Universidad San Jorge

Escuela de Arquitectura y Tecnología

Grado en Ingeniería Informática

Proyecto Final

Rigoberto es un poco limitadito. ¡Hagámoslo listo!

Autor del proyecto: Roberto Tejedor del Río

Director del proyecto: Carlos Cetina

Zaragoza, 09 de septiembre de 2020

Declaración del alumno

Este trabajo constituye parte de mi candidatura para la obtención del título de Graduado en Ingeniería Informática por la Universidad San Jorge y no ha sido entregado previamente (o simultáneamente) para la obtención de cualquier otro título.

Este documento es el resultado de mi propio trabajo, excepto donde de otra manera esté indicado y referido.

Doy mi consentimiento para que se archive este trabajo en la biblioteca universitaria de Universidad San Jorge, donde se puede facilitar su consulta.

Roberto Tejedor del Río 09/09/2020

Dedicatoria y agradecimiento

Quiero dar las gracias a todas aquellas personas que me han ayudado y apoyado no solo durante el proceso de elaboración de este proyecto sino a lo largo de toda la carrera.

En primer lugar, me gustaría dar las gracias a mi tutor del proyecto de la empresa, Carlos Bello, por su cooperación, su apoyo y su paciencia, así como darle las gracias por todo lo que he aprendido a lo largo de estos meses, a pesar de todas las dificultades que han ido surgiendo a lo largo de estos meses tan complicados.

También, me gustaría dar las gracias a mi tutor de la universidad Carlos Cetina, por su paciencia, su apoyo y sus consejos en los momentos difíciles.

Finalmente, me gustaría agradecer a toda mi familia y amigos, en general, por todo el apoyo que me han dado.

¡Muchas gracias a todos!

Tabla de contenidos

Resumen

Este es un proyecto realizado con la empresa Inycom (Instrumentación y Componentes S.A.), con el objetivo de añadir nuevas funcionalidades al robot Elegoo Smart Robot Car Kit V3.0 (Rigoberto). A lo largo del desarrollo se han empleado diferentes dispositivos y software que han permitido implementar la funcionalidad de mapear el entorno en tiempo real.

Para ello, hemos utilizado un dispositivo LIDAR cuya función es la de detectar los obstáculos alrededor del robot para poder mapear el entorno que lo rodea, conectado a una Raspberry Pi, cuya función es la de recibir la información del LIDAR y, a su vez, enviarla a un ordenador para que sea procesada y permita su visualización en un entorno gráfico.

De este modo, podemos observar el tiempo real todos los obstáculos que rodean al robot en un radio de 12 metros, mientras somos capaces de desplazar el robot libremente a través de un smartphone mediante conexión Bluetooth con Rigoberto.

This is a project carried out with the company Inycom (Instrumentación y Componentes S.A.), with the aim of adding new functionalities to the Elegoo Smart Robot Car Kit V3.0 (Rigoberto). Throughout the development, different devices and software have been used that have made it possible to implement the functionality of mapping the environment in real time.

For this, we have used a LIDAR device whose function is to detect obstacles around the robot in order to map the environment around it, connected to a Raspberry Pi, whose function is to receive the information from the LIDAR and, at the same time, send it to a computer so that it can be processed and allowed to be viewed in a graphic environment.

This way, we can observe in real time all the obstacles that surround the robot within a radius of 12 meters, while we are able to move the robot freely through a smartphone via Bluetooth connection with Rigoberto.

Introducción

Antes de hablar del proyecto, comenzaré haciendo una introducción sobre la empresa en la que he estado trabajando para realizar su desarrollo.

Inycom (*Instrumentación y Componentes S.A.*) es una empresa que ofrece soluciones y servicios tecnológicos e innovadores para impulsar el negocio de sus clientes hacia el éxito.
Como empresa tecnológica lleva desde 1982 desarrollando soluciones y servicios para sus clientes en diferentes ámbitos y sectores. Cuenta con un equipo de más de 750 profesionales y una red internacional de *partners* que les permite desarrollar proyectos en diferentes continentes.

- ¿Qué hacen?

  Poner al alcance de sus clientes la transformación tecnológica y global en su organización o negocio.
- ¿Para qué están aquí?

  Para ayudar a sus clientes a impulsar su negocio y alcanzar nuevas cotas de competitividad.
- ¿Cuál es su compromiso?

  Trabajar al lado de sus clientes para conseguir resultados de éxito mediante la innovación y la mejora continua. [1]

El principal objetivo de este proyecto es el desarrollo de nuevas funcionalidades para aumentar las posibilidades del robot Elegoo Smart Robot Car Kit V3.0. La funcionalidad de la que dispone al inicio del proyecto es la de desplazarse siguiendo una línea negra marcada en el suelo.
Por una parte, se empleará un dispositivo Lidar, cuya función en el proyecto será la de obtener información del entorno por el que se desplace el vehículo midiendo las distancias de los obstáculos alrededor del mismo.
Por otro lado, una Raspberry pi, cuya función será la de obtener la información del Lidar (las distancias de los obstáculos detectados en ese momento) y enviarla al ordenador.
Asimismo, el ordenador se encargará de procesar la información recibida por la Raspberry Pi y de representarla en un entorno gráfico, permitiendo al usuario ver un mapa de todos los obstáculos que ha podido detectar el Lidar, sobre el que se visualizarán las diferentes funcionalidades implementadas como el trayecto recorrido por el coche, el costmap o la ruta que deberá seguir el vehículo para explorar nuevo entorno.
Finalmente, el coche sobre el que estará colocado la Raspberry Pi junto con el Lidar, será el vehículo que controlaremos remotamente desde el móvil y nos permitirá descubrir el entorno sin necesidad de movernos.

A continuación, explicaré brevemente las diferentes tecnologías y dispositivos que vamos a emplear durante el proyecto, que nos permitirán implementar esta nueva funcionalidad.

Lidar. LIDAR son las siglas de *Laser Imaging Detection and Ranging*, o en español se podría traducir como sistema de medición y detección de objetos mediante láser. Podríamos diferenciarlo de un RADAR en que mientras que el RADAR emite ondas de radio que rebotan en los objetos para así calcular las distancias a las que se encuentran, el LIDAR emite rayos de luz láser infrarroja (no peligrosa) que rebotan en los objetos para así calcular las distancias a las que se encuentran. [2]

Raspberry Pi. Una Raspberry Pi es un ordenador de bajo coste y tamaño reducido (del tamaño de la palma de una mano), al que puedes conectar un cable HDMI para conectarlo al televisor o dispositivos USB (ratón, teclado,…) para interactuar con ella como si fuera un ordenador.
La Raspberry Pi está compuesta por un SoC (System on Chip), que es un ordenador completo en un solo chip o circuito integrado, CPU, memoria RAM, puertos USB y HDMI, conectividad de red, ranura SD para almacenamiento… Prácticamente lo mismo que un ordenador. [3][4]

Elegoo Smart Robot Car Kit V3.0. Consiste en un kit educativo basado en Arduino UNO. Está diseñado para aprender programación, robótica y electrónica tanto para principiantes como para profesionales. Cuenta con un tutorial para ensamblar el robot, así como de librerías y códigos de ejemplo.
Entre otras cosas, cuenta con un sensor seguidor de línea (que se emplea en la funcionalidad ya implementada del robot), un módulo Bluetooth (que permite que nos conectemos desde el Smartphone para poder controlarlo) y un soporte para sensor ultrasónico y el sensor ultrasónico (que le permite detectar obstáculos a una corta distancia y detenerse antes de colisionar). [5]

Linux. Es el Sistema Operativo con el que vamos a trabajar en el ordenador y en la Raspberry Pi. Es un sistema operativo libre (Open Source) y gratuito. No es propiedad de una compañía sino de gran número de compañías o personas que contribuyen a su desarrollo y crean sus propias distribuciones.[6]

SLAM. SLAM son las siglas de *Simultaneos Localization and Mapping*, o en español se podría traducir como mapeo y localización simultáneos. Es una técnica de navegación que permite a un robot/vehículo, construir un mapa del entorno sobre el que se desplaza, gracias a la información que captan sus sensores en tiempo real. SLAM se utiliza por sensores LIDAR, pero también se puede combinar con otros sensores como GPS, sonar, cámaras… para obtener mejores resultados. [7]

ROS. ROS son las siglas de *Robot Operating System*, o en español se podría traducir como Sistema Operativo Robótico. Está compuesto por herramientas y librerías compatibles con gran variedad de plataformas robóticas y es empleado para escribir software para robots. A pesar de que no es un sistema operativo como tal, proporciona servicios como el control de dispositivos de bajo nivel, la implementación de funcionalidades de uso común, abstracción de hardware, comunicación entre procesos y la administración de paquetes. También cuenta con funcionalidades de alto nivel como las llamadas síncronas y asíncronas, bases de datos centralizadas o un sistema de configuración para el robot. [8]

Catkin. Catkin es una colección de macros CMake (una herramienta multiplataforma de generación o automatización de código) y código asociado usado para construir paquetes empleados en ROS. Se introdujo en el lanzamiento de ROS Fuerte, donde se usó para un pequeño conjunto de paquetes básicos. Más adelante para ROS Groovy y ROS Hydro, fue modificado y utilizado por muchos más paquetes. [9]

Antecedentes y estado del arte

En este apartado hablaré sobre qué hay y qué había en el mercado relacionado con el proyecto que se ha desarrollado, los compararé brevemente con nuestro proyecto y, finalmente, detallaré la tecnología que se ha empleado.

Estos son algunos de los ejemplos que hay en el mercado o que han sido anunciados, relacionados con el proyecto.

Bosch, Livox y Sony.
Bosch, Livox y Sony pretenden acelerar la llegada del coche autónomo con sus nuevos sensores LIDAR de bajo coste.
La industria del automóvil pretende acelerar la llegada del coche autónomo, sin embargo, en los LIDAR tienen el sensor más caro (alcanzando hasta los 100,000 dólares) o bien debido a la dificultad para integrarlos, o bien por el coste del sensor, por lo que no todos los fabricantes optan por ellos.
Estos fabricantes explican que sus sensores podrían revolucionar la industria del automóvil ya que podrían llegar a reducir el coste de producción de estos dispositivos a tan solo 1,000 dólares.

Figura 1 - LIDAR Livox

En los modelos de LIDAR presentados por Livox, los precios oscilan entre los 599 y los 1,499 dólares.

Por otro lado, Luminar, ha presentado su nuevo sensor Hydra, diseñado para detectar objetos a 250 metros y que únicamente requiere 30W de potencia.

Bosch, también presentó un nuevo LIDAR capaz de detectar objetos no metálicos a un largo alcance y de ser producido a gran escala.

La empresa china Robosense, también ha presentado un LIDAR con un coste inferior a los 2,000 dólares, con una precisión de ángulo de 0.1º y un rango de 200m.

Finalmente, entre las propuestas de la empresa Sony, también destacan los sensores LIDAR de estado sólido que miden distancias para obtener una comprensión en 3D de los espacios, aunque no están a la altura con el resto de fabricantes en cuanto al precio. [10]

Volvo.

Volvo ya ha anunciado que sus coches autónomos emplearán tecnología LIDAR a partir de 2022. Este dispositivo determinará distancias con los objetos circundantes creando un mapa 3D del entorno basándose en la luz reflejada para que el vehículo pueda tomar decisiones.

La empresa considera que el LIDAR es la clave del desarrollo de los vehículos autónomos y seguros y ya ha empleado esta tecnología en los Volvo XC90 autónomos.

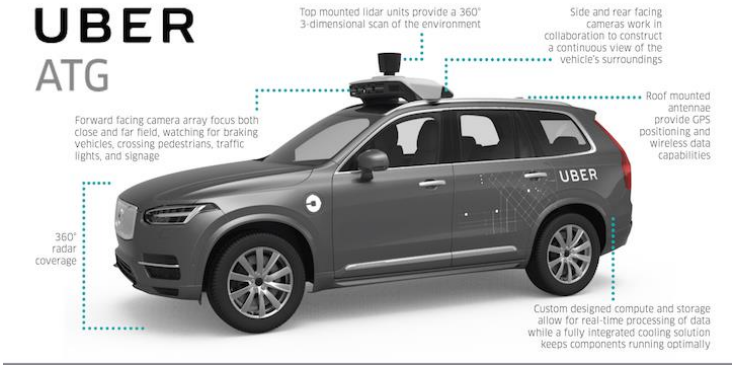

Figura 2 - Volvo XC90 [11]

A partir de 2022, Volvo ofrecerá vehículos con tecnología LIDAR pero no para fines de conducción completamente autónoma, sino para áreas específicas que hayan sido demostradas como seguras. [12]

iPad Pro.

El nuevo iPad Pro que fue lanzado el 24 de marzo de 2020, incluye un sensor LIDAR que Apple ha montado en sus nuevos tablets, acompañado de la doble cámara trasera para mejorar la experiencia de la fotografía así como de la realidad aumentada.

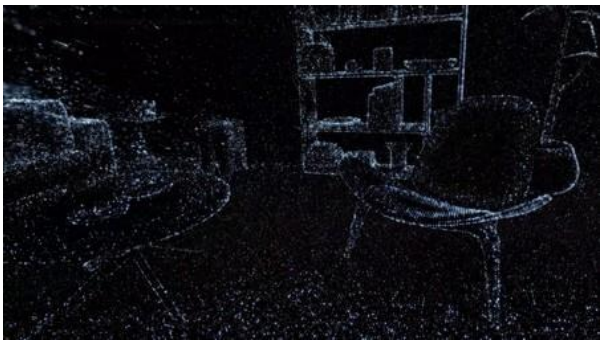

Figura 3 - Lidar iPad Pro

La diferencia entre el LIDAR de los iPad y los LIDAR que usan algunos fabricantes de automóviles, es que mientras que el de los automóviles tiene un rango de 360º (ya que el láser gira constantemente), el de los iPad es fijo y siempre enfoca hacia el mismo sitio (hacia donde apuntamos con la cámara). Esto permite detectar de manera eficiente las profundidades a la hora de realizar una fotografía, por lo que se pueden aplicar de forma eficiente efectos de desenfoque o *bokeh*.
Si bien las tecnologías móviles ya emplean un tipo de sensor conocido como ToF (*Time of Flight*, o Tiempo de Vuelo en español), un LIDAR debería ser mucho más preciso que un ToF normal. [13]

Scanner Sombre.

El videojuego Scanner Sombre, es un videojuego en primera persona en el que exploras el mundo a través de un LIDAR, como los usados en los vehículos autónomos, el cual nos permite movernos por el mundo virtual, mostrando al usuario un entorno creado por miles de puntos de luz que recrean el paisaje paso a paso. Emplea un esquema de colores dentro de la paleta habitual para indicar la distancia y profundidad conforme vamos avanzando en el juego, lo cual nos permite ver un paisaje diferente del que vemos en los videojuegos habituales en los que el entorno se observa de manera muy nítida.[14]

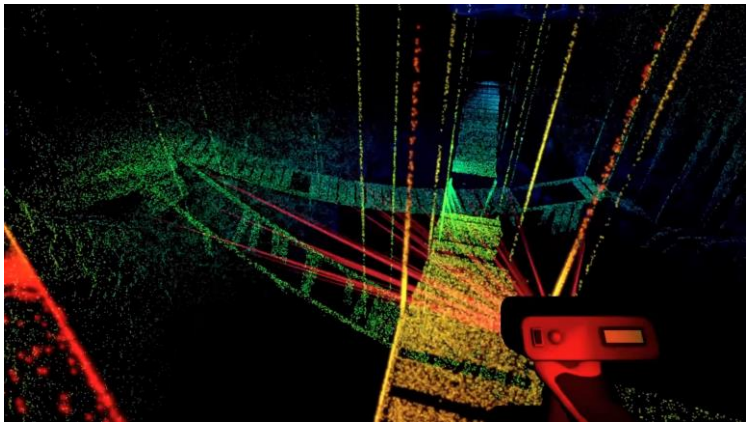

Figura 4 - Scanner Sombre

Dado que son los ejemplos más similares al proyecto que se ha desarrollado, me centraré en explicar las similitudes y diferencias entre los casos de Bosch, Livox y Sony así como el de Volvo con este proyecto.

La similitud más destacable es que el objetivo final es el mismo, pues tanto en los dos ejemplos como en este proyecto, se pretende implementar la conducción autónoma en un vehículo mediante el mapeado de entorno de un dispositivo Lidar.

Sin embargo, la diferencia entre los Lidar empleados por los vehículos de los ejemplos anteriores y el nuestro es notable, ya que la precisión requerida para un vehículo capaz de desplazarse a velocidades de 120km/h es infinitamente mayor que la requerida para un vehículo que se emplea para desplazarse por habitaciones y entornos cerrados. Lo cual hace que los precios entre nuestro dispositivo y los de los ejemplos anteriores sean tan grandes, pasando de los 83€ a 600-2,000€.

Otra diferencia entre los dispositivos Lidar empleados en los ejemplos y el que se ha empleado en este proyecto, es el ángulo de visión. Mientras que el nuestro apenas es capaz de mapear el entorno a una altura determinada (a la altura a la que se encuentre el dispositivo), los Lidar de los vehículos son capaces de mapear el entorno con un cierto ángulo (conocido como *Angular resolution*), que permite detectar no solo los obstáculos que haya justo a la altura a la que esté situado el dispositivo, sino también aquellos que estén por encima y por debajo (hasta que el ángulo lo permita).

Por esto, para un proyecto cuya finalidad es la de obtener de manera remota información sobre el entorno por el que se esté desplazando el vehículo, este proyecto es mucho más adecuado tanto a nivel de trabajo como a nivel de coste de Hardware para su desarrollo. Sin embargo, si nuestro objetivo fuera el de implementar esto en un vehículo de verdad, se requeriría Hardware muchísimo más rápido y preciso puesto que no solo estaríamos hablando de invertir en un proyecto, sino que estaríamos hablando de invertir en seguridad para las personas, o los seres vivos en general.

A continuación, me centraré en hablar sobre las especificaciones de la tecnología empleada en nuestro proyecto.

Raspberry Pi.

Su función es la de actuar como mini ordenador para recibir la información recogida por el LIDAR.

La Raspberry Pi que se ha empleado en este proyecto es la *Raspberry Pi 3 model B+*.

El dispositivo tiene una anchura de 85mm y una largura de 56mm.

Specifications

Processor:	Broadcom BCM2837B0, Cortex-A53 64-bit SoC @ 1.4GHz
Memory:	1GB LPDDR2 SDRAM
Connectivity:	■ 2.4GHz and 5GHz IEEE 802.11.b/g/n/ac wireless LAN, Bluetooth 4.2, BLE ■ Gigabit Ethernet over USB 2.0 (maximum throughput 300Mbps) ■ 4 × USB 2.0 ports
Access:	Extended 40-pin GPIO header
Video & sound:	■ 1 × full size HDMI ■ MIPI DSI display port ■ MIPI CSI camera port ■ 4 pole stereo output and composite video port
Multimedia:	H.264, MPEG-4 decode (1080p30); H.264 encode (1080p30); OpenGL ES 1.1, 2.0 graphics
SD card support:	Micro SD format for loading operating system and data storage
Input power:	■ 5V/2.5A DC via micro USB connector ■ 5V DC via GPIO header ■ Power over Ethernet (PoE)–enabled (requires separate PoE HAT)
Environment:	Operating temperature, 0–50°C
Compliance:	For a full list of local and regional product approvals, please visit www.raspberrypi.org/products/raspberry-pi-3-model-b+
Production lifetime:	The Raspberry Pi 3 Model B+ will remain in production until at least January 2023.

Figura 5 - Raspberry Pi Especificaciones

Lidar.
Su función es la de, mediante la emisión de haces de luz, medir la distancia de los obstáculos en función de lo que el láser tarde en rebotar y volver hasta el dispositivo.

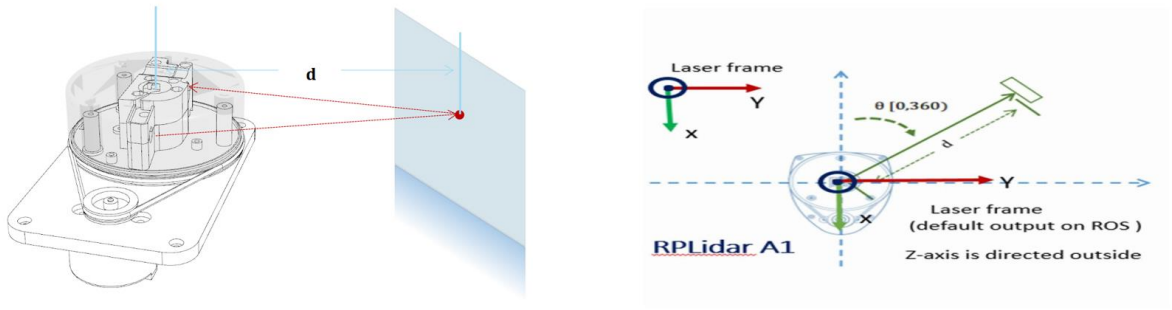

Figura 6 - LIDAR Funcionamiento

El lidar que se ha empleado en este proyecto es el *RPLIDAR A1M8*.

El dispositivo tiene una altura de 60mm, una anchura de 98.5mm, una profundidad de 70mm y un peso de 170g.

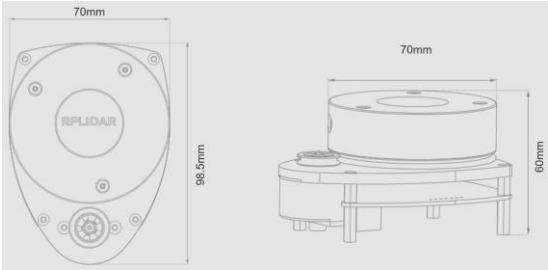

Figura 7 - LIDAR Dimensiones

En cuanto a sus especificaciones, tiene un rango de entre 0.15m y 12m de distancia y gira en torno a 360º. Sin embargo, su ángulo (en altura) es menor a 1º.[15][16]

Item	Unit	Min	Typical	Max	Comments
Measurement Range	Meter(s)	TBD	0.15 - 12	TBD	Test on while reflective objects
Angular Range	Degrees	Not Applicable	0 - 360	Not Applicable	
Measurement Res.	mm	Not Applicable	< 0.5 --- <1% of actual distance *	Not Applicable	Within 1.5 meters --- For whole measurement range *
Angular Res.	Degrees	Not Applicable	≤1	Not Applicable	Scan at 5.5hz
Time for single measurement	ms	Not Applicable	0.5	Not Applicable	
Measurement Freq.	Hz	2000	≥4000	8000	
Scan Freq.	Hz	5	5.5	10	Typical value based on 360 measurements per round

Figura 8 - LIDAR Especificaciones

Figura 9 – LIDAR Especificaciones 2

Elegoo Smart Robot Car Kit V3.0

Su función es la de actuar como vehículo para poder explorar el terreno en función de la información recogida por el LIDAR.[17]

Dimensiones del producto	24.99 x 19 x 8 cm; 1.25 kilogramos
Peso del producto	1.25 kg

Figura 10 - Especificaciones Robot

Arduino

Su función es la de transmitir al robot las órdenes que se ejecuten desde el smartphone para poder conducirlo de forma remota.

El Arduino que se ha empleado para este proyecto es el Arduino UNO Rev3. [18]

Microcontroller	ATmega328P
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
PWM Digital I/O Pins	6
Analog Input Pins	6
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328P) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)
Clock Speed	16 MHz
LED_BUILTIN	13
Length	68.6 mm
Width	53.4 mm
Weight	25 g

Figura 11 - Arduino Especificaciones

Objetivos

El objetivo inicial era coger un robot educativo simple y adentrarlo en el campo de la navegación autónoma partiendo desde el entendimiento del código Arduino que movía el robot hasta hacer que el robot se moviera de forma autónoma explorando el entorno, pero que debido al desconocimiento del esfuerzo que esto podía implicar tanto investigación como de aprendizaje de temas nuevos, así como su implementación en el proyecto, se fijaron varios objetivos a desarrollar conforme se pudiera ir avanzando, pero no se fijó una funcionalidad como objetivo a conseguir sí o sí.

Los objetivos ideales que se marcaron para conseguir en el proyecto son:

1. Comprender el funcionamiento del robot, su electrónica el Arduino.
2. Hacer funcionar el LIDAR permitiendo detectar el entorno.
3. Almacenar el entorno conforme desplazamos el LIDAR.
4. Implementar la funcionalidad del Costmap.
5. Implementar la funcionalidad de Rutas.
6. Implementar la funcionalidad de navegación autónoma en el robot.

Debido a los problemas surgidos a lo largo del año, así como al desconocimiento del tiempo que podía requerir la realización de los objetivos a la hora del planteamiento inicial de los mismos, los objetivos que han sido desarrollados en el proyecto son:

1. Comprender el funcionamiento del robot, su electrónica el Arduino.

Mediante los tutoriales llevados a cabo, así como la implementación de un mando para controlar el robot remotamente.

2. Hacer funcionar el LIDAR permitiendo detectar el entorno.

Mediante la instalación de ROS y Catkin y las librerías de rplidar.

3. Almacenar el entorno conforme desplazamos el LIDAR.

Mediante la instalación de las librerías de hector_slam.

- Otros puntos que se han llevado a cabo:
  - Estudiar el margen de error generado por el LIDAR.
  - Estudiar la viabilidad de trabajar con Roomba sin necesidad de Raspberry Pi y robot.
  - Implementar las funcionalidades del mapeo de entorno y SLAM en la Raspberry Pi.
  - Conectar mediante ssh la Raspberry Pi y el ordenador para dividir el trabajo de manera que sea la Raspberry Pi la encargada de recibir la información del LIDAR y transmitirla al ordenador para que sea este quien procese la información y la represente en el entorno gráfico.

4. Implementar la funcionalidad del Costmap.

Mediante la instalación de las librerías de hector_navigation, así como la instalación de dependencias y modificación de algunos de sus archivos debido a funciones obsoletas.

5. Implementar la funcionalidad de Rutas.

Mediante la modificación de librerías de hector_navigation, para crear así rutas que debería seguir el robot para descubrir nuevo entorno que no ha sido mapeado.

Metodología

He realizado un diagrama de Gantt para que se visualice más fácilmente la estructura temporal del proyecto.

Como se puede observar, las primeras semanas se avanza progresivamente cada semana. Sin embargo, a mediados de marzo el progreso se ve ralentizado y pausado debido al confinamiento, que evita el poder recoger material para poder continuar con el proyecto.
Desde mediados de marzo hasta finales de mayo el proyecto no avanza debido a la imposibilidad para acudir a la empresa a recoger el material necesario para continuar (el robot y la Raspberry Pi).
Tras conseguir ir para coger la Raspberry Pi y comenzar a avanzar, vuelve a ralentizarse por problemas de bajo voltaje hasta que, de nuevo, conseguimos quedar para acudir a la empresa a recoger una fuente de alimentación más potente para que la Raspberry Pi funcione correctamente.
Finalmente, a mediados de junio, el proyecto vuelve a avanzar con normalidad hasta septiembre.

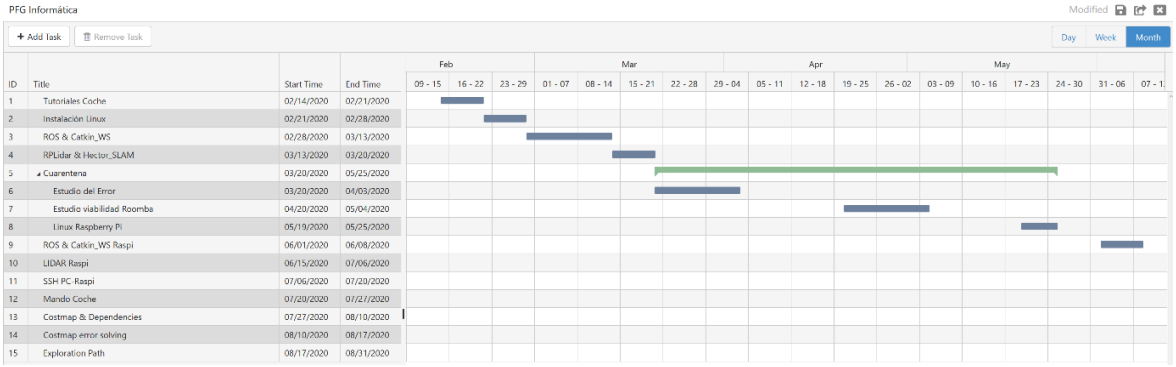

Figura 12 - Diagrama de Gantt 1

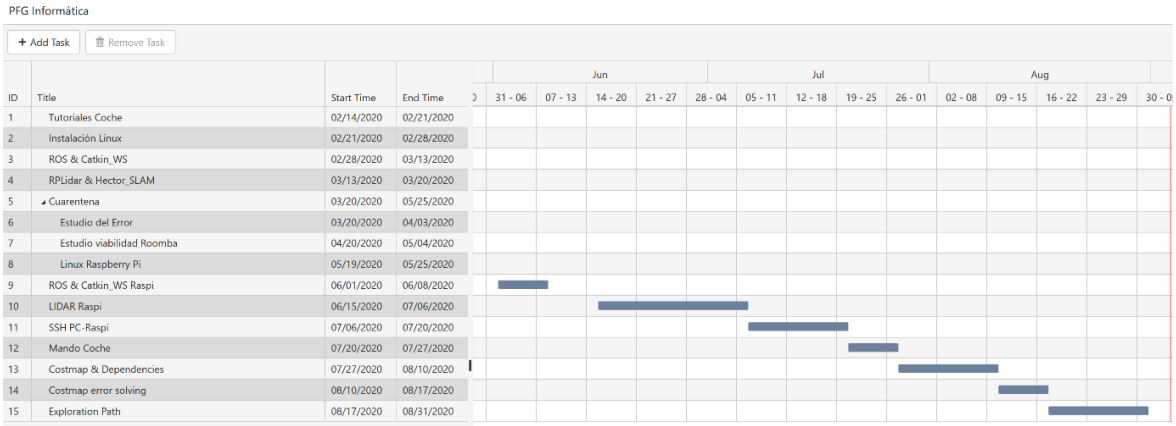

Figura 13 - Diagrama de Gantt 2

Inicialmente, no se seguía ninguna metodología concreta, se realizaban meetings en la empresa semanalmente en los que se aprovechaba para resolver los problemas que surgían a lo largo de la semana, así como para avanzar lo máximo posible en el proyecto durante esas horas.

Para la comunicación hemos estado utilizando el correo electrónico y la aplicación de Whatsapp para comunicarnos.

Comenzamos acudiendo a la empresa para una primera toma de contacto y para ver como deberíamos comenzar con el proyecto.

Dedicamos la primera semana de trabajo a familiarizarnos con el proyecto y algunos de los elementos con los que vamos a trabajar que están disponibles en la empresa, como el coche o el lidar.

También desde casa comienzo a trabajar en el portátil ya que vamos a trabajar con el Sistema Operativo de Linux, para lo cual me recomiendan en la empresa instalar una versión específica de Linux, Linux 16.04 [19].
Debido a incompatibilidades de esta versión con mi ordenador, que hacían que no funcionara la tarjeta de red, por lo que no era capaz de conseguir acceso a internet ni por wifi ni por cable, finalmente instalo el sistema operativo Linux Mint 19.3 Cinnamon.

Una vez instalado Linux en el ordenador que voy a utilizar para trabajar en el proyecto, comenzamos a trabajar con el Lidar.

Sin embargo, dadas las dificultades que han surgido para seguir trabajando con la empresa debido a la cuarentena, se modificó la metodología.

Pasamos a utilizar una metodología tipo Kanban, apoyándonos en la plataforma Trello donde se crearon todas las tareas que habían sido desarrolladas, así como las que considerábamos que teníamos que desarrollar a modo de tarjetas.

Se asignaron colores a cada tarjeta para una mayor organización de manera que aquellas relacionadas con el Lidar, tenían color amarillo, las relacionadas con la Raspberry Pi color verde, las relacionadas con el coche color rojo, las relacionadas con el ordenador morado, las relacionadas con creación de programas naranja y, finalmente, se utilizaba el color negro para otras tareas.
Una vez comenzada la cuarentena y habiendo completado el trabajo con el Lidar y el estudio del margen de error, organizo el proyecto de Trello para ponerlo al día, teniendo el siguiente tablero:

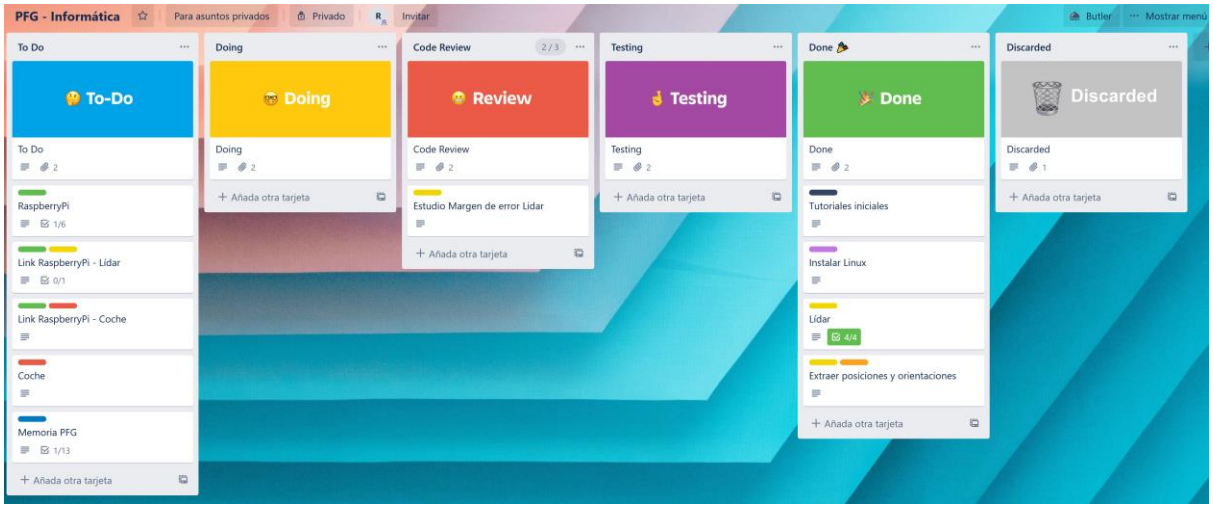

Figura 14 - Trello inicial

Dentro de cada una de las tarjetas asigno las tareas necesarias, para llevar a cabo esa parte del proyecto.

Comenzamos con la tarea de la instalación de Linux, la cual es muy simple y lleva apenas un día.

Figura 15 – Trello - Instalar Linux

El siguiente paso consiste en instalar ROS, crear el Workspace de Catkin y añadir las librerías necesarias para conseguir que el lidar mapee el entorno.

Figura 16 - Trello - Lidar

Una vez instalado y tras testearlo, tenemos el dispositivo capaz de mapear el entorno, lo dejamos en una posición fija (0, 0, 0) grabando datos durante 5 minutos y extrayendo esos datos automáticamente a un archivo de texto. La siguiente tarea, consiste en crear un programa para que todos esos datos del archivo de texto queden separados en diferentes archivos en función de la posición y orientación.

Figura 17 - Trello - Programa

Una vez tenemos todos los datos extraídos en diferentes archivos de texto (uno para cada posición y orientación), comienza el estudio del margen de error para posteriormente estudiar la viabilidad del dispositivo en este proyecto.

Figura 18 - Trello - Margen de Error

En este momento, la tarjeta del estudio del margen de error está pendiente de revisión, pero comienza la cuarentena, por lo que debido a la falta de material para poder continuar, busco alguna alternativa para poder continuar por mi cuenta.

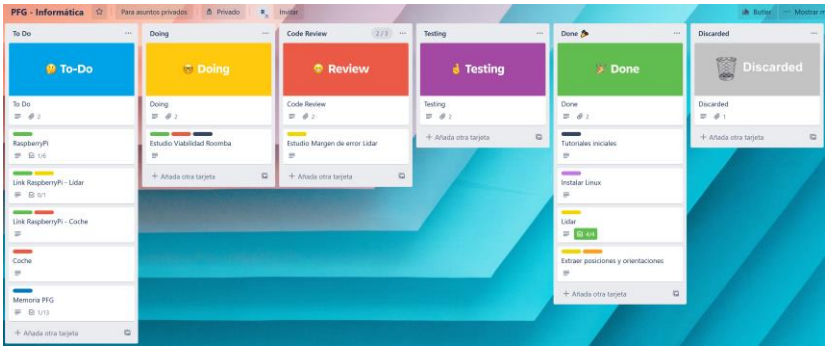

Figura 19 - Trello - Cuarentena

Figura 20 - Trello - Roomba

Finalmente, tras realizar el estudio sobre la viabilidad de utilizar el Roomba en el proyecto, sustituyendo al coche y a la Raspberry pi, la opción queda descartada debido a que sigue siendo necesario el uso de la Raspberry pi para poder hacer de intermediario entre el Roomba y el Lidar. Finalmente decido esperar a que se normalice la situación para continuar con el proyecto.

Una vez normalizada la situación, a mediados de mayo, consigo quedar con la empresa para poder hacerme con el coche y la Raspberry pi y poder seguir trabajando en el proyecto como estaba previsto.

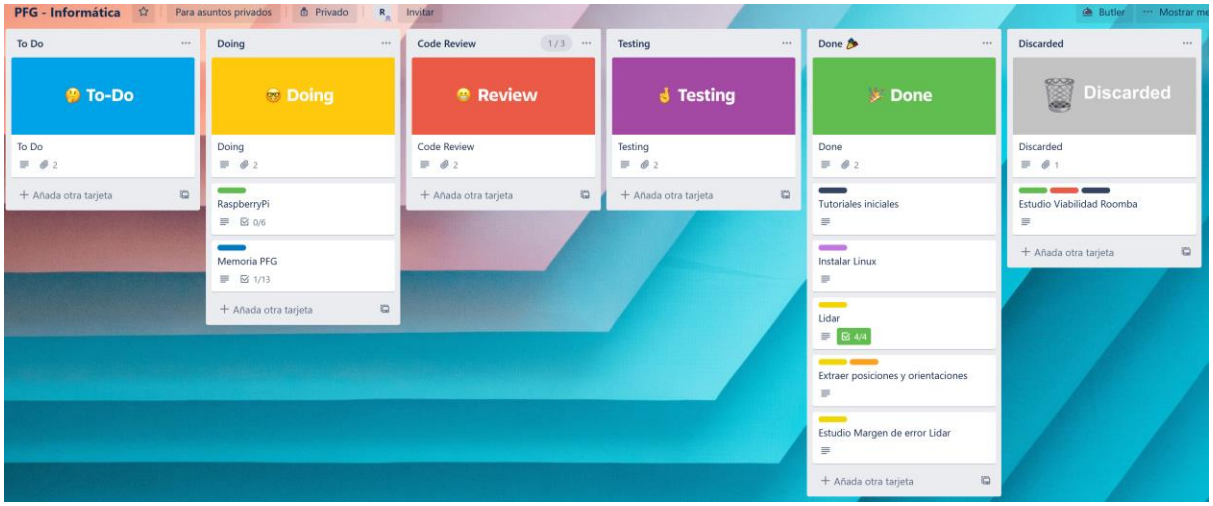

Figura 21 - Trello - Tras cuarentena

Tras haber descartado la opción de trabajar con el Roomba y haber revisado el margen de error provocado por el Lidar, toca implementar en la Raspberry pi un proyecto similar al que habíamos implementado en el ordenador.

Figura 22 - Trello - Raspberry pi

Comenzamos instalando Linux, en este caso, una versión de Ubuntu para Raspberry pi 3. Una vez instalado, seguimos los pasos que habíamos llevado a cabo en el ordenador para hacer funcionar el Lidar.

Sin embargo, tras llevar a cabo todos los pasos, al tratar de testear el proyecto, no conseguimos que funcione.

Figura 23 - Trello - Raspberry pi 2

Se produce un error debido a problemas de bajo voltaje por lo que necesitamos hacernos con una fuente de alimentación más potente.

Una vez conseguida y testeando que el proyecto funciona correctamente, buscamos la manera de conectarnos remotamente desde el ordenador a la Raspberry pi para trabajar desde el ordenador.

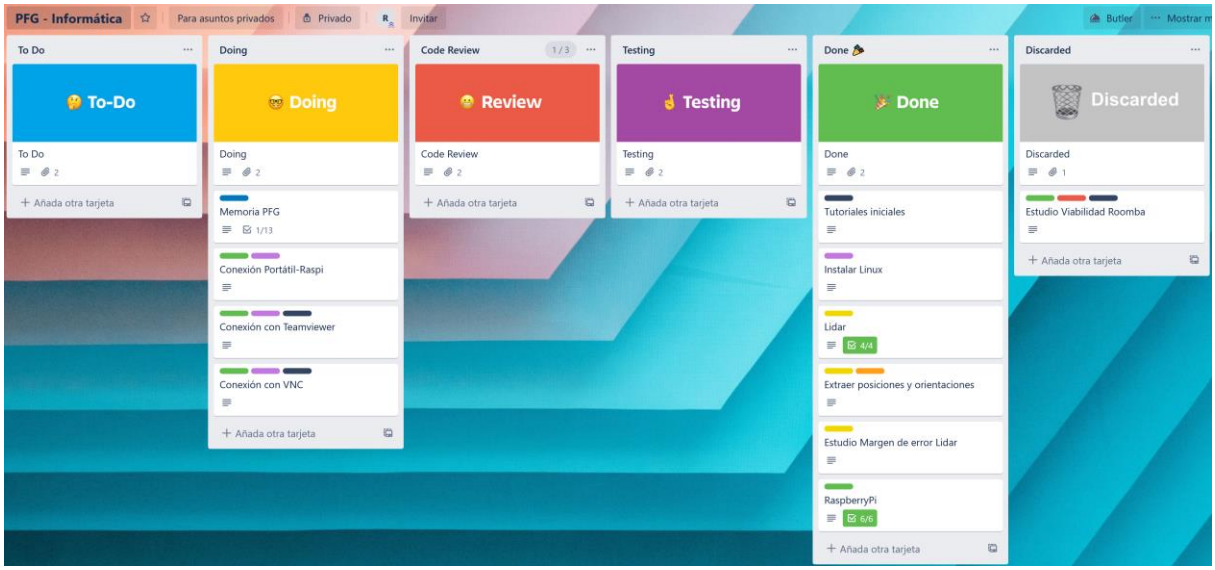

Figura 24 - Trello - Portátil-Raspi

Para ello probamos con los programas TeamViewer y VNC Viewer/Server.

Sin embargo, quedan descartados debido a que al testear, la Raspberry pi no es capaz de procesar la información del lidar, representarla y a la vez en tiempo real, transmitir toda esa información por cualquiera de esos dos programas.

Figura 25 - Trello - Conexión TeamViewer/VNC

Finalmente, ya que la Raspberry pi y el ordenador van a estar conectados a la misma red, decidimos conectarlos mediante ssh, para lo cual necesitamos acceder cada uno a la IP del otro.

Figura 26 - Trello - Conexión ssh

Una vez tenemos el portátil y la Raspberry pi conectados mediante ssh para poder trabajar únicamente desde el portátil sin necesidad de estar cambiando de un dispositivo a otro, buscamos una solución para no saturar la Raspberry pi, por lo que investigamos la posibilidad de que la Raspberry pi se encargue de recoger los datos del lidar, pero que sea el ordenador quien se encargue de procesarlos y representarlo.

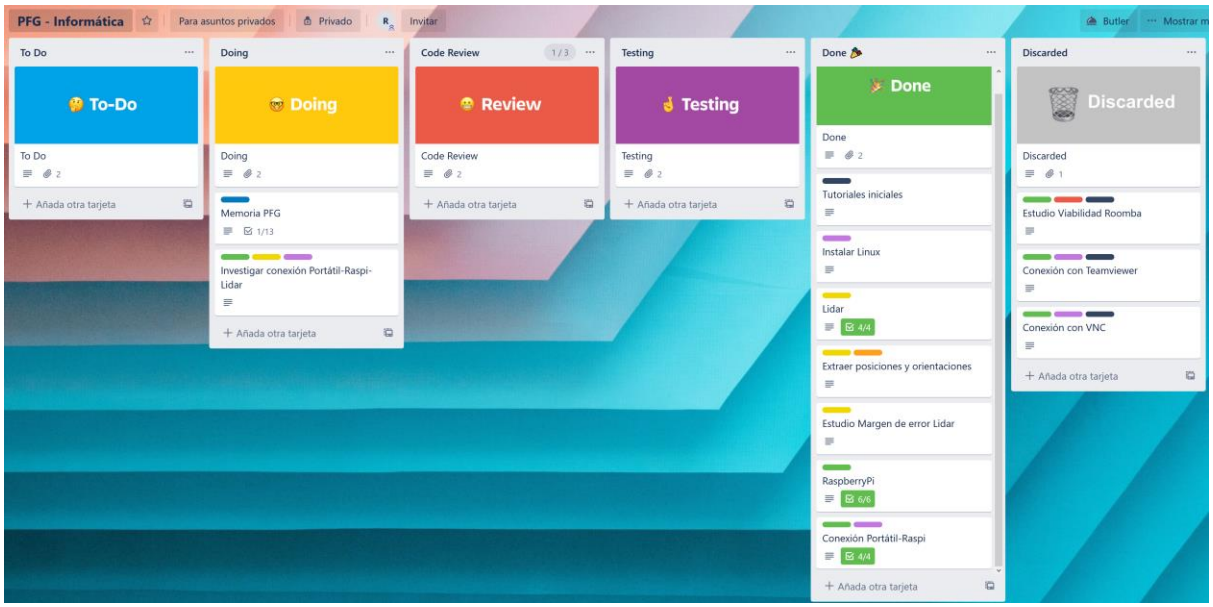

Figura 27 - Trello - Portatil-Raspi-Lidar

Figura 28 - Trello – Conexión Portatil-Raspi-Lidar

Encontramos un tutorial para conectar varias máquinas con ROS para conseguir nuestro objetivo, de modo que añadimos la tarea de testear si funciona correctamente.

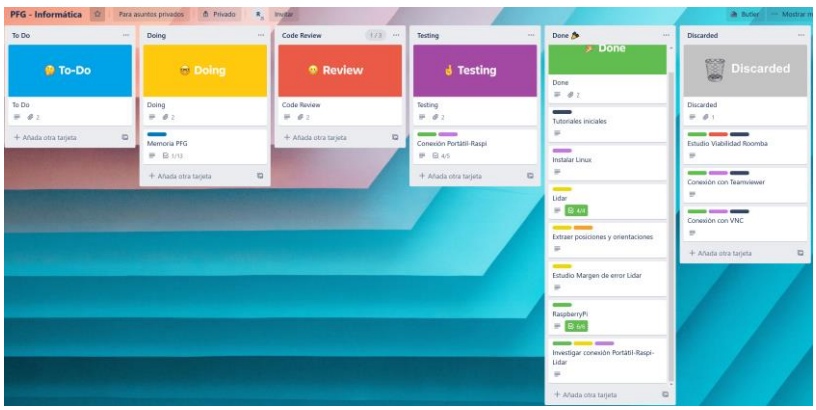

Figura 29 - Trello - Test Conexión

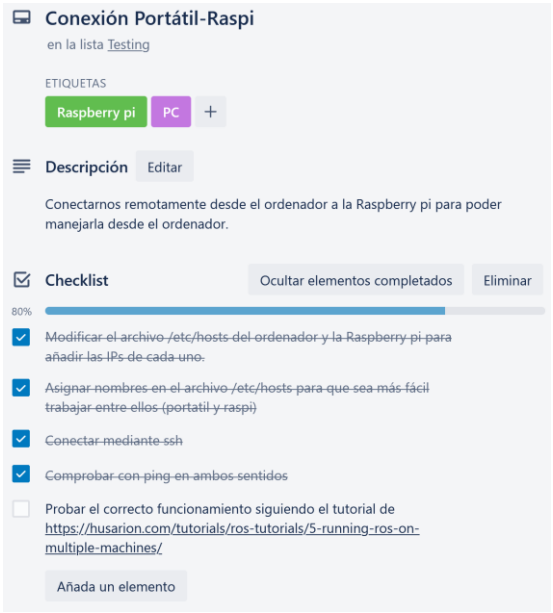

Figura 30 - Trello – Tarea conexión Portátil-Raspi

Tras varios testeos y conseguir que funcionen correctamente, creo una copia de seguridad tanto del portátil como de la Raspberry pi, para evitar cualquier problema que hubiera ya que el proyecto está funcionando correctamente de la manera que nos interesa.

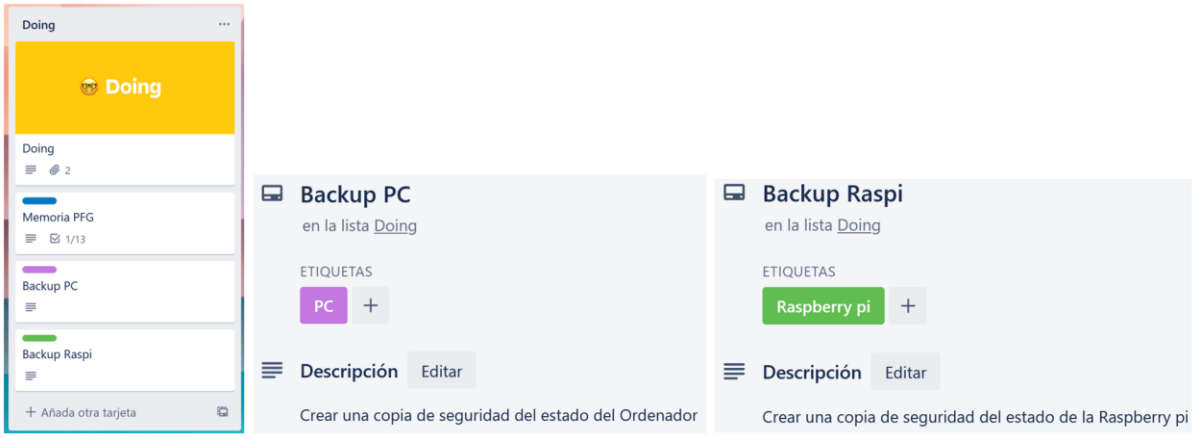

Figura 31 - Trello - Backup

Ahora que tenemos el proyecto funcionando correctamente, creamos un mando en el móvil mediante la APP de Elegoo.

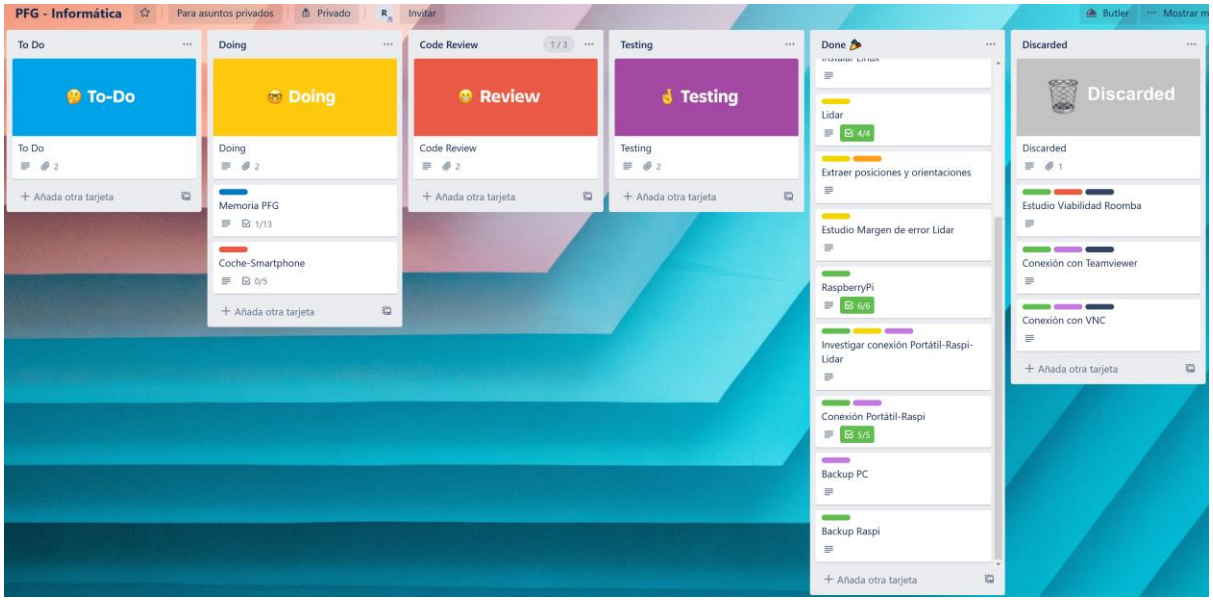

Figura 32 - Trello - Mando Coche

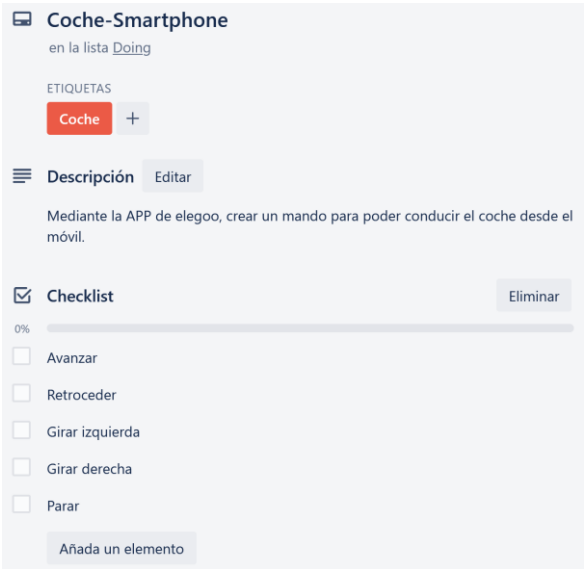

Figura 33 - Trello - Tareas mando

Finalmente, una vez tenemos creado el mando, testeamos que capaces de mover el coche desde el móvil mientras, sobre él, estaría la Raspberry pi y el lidar mapeando el entorno y recibiríamos el mapa creado en el ordenador, por lo que podemos conducir el coche desde el móvil evitando los obstáculos basándonos en el mapa que vemos en la pantalla del ordenador.

A continuación, decidimos implementar una nueva funcionalidad de ROS para crear un "costmap" que coloree los obstáculos de rojo, así como asignarles un radio alrededor para asegurar que se eviten las colisiones entre el robot y los obstáculos.

Figura 34 - Trello - Costmap

Tras importar, las librerías e intentar compilar para posteriormente testearlo, nos damos cuenta de que necesitamos añadir dependencias que son necesarias para el funcionamiento de los paquetes de `hector navigation`.

Figura 35 - Trello - Costmap Dependencias

Una vez importamos las dependencias, al tratar de testear el proyecto nos encontramos con errores de incompatibilidad entre tf y la versión que estamos utilizando de ros (ros-melodic), debido a que tf está obsoleto, por lo que necesitamos migrar manualmente las partes del código que generan errores a tf2.

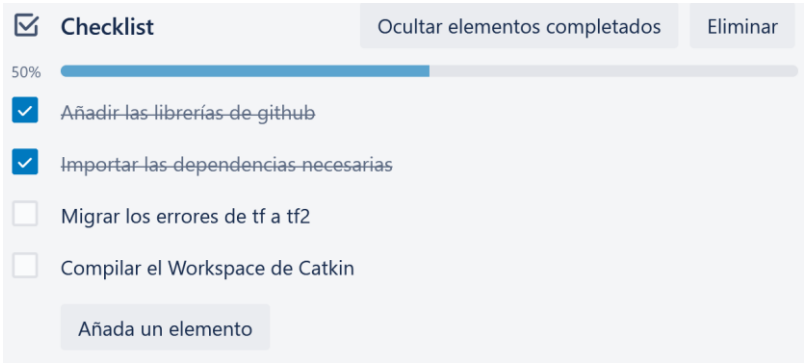

Figura 36 - Trello - Migrar tf a tf2

Tras solventar los errores que aparecen cada vez que intentamos compilar el proyecto, conseguimos lanzarlo correctamente y testear la nueva funcionalidad implementada.

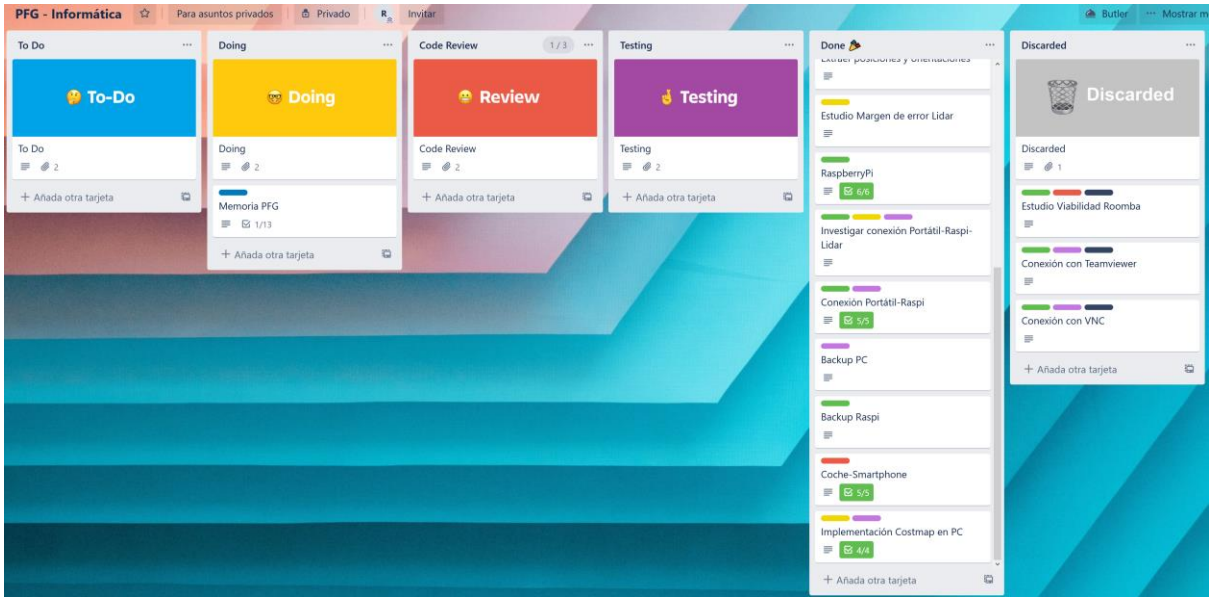

Figura 37 - Trello - Costmap completo

Una vez hemos completado el `costmap` y vemos que funciona correctamente, observamos que podemos realizar modificaciones en las librerías empleados para la representación del `costmap` para ser capaces de añadir la funcionalidad de exploración al robot. De este modo, haremos que se muestre una ruta actualizable en función de la posición, que deberá seguir el robot para ir descubriendo nuevo entorno que todavía no había sido explorado.

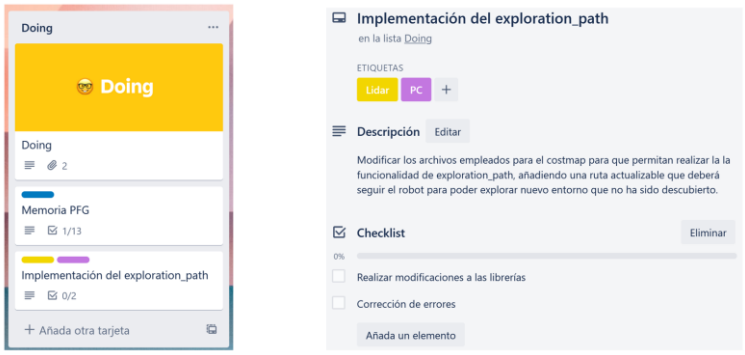

Figura 38 - Trello - Exploration path

Finalmente, tras testear que funciona correctamente, y testear todas las funcionalidades implementadas en el proyecto hasta el momento, realizamos una copia de seguridad del ordenador, para almacenar la última versión funcional del proyecto disponible hasta el momento.

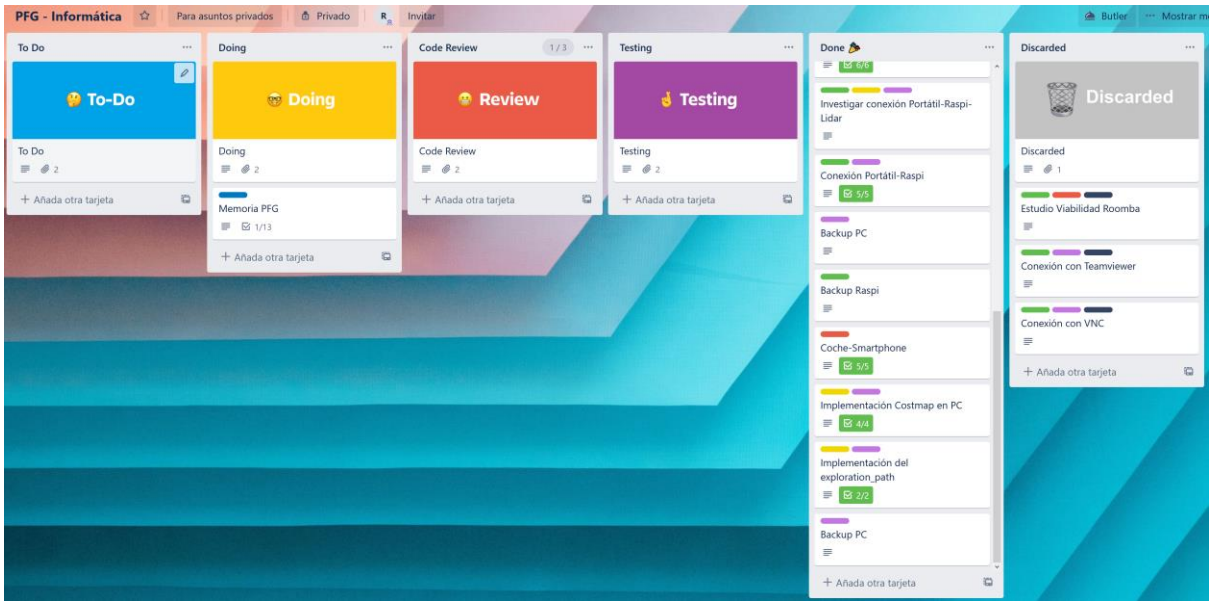

Figura 39 - Trello - Proyecto completo

Análisis, diseño e implementación

He dividido este apartado en un subapartado de análisis y otro de diseño e implementación.

Análisis

Estudio del margen de error del lidar.

He realizado un breve estudio para calcular el margen de error producido por el lidar, grabando datos en una posición fija (0,0,0) durante 5 minutos, por lo que los resultados en una situación ideal deberían ser siempre 0.

Para este estudio, he creado un programa para dividir los datos obtenidos en diferentes archivos según si eran valores de posición x, y, z u orientación w, x, y, z.

Los valores obtenidos inicialmente seguían la siguiente estructura:

test5min.txt

```
.........
header:
  seq: 288
  stamp:
    secs: 1583864570
    nsecs:  38087662
  frame_id: "map"
pose:
  position:
    x: 0.0045280456543
    y: 0.000663757324219
    z: 0.0
  orientation:
    x: 0.0
    y: 0.0
    z: 0.00166534108575
    w: 0.999998629093
---
header:
  seq: 289
  stamp:
    secs: 1583864570
    nsecs: 173361762
  frame_id: "map"
pose:
  position:
    x: -0.00300979614258
    y: 0.00962066650391
    z: 0.0
  orientation:
    x: 0.0
    y: 0.0
    z: -2.87612256216e-05
    w: 1.0
```

Tras usar el programa creado, obtenía únicamente los valores de cada variable:

```
positionX.txt
0.0045280456543
-0.00300979614258
-0.00456619262695
-0.00113677978516
0.00244140625
-0.00474166870117
```

Tras esto, solamente ha habido que trabajar con esos datos para realizar el estudio del margen de error, calculando el valor promedio, la desviación estándar y los máximos y mínimos, así como representando estos resultados.

Posición X

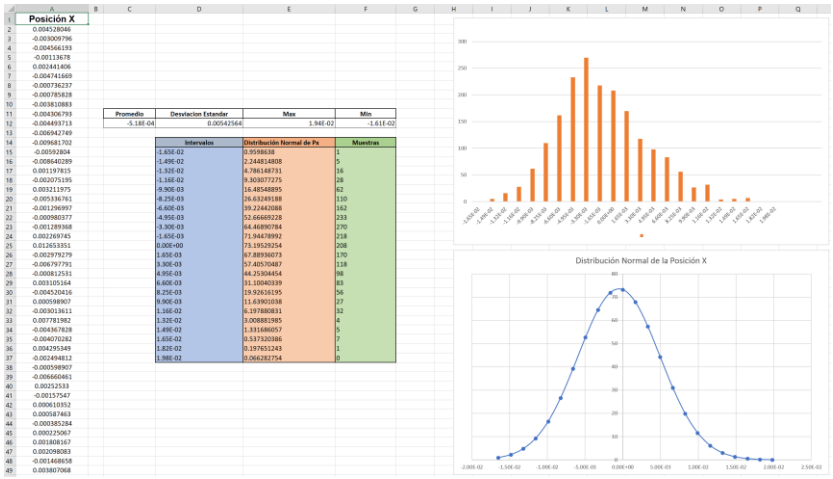

Figura 40 - Error Posición X

Posición Y

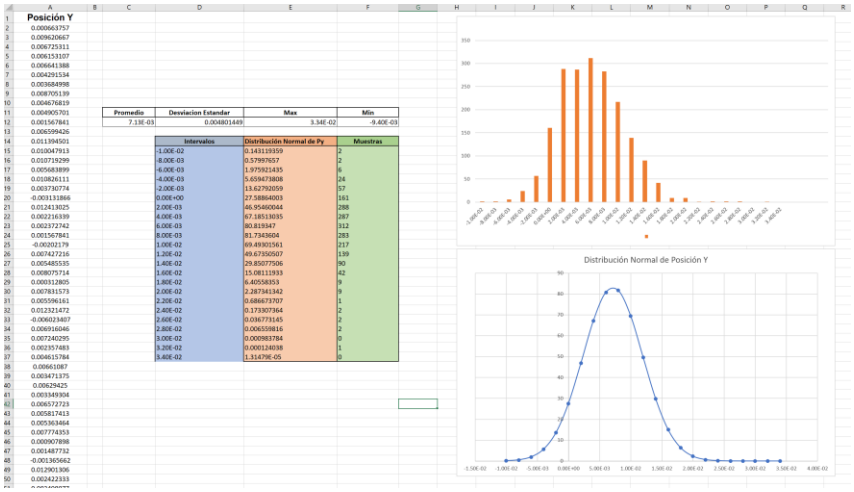

Figura 41 - Error Posición Y

Posición Z

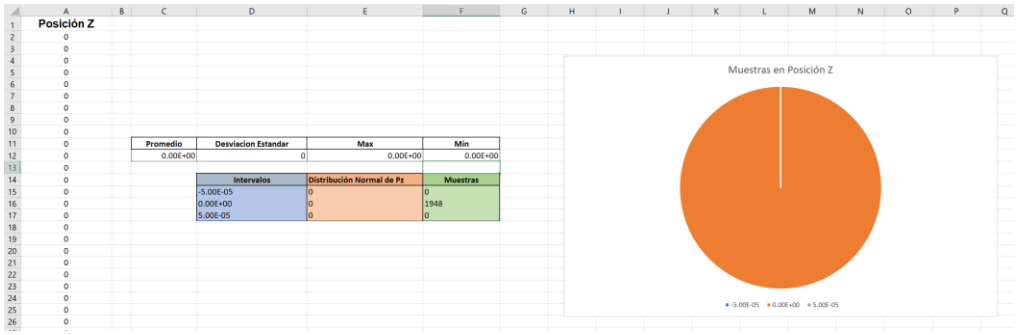

Figura 42 - Error Posición Z

Orientación W

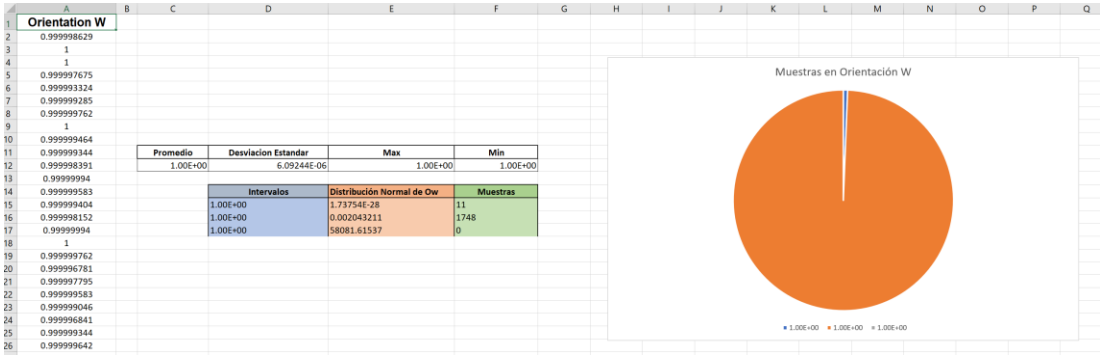

Figura 43 - Error Orientación W

Orientación X

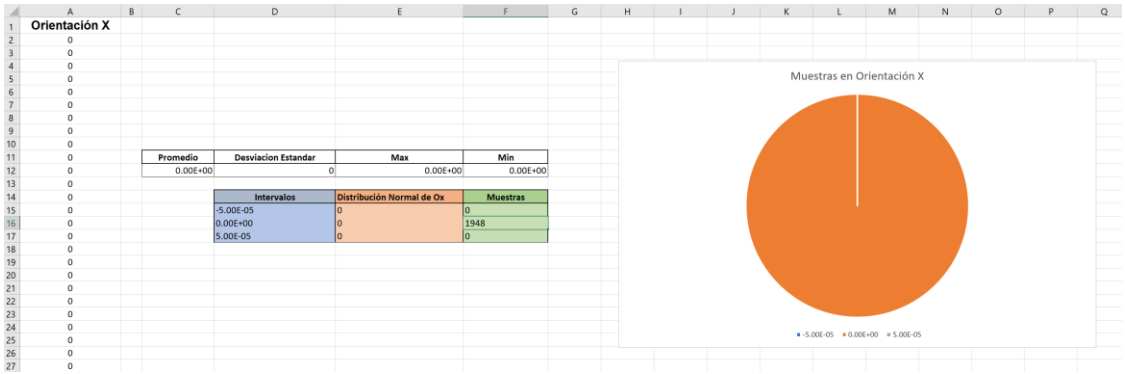

Figura 44 - Error Orientación X

Orientación Y

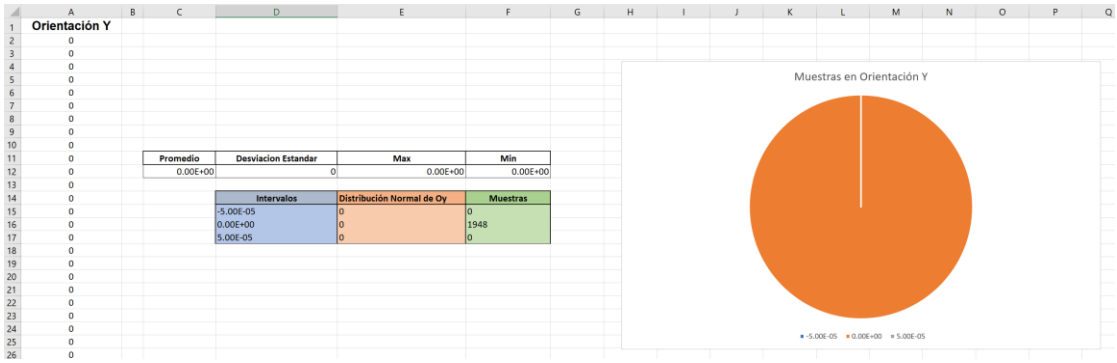

Figura 45 - Error Orientación Y

Orientación Z

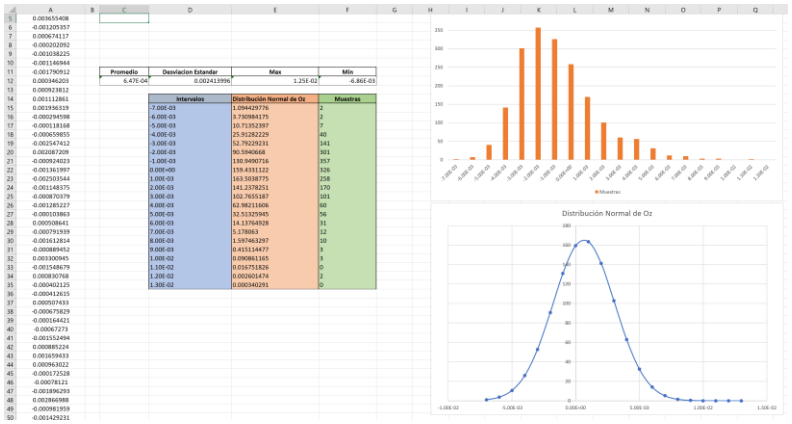

Figura 46 - Error Orientación Z

Con esta información concluimos con que hay un margen de error que es inferior a las milésimas-diezmilésimas, lo cual podemos decir que es un error despreciable para el proyecto en el que vamos a trabajar.

Estudio de la viabilidad del Roomba.

Dispongo de un iRobot Roomba 630[20] en casa, por lo que investigo si es posible trabajar con él sin necesidad de abrirlo o, al menos, sin necesidad de hacer uso de hardware externo, quedando el sistema del siguiente modo:

Figura 47 - Roomba Lidar

Tras investigar y encontrar tutoriales [21] en los que desmontan el dispositivo, para conocer los diferentes métodos de conectividad que emplea para recibir/transmitir información así como el manual de usuario[22], encuentro que no solo no mencionan que tenga Wi-Fi sino que ni si quiera mencionan el Bluetooth. Los únicos sensores que utiliza son un sensor de infrarrojos, sensores de suciedad, sensores de desnivel y un sensor de choque. Por lo que en caso de que sea posible conectarme a la máquina, será necesario abrirla y usar algún tipo de cable para conectarme.

Consigo encontrar el OIS (Open Interface Specification)[23] en el que se pueden observar los pines, su nombre y su utilidad, así como los códigos de los comandos para llevar a cabo las acciones disponibles, como por ejemplo:

Start **Opcode: 128** **Data Bytes: 0**

This command starts the OI. You must always send the Start command before sending any other commands to the OI.

- Serial sequence: [128].
- Available in modes: Passive, Safe, or Full
- Changes mode to: Passive. Roomba beeps once to acknowledge it is starting from "off" mode.

Reset **Opcode: 7** **Data Bytes: 0**

This command resets the robot, as if you had removed and reinserted the battery.

- Serial sequence: [7].
- Available in modes: Always available.
- Changes mode to: Off. You will have to send [128] again to re-enter Open Interface mode.

Stop **Opcode: 173** **Data Bytes: 0**

This command stops the OI. All streams will stop and the robot will no longer respond to commands. Use this command when you are finished working with the robot.

- Serial sequence: [173].
- Available in modes: Passive, Safe, or Full
- Changes mode to: Off. Roomba plays a song to acknowledge it is exiting the OI.

Figura 48 - Open Interface Specification

Con esta información, sí que es posible trabajar con el Roomba y programar su movimiento. Sin embargo, sigo sin encontrar la manera de conectarme.

Finalmente, encuentro por un lado, una guía en la que se modifica la programación del Roomba para poder activarlo remotamente, poder asignar horas de limpieza y que, en caso de quedarse sin batería, vuelva al punto de carga y siga limpiando.[24]

En la misma guía, más adelante también le añaden una tarjeta de red para añadirle conexión a internet, sin embargo, en todos los tutoriales que encuentro, no solo necesitan abrir el dispositivo, sino que además necesitan hardware externo (el cual no dispongo) que emplean ya sea para añadir o para modificar las funcionalidades del Roomba.

En mi caso, probablemente necesitaría un Arduino y/o una Raspberry pi, que pudiera añadir, así como los cables para poder conectarlos al Roomba.

De este modo, el Roomba sustituiría al Elegoo Car, pero seguiría necesitando un dispositivo que hiciera de intermediario para procesar la información y conectarlo al Roomba:

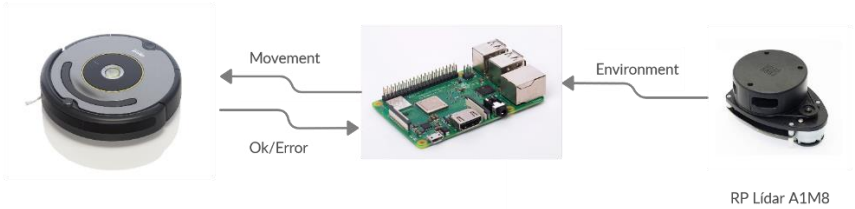

Figura 49 - Lidar - Raspi - Roomba

Debido a la falta de los mismos materiales que necesitaría para trabajar con el coche, me encuentro en la misma situación que al comienzo de la cuarentena, con la diferencia de que el Roomba podría sustituir al coche.

Conclusión, podría sustituir el Roomba por el coche, sin embargo, seguiría necesitando un dispositivo intermedio que actuara como "ordenador" para recibir la información del Lidar y procesarla y otro (o el mismo) para convertir esa información en comandos los comandos correspondientes para mover el coche.

Diseño e implementación

El proyecto consiste inicialmente en la implementación de un dispositivo RPLidar A1M8[16] conectado a una Raspberry Pi 3 Model B+[25] sobre un Elegoo Smart V3.0 Robot Car Kit[26][27] conectado a un Arduino Mega 2560[28], de manera que el Lidar con la Raspberry Pi actuando como procesador, obtenga la información del entorno (obstáculos) y la Raspberry Pi procese esa información creando un mapa del entorno que detecta el Lidar y permitiendo al usuario asignar un punto de destino, enviando esa información al Arduino del Coche para que este, en base a esa información sea capaz de llevar al coche hasta el destino de manera autónoma evitando los obstáculos.

Comenzamos el proyecto viendo el coche, cómo funciona, que funcionalidades se pueden implementar, qué funcionalidades hay ya implementadas, tutoriales… para ver cómo vamos a poder trabajar con él. [29]

Instalamos el programa para trabajar con Arduino, Arduino IDE 1.8.12[30]. Lo conectamos al ordenador mediante USB y comenzamos a trabajar con él, haciendo diferentes tutoriales proporcionados por Arduino[31]. Y, a pesar de haber realizado todos, tan solo podían hacerse funcionar unos pocos, ya que muchos de ellos requerían elementos externos como altavoces para programar el dispositivo para hacer reproducir sonidos en diferentes frecuencias y tiempos, por lo que solo podía hacer funcionar en ese momento los de encender/apagar los leds del Arduino, que eran los primeros tutoriales.

Tras esto, creo una partición en el ordenador portátil para instalar Linux. Comencé instalando la versión de Linux 16.04[19], pero tras instalarla e intentar empezar a trabajar con ella encuentro incompatibilidades con mi tarjeta de red, que me imposibilita trabajar con internet tanto por wifi como por cable, por lo que tengo que instalar una versión más reciente. Finalmente instalo Linux Mint 19.3[32].

Una vez instalado Linux, vuelvo a instalar el Arduino IDE, así como a descargar los tutoriales. También instalo Visual Studio Code. [33]

En este momento, comenzamos a trabajar con el Lidar para ver cómo funciona. Nuestro objetivo es conectar el lidar con el ordenador para ser capaces de representar la información obtenida por el lidar en forma de mapa de puntos en los que colisiona el láser.

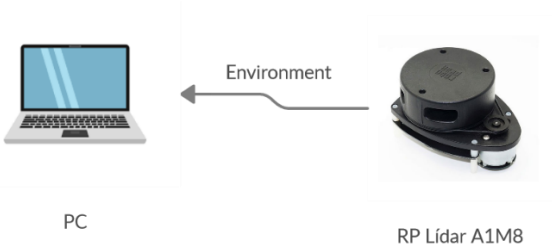

Figura 50 - PC - Lidar

Para ello, comienzo instalando ROS Melodic[34], que nos permitirá mapear el entorno por el que nos movamos.

```
$ sudo apt-get update
$ sudo mintupdate-cli upgrade -r
$ cat /etc/apt/sources.list.d/ros-latest.list
$ cat /etc/apt/sources.list
// Para aceptar paquetes de ROS.
$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main"
> /etc/apt/sources.list.d/ros-latest.list'
// Configuramos las claves
$ sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-key
C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
// Comprobar que está actualizado.
$ sudo apt update
// Instalamos ros-melodic para escritorio
$ sudo apt install ros-melodic-desktop-full
// Comprobamos si hay más paquetes de ros-melodic
$ apt search ros-melodic
// Inicializamos rosdep
$ sudo rosdep init
$ rosdep update
// Guardamos las variables de entorno
$ echo "source /opt/ros/melodic/setup.bash" >> ~/.bashrc
// Nos pide instalar la librería de Python para hacer funcionar ROS
$ sudo apt install python-rosinstall python-rosinstall-generator python-wstool
build-essential
```

[34][35][36][37]

Necesitamos instalar también catkin[38] para crear un workspace para el proyecto, que utilizaremos para instalar cualquier librería que sea necesaria en un futuro.

```
// Instalamos Catkin para la versión de ROS Melodic.
$ sudo apt-get install ros-melodic-catkin
// Creamos la carpeta en la que tendremos el workspace (tfg) y una carpeta (src) que usaremos para añadir todos los paquetes que vayamos a necesitar.
$ mkdir -p ~/tfg/src
// Tras esto volvemos a la carpeta del workspace y ejecutamos el comando catkin_make que hace un build de todos los paquetes localizados en la carpeta src
$ cd ~/tfg/
$ catkin_make
// Después clonamos el repositorio de github de rplidar_ros en la carpeta src
$ cd src
$ git clone https://github.com/Slamtec/rplidar_ros.git
// Una vez tenemos el proyecto en el workspace, volvemos a la carpeta raíz del workspace y ejecutamos catkin_make para hacer un build con los nuevos paquetes.
$ catkin_make
```

[39][40][41]

Una vez hemos hecho esto, ya tenemos el workspace listo para recibir la información del lidar y mapear el entorno. Nos encontramos con un error por falta de permisos del USB, por lo que necesitamos dar permisos cada vez que conectemos el USB del lidar con el comando

```
// Comprobamos a qué usb se conecta el lidar. Para ello accedemos a /dev y hacemos un listado de los dispositivos conectados
$ cd /dev
$ ls
// Comprobamos que el dispositivo que aparece cuando conectamos el lidar es el ttyUSB0, por lo que será el dispositivo que necesitará permisos.
$ sudo chmod 666 /dev/ttyUSB0
```

[42][43]

Llegados a este punto, somos capaces de hacer funcionar el lidar. Mediante los siguientes comandos:

```
// Necesitamos abrir 2 ventanas de comandos y ejecutar los siguientes comandos.
// Accedemos al workspace
$ cd tfg
// Iniciamos la configuración
$ source devel/setup.bash
// Proporcionamos permisos de lectura y escritura al USB para poder acceder al lidar
$ sudo chmod 666 /dev/ttyUSB0

// Una vez hecho esto en las dos ventanas:
//En la primera ventana de terminal ejecutamos ROS:
$ roscore
```

```
//En la siguiente ventana ejecutamos el visor para mapear el entorno:
$ roslaunch rplidar_ros view_rplidar.launch
```

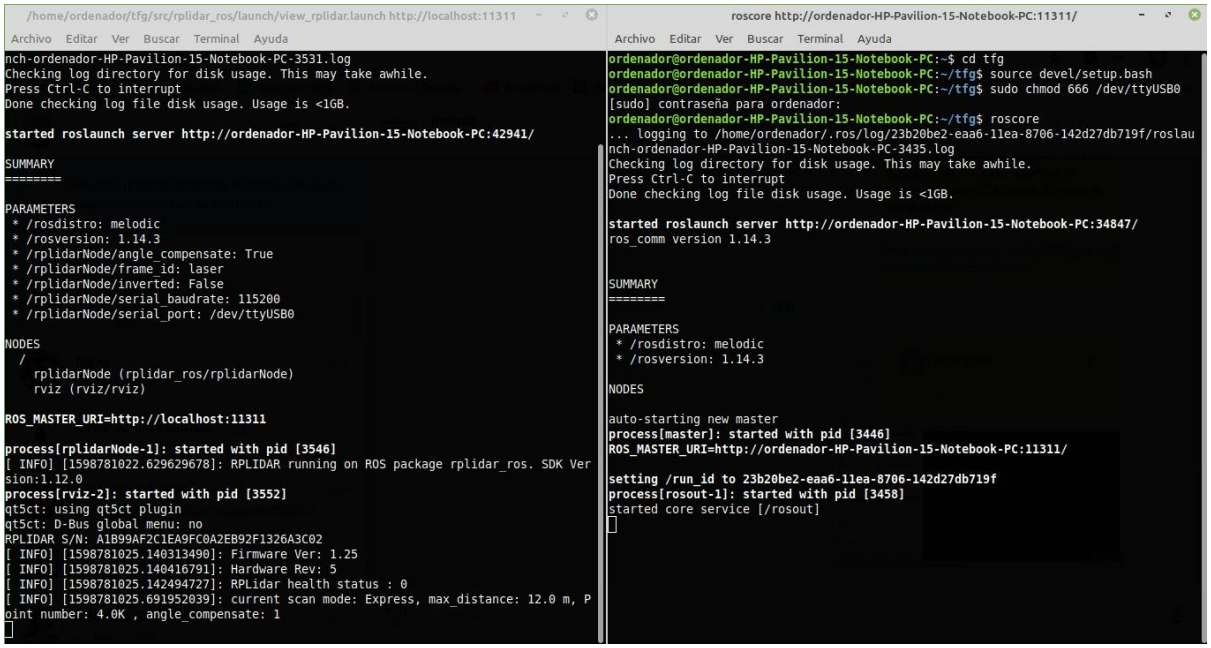

Figura 51 - Consola Roscore y Lidar

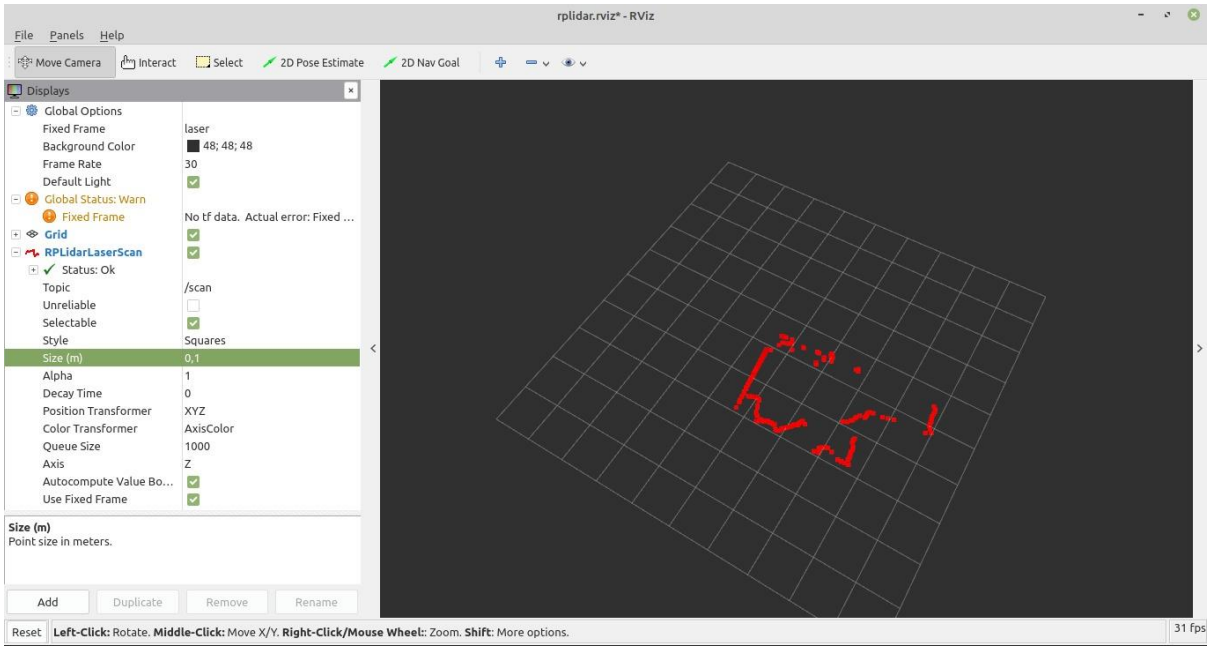

Figura 52 - Visor Lidar

Con esto, conseguimos mapear el entorno en el que nos encontramos conforme movemos el dispositivo lidar. Sin embargo, el lidar se encuentra siempre en la misma posición (supongamos (0, 0, 0)) y es el entorno que se mapea el que se va moviendo en caso de que traslademos o rotemos el lidar. Por lo

que podemos ver el entorno en el estado actual pero no somos capaces de ver cómo era antes de realizar cualquier desplazamiento con el lidar.

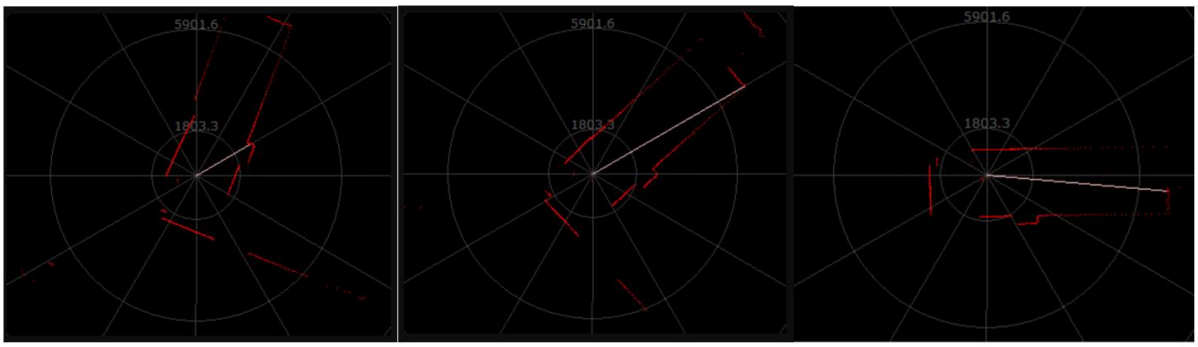

Figura 53 - Lidar Rotando [16]

Sin embargo, lo que queremos es que el entorno permanezca inmóvil y sea el dispositivo el que se mueva, de modo que, si desplazamos el lidar por el entorno, vayamos "descubriendo" nuevas áreas al evitar obstáculos que tapen el resto del entorno.

Para ello, instalamos los paquetes de Hector Slam, que incluye modificaciones sobre los paquetes de ROS que nos permitirán mapear el entorno conforme el lidar se desplaza, así como mapear la ruta que va siguiendo el dispositivo. Este es mucho más simple de instalar, ya que, una vez tenemos configurado y funcionando el Lidar con las instalaciones previas, únicamente necesitamos descargar el repositorio de github y hacer un build en el workspace.

```
// Accedemos a la carpeta src del workspace que es donde guardamos los paquetes del proyecto.
$ cd ~/tfg/src
// Clonamos el repositorio de Hector SLAM
$ git clone https://github.com/NickL77/RPLidar_Hector_SLAM.git
// Una vez tenemos el repositorio en la carpeta src, volvemos a la carpeta raíz del workspace y ejecutamos catkin_make para hacer un build con los nuevos paquetes
$ cd ..
$ catkin_make
```

Ahora, los comandos que deberemos lanzar para hacer que funcione son:

```
// Una vez tenemos hector slam, necesitamos abrir 3 ventanas de comandos y ejecutar los siguientes comandos en cada una de ellas.
// Accedemos al workspace
$ cd tfg
// Iniciamos la configuración
```

```
$ source devel/setup.bash
// Proporcionamos permisos de lectura y escritura al USB para poder acceder al lidar
$ sudo chmod 666 /dev/ttyUSB0

// Una vez hecho esto en las tres ventanas:
// En la primera ventana de terminal ejecutamos ROS:
$ roscore
// En la siguiente ventana ejecutamos el visor para mapear el entorno:
$ roslaunch rplidar_ros view_rplidar.launch
// Finalmente, en la última ventana de comandos, lanzamos hector slam:
$ roslaunch hector_slam_launch tutorial.launch
```

De este modo, sobre el programa para mapear el entorno que usábamos, ejecutamos el de hector slam, que nos permite hacer que las posiciones entorno sean fijas mientras el dispositivo se mueve.

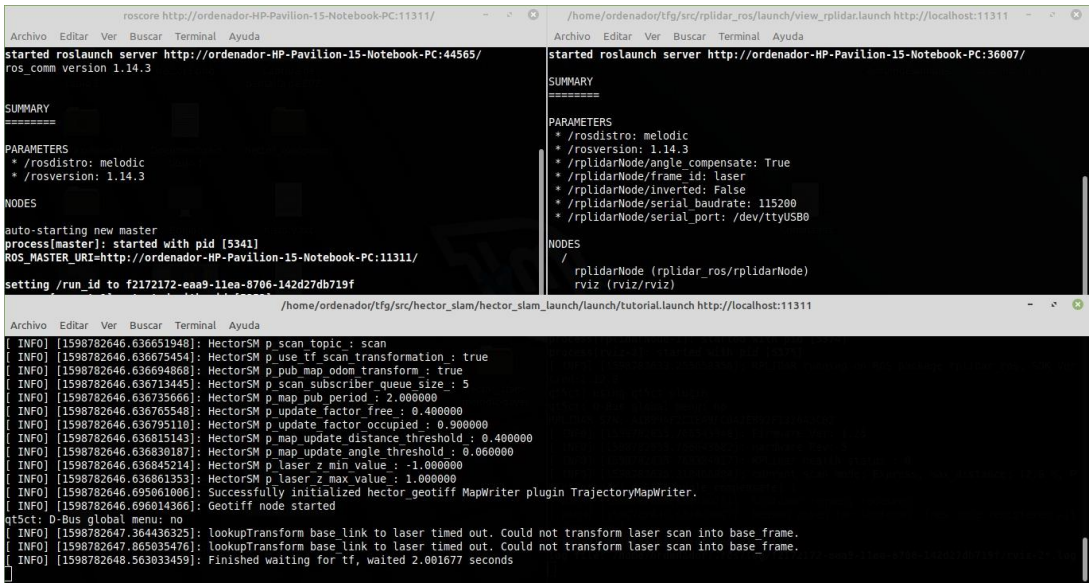

Figura 54 - Consola Roscore, Lidar y HectorSlam

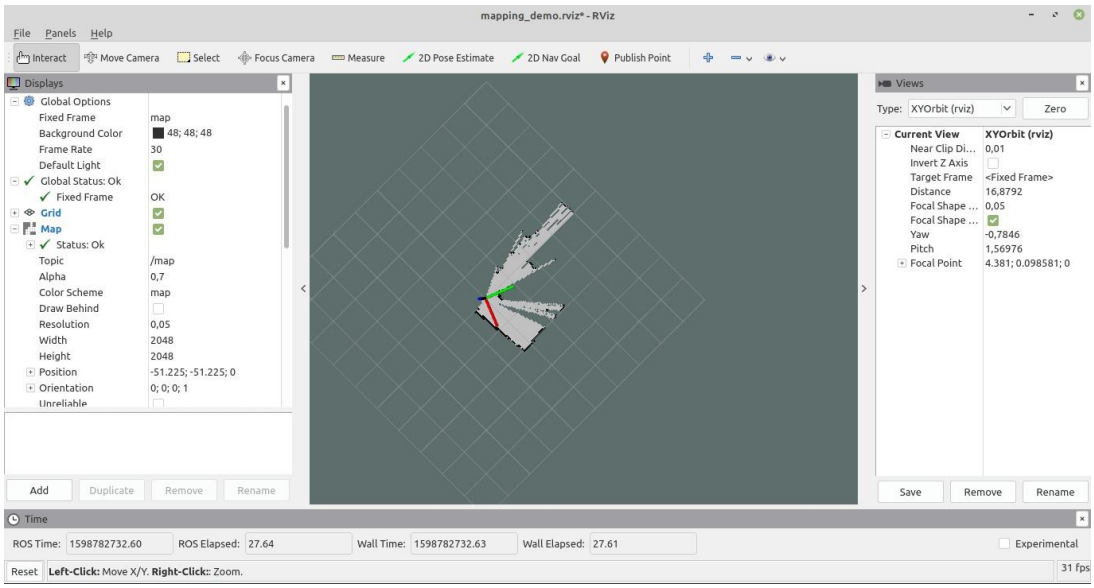

Figura 55 - Visor Hector SLAM

Esta es una captura que hicimos en la empresa tras hacer un breve recorrido. Además de mapear el entorno, también tiene la opción de mapear el recorrido que se ha seguido durante la ejecución del proyecto, que aparece representado con líneas verdes. Al tener el proyecto únicamente en el portátil con el lidar conectado al USB, necesitábamos desplazarnos sujetando tanto el portátil como el lidar.

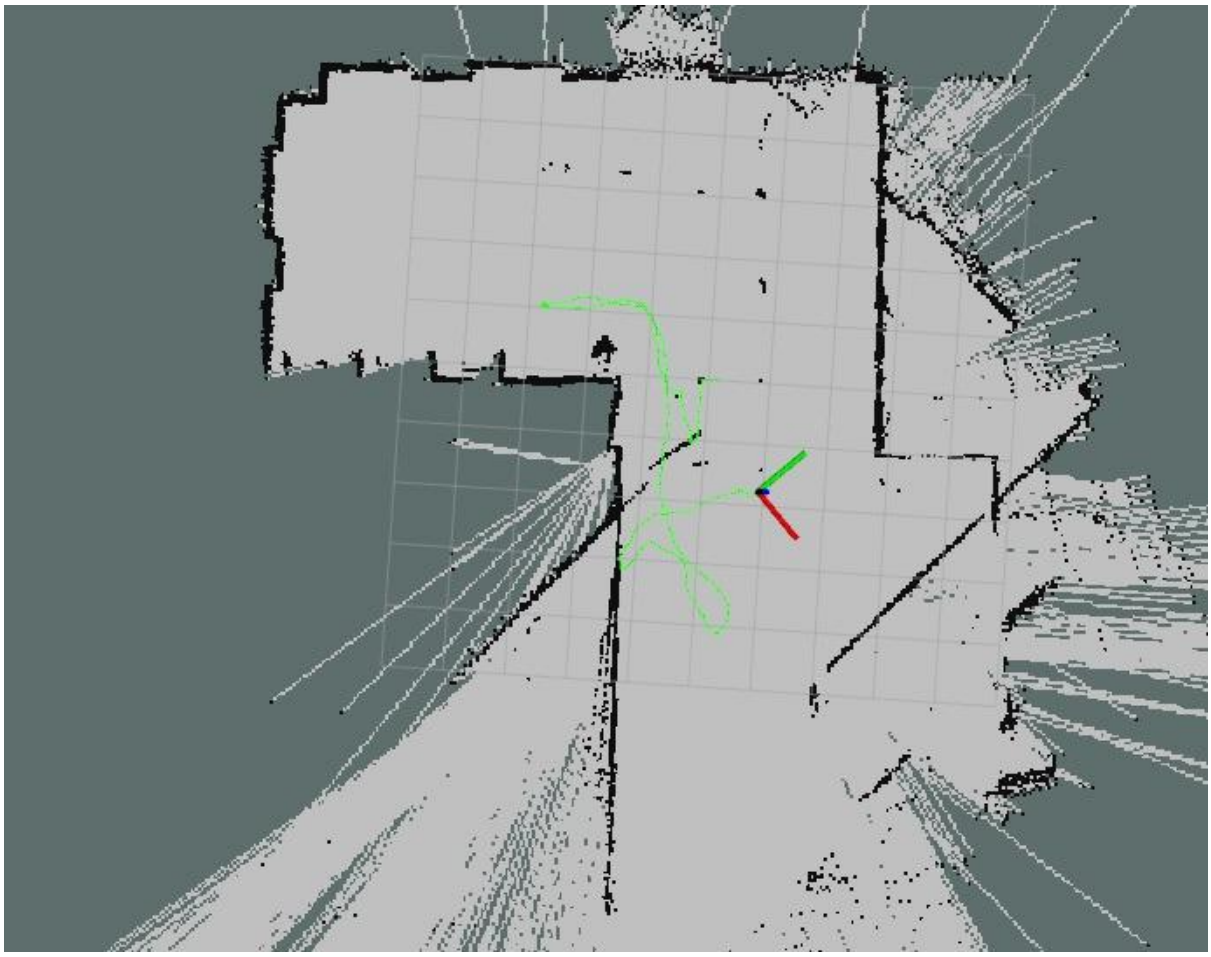

Figura 56 – Visor Hector SLAM Empresa

Las ventanas generan un cierto error (los rayos grises que salen hacia el exterior), debido a que el rayo lanzado por el lidar atraviesa el cristal y no detecta obstáculos.

Teniendo esto desarrollado, realizo una copia de seguridad de Linux, para no perder el proyecto en caso de que hubiera cualquier problema.

Figura 57 - Copia Seguridad 1

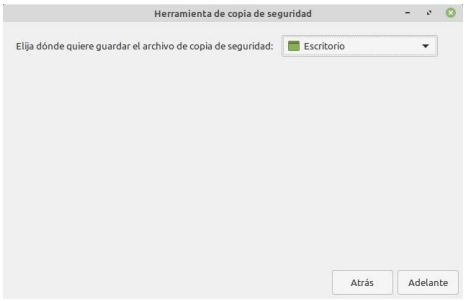

Figura 58 - Copia Seguridad 2

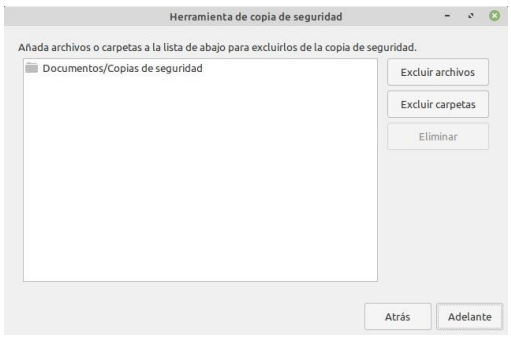

Figura 59 - Copia Seguridad 3

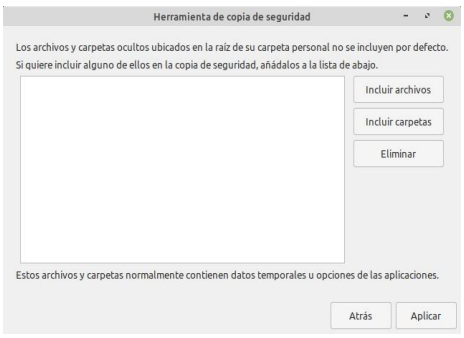

Figura 60 - Copia Seguridad 4

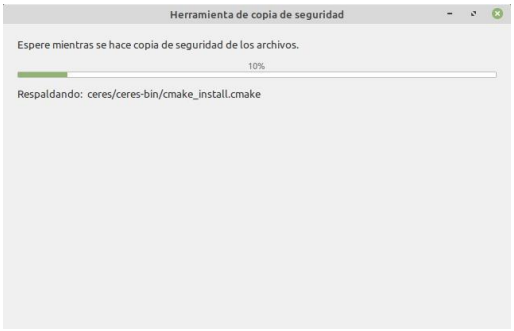

Figura 61 - Copia Seguridad 5

Hasta ahora el trabajo realizado era el funcionamiento del lidar en el ordenador. Sin embargo, no queremos tener el ordenador conectado al lidar y tener que ir persiguiendo al coche con el portátil en la mano, sino que el objetivo es usar un ordenador más pequeño que se encargue del procesamiento de la información recibida por el lidar.

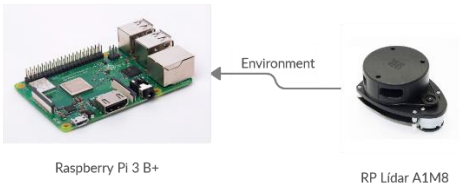

Figura 62 - Raspi - Lidar

Lo primero es instalar Linux en una tarjeta SD para añadirla a la Raspberry pi y tener el sistema operativo en funcionamiento.

El modelo de la Raspberry pi es Raspberry pi 3 model B+[25]. Encuentro en la página de Ubuntu un tutorial para instalar Ubuntu en Raspberry pi (la versión varía en función del modelo de Raspberry pi) e instalo ubuntu 18.04 en la tarjeta SD.

Una vez instalado Ubuntu en la SD, conecto la Raspberry pi a un monitor mediante con HDMI.

Ahora que tenemos la Raspberry pi con linux funcionando, toca instalar los paquetes de ros, catkin y hector SLAM para que el proyecto funcione de manera simular a como lo hacía en el ordenador.

Al trabajar en una Raspberry pi, hay que modificar algún archivo que en el ordenador no daba problemas. Por ejemplo, uno de los problemas que aparece al intentar descargar paquetes es:

```
Virtual memory exhausted: Cannot allocate memory
```

Corregimos esto modificando la configuración por defecto de swapsize que por defecto está a 100MB.

```
$ sudo nano /etc/dphys-swapfile
```

Y modificamos el valor de `CONF_SWAPSIZE` por 1024, para que tenga 1GB de capacidad.

A continuación instalamos los paquetes de ROS.

```
$ sudo apt install ros-melodic-ros-base
$ sudo apt install ros-melodic-robot-state-publisher
$ sudo rosdep init
```

```
$ rosdep update
$ source /opt/ros/melodic/setup.bash
$ roscore
```

Y creamos el workspace para Catkin.

```
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws
$ catkin_make
```

Instalamos los paquetes de ros melodic.

```
$ sudo apt install ros-melodic-rviz
$ sudo apt install ros-melodic-rqt
$ sudo apt install ros-melodic-rplidar
```

Y en la carpeta src del workspace de catkin añadimos los paquetes de rplidar de ros. Al tener linux recién instalado hace falta instalar git.

```
$ sudo apt install git
$ cd catkin_ws/src/
$ git clone https://github.com/robopeak/rplidar_ros.git
```

Una vez añadido el paquete de rplidar, hacemos build del workspace.

```
$ cd ~/catkin_ws
$ catkin_make
```

Finalmente, añadimos los paquetes de hector SLAM y los compilamos.

```
$ cd ~/catkin_ws/src
$ git clone https://github.com/NickL77/RPLidar_Hector_SLAM.git
$ cd ..
$ catkin_make
```

Llegados a este punto, ya tenemos el proyecto instalado, solo falta comprobar que funciona. Para ello seguimos los comandos como lo haríamos en el ordenador:

```
// Lanzamos roscore
$ cd catkin_ws
$ source devel/setup.bash
```

```
$ sudo chmod 666 /dev/ttyUSB0
$ roscore

// Lanzamos rplidar
$ cd catkin_ws
$ source devel/setup.bash
$ sudo chmod 666 /dev/ttyUSB0
$ roslaunch rplidar_ros view_rplidar.launch

// Hasta aquí funciona todo correctamente.
// Lanzamos hector SLAM
$ cd catkin_ws
$ source devel/setup.bash
$ sudo chmod 666 /dev/ttyUSB0
$ roslaunch hector_slam_launch tutorial.launch
```

Dependiendo de la potencia suministrada a la Raspberry Pi, podríamos encontrar este error:

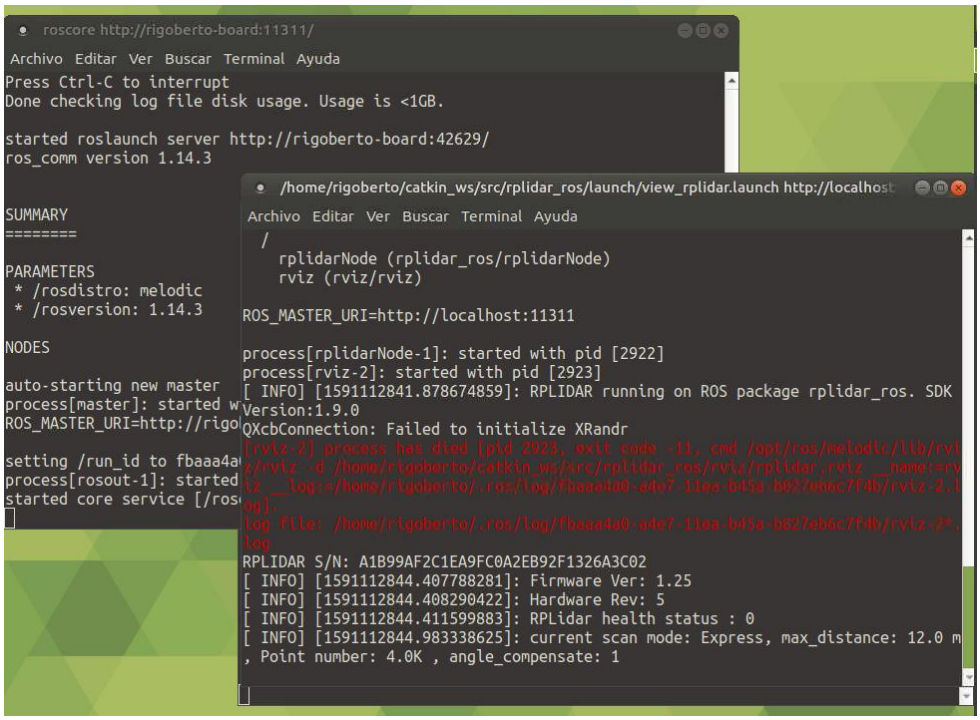

Figura 63 - Hector SLAM Error

```
[rviz-2] process has died [pid 2923, exit code -11, cmd
/opt/ros/melodic/lib/rviz/rviz -d
/home/rigoberto/catkin_ws/src/rplidar_ros/rviz/rplidar.rviz __name:=rviz
```

```
__log:=/home/rigoberto/.ros/log/fbaaa4a0-a4e7-11ea-b45a-b827eb6c7f4b/rviz-
2.log].
log file: /home/rigoberto/.ros/log/fbaaa4a0-a4e7-11ea-b45a-
b827eb6c7f4b/rviz-2*.log
```

Tras investigar el error y mirar los registros del log que aparece en el error, nos damos cuenta de que no es un problema en la instalación de Hector SLAM, sino que el error se debe a problemas de potencia. Lanzamos el comando `dmesg` y encontramos que hay problemas de voltaje.

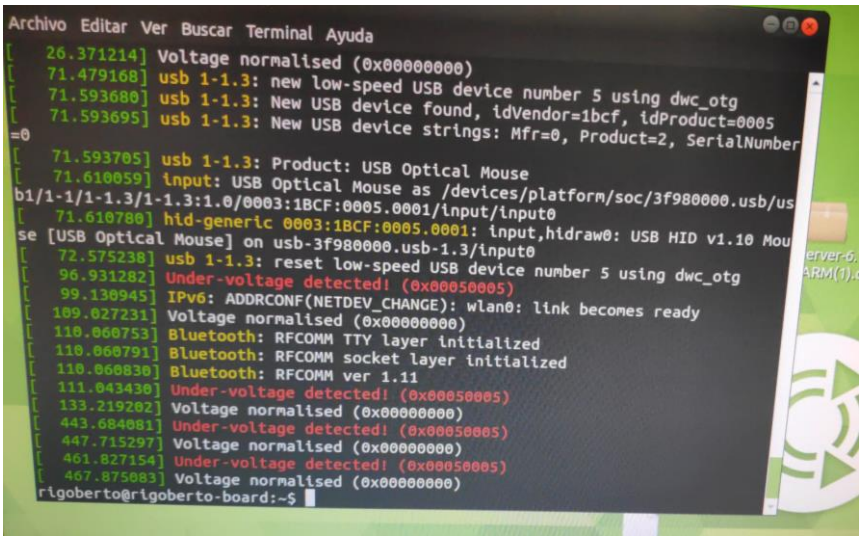

Figura 64 - Under-voltage detected!

Hasta ahora estábamos conectando la Raspberry pi al cargador del móvil, con una intensidad de 2.5A. Tras múltiples pruebas concluimos en que necesitamos una fuente de alimentación de 3A para evitar este problema.

Además, debido a la incomodidad para trabajar en la Raspberry pi utilizando un ratón y el teclado en pantalla, decido instalar Teamviewer para conectarme remotamente. Sin embargo, me encuentro con problemas de rendimiento ya que el procesador de la Raspberry pi no es capaz de conectarse remotamente en tiempo real mientras, a su vez, hace funcionar el lidar, lee los datos y los representa en la interfaz gráfica.

Pruebo a instalar VNC Server y en el portátil instalo VNC Viewer.

```
$ sudo apt-get install realvnc-vnc-server
// Lo ejecuto con el comando
$ vncserver
```

Ya que tanto el ordenador como la Raspberry pi, generalmente, van a estar conectados a la misma red, decidimos conectar la Raspberry pi al ordenador mediante ssh.

Para ello llevamos a cabo los siguientes comandos:

```
// Comprobamos la ip del portátil y de la Raspberry pi
$ ifconfig
// Accedemos a la configuración de /etc/hosts
$ sudo nano /etc/hosts
// Accedemos a la configuración de /etc/hosts
$ sudo cat /etc/hosts
```

[44][45]

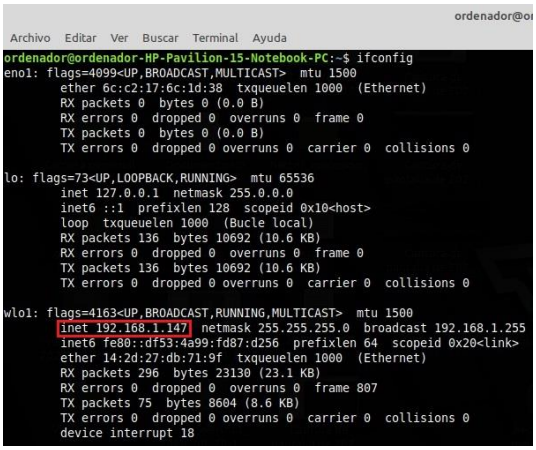

Figura 65 - ifconfig portátil

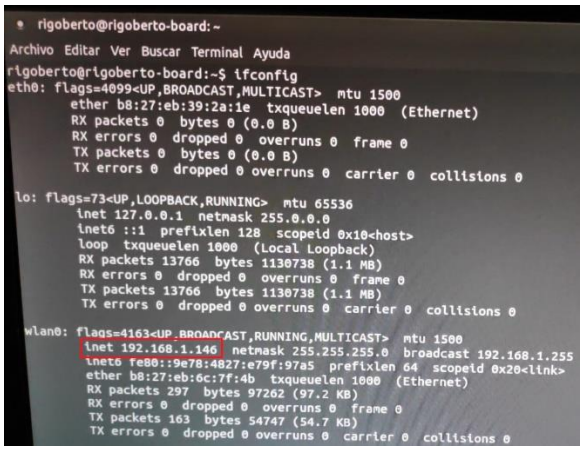

Figura 66 - ifconfig raspi

En el ordenador: añado la ip de la Raspbery pi y le asigno un nombre.

```
192.168.1.146       raspi
```

En la Raspberry pi: añado la ip del ordenador y le asigno un nombre.

```
192.168.1.147       portátil
```

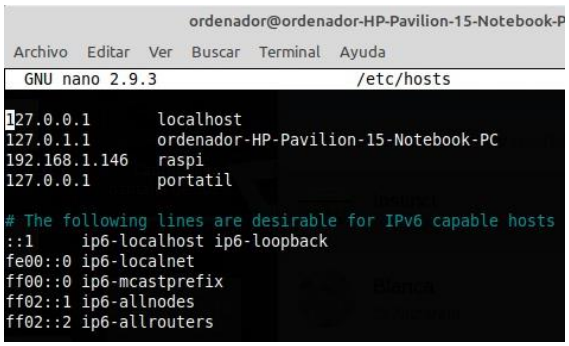

Figura 67 – nano /etc/hosts portátil

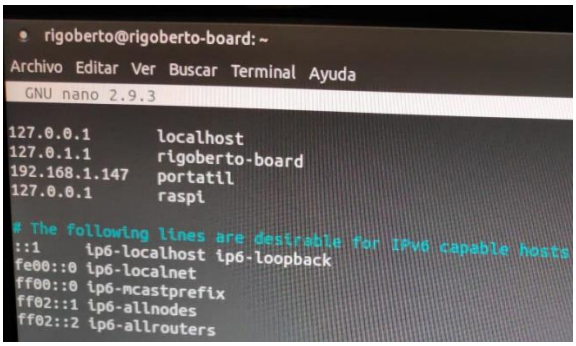

Figura 68 - nano /etc/hosts raspi

Quedando así:

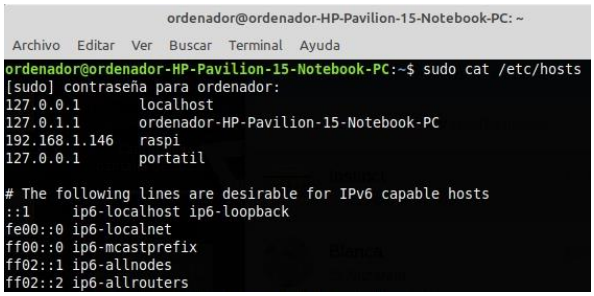

Figura 69 - cat /etc/hosts portátil

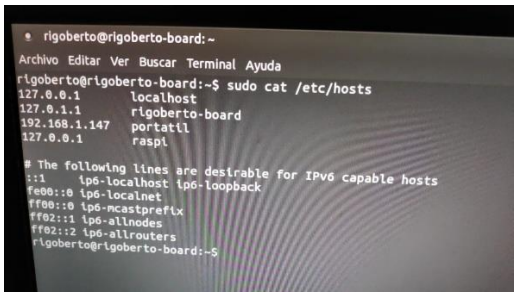

Figura 70 - cat /etc/hosts raspi

De este modo, siempre que estén conectados a la misma red (y la IP no cambie, en caso de que cambiara habría que volver a modificar el archivo /etc/hosts con la nueva IP), podré conectarlos entre ellos mediante el comando ssh.

Lo compruebo en ambos sentidos mediante el comando ping ip. Al haber asignado un nombre a cada uno de ellos, puedo hacer ping nombre.

```
// En el portátil
$ ping raspi
// En la Raspberry pi
$ ping portatil
```

Y veo si los paquetes llegan correctamente al destino.

Una vez tengo esto, puedo conectarme de un dispositivo a otro mediante ssh. Como me interesa conectarme a la Raspberry pi desde el ordenador, uso el comando:

```
// Se usa el nombre del dispositivo (el cual se puede ver mediante el comando whoami) seguido de @ip o @nombre.
$ ssh rigoberto@raspi
```

Al hacer esto la consola pasa a ser la del dispositivo al que me he conectado, permitiéndome usar ambas consolas (tanto la consola del ordenador como la de la Raspberry pi) desde el ordenador.

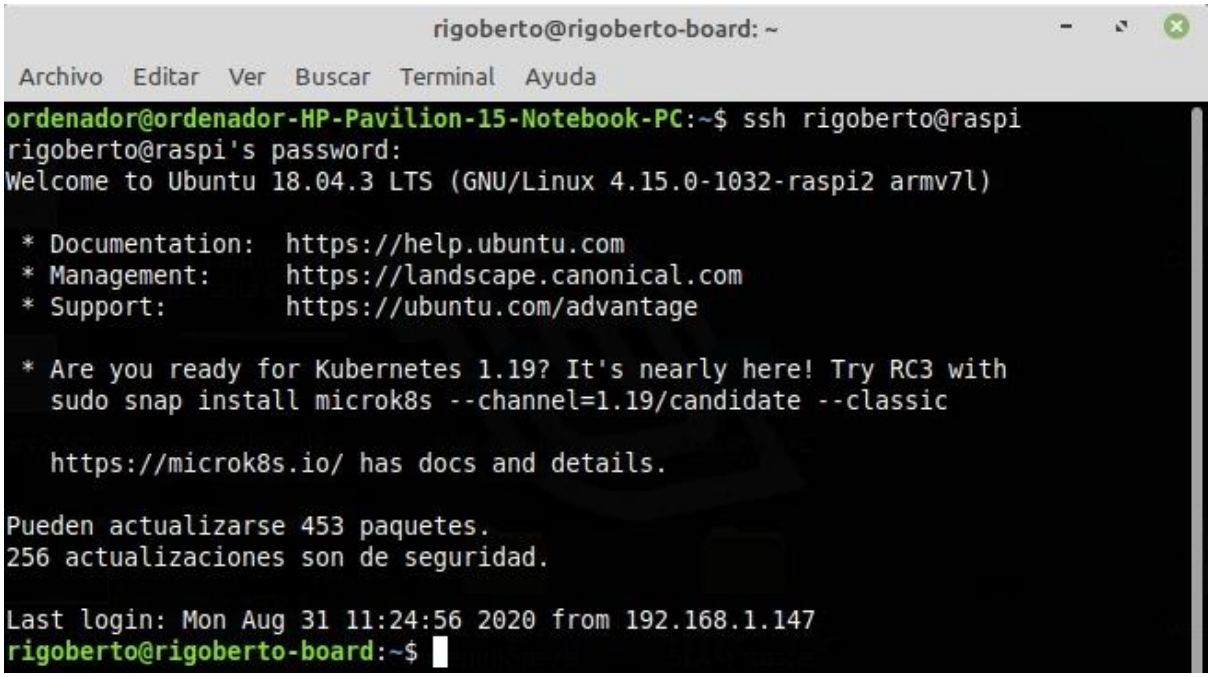

Figura 71 - ssh rigoberto@raspi

Una vez somos capaces de conectar la Raspberry pi al ordenador, nos interesa que los datos se obtengan en la Raspberry pi (que es donde va a estar conectado el lidar), pero que el procesamiento de esos datos se lleve a cabo en el portátil, para no sobrecargar de trabajo a la Raspberry pi.
Investigamos y encontramos la opción de conectar ROS con múltiples dispositivos. Para ello una vez conectamos los dispositivos mediante ssh, necesitamos asignar cuál va a ser el Master y cuál va a transmitir la información, así como definir el IP de cada uno de ellos.[46][47][48]

```
Portátil
$ export ROS_MASTER_URI=http://portatil:11311
$ export ROS_IP=portatil
Raspberry pi
$ export ROS_MASTER_URI=http://portatil:11311
$ export ROS_IP=raspi
```

De este modo conectamos los dispositivos para que los datos se recojan en la Raspberry pi pero sean procesados en el ordenador. Al hacer esto, no es necesario lanzar el comando `roscore` en la raspberry pi, pero sí es necesario lanzarlo en el ordenador ya que es el que va a actuar como ROS_MASTER.

Por tanto en lugar de lanzar los comandos como los lanzábamos hasta ahora, lo haremos del siguiente modo:

```
Portátil
$ export ROS_MASTER_URI=http://portatil:11311
$ export ROS_IP=portatil
Raspberry pi
$ export ROS_MASTER_URI=http://portatil:11311
$ export ROS_IP=raspi
Portátil
$ cd tfg
// Cargamos la configuración
$ source devel/setup.bash
// Añadimos & para no tener que abrir una nueva consola, para que cargue en segundo plano.
$ roscore &
Raspberry pi
$ cd catkin_ws
$ sudo chmod 666 /dev/ttyUSB0
$ source devel/setup.bash
// Lanzamos rplidar.launch para que ponga en funcionamiento el lidar y empiece a recoger datos
$ roslaunch rplidar_ros rplidar.launch
Portátil
// Una vez está funcionando el lidar, podemos comprobar que recibimos correctamente los puntos que encuentra con el comando rostopic echo /scan
// Lanzamos el visor de rplidar (este paso no es necesario).
$ roslaunch rplidar_ros view_rplidar_dist.launch
// Lanzamos hector SLAM
$ roslaunch hector_slam_launch tutorial.launch
```

De este modo estamos capturando la información con el lidar conectado a la Raspberry pi y desde la Raspberry pi enviando los datos al ordenador para que sea en este en el que abramos la interfaz gráfica para ver el mapa que se forma en función de los datos recibidos.

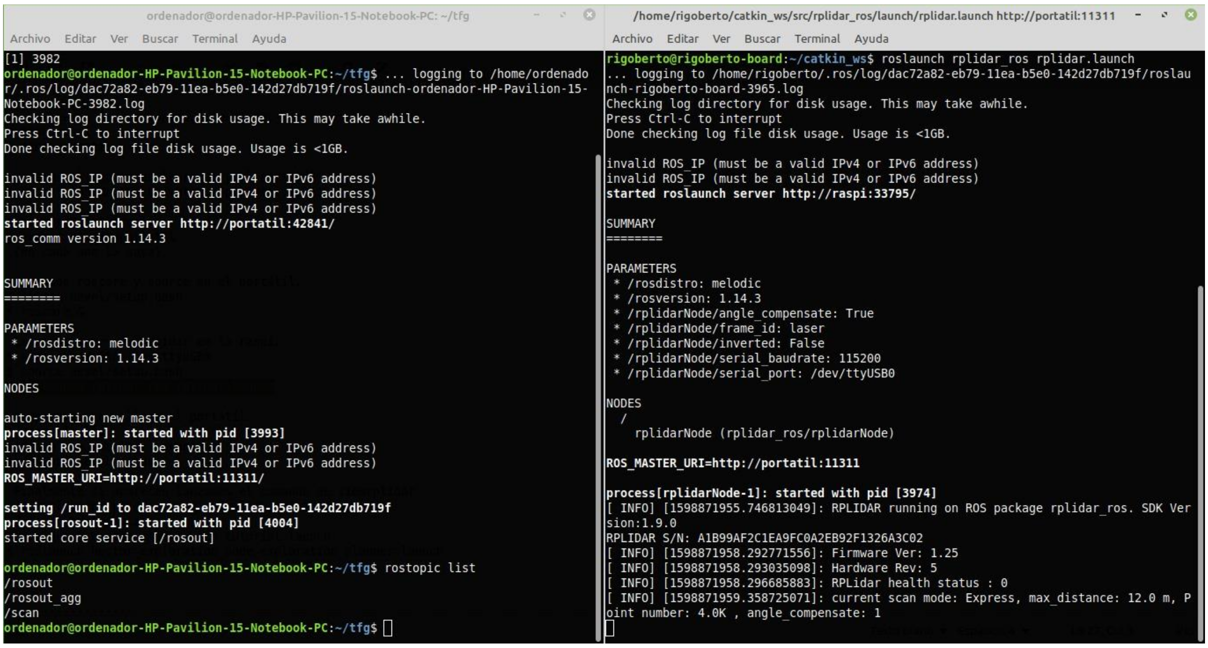

Figura 72 - Portátil – Raspi

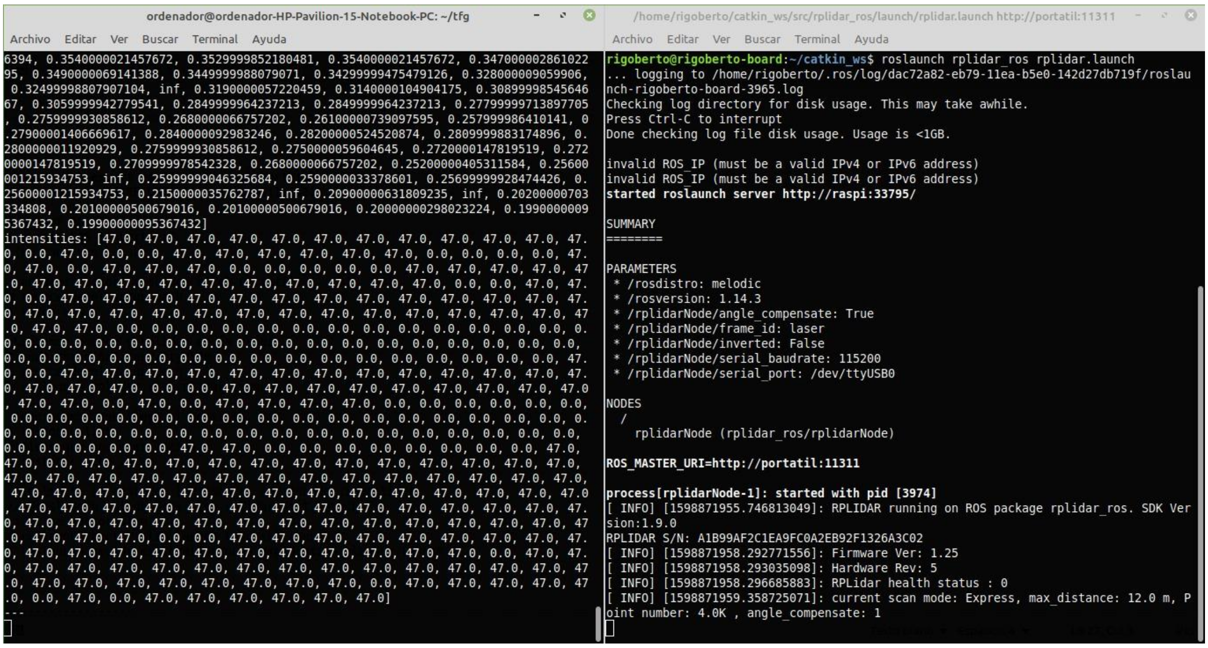

Figura 73 - rostopic echo /scan

El único inconveniente es que cada vez que la IP del ordenador o de la Raspberry pi cambie (generalmente al cambiar de red wifi), habrá que volver a modificar /etc/hosts para actualizar las IPs. Dejamos para más adelante el hacer las IPs (tanto del ordenador como de la Raspberry pi) fijas, para que aunque cambiemos de red, no haya que realizar modificaciones.

En este momento, tenemos una versión en la que somos capaces de conectar el lidar a la Raspberry pi, recibir los datos y, mediante SSH, estando conectados el portátil y la Raspberry pi a la misma red, transmitirlos al portátil para que sea este quien se encargue de procesarlos y representarlos en la interfaz gráfica.

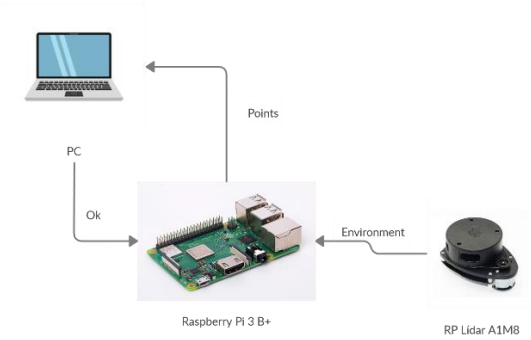

Figura 74 - Portátil - Raspi - Lidar

Ya que tenemos una versión avanzada de esta parte del proyecto, hacemos una copia de seguridad, tanto del ordenador como de la Raspberry pi.
En el caso del ordenador, repetimos los pasos que llevamos a cabo en la última copia de seguridad. Sin embargo, en la Raspberry pi, al estar la información en la tarjeta SD, hacemos la copia de seguridad de otro modo.

Saco la tarjeta micro SD y la pongo en el lector de SD del portátil mediante un adaptador. Instalo en la partición de Windows el programa Win32 Disk Imager. Después, simplemente elijo el nombre de disco de la tarjeta SD y el nombre y ubicación del archivo que se creará al hacer la copia de seguridad. De este modo, obtengo un archivo *.iso* para que, en caso de que haya cualquier problema con la micro SD de la Raspberry pi, pueda recuperar el estado actual instalando el *iso* en una nueva tarjeta SD.[49][50]

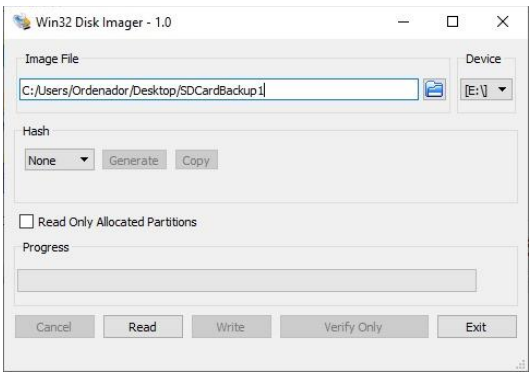

Figura 75 - Win32DiskImager SD Backup

Acudimos a la empresa para comprobar que todo funciona correctamente y probamos a ejecutar el proyecto conectando el lidar a la Raspberry pi y enchufando la Raspberry pi con un alargador para poder desplazarnos lo más libremente posible.

Colocamos la Raspberry pi con el lidar sobre una silla y la desplazamos por la empresa varios metros mientras recibimos toda la información en el ordenador, permitiéndonos ver el entorno por el que nos movemos sin necesidad de estar delante, a pesar de haber tenido que ir moviendo la silla.

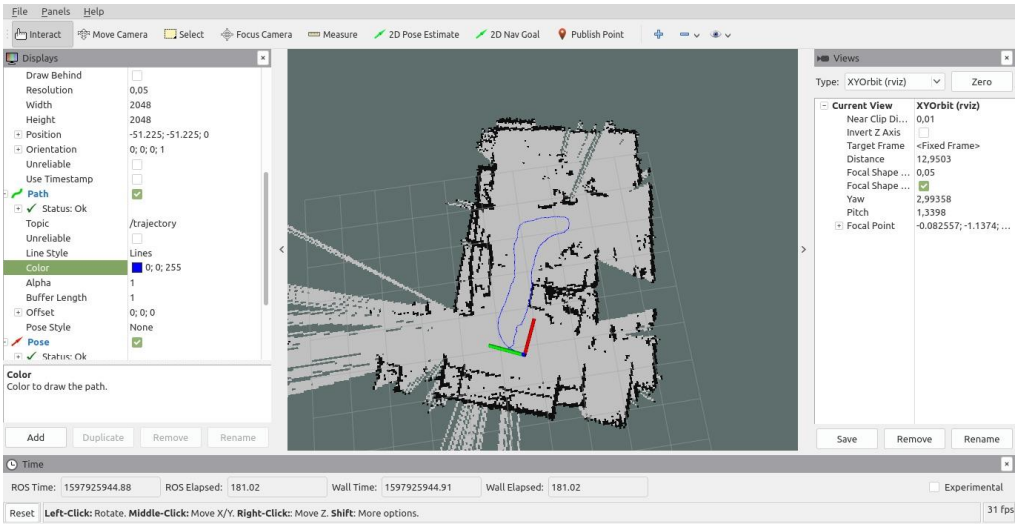

Figura 76 - Mapeo empresa

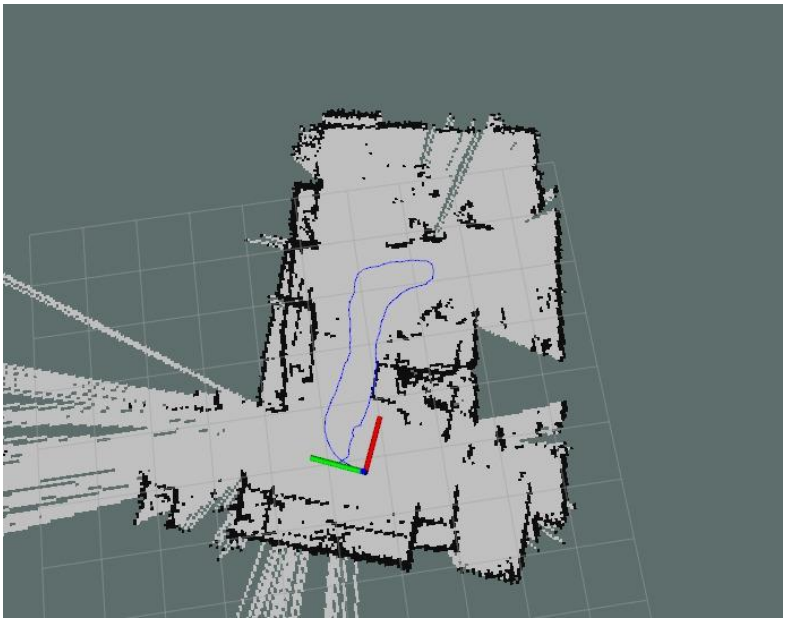

Figura 77 - Imagen Mapeo Empresa

Para la versión real del proyecto, necesitaríamos conectar la Raspberry pi a una batería recargable capaz de proporcionarle al dispositivo la potencia necesaria para funcionar correctamente, evitando los problemas de bajo voltaje que conseguimos solucionar con la fuente de alimentación que sustituyó al cargador del móvil.

Tras esto, comenzamos a trabajar con el coche mediante la app de Elego car (Elegoo BLE Tool (For Android))[29]. Para ello, creo un mando para poder controlar el coche desde el móvil, permitiéndome estar sentado en el ordenador y, mientras muevo el coche, ver el mapeado del entorno que realiza la Raspberry Pi con el lidar sobre el vehículo.

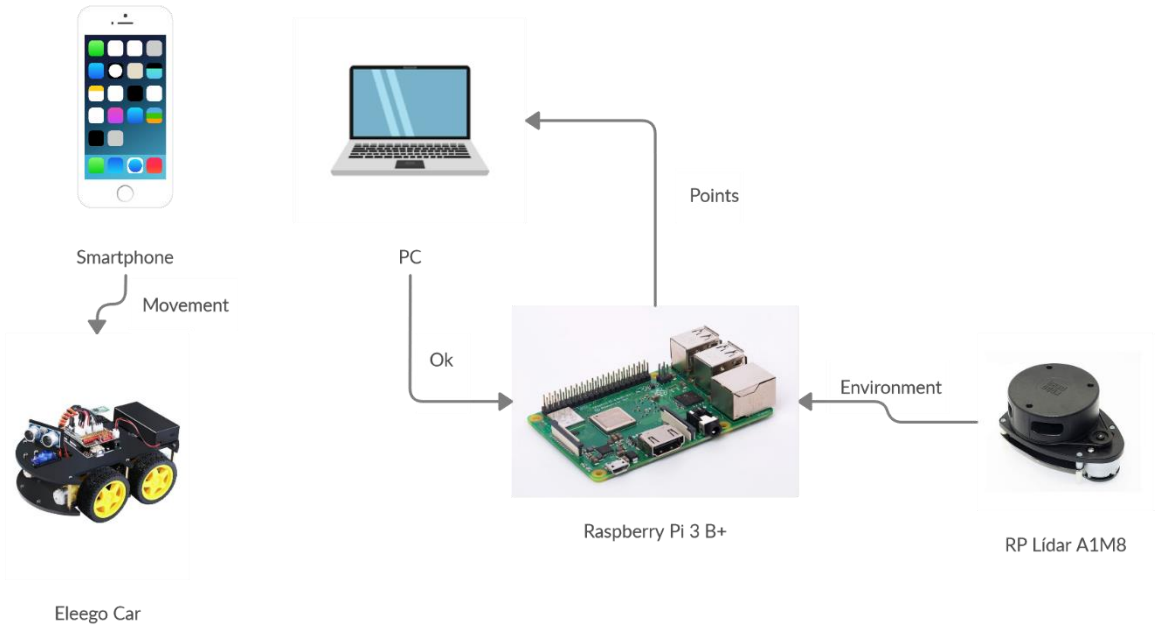

Figura 78 - ROS + Coche

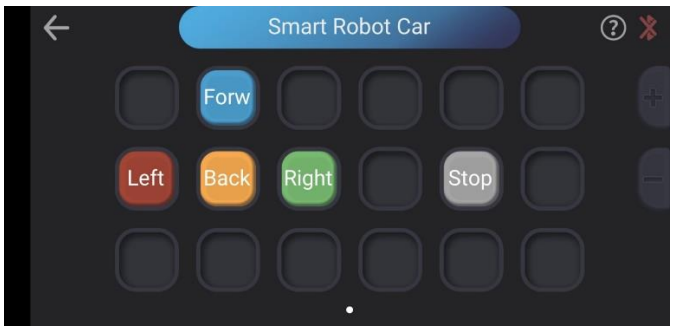

Figura 79 - Mando Coche

Figura 80 - Mando Forward-Left-Right

Figura 81 - Mando Back-Stop

Finalmente encontramos paquetes de ROS para crear un costmap que permite representar los obstáculos, así como asignar un cierto radio para evitar los obstáculos de modo que siempre evitemos que el robot colisione se acerque al radio asignado para evitar de la manera más fiable posible las colisiones con los obstáculos.

Figura 82 – Costmap

En la figura anterior, los puntos rojos representan los obstáculos, el polígono rojo representa la forma del robot y los puntos azules representan los obstáculos a los que se les ha añadido el radio inscrito del robot. De este modo, debemos evitar en todo momento que los bordes del polígono del robot colisionen con los puntos rojos, y que el centro del polígono del robot colisione con los puntos azules, asegurándonos así de que no haya colisiones entre el robot y cualquier obstáculo. [51]

Instalamos las librerías de Hector Navigation en nuestro workspace de catkin del portátil.[52][53]

```
// Accedemos a la carpeta src del Workspace
$ cd tfg/src
// Descargamos el paquete de la rama específica catkin [54]
$ git clone -b catkin https://github.com/tu-darmstadt-ros-pkg/hector_navigation.git
// Una vez lo tenemos, compilamos.
$ cd ~/tfg
$ catkin_make
```

Al compilarlo, nos encontramos con algunas dependencias que son necesarias instalar, pero no estaban en el paquete.

Instalamos:

```
// Costmap 2d [51]
$ sudo apt install ros-melodic-costmap-2d
$ catkin_make
// geodesy [55]
$ sudo apt install ros-melodic-geodesy
$ catkin_make
```

Nos pide instalar ceres, pero necesitamos buscar una guía de cómo instalarlo. [56]

Primero necesitamos instalar algunas dependencias para poder continuar con ceres.

```
$ sudo apt-get install cmake
$ sudo apt-get update
$ sudo apt-get install libgoogle-glog-dev
$ sudo apt-get install libatlas-base-dev
$ sudo apt-get install libeigen3-dev
$ sudo apt-get install libsuitesparse-dev
```

Una vez las hemos instalado, podemos compilar e instalar ceres

```
$ mkdir ~/ceres
$ cd ~/ceres
$ ls
$ tar zxf ceres-solver-1.14.0.tar.gz
$ mkdir ceres-bin
$ cd ceres-bin/
$ cmake ../ceres-solver-1.14.0
$ make -j3
```

```
$ sudo make install
```

De nuevo, compilamos el workspace para ver si todo funciona correctamente

```
$ cd ~/tfg
$ catkin_make
```

Seguimos necesitando más dependencias, por lo que seguimos añadiéndolas a nuestro Workspace y compilando.

```
$ cd ~/tfg/src
// ceres_catkin [57]
$ git clone https://github.com/ethz-asl/ceres_catkin.git
$ cd ..
$ catkin_make
$ cd src
// ceres_catkin [58]
$ git clone https://github.com/catkin/catkin_simple.git
$ cd ..
$ catkin_make
// nav-core [59]
$ sudo apt install ros-melodic-nav-core
$ catkin_make
$ cd src
$ git clone https://github.com/ethz-asl/glog_catkin.git
$ git clone https://github.com/ethz-asl/gflags_catkin.git
$ git clone -b feature/depend_on_libsuitesparse-dev https://github.com/ethz-asl/suitesparse.git
$ cd ..
$ catkin_make
$ sudo apt-get install autoconf
$ catkin_make
$ sudo apt-get install autoconf
$ catkin_make
$ autoreconf
$ sudo apt install automake
$ catkin_make
$ sudo apt-get install libtool libtool-bin
$ catkin_make
```

Llegados a este punto, encontramos errores de incompatibilidad entre tf y la versión de ROS que estamos utilizando (ros-melodic), ya que está obsoleto, por lo que necesitamos migrarlo a tf2 manualmente. [60]

Instalamos el paquete de tf2 y manualmente, modificamos el código de los archivos del proyecto conforme van apareciendo los errores al compilar el workspace.

```
$ sudo apt install ros-melodic-tf2-sensor-msgs
```

Finalmente, conseguimos que el `catkin_make` funcione, sin dar errores de compilación, permitiéndonos lanzar el proyecto con la nueva funcionalidad del costmap. Para ello, tras lanzar el comando

```
$ roslaunch hector_slam_launch tutorial.launch
```

Lanzamos en una nueva ventana

```
$ source devel/setup.bash
$ roslaunch hector_exploration_node exploration_planner.launch
```

Después, en el entorno gráfico, duplicamos el mapa `map` para en lugar de hacer un `/map` hagamos `/hector_exploration_node/costmap` .

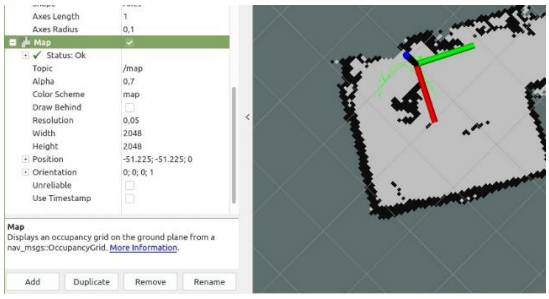

Figura 83 – /map

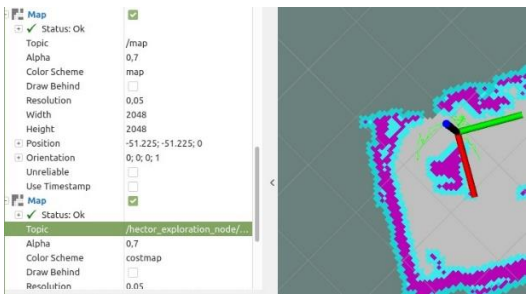

Figura 84 - /costmap

También podemos modificar el `Color Scheme` por `costmap` para ver el mapa los colores representando los obstáculos y el radio añadido para evitar colisiones.

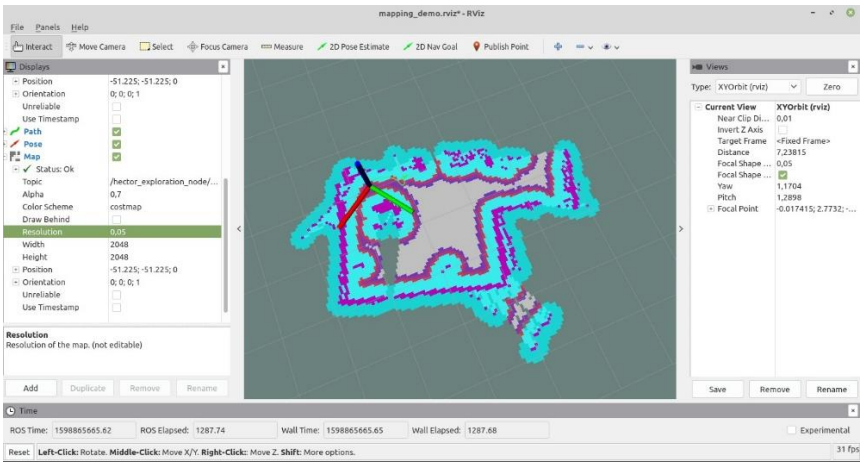

Figura 85 - Costmap 1

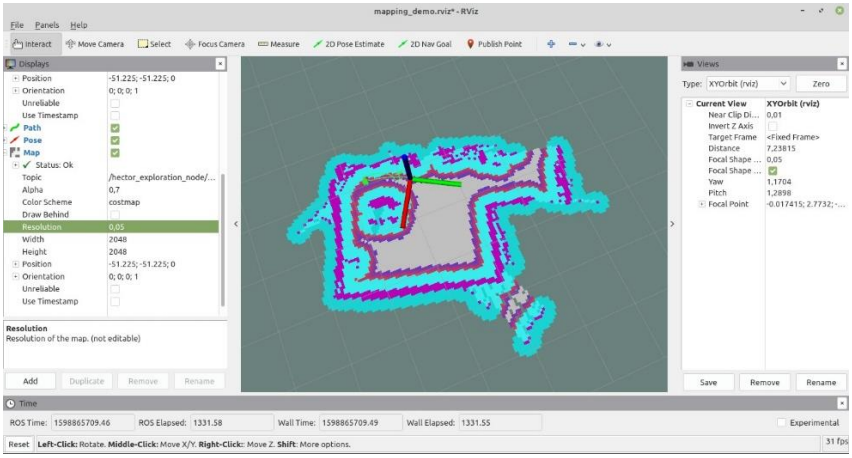

Figura 86 - Costmap 2

Finalmente, modificamos los archivos de `exploration_controller` ya que desde ahí se llama a `get_exploration_path` que es el servicio que publica el `exploration_node`. Este servicio invoca a `ExplorationServiceCallback`, que llama a la función `doExploration` que es donde se calculan los path.

Para lanzar esta nueva funcionalidad, creamos una nueva pestaña en la consola y lanzamos los comandos:

```
$ source devel/setup.bash
$ roslaunch hector_exploration_controller exploration_controller.launch
```

Al tratar de lanzarlo por defecto podemos encontrarnos con el siguiente error en la consola:

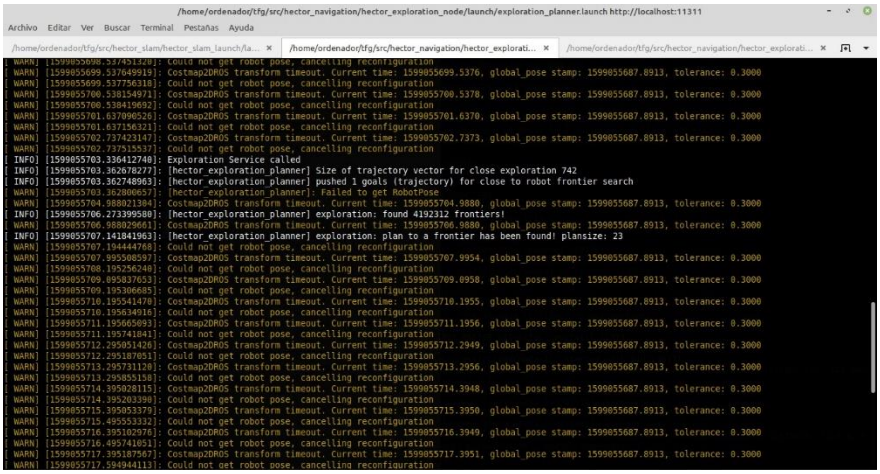

Figura 87 - eploration controller error

Este error lo solucionamos aumentando el valor de la variable `transform_tolerance` en el archivo `costmap.yaml`.
Con esto conseguimos solucionar el error y lanzar correctamente el `exploration_controller`.

Tras lanzarlo, al igual que hacíamos con el `costmap`, duplicamos esta vez el `path` y lo modificamos para que en lugar de usar `/trajectory` para representar el recorrido por donde nos hemos desplazado, use `/exploration_path` representando así el siguiente punto al que debería desplazarse el robot para poder seguir descubriendo el entorno que no se ha conseguido mapear todavía.

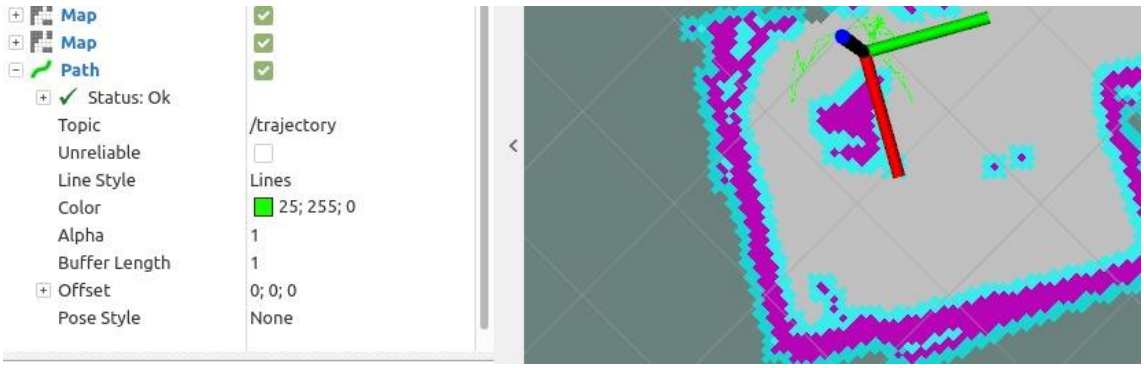

Figura 88 - /trajectory

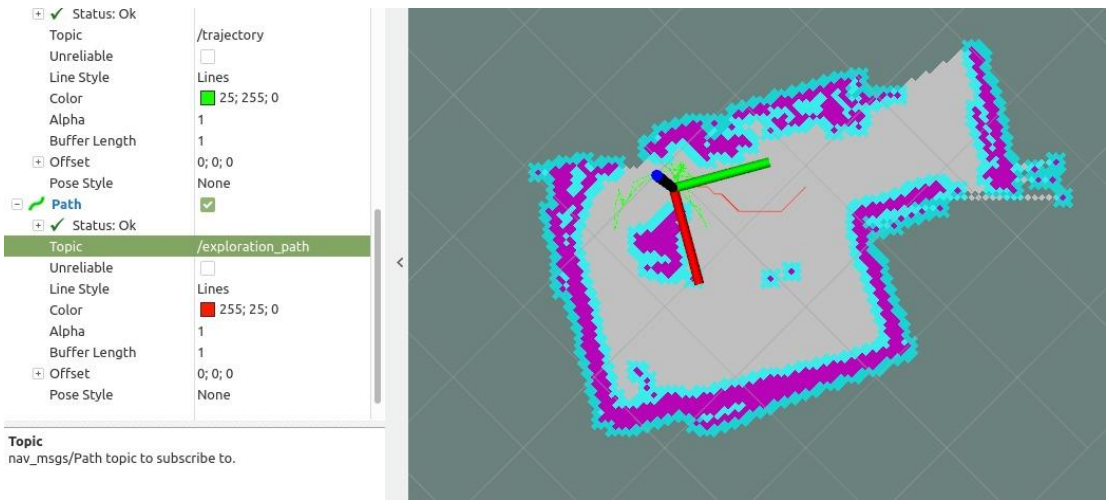

Figura 89 - /exploration_path

También modificamos el color de `topic` por un color diferente (por ejemplo: rojo o azul), para evitar confundirnos con el color del recorrido que ha realizado el robot, ya que al duplicarlo por defecto se queda del mismo color.

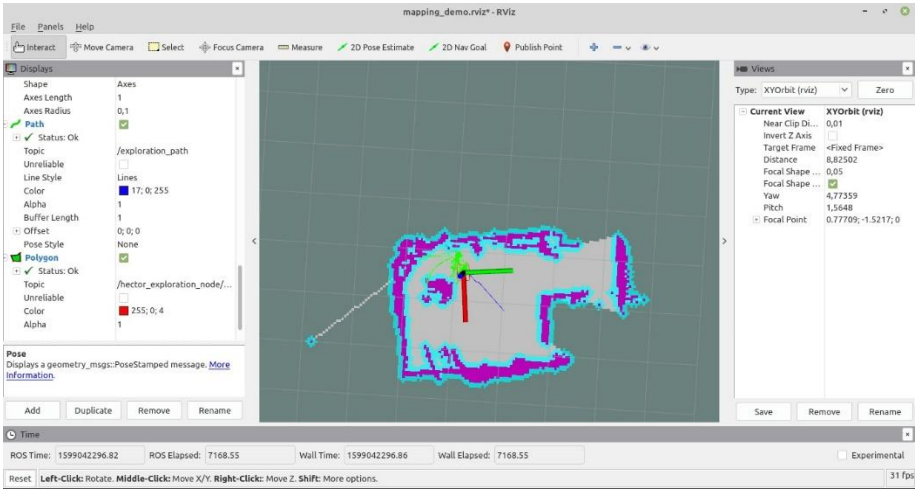

Figura 90 - Exploration path 1

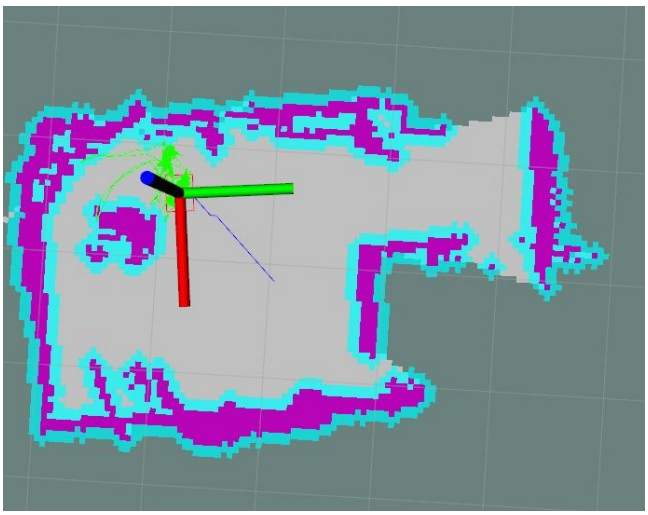

Figura 91 – Exploration path 2

Finalmente, para concluir el proyecto, realizo una copia de seguridad del estado del ordenador para evitar cualquier problema que pudiera surgir ya que esta es la última versión funcional del proyecto de la que disponemos.

No es necesario realizar una copia de seguridad de la tarjeta SD de la Raspberry Pi, ya que no ha sido necesario realizar más modificaciones desde la última copia de seguridad realizada.

Estudio Económico

El proyecto, hasta el momento ha requerido un total aproximado de unas 320 horas de investigación y desarrollo.

Teniendo en cuenta que el sueldo bruto anual de un analista/programador es de 27,543€ al año [61] y que se trabajan unas 1764 horas anuales (para contratos de 39h semanales)[62], podemos ver que el coste aproximado por hora sería de unos 15.61€, por lo que el coste del empleado (*sueldo bruto*) de este proyecto serían unos 4,996.46€.

Teniendo en cuenta que el sueldo bruto del trabajador para este proyecto son 4,996.46€, deberemos añadir el coste de la seguridad social que debe pagar la empresa. La seguridad social supone un coste añadido del 28% del sueldo del trabajador, lo cual se comparte entre el trabajador y la empresa, siendo la empresa la que asume un 23% del pago.
Por tanto, haciendo los cálculos para aplicar ese 23% al sueldo bruto del trabajador (4,996.46€), vemos que la empresa debe pagar 1,149.19€ extra, teniendo como resultado un coste total para la empresa de 6,145.65€.[63]

No sería necesario añadir gastos de software ya que ha sido suficiente con el uso de la licencia gratuita de Visual Studio, y el sistema operativo que se ha empleado ha sido Linux.

Por otra parte, sí que necesitamos añadir gastos de Hardware ya que hay varios dispositivos que han sido necesarios para el proyecto.

Gastos fijos:
Raspberry Pi 3 model b+: 34.79£ → 39.13€
LIDAR: 99 USD → 83.39€
Elegoo Smart Robot Car Kit V3.0: 75€
Arduino UNO: 20€

Lo que suma un gasto fijo en Hardware de 217.52

Gastos variables:
Fuente de alimentación (Raspberry Pi): Mean Well MDR 20-5 – 12.74€
Monitor (Raspberry Pi): LG Flatdron M2080D-PZ - 67.89€
Teclado (Raspberry Pi): Mars Gaming MK4 – 34.90€
Ratón (Raspberry Pi): Mars Gaming MRM0 – 7.90€
Tarjeta Micro SD (Raspberry Pi): Samsung EVO Plus 32GB – 10.99€
Smartphone: Xiaomi Pocophone f1 - 302€

Lo que suma un gasto variable de Hardware de 436.42€.
(El hardware variable con el que se han realizado los cálculos, es el que se ha usado a lo largo del proyecto, que podría ser sustituido por otro de mayor o menor precio.)

Por tanto, teniendo en cuenta los gastos de investigación y desarrollo, así como los gastos de Hardware fijos, el proyecto ha tenido un coste total de 6,363.17€.
A lo que deberíamos añadir el coste de Hardware variable, que en este caso ha sido un coste de 436.42€. Lo que suma un coste total de 6,799.59€.

En cuanto a los beneficios del proyecto, al tratarse de un proyecto de investigación y no de un proyecto desarrollado por petición de un cliente, no generaría beneficios de forma inmediata. Sin embargo, el proyecto podría ser aprovechado de cara a futuros proyectos relacionados con cualquiera de las implementaciones desarrolladas aquí, de los cuales sí que sería posible obtener beneficio económico.

Resultados

He dividido este apartado en dos puntos: (1) errores, en el que hablaré de los errores con los que nos hemos ido encontrado a lo largo del proyecto; y (2) resultados, en el que hablaré de los resultados obtenidos.

Errores

ROS – Catkin – RPLidar: Durante la primera implementación, instalando ROS y Catkin en el ordenador, fuimos realizando los tutoriales que encontramos tras una investigación. Encontrábamos errores en la consola que éramos capaces de solucionar leyendo los motivos de los errores que, generalmente, eran debido a faltas de librerías, ya que estábamos trabajando en una nueva instalación de Linux y necesitábamos hacer `sudo apt-get install` de los paquetes que se nos solicitaban en consola.

Hector_SLAM: Tras testear la funcionalidad, observamos los errores que se producen debido superficies reflectantes, como pantallas, espejos o cristales.
Las pantallas, por ejemplo, debido al "efecto espejo" que producen por la reflexión de la luz, no se detectan como obstáculos. Los espejos hacen que los obstáculos se detecten en otra posición y las ventanas hacen que los haces del láser a veces las traviesen generando proyecciones "infinitas" (hasta los 12 metros si no detecta obstáculos) o hasta que colisionan con algún obstáculo al otro lado del cristal o bien que no atraviesen las ventanas y colisionen con ellas haciendo que sean detectadas como obstáculos.

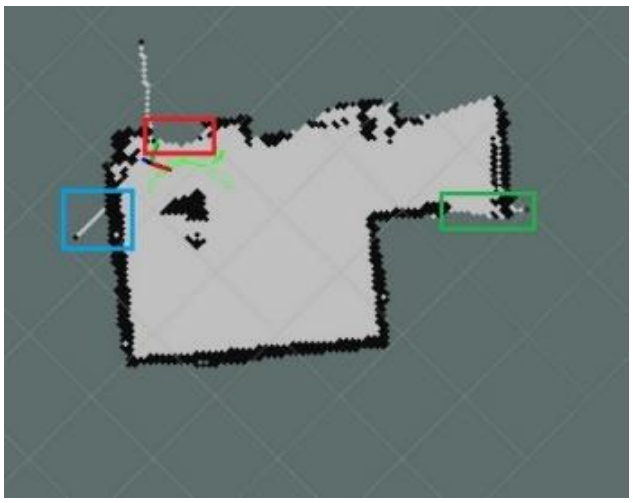

Figura 92 – Errores Lidar

En el rectángulo marcado con color rojo está el monitor del ordenador, con el que colisiona el lidar pero no lo marca como una pared u obstáculo (no lo marca en negro como el resto de paredes u obstáculos). En el rectángulo marcado con color verde hay espejos que como se observa hacen que no detecte que hay obstáculo en esa posición, pero sí proyecta puntos de colisión más a la derecha debido al reflejo de la pared.
Finalmente, en el rectángulo marcado con color azul hay una ventana cubierta con estores de láminas. Entre alguno de ellos ha pasado el láser generando un punto que colisiona en la terraza.

Margen de error LIDAR: Tras tener todos los datos almacenados y divididos en archivos en función de la posición y orientación, pasamos a un Excel en diferentes hojas los datos de cada uno de los archivos. Nos encontramos con que, debido a problemas de Excel, nos encontramos con errores ya que valores muy pequeños que aparecen en los archivos de texto como `-6.103515625E-05` al pegarlos quedaban como `-6.10E+04`, es decir, de `-0.000061` a `-61000`, desvirtuando completamente los cálculos. A pesar de escribir manualmente esos números que se pegaban de manera errónea desde el archivo de texto, se modificaban por números erróneos, por lo que decidimos eliminarlas esas muestras erróneas (entre 30 y 50 de las cerca de 2000 muestras de cada archivo).

Virtual Memory Exhausted: Después de haber instalado Linux, comenzamos con la instalación de los paquetes de ROS y Catkin, pero nos encontramos con el error de `Virtual memory exhausted: Cannot allocate memory`, el cual buscando el error en internet corregimos accediendo el archivo `/etc/dphys-swapfile` y modificando el valor de la variable `CONF_SWAPSIZE` por 1024.

Under-voltage: Tras instalar rplidar y hector_SLAM en la Raspberry Pi, al tratar de lanzar el proyecto nos encontramos con este error:

```
[rviz-2] process has died [pid 2923, exit code -11, cmd
/opt/ros/melodic/lib/rviz/rviz -d
/home/rigoberto/catkin_ws/src/rplidar_ros/rviz/rplidar.rviz __name:=rviz
__log:=/home/rigoberto/.ros/log/fbaaa4a0-a4e7-11ea-b45a-b827eb6c7f4b/rviz-
2.log].
log file: /home/rigoberto/.ros/log/fbaaa4a0-a4e7-11ea-b45a-
b827eb6c7f4b/rviz-2*.log
```

Decidimos comprobar la potencia que recibe la Raspberry Pi mediante el comando `dmesg`, ya que estaba conectada mediante USB al cargador del móvil y encontramos que hay problemas de bajo voltaje.

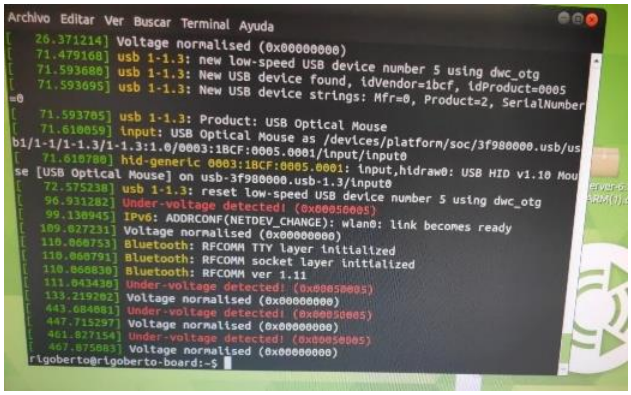

Figura 93 - Under-voltage

Conseguimos una fuente de alimentación que utilizar que proporcione más potencia para enchufar la Raspberry Pi. También comprobamos que los cables que conectan el lidar a la Raspberry Pi no han sufrido ningún cortocircuito.

Ralentización Raspberry Pi: Al tratar de lanzar el proyecto en la Raspberry Pi nos encontramos con que el dispositivo se sobrecaliente y se ralentiza en exceso. Este problema lo solucionamos tras conectar la Raspberry Pi y el ordenador mediante SSH para que la Raspberry Pi reciba la información recogida por el lidar y la envíe al ordenador para que la procese.

Costmap: Durante su instalación y compilación, encontramos nuevos errores en la consola que nos piden instalar dependencias para que el proyecto funcione. Una vez deja de solicitar la instalación de librerías encontramos nuevos errores de incompatibilidad entre tf y ros-melodic, por lo que debemos modificar los archivos en los que aparecen errores para migrar de tf a tf2.

Exploration controller: Al tratar de lanzar el proyecto nos encontramos con los errores `transform timeout` y `Could not get robot pose, cancelling reconfiguration`

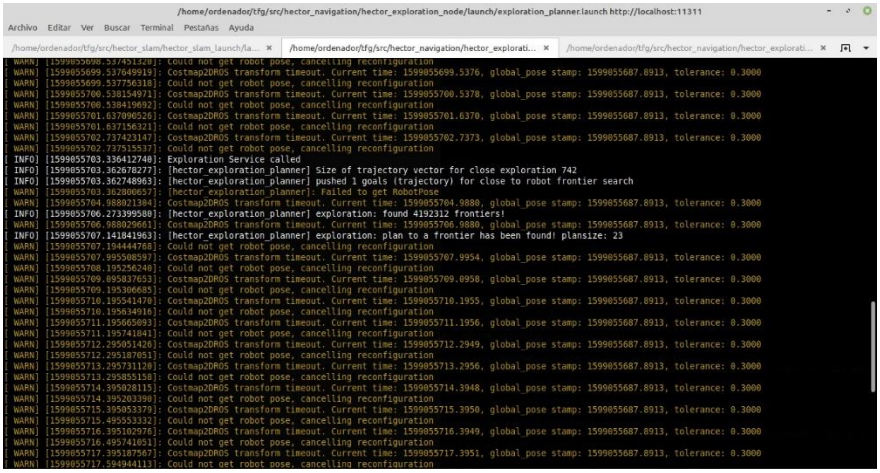

Figura 94 - eploration controller error

Encontramos que el error se debe a una variable llamada `transform_tolerance` en el archivo `costmap.yaml` que asignaba a cada transform un timestamp asociado a la hora en la que se calculó. Con ese parámetro decías que no confiabas en los transform que se hubieran calculado hace más de 0.3s. Como era demasiado poco tiempo, al sistema no le daba tiempo a calcularlo tan rápido, por lo que, al no saber cuánto tiempo necesitaba aproximadamente, decidimos subirlo a 10s (sabiendo que era mucho, pero para asegurarnos.

Con esto conseguimos solucionar el error y lanzar correctamente el `exploration_controller`.

Resultados

ROS – Catkin – RPLidar. Obtenemos los resultados que queríamos, que era el lidar mapeando el entorno mientras este, se mapeaba gráficamente en el entorno gráfico, pero manteniendo el lidar en una posición fija, siendo el entorno el que rota o se desplaza en torno al mismo.

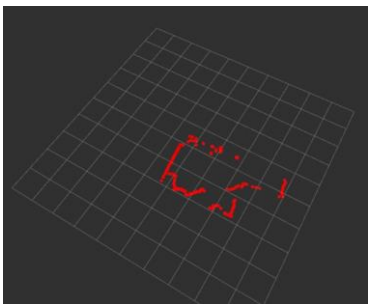

Figura 95 - rplidar

Hector_SLAM. Conseguimos mapear el entorno dejando el entorno en una posición fija, permitiendo almacenar y representar el entorno que vamos descubriendo con el lidar, así como representar los desplazamientos que hemos hecho con el dispositivo.

Figura 96 - hector_slam

Costmap. Conseguimos añadir la funcionalidad de mapear el entorno añadiendo la representación del costmap.

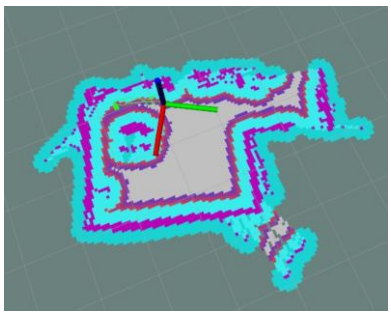

Figura 97 - costmap

Exploration path. Añadimos la funcionalidad de crear un recorrido actualizable que debería seguir el robot para mapear nuevo entorno que no ha sido explorado todavía.

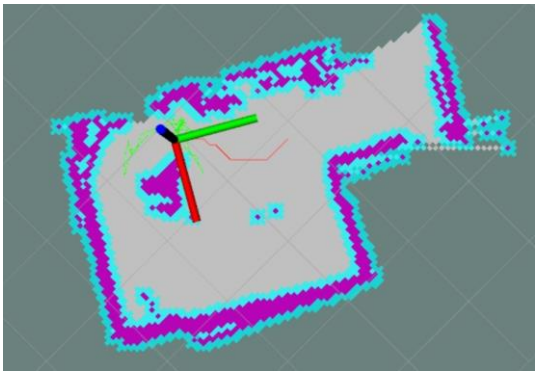

Figura 98 - Exploration path

Concluyendo con los resultados:

- Se ha implementado el mapeo con el Lidar desplazando el entorno (rplidar) en el ordenador.
- Se ha implementado el mapeo del entorno desplazando el Lidar (hector_slam) en el ordenador.
- Se ha realizado el estudio del error del Lidar.
- Se ha implementado rplidar y hector_slam en la Raspberry Pi siendo esta la que recibe la información del Lidar y quién la transmite al ordenador.
- Se ha creado un mando para controlar el vehículo desde el móvil mediante la APP del robot.
- Se ha implementado la funcionalidad de visualizar el costmap en el entorno mapeado.
- Se ha implementado la funcionalidad de visualizar la ruta que debería seguir el robot para explorar nuevo entorno que no ha sido mapeado.

Conclusiones

Hemos conseguido trabajar con el lidar para que sea capaz de mapear el entorno de forma simple, permitiendo al usuario visualizarlo en un entorno gráfico, manteniendo el lidar en una posición fija y desplazando/rotando el entorno a su alrededor.

Se ha implementado la funcionalidad de almacenar el entorno que mapea el dispositivo mientras lo desplazamos, permitiéndonos crear un mapa del mismo y visualizarlo en el entorno gráfico, así como visualizar la ruta por la que se ha ido desplazando el dispositivo.

Se ha realizado un estudio del margen de error del dispositivo para estudiar su viabilidad en el proyecto. Al tratarse de un margen de error menor al milímetro, concluimos con que es un margen de error despreciable de cara al proyecto en el que vamos a trabajar.

Se ha estudiado la posibilidad de trabajar con un Roomba durante la cuarentena para sustituir la Raspberry Pi y el robot. Se ha concluido que era necesario un dispositivo intermedio (Raspberry Pi o Arduino) para ser capaces de sustituir el Roomba por el robot, ya que el objetivo era poder trabajar sin necesidad de esperar a conseguir el material necesario.

Se han realizado las implementaciones en la Raspberry Pi para que sea mucho más fácil de desplazar el lidar, dividiendo el trabajo entre la Raspberry Pi y el ordenador para que la Raspberry Pi reciba la información del lidar y la envíe al ordenador, siendo este último quien se encargue de procesarla y representarla en el entorno gráfico.

Se ha creado un mando en el Smartphone para poder controlar el robot de forma remota mediante conexión Bluetooth.

Se ha implementado la funcionalidad de añadir en el entorno gráfico la visualización del costmap sobre el mapa del entorno explorado por el lidar para asegurar evitar colisiones.

Finalmente, se ha implementado la funcionalidad de crear una ruta actualizable que debería seguir el lidar para descubrir entorno que todavía no ha sido explorado.

Sin embargo, no se ha implementado la funcionalidad de navegación autónoma del coche.

Futuras implementaciones y mejoras para el proyecto hay muchas. Las siguientes implementaciones que podrían llevarse a cabo son:

En primer lugar y para que se pueda hacer un uso "real" del proyecto, el sustituir la fuente de alimentación de la Raspberry Pi por una batería recargable, para no limitar su uso al tener que estar conectada mediante un enchufe, permitiendo que se desplace libremente sobre el coche con el lidar.

Tras esto, traducir la información procesada por el lidar a movimiento del robot, de manera que sea capaz de "entender" dónde están los obstáculos y el espacio libre.

Una vez el robot comprende la información del lidar, implementar la funcionalidad de navegación autónoma para hacer que el robot siga la ruta creada por la última implementación realizada en el proyecto (exploration_path), de modo que vaya explorando de manera autónoma todo el entorno disponible hasta haber mapeado todo, siguiendo la ruta que se irá actualizando cada vez que detecte que hay entorno para explorar.

También se puede implementar una funcionalidad que permita al usuario introducir en el entorno descubierto por el robot un punto marcado como destino, estableciéndose así la ruta más corta desde la posición inicial del robot hasta el destino (algoritmos de rutas como A*), de modo que el robot se desplace de forma autónoma hasta ese punto siguiendo la ruta.

Bibliografía

[1] Conócenos | Inycom. *Inycom innovation Technologies* [en línea]. [consulta: septiembre 2020]. Disponible en: https://www.inycom.es/conocenos

[2] IBAÑEZ. Qué es un LIDAR, y cómo funciona el sensor más caro de los coches autónomos. *Motorpasión* [en línea]. 30 septiembre 2017 [consulta: septiembre 2020]. Disponible en: https://www.motorpasion.com/tecnologia/que-es-un-lidar-y-como-funciona-el-sistema-de-medicion-y-deteccion-de-objetos-mediante-laser

[3] RODRIGUEZ DE LUIS, Eva. De cero a maker: todo lo necesario para empezar con Raspberry Pi. *Xataka* [en línea]. 18 septiembre 2018 [consulta: septiembre 2020]. Disponible en: https://www.xataka.com/makers/cero-maker-todo-necesario-para-empezar-raspberry-pi

[4] MANUTI. SoC. *Raspberry Pi para torpes* [en línea]. 5 marzo 2013 [consulta: septiembre 2020]. Disponible en: https://raspberryparatorpes.net/glossary/soc/

[5] Robot Inteligente ELEGOO V3.0. *MCI Electronics* [en línea]. [consulta: septiembre 2020]. Disponible en: https://www.mcielectronics.cl/shop/product/robot-inteligente-elegoo-v3-0-manual-en-espanol-25769

[6] ADEVA, Roberto. Todo sobre Linux, el sistema operativo de código abierto. *Adsl Zone* [en línea]. 28 marzo 2020 [consulta: septiembre 2020]. Disponible en: https://www.adslzone.net/reportajes/software/que-es-linux/

[7] MARTÍNEZ, José Luis. Navegación SLAM: robots que construyen mapas. *CLEM* [en línea]. 11 febrero 2020 [consulta: septiembre 2020]. Disponible en: https://clem.es/noticia/navegacion-slam-robots-que-construyen-mapas-514

[8] VARGAS, César. ¿Qué es ROS? *Mariachi Robot, la robótica al alcance de todos* [en línea]. 10 enero 2019 [consulta: septiembre 2020]. Disponible en: http://mariachirobot.com/que-es-ros/

[9] Catkin. *Catkin 0.7.28 Documentation* [en línea]. [consulta: septiembre 2020]. Disponible en: https://docs.ros.org/api/catkin/html/

[10] PÉREZ, Enrique. Bosch, DJI y Sony quieren acelerar la llegada del coche autónomo: así son sus nuevos sensores LIDAR de bajo coste. *Xataka* [en línea]. 15 enero 2020 [consulta: septiembre 2020]. Disponible en: https://www.xataka.com/alta-definicion/bosch-dji-sony-quieren-acelerar-llegada-coche-autonomo-asi-sus-nuevos-sensores-lidar-coste

[11] DANS, Enrique. Uber buys 24,000 Volvo XC90s … and this time, it's not a rumor. *Medium, Dive deeper on topics that matter to you* [en línea]. 21 noviembre 2017 [consulta: septiembre 2020]. Disponible en: https://medium.com/enrique-dans/uber-buys-24-000-volvo-xc90s-and-this-time-its-not-a-rumor-2970a7dae40e

[12] RUBIO, Claudia. Volvo empleará la tecnología LiDAR en sus coches autónomos a partir de 2022. *NEOMOTOR* [en línea]. 11 mayo 2020 [consulta: septiembre 2020]. Disponible en: https://www.neomotor.com/coches/volvo/volvo-empleara-la-tecnologia-lidar-en-sus-coches-autonomos-a-partir-de-2022.html

[13] FERNÁNDEZ, Samuel. Así funciona un LiDAR, el sensor láser 3D que Apple ha montado en los nuevos iPad Pro. *Xataka* [en línea]. 20 marzo 2020 [consulta: septiembre 2020]. Disponible en: https://www.xatakamovil.com/apple/asi-funciona-lidar-sensor-laser-3d-que-apple-ha-montado-nuevos-ipad-pro

[14] SOLÉ, Roberto. Scanner Sombre, un juego que nos muestra el mundo que nos rodea basado en un escáner LIDAR. *Hardwaresfera* [en línea]. 25 abril 2017 [consulta: septiembre 2020]. Disponible en: https://hardwaresfera.com/noticias/scanner-sombre-juego-nos-muestra-mundo-nos-rodea-basado-escaner-lidar/

[15] RPLIDAR A1 Dimension and Weight. *Slamtec* [en línea]. [consulta: septiembre 2020]. Disponible en: http://www.slamtec.com/en/Lidar/A1Spec

[16] RPLIDAR A1 360º Laser Range Scanner. *Slamtec* [en línea]. [consulta: septiembre 2020]. Disponible en: http://www.slamtec.com/en/Lidar/A1

[17] What's new with Elegoo Smart Robot Car Kit V3.0. *Elegoo* [en línea]. [consulta: septiembre 2020]. Disponible en: https://www.elegoo.com/whats-new-with-elegoo-smart-robot-car-kit-v3-0/

[18] Arduino UNO REV3. *Arduino* [en línea]. [consulta: septiembre 2020]. Disponible en: https://store.arduino.cc/arduino-uno-rev3

[19] Install Ubuntu 16.04 desktop. *Ubuntu Tutorials* [en línea]. [consulta: septiembre 2020]. Disponible en: https://ubuntu.com/tutorials/install-ubuntu-desktop-1604#1-overview

[20] Stephen. IRobot Roomba 630. *Aspirame* [en línea]. [consulta: septiembre 2020]. Disponible en: https://aspirame.es/irobot/roomba-630/

[21] How To Clean, Maintain, Repair and Disassemble EVERYTHING on a Roomba 600 Series (690 675 650 etc). *Youtube* [en línea]. 20 enero 2019 [consulta: septiembre 2020]. Disponible en: https://www.youtube.com/watch?v=s8K2gCzfTxE

[22] iRobot Roomba, Robot de limpieza aspirador, manual del usuario. *iRobotweb* [en línea]. [consulta: septiembre 2020]. Disponible en: https://www.irobotweb.com/-/media/Files/Support/Home/Roomba/600/Roomba-600-Manual.pdf?sc_lang=es-CO

[23] iRobot® Create® 2 Open Interface (OI) Specification based on the iRobot® Roomba® 600. *iRobotweb* [en línea]. 10 agosto 2016 [consulta: septiembre 2020]. Disponible en: https://www.irobotweb.com/~/media/MainSite/PDFs/About/STEM/Create/iRobot_Roomba_600_Open_Interface_Spec.pdf

[24] Upgrade Your Old Roomba to a Smart BotVac for $5. *Youtube* [en línea]. 25 abril 2018 [consulta: septiembre 2020]. Disponible en: https://www.youtube.com/watch?v=t2NgA8qYcFI

[25] Raspberry Pi 3 Model B+. *Raspberrypi* [en línea]. [consulta: septiembre 2020]. Disponible en: https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/

[26] LOHS, Ingo. Elegoo Smart Robot Car Kit v3.0. *hackster.io. An avnet community* [en línea]. 17 septiembre 2018 [consulta: septiembre 2020]. Disponible en: https://www.hackster.io/ingo-lohs/elegoo-smart-robot-car-kit-v3-0-757c06

[27] ELEGOO ROBOT CAR KIT 3.0 | MONTAJE PASO A PASO. *Youtube* [en línea]. 12 marzo 2019 [consulta: septiembre 2020]. Disponible en: https://www.youtube.com/watch?v=hRLR1J1rveE

[28] Arduino Mega 2560 Rev3. *Arduino* [en línea]. [consulta: septiembre 2020]. Disponible en: https://store.arduino.cc/arduino-mega-2560-rev3

[29] Elegoo Download. *Elegoo* [en línea]. [consulta: septiembre 2020]. Disponible en: https://www.elegoo.com/download/

[30] Download the Arduino IDE. *Arduino* [en línea]. [consulta: septiembre 2020]. Disponible en: https://www.arduino.cc/en/Main/Software

[31] Built-In Examples. *Arduino* [en línea]. [consulta: septiembre 2020]. Disponible en: https://www.arduino.cc/en/Tutorial/BuiltInExamples

[32] Linux Mint 19.3 "Tricia" - Cinnamon (64-bit). *Linux Mint from freedom came elegance* [en línea]. [consulta: septiembre 2020]. Disponible en: https://linuxmint.com/edition.php?id=274

[33] Download Visual Studio Code. *Visual Studio Code* [en línea]. [consulta: septiembre 2020]. Disponible en: https://code.visualstudio.com/download

[34] TULLYFOOTE. Ubuntu install of ROS Melodic. *ROS* [en línea]. 25 marzo 2020 [consulta: septiembre 2020]. Disponible en: https://wiki.ros.org/melodic/Installation/Ubuntu

[35] ALIOZCAN096. ROS Tutorials. *ROS* [en línea]. 08 agosto 2020 [consulta: septiembre 2020]. Disponible en: http://wiki.ros.org/ROS/Tutorials

[36] KINTZHAO. Robopeak/rplidar_ros. *Github* [en línea]. 24 agosto 2018 [consulta: septiembre 2020]. Disponible en: https://github.com/robopeak/rplidar_ros

[37] Install ROS. *Robots uc3m* [en línea]. [consulta: septiembre 2020]. Disponible en: http://robots.uc3m.es/gitbook-installation-guides/install-ros.html

[38] GVDHOORN. Catkin. *ROS* [en línea]. 26 julio 2017 [consulta: septiembre 2020]. Disponible en: http://wiki.ros.org/catkin#installing_catkin

[39] GVDHOORN. Catkin. *ROS* [en línea]. 26 julio 2017 [consulta: septiembre 2020]. Disponible en: http://wiki.ros.org/catkin

[40] DIRKTHOMAS. Building and using catkin packages in a workspace. *ROS* [en línea]. 30 enero 2017 [consulta: septiembre 2020]. Disponible en: http://wiki.ros.org/catkin/Tutorials/using_a_workspace

[41] WUBINXIA. Slamtec/rplidar_ros. *Github* [en línea]. 31 diciembre 2019 [consulta: septiembre 2020]. Disponible en: https://github.com/Slamtec/rplidar_ros

[42] Changing permissions on serial port. *AskUbuntu* [en línea]. 07 octubre 2017 [consulta: septiembre 2020]. Disponible en: https://askubuntu.com/questions/58119/changing-permissions-on-serial-port

[43] ADMIN101. Linux: El directorio "/dev". *Medium* [en línea]. 10 octubre 2017 [consulta: septiembre 2020]. Disponible en: https://medium.com/@admin101/linux-el-directorio-dev-parte-4-5cdb140ac28

[44] The /etc/hosts file. *Faqs, Internet FAQ Archives* [en línea]. [consulta: septiembre 2020]. Disponible en: http://www.faqs.org/docs/securing/chap9sec95.html

[45] Modifying /etc/hosts to easily access a domain name (or ip address) for amazon EC2 instance. *Serverfault* [en línea]. 25 julio 2013 [consulta: septiembre 2020]. Disponible en: https://serverfault.com/questions/526164/modifying-etc-hosts-to-easily-access-a-domain-name-or-ip-address-for-amazon-e

[46] AUSTINHENDRIX. Running ROS across multiple machines. *ROS* [en línea]. 20 marzo 2019 [consulta: septiembre 2020]. Disponible en: http://wiki.ros.org/ROS/Tutorials/MultipleMachines

[47] JANOR. Specification of ROS_MASTER_URI and ROS_HOSTNAME. *ROS Answers* [en línea]. 02 octubre 2017 [consulta: septiembre 2020]. Disponible en: https://answers.ros.org/question/272065/specification-of-ros_master_uri-and-ros_hostname/

[48] MITKA, Łukasz y KRAWCZYK, Adam. Running ROS on multiple machines. *Husarion Docs* [en línea]. [consulta: septiembre 2020]. Disponible en: https://husarion.com/tutorials/ros-tutorials/5-running-ros-on-multiple-machines/

[49] Backing up and Restoring your Raspberry Pi's SD Card. *The Pi Hut* [en línea]. 18 marzo 2015 [consulta: septiembre 2020]. Disponible en: https://thepihut.com/blogs/raspberry-pi-tutorials/17789160-backing-up-and-restoring-your-raspberry-pis-sd-card

[50] GRUEMASTER, TUXINATOR2009. Win32 Disk Imager. *Sourceforge* [en línea]. 12 octubre 2018 [consulta: septiembre 2020]. Disponible en: https://sourceforge.net/projects/win32diskimager/

[51] NICKLAMPRIANIDIS. Costmap_2d. *ROS* [en línea]. 10 enero 2018 [consulta: septiembre 2020]. Disponible en: http://wiki.ros.org/costmap_2d

[52] JOHANNESMEYER. Hector_navigation. *ROS* [en línea]. 16 mayo 2013 [consulta: septiembre 2020]. Disponible en: http://wiki.ros.org/hector_navigation

[53] MARTIN-OHELER. tu-darmstadt-ros-pkg/hector_navigation. *Github* [en línea]. 6 mayo 2019 [consulta: septiembre 2020]. Disponible en: https://github.com/tu-darmstadt-ros-pkg/hector_navigation/tree/catkin

[54] Git clone. *Atlassian Bitbucket* [en línea]. [consulta: septiembre 2020]. Disponible en: https://www.atlassian.com/es/git/tutorials/setting-up-a-repository/git-clone

[55] JACKOQUIN. Geodesy. *ROS* [en línea]. 04 julio 2017 [consulta: septiembre 2020]. Disponible en: http://wiki.ros.org/geodesy

[56] Installation. *Ceres Solver 1.14* [en línea]. [consulta: septiembre 2020]. Disponible en: http://ceres-solver.org/installation.html

[57] MFEHR. Ethz-asl/ceres_catkin. *Github* [en línea]. 06 mayo 2019 [consulta: septiembre 2020]. Disponible en: https://github.com/tu-darmstadt-ros-pkg/hector_navigation/tree/catkin

[58] WJWWOOD. Catkin/catkin_simple. *Github* [en línea]. 25 junio 2015 [consulta: septiembre 2020]. Disponible en: https://github.com/catkin/catkin_simple

[59] CHRISTOPHROESMANN. Nav_core. *ROS* [en línea]. 02 marzo 2020 [consulta: septiembre 2020]. Disponible en: http://wiki.ros.org/nav_core

[60] JAVISSCHULTZ. Tf. *ROS* [en línea]. 02 octubre 2017 [consulta: septiembre 2020]. Disponible en: http://wiki.ros.org/tf

[61] Salarios para empleos de Analista programador/a en España. *Indeed* [en línea]. 31 agosto 2020 [consulta: septiembre 2020]. Disponible en: https://es.indeed.com/salaries/analista-programador-Salaries

[62] OKDIARIO. Cómo calcular las horas trabajadas en un año. *Okdiario How To* [en línea]. 10 julio 2018 [consulta: septiembre 2020]. Disponible en: https://okdiario.com/howto/como-calcular-horas-trabajadas-ano-2559566

[63] TRECET, Jose. Cuánto cotizas a la Seguridad Social según tu salario. *Finect* [en línea]. 29 mayo 2020 [consulta: septiembre 2020]. Disponible en: https://www.finect.com/usuario/Josetrecet/articulos/cuanto-cotizas-seguridad-social-segun-salario

Anexo I – Figuras