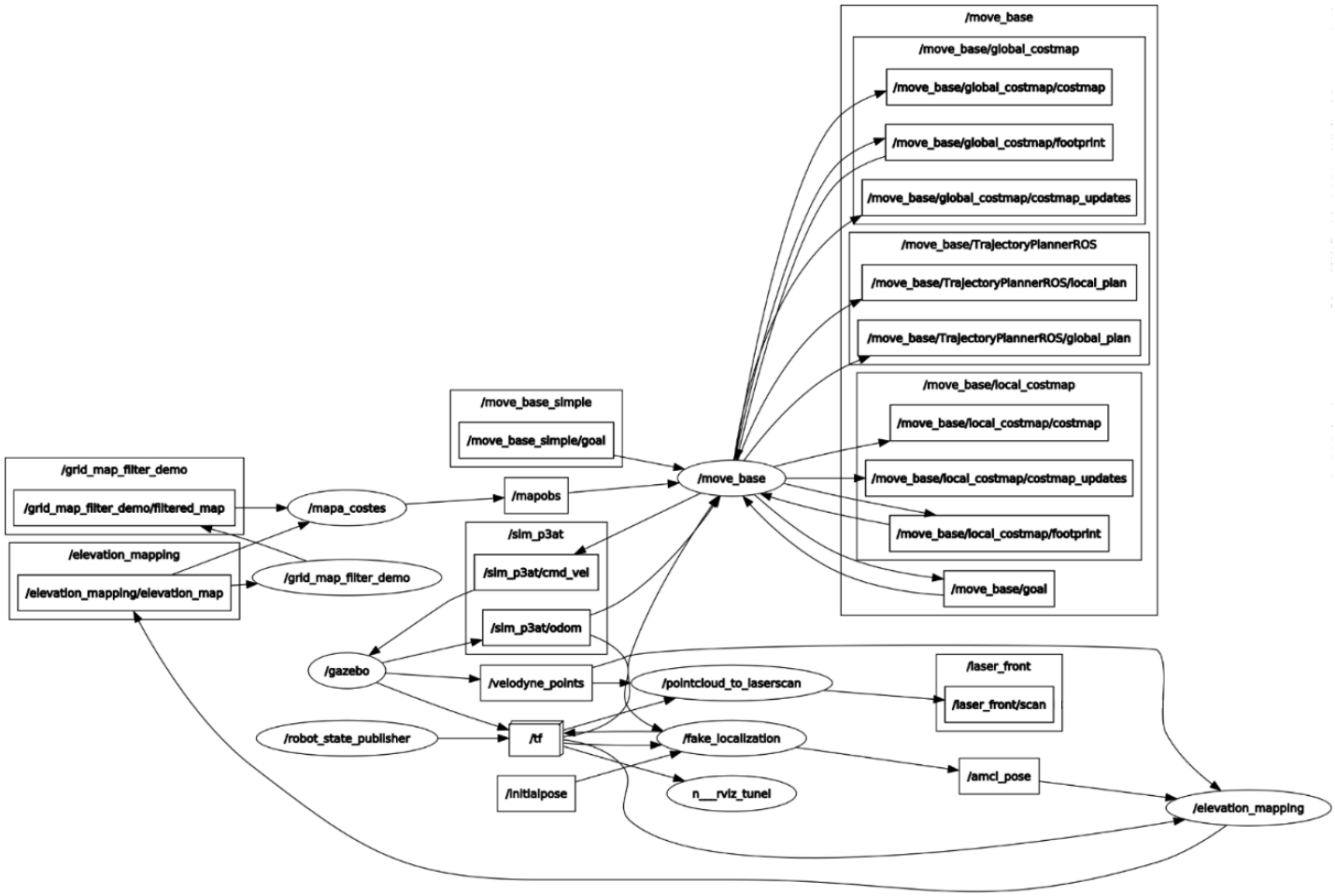

Figura 17. Los nodos de ROS activos son los contenidos en elipses y los topics se hallan dentro de un rectángulo

6.2. EXPLICACIÓN DE CADA NODO

A continuación, se va a proceder a explicar cada nodo.

Nodos de visualización

- ***/gazebo***. Se trata del simulador que se utiliza.
- ***/rviz_tunel.*** Se trata del visualizador que se utiliza.

Nodos de toma de datos

- ***/velodyne_points.*** Se trata del nodo relativo al sensor lidar.

- ***/pointcloud_to_laserscan.*** Este nodo da la posibilidad de proyectar sobre la horizontal del sensor la nube de puntos del lidar para una navegación menos compleja. Se utilizó al principio para empezar a manejar la navegación y hacer pruebas.

Nodos informativos

- ***/rosout.*** Nodo que da información sobre el proceso en la consola.
- ***/robot_state_publisher.*** Nodo que te permite publicar el estado de los *frames* (transformadas) del robot.

Nodos de creación de mapas

- ***/elevation_mapping.*** Nodo encargado de crear el mapa de elevación.
- ***/grid_map_filter_demo.*** Nodo encargado de obtener el mapa de inclinación.
- ***/mapa_costes.*** Nodo encargado de crear el mapa de costes

Nodos de localización y navegación

- ***/fake_localization.*** Nodo que sustituye el sistema de localización, ya que se usa los valores de localización que devuelve el simulador, en este caso Gazebo. Se suele usar para simulaciones ya que la localización que devuelve es perfecta.
- ***/move_base.*** Nodo encargado de llevar a cabo la planificación para la navegación. Se encarga de calcular la trayectoria para llegar al punto especificado con la velocidad configurada en este nodo.

7. EVALUACIÓN EXPERIMENTAL

Se han realizado varios experimentos para poner a prueba el código implementado, así como los paquetes que se han instalado. Todas estas pruebas se han realizado en simulación sobre Gazebo y visualización en Rviz. Vamos a ver todos los ejemplos con diferentes tipos de obstáculos:

7.1. EXPERIMENTO 1. Obstáculo en forma de cubo.

En esta prueba, se ha querido mostrar la detección de un claro obstáculo como podría ser una pared o piedra en el túnel. En este caso, se ha simulado como un cubo de dimensiones 1x1x1 metros.

Una vez que el obstáculo entra en la zona que el lidar puede alcanzar, la capa de elevación devuelve valores altos, como ocurre en con el caso de las paredes del túnel. Mediante la detección de escalón, se clasifica como obstáculo. En las siguientes figuras, se puede observar por un lado, la capa de elevación y por otro, el mapa de costes con la nube de puntos.

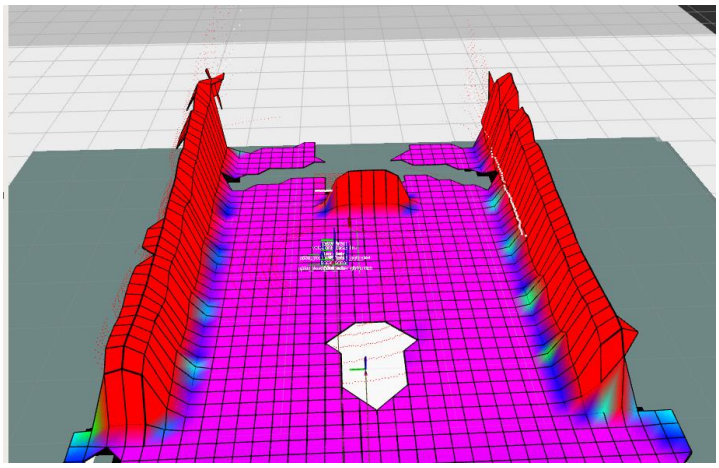

Figura 18. Mapa de elevación de un objeto en forma de cubo. La zona blanca es la posición inicial del robot, que como ya se ha comentado, no tiene valor de elevación hasta que se ha avanzado y el sensor ha comenzado a recibir datos de la zona

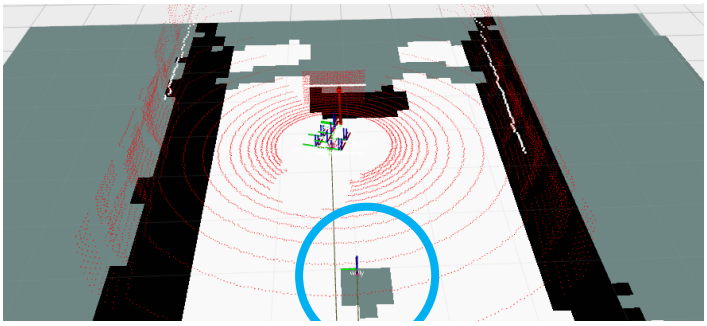

Figura 19. Mapa de costes con nube de puntos del sensor frente a un objeto en forma de cubo. En el círculo azul, ángulo muerto de la zona inicial

Tal y como se ha comentado anteriormente, la zona gris en el círculo azul corresponde a una zona inicialmente no visible por el sensor.

7.2. EXPERIMENTO 2. Rampa atravesable.

Tras varias pruebas experimentales con rampas de diferentes inclinaciones, simulando en Gazebo, se ha llegado a la conclusión que el robot no puede subir rampas más inclinadas de unos 8°, ya que el par del motor no es suficiente para atravesarlas.

Para este experimento, se ha utilizado una rampa de 5°, que se sabe que es atravesable, para ver cómo la etiqueta el robot. Como el valor de rampa crítico impuesto para la realización del mapa de costes es de 8°, en este caso el robot debe marcarla como atravesable.

Lo que vemos en las imágenes es, por un lado, el mapa de elevación realizado por el lidar, por otro, la capa de *slope* que se realiza a partir de la capa de vectores normales como se ha descrito anteriormente y, finalmente, el mapa de costes obtenido. Como vemos, la rampa no es etiquetada como obstáculo y por tanto es transitable, como indica el mapa de costes.

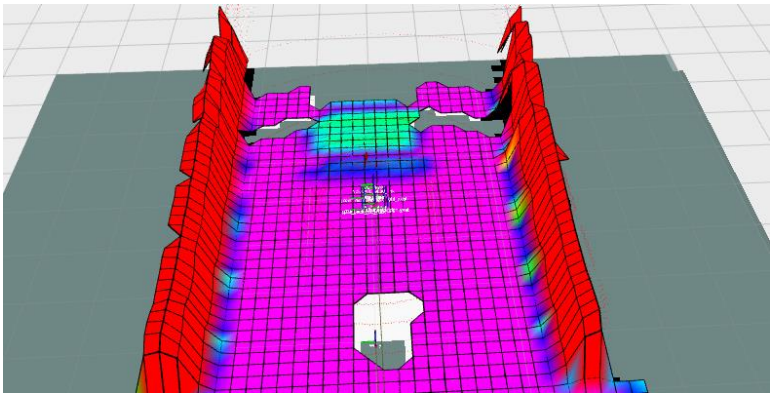

Figura 20. Mapa de elevación frente a una rampa atravesable

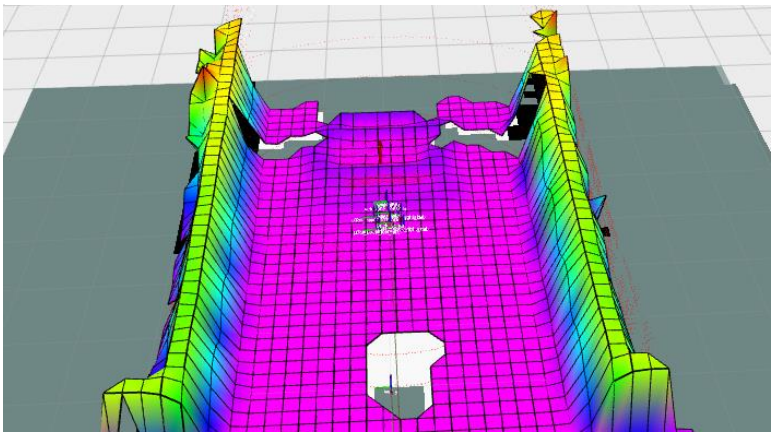

Figura 21. Mapa de inclinación frente a una rampa atravesable

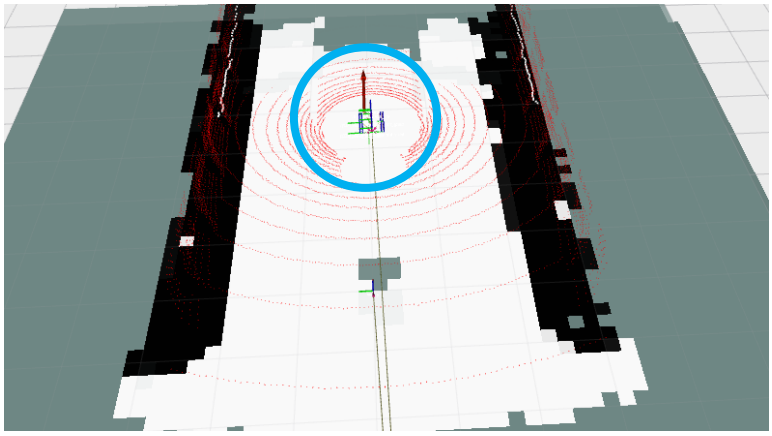

Figura 22. Mapa de costes frente a una rampa atravesable. En el círculo verde se localizaría la rampa, etiquetada como transitable

7.3. EXPERIMENTO 3. Rampa no atravesable.

Como se ha comentado anteriormente, se ha probado que el robot no puede subir rampas mayores a 8°, por lo que se va a probar que las rampas mayores a esa inclinación se marcan como obstáculo.

Para ello, se ha realizado una prueba con una rampa de 10°. Mediante la capa de inclinaciones, se obtiene que las casillas de la zona donde se encuentra la rampa toman valores mayores que el valor de rampa crítica, por lo que esta rampa se clasificará como obstáculo. Esto es exactamente lo que ocurre, como se puede ver en las figuras.

En el mapa de costes se crea una barrera (en negro en la **Figura 25**) en el contorno del obstáculo, que evita que el navegador pueda planificar y consecuentemente navegar sobre ella.

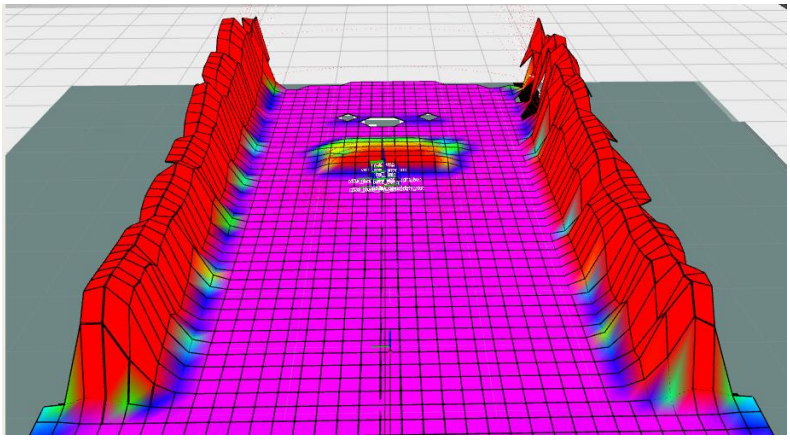

Figura 23. Mapa de elevación frente a una rampa no atravesable

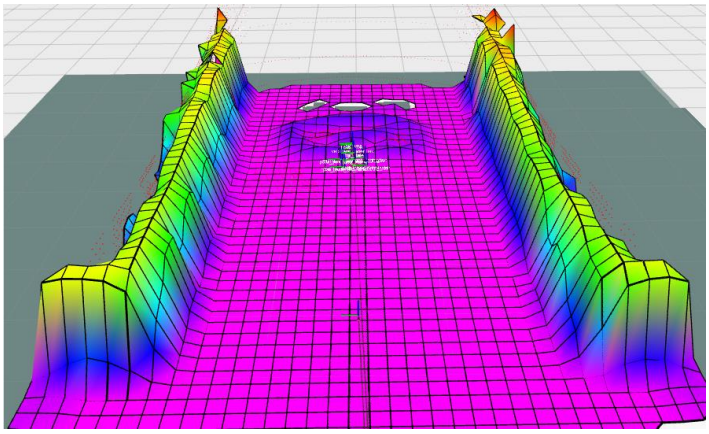

Figura 24. Mapa de inclinaciones frente a una rampa no atravesable

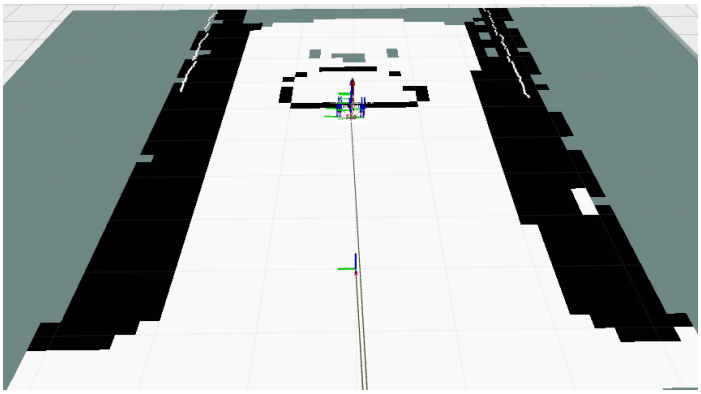

Figura 25. Mapa de costes frente a una rampa no atravesable

7.4. EXPERIMENTO 4. Escalón de bajada.

Tras varias pruebas sobre simulación, se ha descubierto que el robot no puede bajar escalones de más de 10 cm sin perder el equilibrio y acabar volcando. Por lo tanto, en este caso el valor de escalón crítico se definirá a 10 cm.

En este caso se lleva a cabo la prueba con un escalón de 20 cm. Como era de esperar, el robot toma la parte posterior del escalón como desconocido. Esto ocurre debido a que, al estar el sensor colocado sobre el robot y tener un ángulo muerto, no puede ver que hay en la casilla inmediatamente a continuación del escalón. La navegación en este caso se realizará volviendo por donde había venido. En el apartado de mejoras se comentará como solventar este problema.

En este caso, las figuras muestran tanto la capa de elevación como el mapa de costes obtenido.

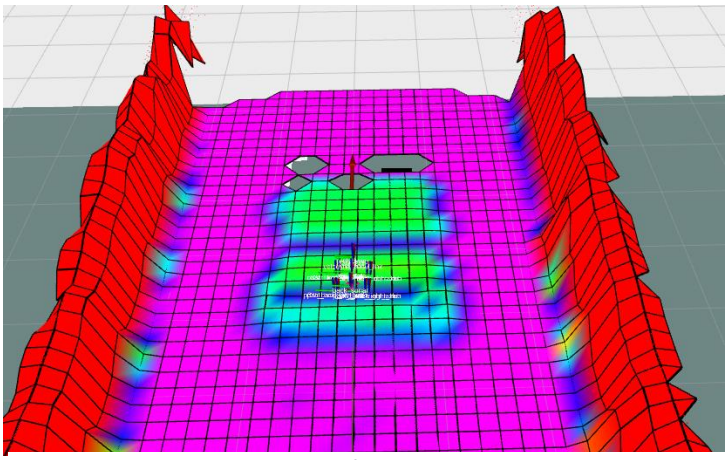

Figura 26. Mapa de elevación de rampa inicialmente y con escalón al final. Las zonas donde las casillas aparecen vacías se deben a lo explicado anteriormente. Debido a las características y posición del robot, no se puede identificar el valor de las celdas.

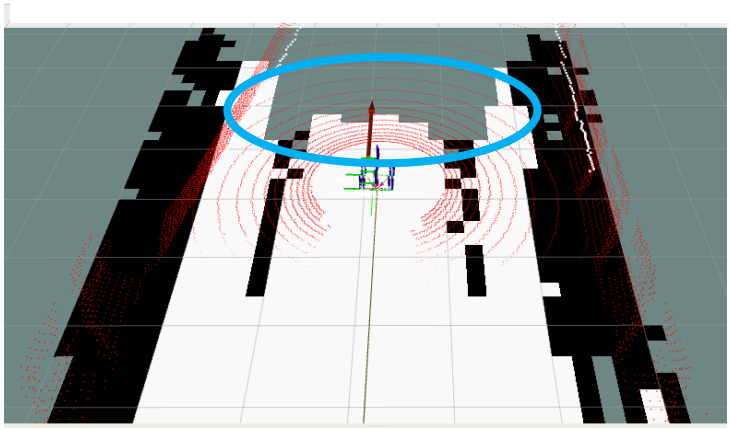

Figura 27. Mapa de costes que refleja el escalón. Los lados han sido reconocidos con anterioridad. Sin embargo, la zona inmediatamente posterior al escalón se encuentra desconocida, en gris.

7.5. EXPERIMENTO 5. Escalón de subida atravesable

Llevando a cabo varias simulaciones para determinar qué valor tomaba el escalón de subida máximo atraversable por el robot, se ha concluido que el robot apenas puede subir escalones sin perder el equilibrio. Sin embargo, no se puede tomar valor crítico 0. Para el caso práctico de la programación, se utilizará un valor de 3 cm, ya que los valores del sensor para obtener el mapa de elevación no son exactos, sino que tienen una incertidumbre. Si imponemos que ninguna celda vecina puede ser mayor que la celda en cuestión, si el valor de elevación de la casilla vecina fuese superior en una milésima, por ejemplo, sería tomado como obstáculo. El valor de 3 cm es atravesable, aunque con dificultades.

En este caso, se ha probado con un escalón de 1cm, y es marcado como libre.

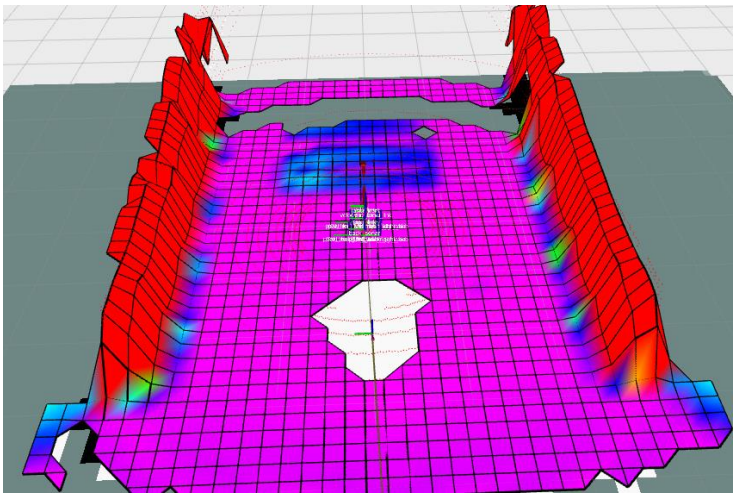

Figura 28. Mapa de elevación de un escalón de 1 cm de alto, atravesable

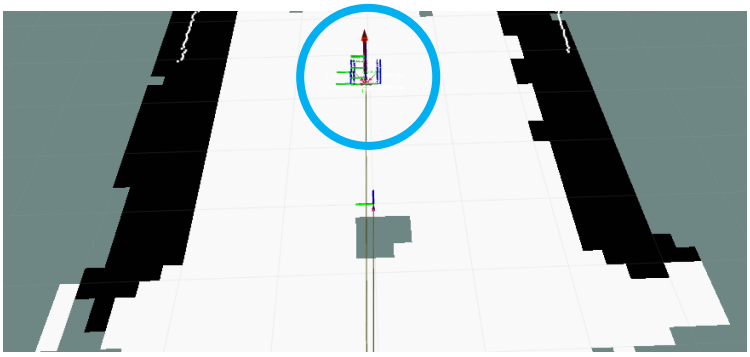

Figura 29. Mapa de costes obtenido del escalón atravesable

7.6. EXPERIMENTO 6. Escalón de subida no atravesable

Como hemos determinado antes, el robot no puede subir escalones de más de unos pocos centímetros. Por lo tanto, para hacer esta prueba se ha usado un escalón de 20 cm, obteniendo los siguientes resultados:

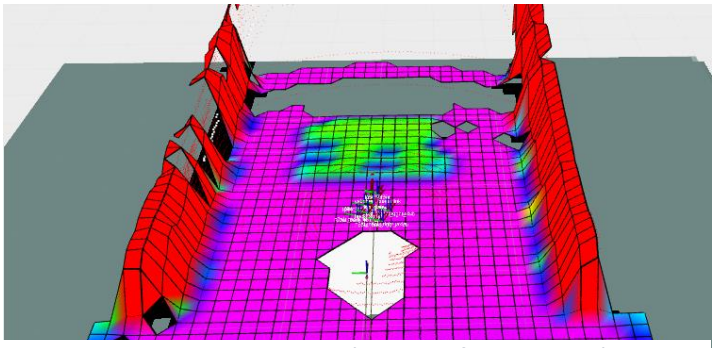

Figura 30. Mapa de elevación que refleja el obstáculo en forma de escalón no atravesable.

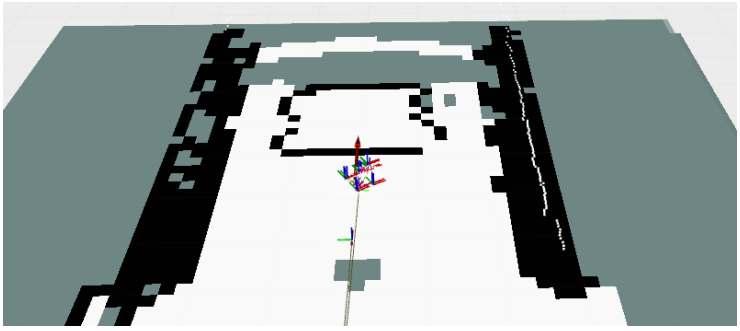

Figura 31. Mapa de costes donde se ve cómo se marcan el escalón como obstáculo

7.7. EXPERIMENTO 7. Rampa con doble inclinación

En este caso se va a poner a prueba el funcionamiento del IMU. Se va a utilizar una rampa que al principio tenga 5° de inclinación y luego pase a 11°. Teóricamente, la segunda parte es no transitable, ya que supera la inclinación crítica. Sin embargo, cuando el robot la vea, se encontrará subiendo la rampa de 5°, por lo que él solo apreciará la inclinación relativa de 6º, que para él es transitable. Para tomar la decisión de marcarlo o no como obstáculo hace falta conocer la inclinación absoluta, que como hemos dicho anteriormente se calcula sumando la inclinación del robot y la inclinación que percibe el sensor. Por lo tanto, la primera parte será transitable y la segunda no. Esto se ha podido observar en las siguientes figuras:

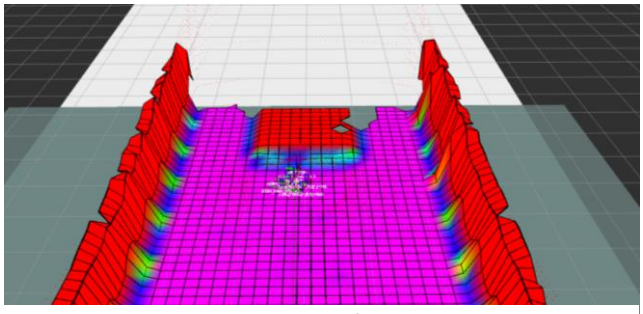

Figura 32. Mapa de elevación, donde se aprecia el cambio de inclinación de la rampa.

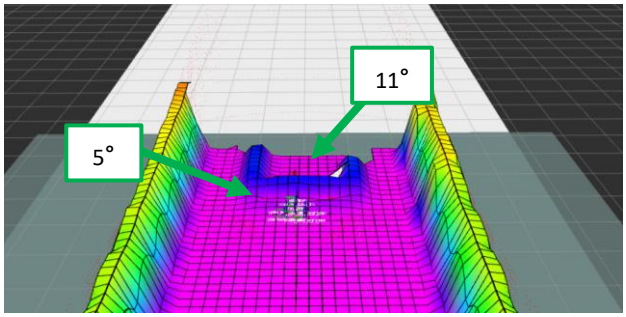

Figura 33. Mapa de inclinación, primero de 5° y luego de 11°

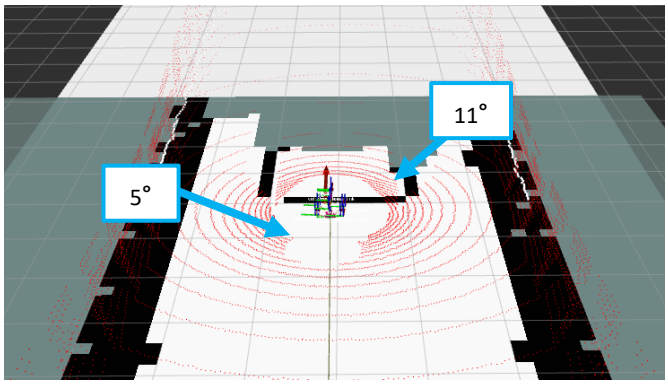

Figura 34. Mapa de costes, primero de 5° y luego de 11°

7.8. EXPERIMENTO 8. Conjunto de varios obstáculos.

En este caso se va a simular una escena con varios obstáculos que ya se han visto anteriormente: escalón atravesable y no atravesable y rampa atravesable y no atravesable.

El escenario es el siguiente:

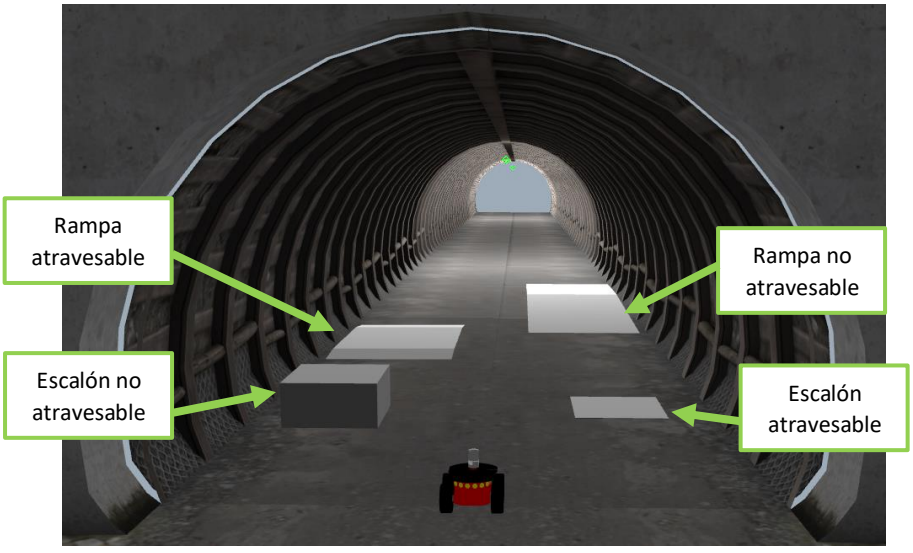

Figura 35. Escena Gazebo de la simulación.

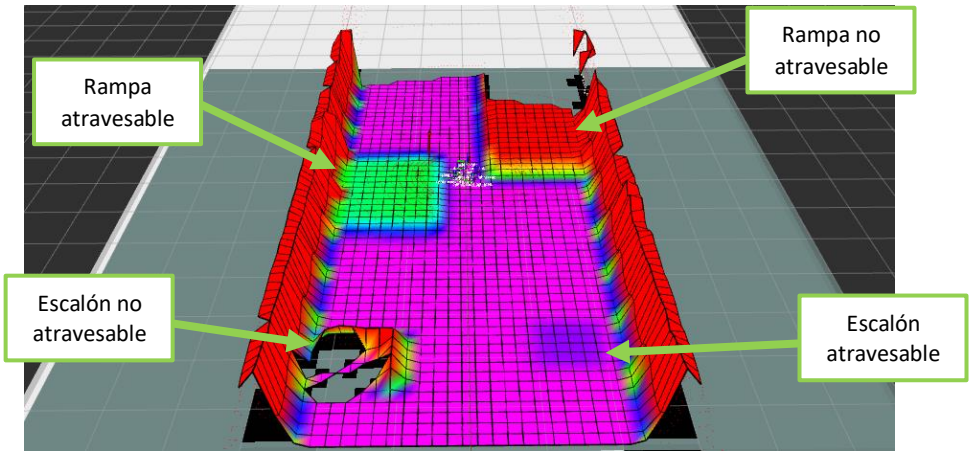

Figura 36. Mapa de elevación de la escena.

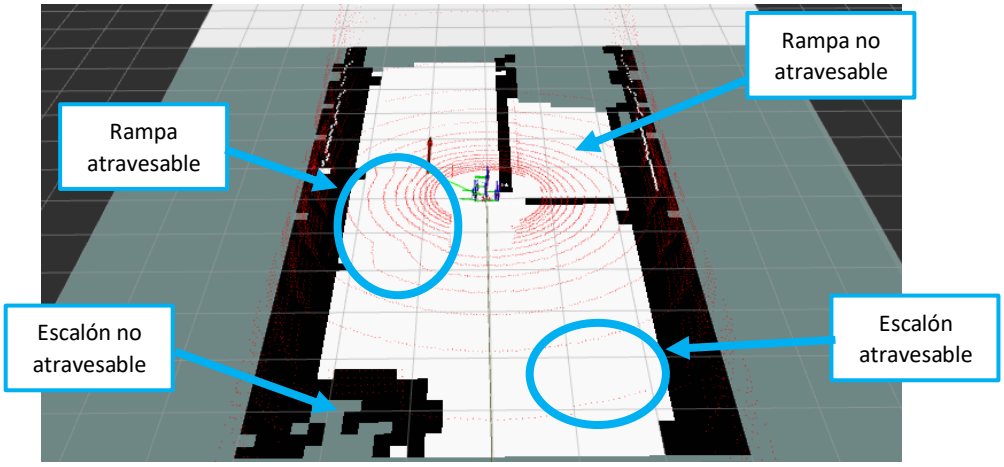

Figura 37. Mapa de costes de la escena

8. CONCLUSIONES

8.1. PROBLEMAS ENCONTRADOS

Respecto a problemas encontrados durante la realización de este proyecto, cabe destacar la velocidad de procesamiento requerida para realizar estas simulaciones. Se intentaron llevar a cabo en un ordenador portátil con procesador *Intel Core i7*, pero las simulaciones se hacían tan lentas que, por mucho que se disminuyese la velocidad del robot, el mapa de elevación se creaba más lento que el robot adelantaba al mapa. El factor temporal, que representa el cociente entre el tiempo real y el de simulación, en este caso era de menos de 0.3, siendo 1 el caso ideal donde la simulación y el tiempo real van a la par.

Haciendo las simulaciones en otro ordenador con el mismo procesador, pero con tarjeta gráfica más potente, se consiguió mejorar algo este problema, consiguiendo generar un mapa de costes que fuese adelantado en 2 o 3 metros al robot. En este caso, se consiguió subir el valor del factor temporal a 0.5-0.6 aproximadamente, cada segundo de la realidad se simula en unos 2 segundos.

Respecto a lo anterior, entra en juego el tamaño de las casillas del *grid map.* Si las casillas son muy grandes, el procesamiento y obtención del mapa de costes es mucho más rápido. Sin embargo, la precisión es muy baja y la zona marcada como obstáculo es siempre mayor que el obstáculo real. Por el contrario, si el tamaño de casillas es pequeño, el mapa es mucho más fiel a la realidad en cuanto a marcar los obstáculos, pero el tiempo de procesamiento es muy largo, por lo que no se consigue sacar a tiempo el mapa de costes y el robot se para a esperar a que aparezca.

Haciendo pruebas, se ha llegado a la conclusión que un buen tamaño de celdas para la simulación es de 20 cm.

Otra dificultad encontrada fue ROS. Se trata de un *software* muy completo y en vías de desarrollo. Es una herramienta muy potente, y por lo tanto muy compleja. Los paquetes que se han utilizado en este proyecto son muy concretos y con muy poca o nula información sobre ellos. Por lo tanto, hacer que funcionen de la forma que se desea ha sido la tarea más compleja de todo el trabajo. Una vez conocido bien el entorno de trabajo, el posible programar misiones complicadas, con unas herramientas de visualización muy potentes.

Otro problema fue el intentar realizar el mapa de costes con el filtro de escalón del paquete *traversability estimation*. Como se ha nombrado, sería la solución ideal en el caso de que el robot pudiese subir y bajar la misma altura de escalones, al poder dar valor crítico general para casos de subida o bajada, ya que el funcionamiento del filtro es muy acertado. Sin embargo, no se ha usado para llevar a cabo los experimentos.

8.2. VALORACIÓN GENERAL Y PERSONAL

El objetivo principal del proyecto ha sido poder navegar por un terreno irregular desconocido, etiquetando los objetos encontrados durante el trayecto como obstáculos, transitables o desconocidos para poder planificar la navegación.

Para ello, se han simulado varios casos del robot Pioneer 3-AT con el sensor lidar *Velodyne HDL-32E*, donde este sensor obtenía mapas de elevación y, tras su procesamiento, se etiquetaban los obstáculos para la elaboración del mapa de costes, incluyendo rampas de subida y bajada, y escalones de subida y bajada también.

La navegación se realizaba evitando esas casillas categorizadas como no transitables ya fuese porque superaba el valor de rampa crítica o el valor de escalón crítico, valores parametrizables en función del robot. En el caso de cambiar de robot a otro con diferentes características, el mapa de costes variaría en consecuencia. Sin embargo, el mapa de elevación permanecería constante si se tratase de la misma escena.

Esta investigación tiene muchas aplicaciones importantes que, poco a poco, nos van a ayudar a mejorar en muchos aspectos, ya que se pueden utilizar para investigación de cuevas o zonas no accesibles para el ser humano, robots para salvamento, entre otras. Cada vez se irán mejorando las técnicas de desarrollo en este campo y se irá mejorando la calidad de vida.

En este proyecto he descubierto mi gran interés sobre el tema de robots y su navegación autónoma, además de con la asignatura de Robots Autónomos de la que tanto disfruté. Aunque los problemas encontrados a lo largo del desarrollo no han sido pocos, con ayuda de tutoriales de ROS y artículos sobre el tema he conseguido la realización de una navegación autónoma sobre terreno desconocido, que a simple vista no es fácil. Con ayuda de conocimientos adquiridos a lo largo de la carrera, he conseguido programar, ajustar parámetros, simular satisfactoriamente y saber documentar todo lo que he realizado en los últimos meses, además de aprender cómo funciona la herramienta ROS.

Y como conclusión final, aunque todo parezca muy negro en un principio, siempre consigo lo que me propongo con constancia y perseverancia, dando todo de mí para llegar al resultado que me había marcado.

8.3. POSIBLES MEJORAS Y FUTURA INVESTIGACIÓN

Durante la realización de este trabajo he encontrado algunos hechos que se podrían mejorar.

Uno de ellos sería la eliminación del ángulo muerto del sensor. Para ello, sería buena idea instalar nuevos sensores lidar en todos los costados del robot y mirando al suelo, de tal forma que el robot conociese en todo momento lo que ocurre a su alrededor.

De esta forma, se conseguiría conocer con exactitud los valores de escalones que en estos momentos se marcan como desconocidos debido al corto campo de visión del sensor.

Otra idea para mejorar esto sería la utilización de un espejo en el sensor colocado de la siguiente forma:

Figura 38. Sensor Velodyne con espejo para ampliar el campo de visión de una forma barata y fácil.

El procesamiento de los datos de la nube sería distinto, pero la creación del mapa de costes se haría de la misma manera.

Una futura línea de investigación incluiría estas mejoras comentadas. También estaría bien añadir alguna forma de localización más fiable, técnicas de SLAM (*simultaneous localization and mapping)* ya conocidas. Existen en ROS diversos algoritmos utilizables, que servirán para integrar el mapeo, la navegación y la localización, pudiendo simular misiones más realistas al tener en cuenta los problemas de localización en robótica.

8.4. AGRADECIMIENTOS

Agradecer primeramente a Luís Montano ser mi tutor durante todo el proyecto y por darme ideas para llevar a cabo este trabajo. Por otro lado, a Luís Riazuelo, por ayudarme con cualquier duda que me surgía en relación con ROS y con todas las simulaciones. Agradecer también a todos y cada uno de los profesores que me han impartido clases durante estos 4 años, he aprendido de todos y cada uno de ellos. Agradecer también a mis compañeros, con los que he compartido todo este largo camino. Finalmente, agradecer también a mi familia, por aguantar todas mis quejas diarias sobre la dificultad de la carrera y por todos los consejos y ánimos que me han dado para llegar hasta este punto, convertirme en lo que es mi sueño, ser ingeniera.

BIBLIOGRAFÍA

[1] Wiki.ros.org. (2019). *es - ROS Wiki*. [online] Available at: http://wiki.ros.org/es [Accessed 13 Nov. 2019].

[2] Anon, (2019). [online] Available at: https://www.generationrobots.com/en/402397-robot-mobile-pioneer-3-at.htm [Accessed 13 Nov. 2019].

[3] Gazebosim.org. (2019). *Gazebo*. [online] Available at: http://gazebosim.org/ [Accessed 13 Nov. 2019].

[4] GitHub. (2019). *ANYbotics/grid_map*. [online] Available at: https://github.com/ANYbotics/grid_map [Accessed 13 Nov. 2019].

[5] GitHub. (2019). *ANYbotics/elevation_mapping*. [online] Available at: https://github.com/ANYbotics/elevation_mapping [Accessed 13 Nov. 2019].

[6] GitHub. (2019). *leggedrobotics/traversability_estimation*. [online] Available at: https://github.com/leggedrobotics/traversability_estimation [Accessed 13 Nov. 2019].

[7] Wiki.ros.org. (2019). *move_base - ROS Wiki*. [online] Available at: http://wiki.ros.org/move_base [Accessed 14 Nov. 2019].

[8] Velodynelidar.com. (2019). *HDL-32E*. [online] Available at: https://velodynelidar.com/hdl-32e.html [Accessed 13 Nov. 2019].

[9] Oíza Cólera, J. (2018). *Navegación autónoma de un robot utilizando mapas de elevación*. Trabajo de Fin de Grado. Universidad de Zaragoza.

[10] Answers.ros.org. (2019). *Questions - ROS Answers: Open Source Q&A Forum*. [online] Available at: https://answers.ros.org/questions/ [Accessed 13 Nov. 2019].

[11] Fankhauser, Péter & Hutter, Marco. (2016). A Universal Grid Map Library: Implementation and Use Case for Rough Terrain Navigation. 10.1007/978-3-319-26054-9_5.

[12] team, B. (2019). *DARPA Subterranean Challenge: Unearthing the Subterranean Environment*. [online] DARPA Subterranean Challenge. Available at: https://www.subtchallenge.com/ [Accessed 14 Nov. 2019].

[13] Medium. (2019). *An Introduction to LIDAR: The Key Self-Driving Car Sensor*. [online] Available at: https://news.voyage.auto/an-introduction-to-lidar-the-key-self-driving-car-sensor-a7e405590cff [Accessed 14 Nov. 2019].

[14] Explain that Stuff. (2019). *LIDAR: a simple introduction*. [online] Available at: https://www.explainthatstuff.com/lidar.html [Accessed 14 Nov. 2019].

ANEXO I. ROS

ROS (*"Robotic Operating System"*) es un *framework* para el desarrollo de software para robots que provee la funcionalidad de un sistema operativo aún sin serlo. Se desarrolló originalmente en 2007 bajo el nombre de *switchyard* por el Laboratorio de Inteligencia Artifical de Stanford. Está orientado para el sistema operativo Linux (Ubuntu). Se trata de un *software* libre.

ROS provee los servicios estándar de un sistema operativo tales como abstracción del hardware, control de dispositivos de bajo nivel, implementación de funcionalidad de uso común, paso de mensajes entre procesos y mantenimiento de paquetes.

ROS consta de dos partes básicas: la parte del sistema operativo ROS (explicado anteriormente) y los *ros-pkg*, que son los paquetes aportados por los usuarios que implementan las funcionalidades tales como localización, mapeo simultáneo (SLAM), planificación, percepción, simulación, etc.

Está basado en una arquitectura de grafos. El funcionamiento se explica en la **Figura 39**.

ROS Master se encarga de controlar las comunicaciones, así como de poner en contacto a los diferentes nodos. Los nodos son las unidades que realizan los cálculos y se pueden combinar para crear funciones más complejas. Se comunican entre ellos mediante los *topics* a los que cada nodo se suscribe o publica. Los *topics* son los canales de comunicación entre nodos, y cada uno envía un tipo de mensaje definido en ROS. Todos los nodos, servicios y *topics* en ejecución son registrados por el *ROS Master.*

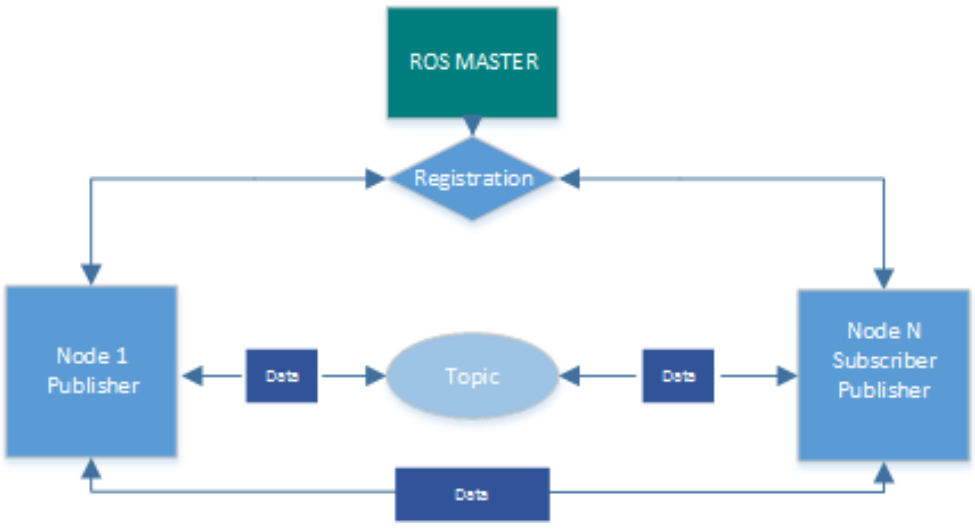

Figura 39. Funcionamiento ROS

ANEXO II. PIONEER 3-AT

El robot Pioneer 3-AT es un robot móvil ideal para todo tipo de terrenos con irregularidad media. Se trata de un robot de cuatro ruedas. Los robot Pioneer ampliamente utilizados en investigación y educación debido a su versatilidad, fiabilidad y durabilidad. Son personalizables en términos tanto de *software* como de *hardware.*

Este robot de cuatro ruedas tiene neumáticos reforzados y un cuerpo de aluminio lacado. Puede usarse para superficies de baldosa, arena, asfalto y barro, perfecto para la exploración.

Figura 40. Robots Pioneer 3-AT en un túnel

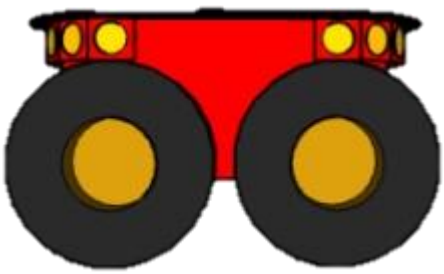

Figura 41. Robot Pioneer utilizado para la simulación

ANEXO III. VELODYNE HDL-32E

El sensor *Velodyne HDL-32E* se trata de un lidar que crea imágenes 3D usando 32 pares de láser para escanear el escenario. Su diseño permite que cada láser dispare miles de veces por segundo, proporcionando una completa nube de puntos 3D.

Sus principales características son:

a) Campo de visión horizontal de 360°

a) Campo de visión vertical de 41.3°

b) Distancias hasta 70 metros

c) Frecuencia de imágenes por segundo de 10 Hz

e) Precisión < 2cm

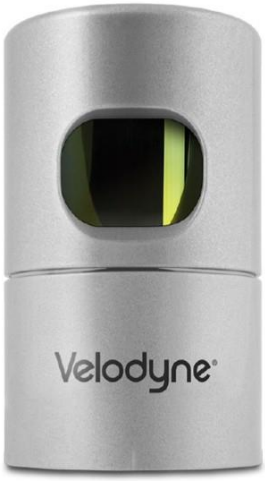

Figura 42. Sensor Velodyne HDL-32E

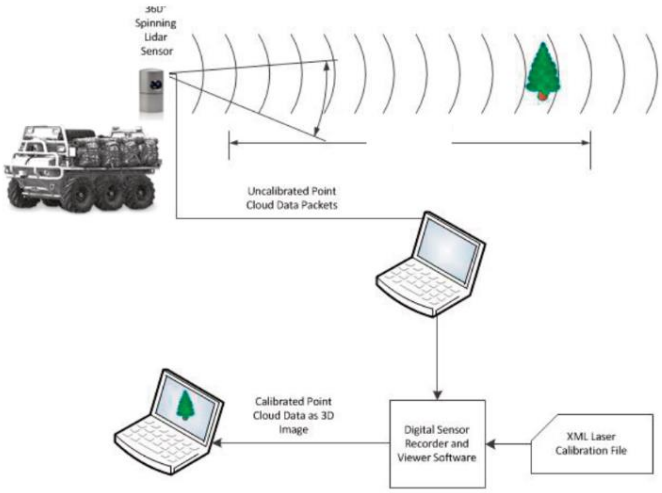

Figura 43. Funcionamiento lidar

ANEXO IV. GRID MAP (PACKAGE)

Es un paquete en lenguaje C++ creado para manejar mapas en dos dimensiones con varias capas de datos. Se ha diseñado para la creación de mapas de robots en movimiento y almacena datos como elevación, color, coeficientes de fricción, normales a la superficie e inclinaciones, entre otros.

Este paquete contiene demostraciones y filtros con diferentes objetivos, que trabajan sobre los mapas.

Un paquete popular que se basa en *Grid Map* es el *costmap_2d.* Se usa para la navegación y su función es procesar medidas para construir un mapa de ocupación o mapa de costes, donde da valores a las celdas de ocupado, libre o desconocido.

En este paquete encontramos los filtros de vectores normales y de inclinaciones utilizados para crear el mapa de ocupación. Los parámetros utilizados han sido los siguientes:

```
grid_map_filters:

  # Compute surface normals.
  - name: surface_normals
    type: gridMapFilters/NormalVectorsFilter
    params:
      input_layer: elevation
      output_layers_prefix: normal_vectors_
      radius: 0.3
      normal_vector_positive_axis: z


  # Compute slope from surface normal.
  - name: slope
    type: gridMapFilters/MathExpressionFilter
    params:
      output_layer: slope
      expression: acos(normal_vectors_z)
```

Los paquetes *Elevation Mapping* y *Traversability Estimation* basan su funcionamiento en éste.

ANEXO V. ELEVATION MAPPING (PACKAGE)

Este paquete, basado en *Grid Map* y sus mapas con celdas y capas, obtiene un mapa de elevación del terreno a partir de puntos obtenidos por un sensor (en este caso es lídar, pero podría tratarse de una cámara) conociendo la pose del robot. Al concepto de localización y mapeo de forma simultánea se lo conoce como SLAM.

Con este paquete, tenemos la posibilidad de guardar en mapa de elevación para futuros experimentos, en formato *rosbag.*

El mapa de elevación obtenido tiene formato *Grid Map* por lo que obtenemos un mapa dividido en celdas donde en cada celda está definida su altura.

El paquete *Elevation Mapping* utiliza varios ficheros de configuración, donde se da valor a los diferentes parámetros del robot, mapa y sensor.

- Fichero de configuración del robot. En éste se ajustan parámetros relativos al robot y se definen los topics donde se publican los puntos obtenidos por el láser, entre otros. Son los siguientes:

```
point_cloud_topic: "/velodyne_points"
sensor_frame_id: "/velodyne"
map_frame_id: "wally"
robot_base_frame_id: "base_link"
robot_pose_with_covariance_topic: "/amcl_pose"
robot_pose_cache_size: 200
track_point_frame_id: "base_link"
track_point_x: 0.0
track_point_y: 0.0
track_point_z: 0.0
```

- Fichero de configuración del mapa. Aquí se definen parámetros relativos a longitud de mapa, posición, resolución, etc. Los parámetros utilizados son los siguientes:

```
length_in_x: 12.0
length_in_y: 12.0
position_x: 0.0
position_y: 0.0
resolution: 0.2
min_variance: 0.000009
max_variance: 0.01
#min_update_rate: 10.0
update_rate: 10.0
mahalanobis_distance_threshold: 2.5
multi_height_noise: 0.0000009
sensor_cutoff_max_depth: 1
sensor_cutoff_min_depth: 0.1
surface_normal_estimation_radius: 0.015
surface_normal_positive_axis: z
```

- Fichero de configuración del sensor *Velodyne HDL-32E*. Aquí se definen los parámetros del láser. Los valores utilizados son los siguientes:

```
# Velodyne_HDL-32E

sensor_processor/type: laser
sensor_processor/min_radius: 0.018
sensor_processor/beam_angle: 0.0006
sensor_processor/beam_constant: 0.0015
```

ANEXO VI. FICHEROS DE CONFIGURACIÓN PARA LA NAVEGACIÓN (MOVE_BASE)

En este paquete encontramos varios ficheros para la configuración de la navegación sobre el mapa de costes. Los más relevantes son los siguientes:

- Fichero de configuración de las velocidades de navegación

```
TrajectoryPlannerROS:
  max_vel_x: 0.22 #m/s
  min_vel_x: 0.02 #m/s
  max_rotational_vel: 0.06 #rad/s
  min_in_place_rotational_vel: 0.03 #rad/s
  escape_vel: -0.08 #m/s
  meter_scoring: true

  sim_time: 2.0
  path_distance_bias: 0.6
  goal_distance_bias: 0.6

  acc_lim_theta:  0.4 #rad/s^2
  acc_lim_x:  0.8 #m/s^2
  acc_lim_y: 0.8 #m/s^2

  holonomic_robot: false
```

- Fichero de parametrización de los *costmaps*

```
#---standard pioneer footprint (in meters)---
footprint: [ [0.3302, -0.0508], [0.254, -0.0508], [0.254, -0.254], [-0.254, -
0.254], [-0.254, 0.254], [0.254, 0.254], [0.254, 0.0508], [0.3302, 0.0508] ]

obstacle_layer:
    obstacle_range: 10
    raytrace_range: 5
    transform_tolerance: 0.2
    map_type: costmap
    max_obstacle_height:    0.57
    min_obstacle_height: 0.02
    observation_sources: laser_scan_sensor
    laser_scan_sensor:
        sensor_frame: laser_front
        data_type: LaserScan
        topic: /laser_front/scan
        marking: true
        clearing: true


#cost_scaling_factor and inflation_radius were now moved to the
inflation_layer ns
inflation_layer:
  enabled:              true
  cost_scaling_factor:  15.0  # exponential rate at which the obstacle cost
drops off (default: 10)
  inflation_radius:     1.0  # max. distance from an obstacle at which costs
are incurred for planning paths.

static_layer:
  enabled:              true
  map_topic:            /mapobs
  first_map_only:       false
```

- Configuración del *global costmap*

```
global_costmap:
  global_frame: wally
  robot_base_frame: base_link
  update_frequency: 2.0
  publish_frequency: 2.0
  static_map: true
  rolling_window: true
  width: 40.0
  height: 20.0
  resolution: 0.1
  transform_tolerance: 1

  plugins:
     - {name: static_layer,       type: "costmap_2d::StaticLayer"}
    # - {name: obstacle_layer,          type: "costmap_2d::ObstacleLayer"}
    # - {name: obstacle_layer,          type: "costmap_2d::VoxelLayer"}
    # - {name: inflation_layer,         type: "costmap_2d::InflationLayer"}
```

- Configuración del *local costmap*

```
local_costmap:
   global_frame: wally
   robot_base_frame: base_link
   update_frequency: 5.0
   publish_frequency: 2.0
   static_map: true
   rolling_window: true
   width: 20.0
   height: 20.0
   resolution: 0.2
   transform_tolerance: 3.5

   plugins:
     - {name: static_layer,       type: "costmap_2d::StaticLayer"}
   # - {name: obstacle_layer,      type: "costmap_2d::ObstacleLayer"}
#obstaculos 2dim
   # - {name: obstacle_layer,      type: "costmap_2d::VoxelLayer"} #obstaculos
3dim
   # - {name: inflation_layer,     type: "costmap_2d::InflationLayer"} #an
optimization that adds new values around lethal obstacles
```

ANEXO VII. CÓDIGO RELEVANTE

Nodo de creación del mapa de costes mapa_costes.cpp

```
#include "nav_msgs/OccupancyGrid.h"
#include <vector>
#include <string>
#include <cmath>
#include <limits>

// Grid Map
#include "grid_map_msgs/GridMap.h"
#include <grid_map_ros/grid_map_ros.hpp>
#include <grid_map_msgs/GetGridMap.h>
#include <grid_map_core/GridMap.hpp>
#include <grid_map_core/Polygon.hpp>
#include "grid_map_ros/GridMapMsgHelpers.hpp"
#include <grid_map_cv/grid_map_cv.hpp>
#include "grid_map_core/iterators/GridMapIterator.hpp"

// ROS
#include <filters/filter_chain.h>
#include <ros/ros.h>
#include <sensor_msgs/Image.h>
#include <std_srvs/Empty.h>
#include <tf/transform_listener.h>

// ROS
#include <sensor_msgs/point_cloud2_iterator.h>
#include <geometry_msgs/PolygonStamped.h>
#include <geometry_msgs/Pose.h>
#include <ros/package.h>
#include <geometry_msgs/Pose.h>
#include <xmlrpcpp/XmlRpcValue.h>


// STL

#include <algorithm>

// kindr
#include <kindr/Core>

// Eigen
#include <Eigen/Core>
#include <Eigen/Geometry>

#include <message_filters/subscriber.h>
#include <message_filters/synchronizer.h>
#include <message_filters/sync_policies/approximate_time.h>

using namespace sensor_msgs;
using namespace message_filters;
using namespace Eigen;
using namespace grid_map;
using namespace std;


class Node
{

 public:
  Node()
  {
    sub_slope.subscribe(nh_, "/grid_map_filter_demo/filtered_map", 30);
```

```
    sub_step.subscribe(nh_, "/elevation_mapping/elevation_map", 40);
    sub_imu = nh_.subscribe("/imu", 1, &Node::callback1, this); // queue de 1
para que siempre se coja el valor que corresponde a ese momento ya que llegan
varios valores cada segundo
    pub_ = nh_.advertise<nav_msgs::OccupancyGrid>("/mapobs", 30);

    double pitch;

// Sincronizar los mapas para la llamada del callback
    sync_.reset(new Sync(MySyncPolicy(30), sub_slope, sub_step));
    sync_->registerCallback(boost::bind(&Node::callback2, this, _1, _2));

  }
void callback1(const sensor_msgs::Imu::ConstPtr& input_imu)
  {
// datos IMU
    double roll,yaw;
    tf::Quaternion cuaternio;
    tf::quaternionMsgToTF(input_imu->orientation,cuaternio);
    tf::Matrix3x3(cuaternio).getRPY(roll, pitch, yaw);

}

  void callback2(const grid_map_msgs::GridMap::ConstPtr& input, const
grid_map_msgs::GridMap::ConstPtr& input2)
  {


    ROS_INFO("entrando callback");

// Grid map de slope
    grid_map::GridMap gridMap;
    GridMapRosConverter::fromMessage( *input,gridMap);


// Grid map de elevación
    grid_map::GridMap gridStep;
    GridMapRosConverter::fromMessage( *input2,gridStep);


// Objetos tipo position
    Position position = gridMap.getPosition() - 0.5 *
gridMap.getLength().matrix();//posicion en el frame (centro mapa)-longitud,
por lo tanto el 0,0 deja de estar en el frame sino en la esquina de arriba a
la izquierda
    Position positionnow, positionUP, positionDOWN, positionRIGHT,
positionLEFT;


// Crear mapa de costes con los mismos parametros que el de slope
    nav_msgs::OccupancyGrid output;
    output.header.frame_id = gridMap.getFrameId();
    output.header.stamp.fromNSec(gridMap.getTimestamp());
    output.info.map_load_time = output.header.stamp;
    output.info.resolution = gridMap.getResolution();
    output.info.width = gridMap.getSize()(0);
    output.info.height = gridMap.getSize()(1);
    output.info.origin.position.x = position[0];
    output.info.origin.position.y = position[1];
    output.info.origin.position.z = 0.0;
    output.info.origin.orientation.x = 0.0;
    output.info.origin.orientation.y = 0.0;
    output.info.origin.orientation.z = 0.0;
    output.info.origin.orientation.w = 1.0;
    size_t nCells = gridMap.getSize()prod();
    output.data.resize(nCells);
    const float resolution =gridMap.getResolution();
```

```
// Valores que pueden tomar las casillas
    const int obs=100;//obstaculos
    const int desc=-1.0;//mapa desconocido
    const int blanco=0.0;//zonas libres

// Valor inicial la casilla final del occupancy grid
    float casilla = 0.0;

// Valor critico de rampa
    const float critical_slope = 0.1396; //8°

// Valor critico de escalon
    const float critical_step_down = 0.10; //cm
    const float critical_step_up = 0.03; //cm


for (GridMapIterator iterator(gridMap); !iterator.isPastEnd(); ++iterator){

          if(gridMap.getPosition(*iterator, positionnow)==true){


             float value = (gridMap.at("slope", *iterator)); //valor de rampa
en cada casilla
             float step = (gridStep.at("elevation", *iterator)); // valor de
elevación de cada casilla
// Actualizacion posiciones de las vecinas

float valueUP_step, valueDOWN_step, valueRIGHT_step, valueLEFT_step = -10.0;//
valor que no va a tomar nunca para inicializar


             positionUP[0] = positionnow[0];
             positionUP[1] = positionnow[1]+resolution;
             if(gridStep.isInside(positionUP)==true)
                    valueUP_step = (gridStep.atPosition("elevation",
positionUP));


             positionDOWN[0] = positionnow[0];
             positionDOWN[1] = positionnow[1]-resolution;
           if(gridStep.isInside(positionDOWN)==true)
                  valueDOWN_step = (gridStep.atPosition("elevation",
positionDOWN));


             positionRIGHT[0] = positionnow[0]+resolution;
             positionRIGHT[1] = positionnow[1];
             if(gridStep.isInside(positionRIGHT)==true)
                     valueRIGHT_step = (gridStep.atPosition("elevation",
positionRIGHT));


             positionLEFT[0] = positionnow[0]-resolution;
             positionLEFT[1] = positionnow[1];
                    if(gridStep.isInside(positionLEFT)==true)
                     valueLEFT_step = (gridStep.atPosition("elevation",
positionLEFT));


             if (isnan(value) && (((positionnow[0]>position[0]+5.5) &&
(positionnow[0]<position[0]+6.5)) && ((positionnow[1]>position[1]+5.5) &&
(positionnow[1]<position[1]+6.5)))){    //da valor a las zonas de alrededor al
principio, el radio ciego del robot es aprox 0.5m, mapa de longitud 12x12m, el
centro en 6m

                 casilla = blanco;
```

```
          }else if (isnan(value)|| valueUP_step == -10.0 || valueDOWN_step ==
-10.0 || valueRIGHT_step == -10.0 || valueLEFT_step == -10.0){// si no tiene
valor asignado
                   casilla = desc;

             }else if(((value + pitch) < critical_slope)&& ((value +
pitch)>=1.57)){// angulos menores que el critico y mayores de 90°, ya que
puede bajar cualquier inclinacion

                 casilla = blanco;

             }else if ((((step + critical_step_up >=  valueUP_step )&&( step -
critical_step_down <= valueUP_step))&&((step + critical_step_up >=
valueDOWN_step )&&( step - critical_step_down <= valueDOWN_step))&&((step +
critical_step_up >=  valueRIGHT_step )&&( step - critical_step_down <=
valueRIGHT_step))&&((step + critical_step_up >=  valueLEFT_step )&&( step -
critical_step_down <= valueLEFT_step)))&&(valueUP_step != -10.0 &&
valueDOWN_step != -10.0 && valueRIGHT_step != -10.0 && valueLEFT_step != -
10.0)){// escalon de subida o bajada dentro de los limites establecidos y
existe valor de las vecinas

                 casilla = blanco;

             }else if ((((step + critical_step_up <  valueUP_step )&&( step -
critical_step_down > valueUP_step))&&((step + critical_step_up <
valueDOWN_step )&&( step - critical_step_down > valueDOWN_step))&&((step +
critical_step_up <  valueRIGHT_step )&&( step - critical_step_down >
valueRIGHT_step))&&((step + critical_step_up <  valueLEFT_step )&&( step -
critical_step_down > valueLEFT_step)))&&(valueUP_step != -10.0 &&
valueDOWN_step != -10.0 && valueRIGHT_step != -10.0 && valueLEFT_step != -
10.0)){// escalon de subida o bajada considerados como obstaculo

                 casilla = obs;

             }else if(((value + pitch) >= critical_slope)&&((value+pitch) <
1.57)){
                 casilla = obs;

             }

                  size_t index =
getLinearIndexFromIndex(iterator.getUnwrappedIndex(), gridMap.getSize(),
false);
           //El orden entre los dos mapas de las celdas no es lo mismo por eso
cambia del index 2dim a un indice 1dim
                  output.data[nCells - index - 1] = casilla;

          }

}// Publicar salida en el mapa de costes
   pub_.publish(output);

   ROS_INFO("saliendo callback");

}

 private:
  ros::NodeHandle nh_;

// suscripciones

  message_filters::Subscriber<grid_map_msgs::GridMap> sub_slope;
  message_filters::Subscriber<grid_map_msgs::GridMap> sub_step;
  ros::Subscriber sub_imu;

// publicacion

  ros::Publisher pub_;
```

```
// sincronización

  typedef
message_filters::sync_policies::ApproximateTime<grid_map_msgs::GridMap,grid_ma
p_msgs::GridMap> MySyncPolicy;
  typedef message_filters::Synchronizer<MySyncPolicy> Sync;
  boost::shared_ptr<Sync> sync_;// Sincronización de los topics a los que se
suscribe el nodo, ya que tienen valores de actualización parecidos
};



int main(int argc, char **argv)
{
  ros::init(argc, argv, "subscribe_and_publish");
  Node synchronizer;
  ros::spin();
  return 0;
}
```

ANEXO VIII. TOPICS RELEVANTES DE ROS

Se explicará brevemente los *topics* más interesantes para la comprensión del proyecto. (en la Figura del mapa de nodos y *topics* del capítulo 6 se puede observar qué nodos los publican y cuáles se suscriben a cada uno)

- ***/amcl_pose.*** *Topic* que contiene la localización que le ha suministrado el simulador.
- ***/elevation_mapping/elevation_map.*** *Topic* publicado por el nodo */elevation_mapping* y representa la elevación del terreno captada por la nube de puntos del lidar.
- ***/grid_map_filter_demo/filtered_map.*** *Topic* publicado por el nodo */grid_map_filter_demo* que contiene el mapa con la capa de inclinaciones.
- ***/imu.*** *Topic* donde se publican los valores relativos al sensor IMU.
- ***/mapobs.*** *Topic* publicado por el nodo */mapa_costes* que contiene el mapa de costes sobre el que se navega.
- ***/move_base/global_costmap/costmap.*** *Topic* publicado por el nodo */move_base* que representa el mapa de costes global.
- ***/move_base/local_costmap/costmap.*** *Topic* publicado por el nodo */move_base* que representa el mapa de costes local.
- ***/sim_p3at/odom.*** *Topic* publicado por el simulador, por el nodo */gazebo* que representa la odometría del robot.
- ***/sim_p3at/cmd_vel.*** *Topic* publicado por el simulador, por el nodo */gazebo* que representa la velocidad del robot en simulación.
- ***/velodyne_points.*** *Topic* publicado por el simulador, por el nodo */gazebo* que contiene la nube de puntos que se utilizará para la obtención del mapa de elevación.
-

También cabe comentar las frecuencias de actualización de los *topics* utilizados en mapa_costes.cpp para justificar los valores de *queue* de los *topics* escogidos.

- **/elevation_mapping/elevation_map.** Frecuencia de 0.16, mapa cada 6 segundos. *Queue* escogida de 40.
- ***/grid_map_filter_demo/filtered map.*** Frecuencia de 0.19, mapa cada 5 segundos. *Queue* escogida de 30, menor que en el caso anterior ya que se actualiza más rápido.
- ***/imu.*** Frecuencia de 9.7, valores nuevos cada 100 ms. *Queue* escogida de 1, para que cuando se actualice el mapa de costes se coja el valor exacto de inclinación del robot.
- ***/mapobs.*** Frecuencia de 0.17, mapa cada 6 segundos. *Queue* escogida de 40.

ANEXO IX. RAMPAS EN AUTODESK FUSION 360 / SKETCHUP

Las rampas que se han utilizado para la simulación en Gazebo han sido creadas mediante el programa Autodesk Fusion 360 y la her.

Autodesk Fusion 360 se trata de una herramienta de CAD 3D que combina el diseño industrial y mecánico, usando comandos de modelación directa para llegar a la geometría deseada.

Se han creado rampas de diferentes inclinaciones y escalones de bajada para llevar a cabo los diferentes experimentos.

Una vez creado el modelo 3D, se creaba el modelo .OBJ, que se trata de un formato de archivo usado para un objeto tridimensional que contiene coordenadas 3D (líneas poligonales y puntos), mapas de textura y otra información de objetos.

En SketchUp, se pueden crear modelos 3D online y posteriormente pasarlos a .STL para luego utilizarlos en Gazebo.

En el momento de pasarlos a Gazebo, se les da el valor de centro de gravedad.

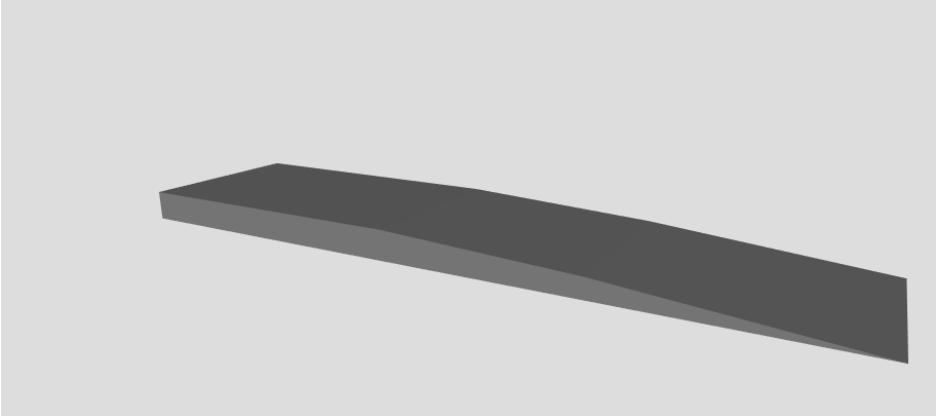

Figura 44. Rampa creada en Autodesk Fusion 360

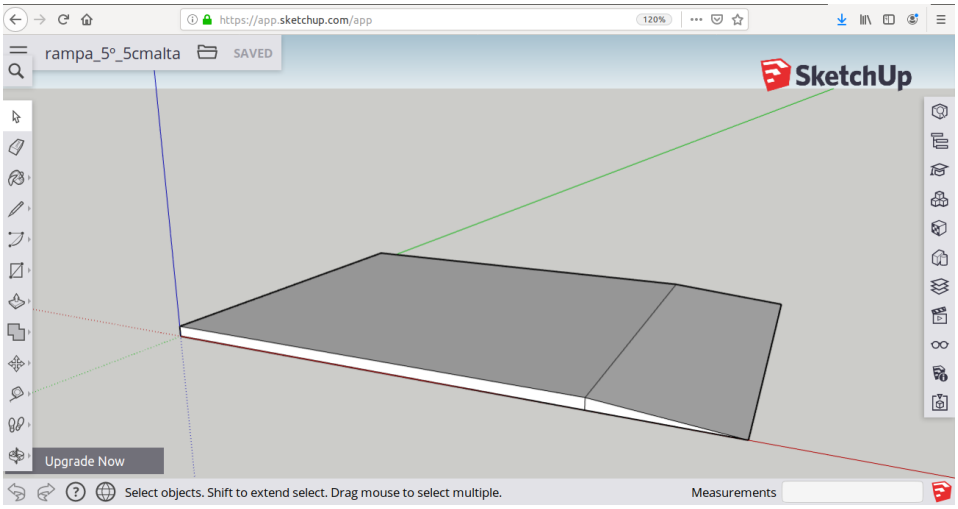

Figura 45. Rampa creada en SketchUp online