Universidad del País Vasco Euskal Herriko Unibertsitatea

BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

MÁSTER EN INGENIERÍA INDUSTRIAL

TRABAJO FIN DE MÁSTER

SISTEMA DE RECONOCIMIENTO AUTOMÁTICO DEL ENTORNO BASADO EN LA PLATAFORMA SUMMIT ACTUALIZADA

Documento 1 - Memoria

Alumna: Gracia Zarraga, Nerea
Director: Irigoyen Gordo, Eloy

Curso: 2020-2021

Fecha: 22-09-2021

RESUMEN

El robot autónomo terrestre SUMMIT de Robotnik, se gobierna mediante una placa controladora del mismo fabricante. El PC integrado al que se conecta tal controladora, ha quedado obsoleto y por tanto, se ha sustituido por una placa con mejores prestaciones. Esta sustitución ha dado lugar a la implantación de nuevo hardware y software para el control de los motores. Para la mejora de la sensorización, la cámara PTZ integrada en el robot ha sido sustituida por un sensor rpLidar con el que poder crear un sistema de mapeado. La solución creada a partir de estas modificaciones se presenta detalladamente en este documento.

Palabras clave: SUMMIT, Jetson, rpLidar, navegación, autónomo.

LABURPENA

Robotnik-ek sortutako SUMMIT lurreko robot autonomoa fabrikatzaile berdineko plaka kontroladore baten bidez gobernatzen da. Plaka hau zaharkituriko prozesagailu batera konektaturik dago. Ordenagailu hori, ezaugarri hobedun beste batekin ordezkatu da. Ordezkapen honek, motoreen kontrolerako hardware eta software berriak erabiltzea eragin du. Sentsorizazioa hobetzeko asmoz, errobotaren PTZ kamera rpLidar sentsore batez ordeztu da, zeinekin mapeo sistema bat era daitekeen. Memoria honetan, modifikazio hauekin sortutako soluzio berria aurkezten da.

Gako-hitzak: SUMMIT, Jetson, rpLidar, nabigazioa, autonomoa.

ABSTRACT

The Robotnik SUMMIT autonomous terrestrial robot is managed by a controller board created by the same manufacturer. This one is connected to an out-dated PC which has been replaced by a computer with better features. This has concluded in the use of new hardware and software for the control of the motors. For a better sensorization system, the PTZ camera integrated on the robot, has been replaced by the rpLidar sensor which is capable to build a map system. In this memory, it is presented a new solution created from the previous modifications.

Keywords: SUMMIT, Jetson, rpLidar, navigation, autonomous.

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1.INTRODUCCIÓN

Con el paso del tiempo, la sociedad se sumerge en un mundo cada vez más robotizado. Lo que hace años podía verse como un futuro lejano ha llegado al día a día, y se encuentra en un desarrollo continuo. En la vida cotidiana pueden encontrarse claros ejemplos de ello, tales como aspiradoras autónomas o asistentes virtuales controlados por voz.

A medida que la tecnología, y con ello el conocimiento de la sociedad, avanza, cada vez son más los aparatos que pueden llegar a ser autónomos. En el tema automovilístico pueden apreciarse varios avances de este tipo, por ejemplo, coches con sistema de aparcamiento automático, que aunque lleven el nombre "automático" aún no lo son completamente, ya que necesitan de movimientos ejecutados por una persona para poder completar su objetivo.

Centrando el tema en robots autónomos y yendo un poco más allá de las aspiradoras, se podría pensar en las dificultades que pueden tener diariamente las personas con movilidad reducida o simplemente de avanzada edad. Para estas, un robot autónomo podría ser de gran utilidad en diferentes ámbitos de la vida cotidiana. Como sería un robot ayudante dentro de una vivienda, el cual daría apoyo a la hora de realizar acciones tan simples como alcanzar un objeto inalcanzable para algunas personas, incluso sería posible introducir comandos por voz, facilitando así su uso.

En el presente proyecto se trabaja en un laboratorio con el robot SUMMIT con el que poder estudiar la autonomía de los robots mencionados en los párrafos anteriores. Se trata de hacer una actualización tanto del hardware como del software, consiguiendo con ello estar acorde con las necesidades del presente. Con este fin, la idea principal será integrar una nueva placa base más potente que la anterior, con la que sea posible hacer uso de las últimas actualizaciones tanto del sistema operativo como de otras aplicaciones. Además, se va a instalar un sensor rpLidar con el que se logra una conducción autónoma mejorada.

En el siguiente capítulo se va a contextualizar el problema planteado. A continuación, se definirá el objetivo principal, y también los objetivos secundarios a abordar para lograr este. Una vez clara la finalidad del trabajo se hablará de los beneficios que puede aportar lograr tales objetivos. En el quinto capítulo, se describirán brevemente los requerimientos del sistema, los que será necesario

cumplir. Seguidamente se hará un análisis de alternativas, analizando tanto las opciones de hardware como de software que se deben actualizar, donde se concluirá con que equipos se ha de trabajar. Los capítulos 7,8 y 9 podrían considerarse el núcleo de la memoria, donde se encuentran respectivamente, la descripción de la solución que se plantea, el diseño que se ha llevado a cabo para ello y la descripción de los resultados obtenidos. Al final del documento se encuentran tres capítulos para la organización del proyecto, que corresponden a un plan de trabajo, un diagrama de gantt basado en el capítulo anterior y el presupuesto a abordar. Para finalizar serán descritas las conclusiones finales de este Trabajo Fin de Máster.

2.CONTEXTO

Se podría definir un robot como una máquina programable capaz de realizar acciones comúnmente ejecutadas por los seres humanos. [1] Esta entidad autómata se compone por mecánica artificial y un sistema electromecánico. [2]

Dentro de la robótica existen diversos tipos de robots, entre los que se encuentran los robots móviles. Se definen como máquinas automáticas con capacidad de desplazamiento, dirigidas mediante telemando o autónomas, guiadas gracias a la información recibida del entorno mediante un software avanzado. [3]

Un claro ejemplo de robot autónomo extensamente conocido, es el de las aspiradoras autónomas. La base de funcionamiento de un robot aspirador es el empleo de sensores. Mediante estos, se detectan todos los objetos de la casa para después poder evitarlos. Es un mercado innovador, existen aspiradoras simples que al encontrar un objeto vuelven en la dirección opuesta, pero también los hay con tecnología más avanzada, capaces de detectar si hay un vacío como podría ser una escalera. La última novedad de estos robots es su control mediante wifi, facilitando así activar su funcionamiento desde cualquier lugar, haciendo uso de una aplicación de móvil. [4]

El robot SUMMIT es un robot móvil terrestre autónomo creado por la empresa Robotnik. Es capaz de navegar, tanto controlado telemáticamente como de manera autónoma. Para ayudar en la navegación dispone de una cámara Pan-Tilt-Zoom con la que se puede recibir video del entorno en tiempo real. Además, la arquitectura de control de la que dispone en su PC integrado de Intel es abierta y modular y está basada en ROS. [5]

Figura 1: Robot SUMMIT en el laboratorio

El robot original, tal y como lo distribuye Robotnik, se compone de los siguientes elementos, que pueden dividirse en elementos mecánicos y electrónicos.

ELEMENTOS MECÁNICOS

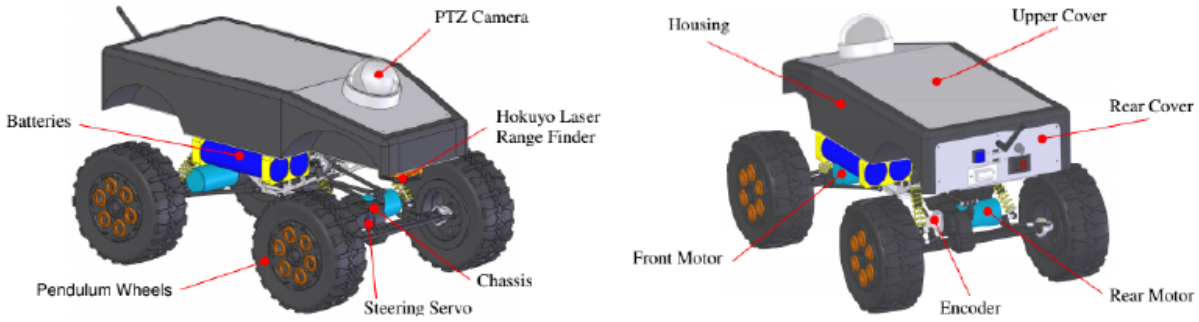

Figura 2: Elementos mecánicos del SUMMIT

- La **carcasa** que envuelve la parte electrónica está compuesta por una cubierta, una tapa superior y una tapa posterior. En la tapa superior va colocada una cámara PTZ, la cual en este caso no se va a utilizar. La tapa posterior dispone de 2 entradas USB, una entrada para el cargador de baterías, un interruptor de encendido para las baterías y un botón de power para la placa base.

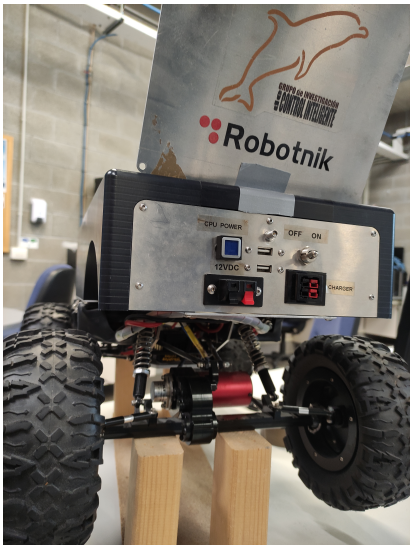

Figura 3: Parte trasera del SUMMIT

- Las **ruedas** son del tipo de bloqueo de cuentas de péndulo pesado, con las que se logra que el peso siempre quede en el punto más bajo de la rueda.

Figura 4: Ruedas del SUMMIT

- Dos **baterías** Mamba Monster Xtreme ⅛ colocadas en la parte inferior del SUMMIT.
- **Chasis** de aluminio con fuertes amortiguadores.
- Un **motor** eléctrico sin escobillas en cada eje, uno para las ruedas delanteras y otro para las traseras, proporcionando así tanto tracción delantera como trasera. Motores Leopard Hobby, LBP 4065 de 1400 kV cada uno.
- Un **encoder** RLS.
- Un **servomotor** de dirección en el eje delantero del robot, puesto que el eje trasero se mantiene fijo. Servo Hitec Monster Torque HS-7980TH que dispone de una potencia de 38kg/cm.

ELEMENTOS ELECTRÓNICOS

- **PC empotrado** Intel D945GSEJT que dispone de un procesador de 32 bit.
- Un módulo **Router WiFi** para poder conectar la placa base a la red.
- Un **convertidor** DC/DC.
- Una caja de **fusibles**.

- La placa **controladora** AGVCTRL-V2 conectada mediante conexión serie al PC empotrado. Para esta conexión se ha utilizado un cable serie RS232 de 9 pines. Esta placa a su vez está conectada al servomotor de dirección, a los motores y a las baterías. Sus funciones principales son regular la velocidad de los motores y la posición del servomotor, haciendo uso de la información recibida del PC.

Figura 5: Parte delantera del SUMMIT

- Un **disco duro** donde poder guardar los datos necesarios.

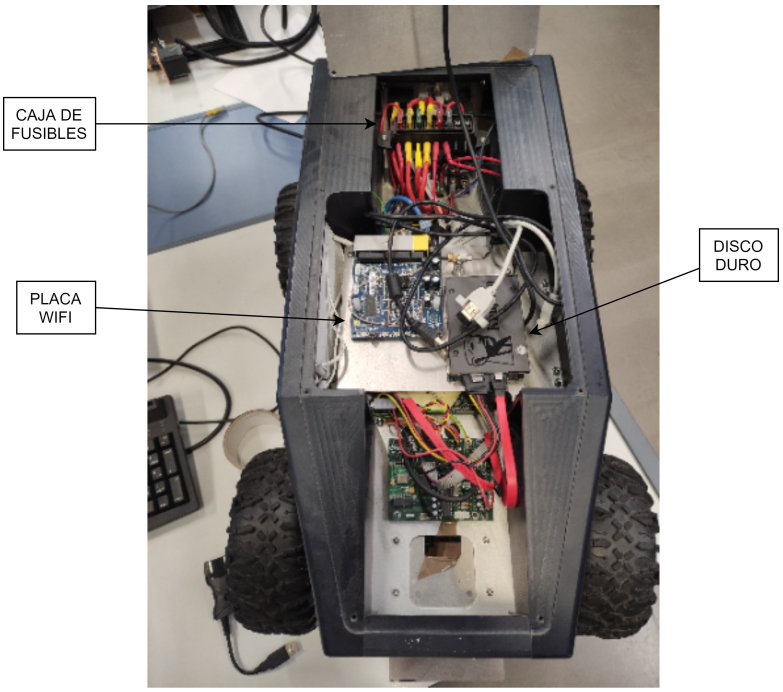

Figura 6: Interior del SUMMIT 1

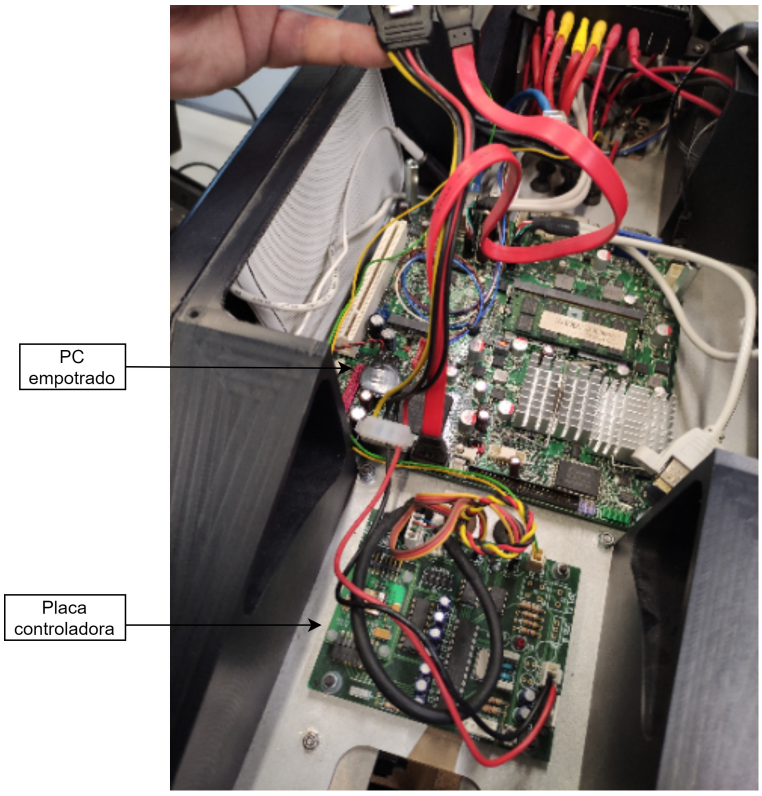

Figura 7: Interior del SUMMIT 2

Tanto el PC integrado como la cámara se han quedado obsoletos al tratar de avanzar con la investigación en el entorno robótico. Es aquí donde entra este trabajo de fin de máster (TFM) que consistirá en actualizar los componentes del SUMMIT, incorporando una PC más potente y un sensor con el que ser capaz de reconocer el entorno y posibilitar la conducción de forma autónoma.

La placa que se va a instalar, sustituyendo la Intel, es la JETSON TX2 de Nvidia. Está compuesta por un procesador de 64 bits siendo así más veloz que la anterior. Además, esta serie de módulos permite aplicaciones de deep learning para elementos de pequeño tamaño como podría ser un dron. [11]

La nueva sensorización se trata de un sensor rpLidar capaz de captar datos de 360° a su alrededor. Este sensor se colocará en la parte superior del robot y mediante la conducción del robot por un entorno concreto, el sensor irá detectando los diferentes objetos existentes. Finalmente, se logrará crear un mapa completo del lugar.

Para la creación de nodos y comunicación entre ellos se va a hacer uso de la plataforma ROS. Es un sistema operativo proveedor de librerías y herramientas, útiles a la hora de desarrollar el software necesario para la creación de aplicaciones para robots. [13]

Además, en la preparación previa al proyecto, se ha utilizado la aplicación Gazebo en la que hacer una simulación del robot SUMMIT con el sensor rpLidar integrado y así lograr los resultados que se deberán obtener mediante la navegación real.

A su vez, se utiliza la plataforma rviz para diferentes funciones, tanto en la simulación como en la navegación real por igual. Primeramente, en esta aplicación se visualizará el proceso de creación del mapa, es decir, el sensor irá captando objetos poco a poco y se podrá ver cómo se va creando el mapa. Después, se plasmará el mapa que se ha cargado en el nodo correspondiente a la navegación y el robot se colocará aleatoriamente en este. Desde esta aplicación será posible indicarle al robot en que posición del mapa se encuentra y su orientación instantánea. Por último, mediante otra herramienta de rviz, se le podrá indicar al SUMMIT su destino y una vez hecho esto el robot deberá moverse hasta el lugar indicado por sí solo.

3.OBJETIVOS Y ALCANCE DEL TRABAJO

El objetivo principal de este TFM es la actualización de los componentes de la plataforma móvil terrestre SUMMIT para posteriormente poder desarrollar una solución de reconocimiento del entorno. Para ello se deberán identificar los elementos clave de dicho entorno, por lo que se añadirá nueva sensorización. De esta manera, se creará un mapa estático y finalmente se posibilitará la navegación autónoma del robot.

El alcance y los objetivos parciales a abordar en este TFM son los siguientes:

- Adquirir destreza en la búsqueda bibliográfica y localización de fuentes de consulta.
- Hacer uso con soltura de Ubuntu en Linux.
- Adquirir conocimiento y experiencia en ROS.
- Actualizar el hardware actualmente instalado en la plataforma SUMMIT.
- Incorporar la nueva sensorización.
- Diseñar las conexiones necesarias entre la nueva placa y los componentes del robot.
- Diseñar una conexión serie nueva con la controladora.
- Crear un mapa en el que navegar.
- Desarrollar una solución de navegación autónoma.
- Aprender a redactar un informe formal en formato técnico/científico.
- Practicar el inglés como lenguaje de trabajo.

4.BENEFICIOS QUE APORTA EL TRABAJO

En el mundo de la tecnología y la robótica es importante estar siempre al día, aprendiendo e innovando a la vez que los recursos y capacidades brotan. A día de hoy, los robots móviles autónomos están revolucionando el mundo de la automatización en la industria, mayormente a la hora del transporte de material. Gracias a estos, empieza a ser posible por ejemplo unir procesos de producción, haciendo el transporte de los productos de una cinta transportadora a otra. [6] Por otra parte, dejando de un lado la industria y mirando a la sociedad, están las aspiradoras autónomas, las cuales facilitan el trabajo diario de cada vez más personas. Tanto estos robots domésticos como los industriales mencionados, se basan en la misma arquitectura base que el robot presentado en este trabajo. Disponen de una base móvil, mediante la que se desplazan, y sensorización, a través de la cual se transmite información a un software avanzado desde el que poder ejecutar la navegación autónoma.

El robot del que se dispone es útil para el estudio de este tipo de robots. Para este estudio, es importante que el robot esté actualizado y que soporte las innovaciones pertinentes. Con ese fin se crea el presente proyecto, con el que se logra la conexión de la placa controladora con una nueva computadora, la placa JETSON TX2, más potente que la anterior. Al tener mejores capacidades y actualizaciones más novedosas, será posible la integración de nuevos sensores y la mejora de la navegación autónoma.

Mirando un poco más allá, en un futuro podrían aplicarse este tipo de robots en uso doméstico enfocado mayormente en personas con discapacidades físicas o personas de avanzada edad que podrían necesitar ayuda en labores cotidianas como alcanzar algún objeto. Podría plantearse también llevar la idea del robot autónomo a una silla de ruedas autónoma con la que poder transportarse indicando simplemente el destino deseado.

5.DESCRIPCIÓN DE REQUERIMIENTOS

5.1 Requisitos del sistema.

Los requisitos del sistema corresponden a las especificaciones impuestas previas al inicio del trabajo. Estas se resumen en los siguientes puntos:

- Se va a hacer uso del robot autónomo terrestre SUMMIT fabricado por la empresa Robotnik. Este, está compuesto de 2 motores de tracción y un servomotor de dirección, todos ellos gobernados por una placa controladora del mismo fabricante.
- En comunicación con la controladora se instalará la placa JETSON TX2 fabricada por Nvidia.
- Para el control de los movimientos se hará uso de la plataforma ROS implementada en el sistema operativo Ubuntu 18.04 de Linux.
- Para la navegación autónoma se instalará un sensor que tenga la capacidad de analizar 360° a su alrededor.

5.2 Requisitos funcionales.

Los requisitos funcionales son las especificaciones que podría dar el cliente o en este caso las que cumplen con los objetivos del proyecto.

Se busca la actualización de los componentes del robot para posteriormente poder conseguir la navegación autónoma. Para ello, haciendo uso de la plataforma ROS, se crearán diferentes nodos, los cuales comunicándose entre ellos deberán recoger la información que reciba el sensor, procesarla y transmitirla a la controladora. Esta última será la encargada de controlar tanto los motores como el servomotor, con el fin de transmitirles las órdenes adecuadas de movimiento facilitando así seguir el camino correcto. La transmisión de datos mencionada se hará a través de la computadora, que en este caso será nueva, y por tanto será imprescindible diseñar correctamente las conexiones necesarias.

6.ANÁLISIS DE ALTERNATIVAS

6.1 Alternativas a la distribución de ROS

La utilización de la plataforma ROS es uno de los requisitos del sistema, aun así se ha analizado la distribución más adecuada a implementar para este proyecto. Las características analizadas han sido las siguientes:

- Soporte: Será importante disponer del mayor tiempo de soporte posible puesto que este determinará la cantidad de actualizaciones disponibles para resolver los problemas que puedan surgir.
- Comunidad: Para solventar las dudas y problemas que van surgiendo durante el proyecto es de gran ayuda la comunidad que respalda a cada distribución, la cual dispondrá de más recursos cuanto más grande sea.
- Compatibilidad con Ubuntu 18.04: La versión de Ubuntu es otro de los requisitos del sistema y por tanto se deberá disponer de una distribución de ROS compatible con esta o al menos dentro de los paquetes que disponga, que los necesarios para el proyecto sean compatibles.

Una vez claras las características que se deben analizar se enfocan en cada una de las alternativas.

- ROS Kinetic Kame: Esta versión fue lanzada en 2016 y se amplió hasta abril de 2021 por lo que no dispondrá de nuevas actualizaciones de este año en adelante. Su gran periodo de soporte implica que tenga una gran comunidad que la respalda. Tiene un diseño enfocado a Ubuntu 16.04.
- ROS Melodic Morenia: Es la versión lanzada en 2018 que se ampliará hasta 2023. Está principalmente diseñada para implantarse en Ubuntu 18.04. Al igual que la anterior dispone de una gran comunidad.
- ROS Noetic Ninjemys: Es la versión más reciente, lanzada en 2020 y ampliada hasta 2025. Está enfocada para ser implantada en Ubuntu 20.04, que corresponde a la versión más actual. Al ser tan reciente, la comunidad de la que dispone está aún en crecimiento.

Resumiendo estas características en una tabla:

Tabla 1: Características de las distribuciones de ROS

Versión de ROS	EOL	Comunidad	Compatibilidad con Ubuntu 18.04
ROS Kinetic	2021	Amplia	No
ROS Melodic	2023	Amplia	Si
ROS Noetic	2025	Reducida	No

Para tomar la mejor decisión se va a valorar cada una de las características haciendo uso de la siguiente ponderación:

Tabla 2: Ponderación de las características de ROS

	Soporte	Comunidad	Compatibilidad con Ubuntu 18.04
Ponderación	1	2	3

Es decir, en este caso se le va a dar un peso mayor a la compatibilidad con Ubuntu 18.04, seguido de la amplitud de la comunidad que apoya la versión y dando la menor importancia al soporte que se oferta.

En la tabla siguiente se puntuará cada uno de estos tres aspectos para cada uno de los soportes en un rango de 1 a 3 puntos, y multiplicando esta puntuación por la ponderación mencionada se logrará la puntuación total para cada soporte de ROS. Finalmente, en base a esto se hará la elección.

Tabla 3: Puntuación a las distribuciones de ROS

	Soporte	Comunidad	Compatibilidad con Ubuntu 18.04	TOTAL
Ponderación	*1*	*2*	*3*	
ROS Kinetic	1	3	1	10
ROS Melodic	2	3	3	17
ROS Noetic	3	1	1	8

Observando los resultados se concluye que la mejor opción es instalar ROS Melodic.

En un futuro próximo, lo ideal sería poder actualizar todo el sistema. Es decir, trabajar con Ubuntu 20.04 en el que instalar ROS Noetic y hacer uso del nuevo ROS 2. Todo esto es aún muy nuevo y está en desarrollo continuo, por lo que dentro de un tiempo dispondrá de más recursos, como puede ser una comunidad algo más amplia que pueda facilitar el trabajo.

6.2 Alternativas de la sensorización a implementar

En un principio, se propuso implementar tanto un sensor como una cámara para el reconocimiento del entorno del SUMMIT. Finalmente, se decidió prescindir de la cámara puesto que no es necesaria para la conducción autónoma. Aún así, se va a añadir en el análisis de alternativas por si fuera de ayuda en una próxima instalación.

6.2.1 Sensores láser

- Sensor rpLidar A2: Sensor perteneciente a la empresa Slamtec con capacidad de analizar 360° a su alrededor y detectar objetos a una distancia de hasta 6 metros. Su uso se extiende tanto para interiores como exteriores. Utiliza un sistema de medición por triangulación láser tomando hasta 4000 muestras de láser por segundo con una velocidad de rotación de 600 rpm. Su precio ronda los 500 €. [7]

- Sensor Hokuyo UST-20LX: El fabricante de este sensor es Hokuyo Automatic. Tiene un rango de análisis de 270° y de hasta 20m de distancia. Está diseñado únicamente para interiores. Escanea con una frecuencia de 40 Hz, es decir, con una velocidad de rotación de 2400 rpm. Su precio ronda los 2300 €. [8]

Estas propiedades pueden resumirse en la siguiente tabla:

Tabla 4: Características de los sensores láser

	Rango	Distancia máx.	Velocidad de rotación	Precio
rpLidar	360°	6 m	600 rpm	500 €
Hokuyo	270°	20 m	2400 rpm	2300 €

Las características descritas se van a valorar teniendo en cuenta la siguiente ponderación, dándole el valor más alto a la propiedad más significativa para el proyecto que en este caso será el precio, seguido del rango de visión, la distancia máxima alcanzable y la velocidad de rotación:

Tabla 5: Ponderación de las características de los sensores

	Rango	Distancia máx.	Velocidad de rotación	Precio
Ponderación	3	2	1	4

Puntuando cada característica de cada uno de los sensores en un rango de [1-3] siendo el valor más alto para la característica que más se adecúe al proyecto, y a continuación, aplicando la ponderación nos queda el siguiente resultado:

Tabla 6: Puntuación a los sensores láser

	Rango	Distancia máx.	Velocidad de rotación	Precio	TOTAL
Ponderación	*3*	*2*	*1*	*4*	
rpLidar	3	1	1	3	24
Hokuyo	2	2	3	1	17

Para este proyecto en concreto será más conveniente utilizar el sensor rpLidar. Aunque en general el láser Hokuyo tiene mejores características, el precio es muchísimo mayor y esto es un gran inconveniente.

6.2.2 Cámaras

- Cámara Orbbec Astra: Cámara RGBD fabricada por Orbbec, desarrollada para su compatibilidad con aplicaciones en OpenNI. Dispone de un campo de visión de 60° en horizontal y 49,5° en vertical y capta la imagen a una distancia de hasta 8 metros. Capta la imagen con una resolución de 640x480 para la imagen en profundidad y de 1280x720 para la imagen en RGB. Su precio ronda los 150 €. [8]
- Cámara Kinect-2: El fabricante de esta cámara es Microsoft. Dispone de un campo de visión de 70° en horizontal y 60° en vertical. Tiene un sensor de distancia capaz de detectar hasta 6 personas a la vez. Capta la imagen con una resolución de 1920x1080. Su precio ronda los 100 €. [9]
- Cámara Intel Realsense D435: Cámara con sensor de profundidad y sensor RGB fabricada por Intel. Su intervalo de operación es de hasta 10 metros y dispone de un campo de visión de 85,2° en horizontal y 58° en vertical. Capta la imagen con una resolución de 1280x720 para la imagen en profundidad. Su precio ronda los 200 €. [10]

Estas características se resumen en la siguiente tabla:

Tabla 7: Características de las cámaras

	Campo de visión	Resolución	Precio
Orbbec Astra	60°H ; 49.5°V	640x480	150 €
Kinect-2	70°H ; 60°V	1920x1080	100 €
Intel RealSense	85.2°H ; 58°V	1280x720	200 €

La valoración de las características se va a hacer teniendo en cuenta la siguiente ponderación, en la que se da mayor importancia a la resolución de la cámara:

Tabla 8: Ponderación de las características de las cámaras

	Campo de visión	Resolución	Precio
Ponderación	2	3	1

Para este caso se ha dado una mayor importancia a las características que al precio puesto que las tres cámaras a evaluar están en el mismo rango de precios.

Puntuando las características y aplicando la ponderación a las tres cámaras obtenemos el siguiente resultado:

Tabla 9: Puntuación a las cámaras

	Campo de visión	Resolución	Precio	TOTAL
Ponderación	*2*	*3*	*1*	
Orbbec Astra	3	1	2	11
Kinect-2	2	3	3	16
Intel RealSense	1	2	1	9

Según los criterios analizados en este apartado la elección de la cámara sería la Kinect-2.

7.DESCRIPCIÓN DE LA SOLUCIÓN

Habiendo quedado elegidas la distribución de ROS a utilizar y la sensorización a implementar, se debe definir la metodología necesaria para cumplir con los objetivos del presente proyecto. El objetivo principal, mencionado anteriormente, es sustituir la placa base y añadir nueva sensorización para posteriormente lograr un reconocimiento del entorno y la navegación autónoma del robot. A continuación se va a describir la solución propuesta, especificando tanto el hardware como software necesarios para concluir el proyecto.

7.1 Hardware

Se hará uso de un ordenador con su pantalla, teclado y ratón, con la que se creará una conexión remota desde la que ejecutar los nodos en el PC empotrado del SUMMIT. En gran parte del proyecto, durante su desarrollo, se le han añadido al robot una pantalla, un teclado y un ratón para poder acceder y hacer las ediciones correspondientes directamente sobre el PC empotrado. En esta segunda configuración no será necesario el ordenador remoto. Por tanto, habrá dos esquemas posibles durante el proyecto, los cuales se muestran en las próximas páginas.

En tales esquemas algunas de las conexiones estarán representadas con colores. La flecha roja corresponde a la conexión serie entre la placa controladora y la Jetson TX2. Por otro lado, la azul indica los pines desde los que la controladora es alimentada. Por último, los motores, el servomotor y el encoder se conectan mediante conectores de 3 pines.

En el segundo esquema existen dos nuevas conexiones: la que conecta la Jetson TX2 con el botón Power situado en la parte trasera del robot, indicado en verde en la imagen; y la conexión WiFi de los dispositivos, mediante la que se realiza la conexión remota, indicado con líneas discontinuas.

DURANTE EL DESARROLLO

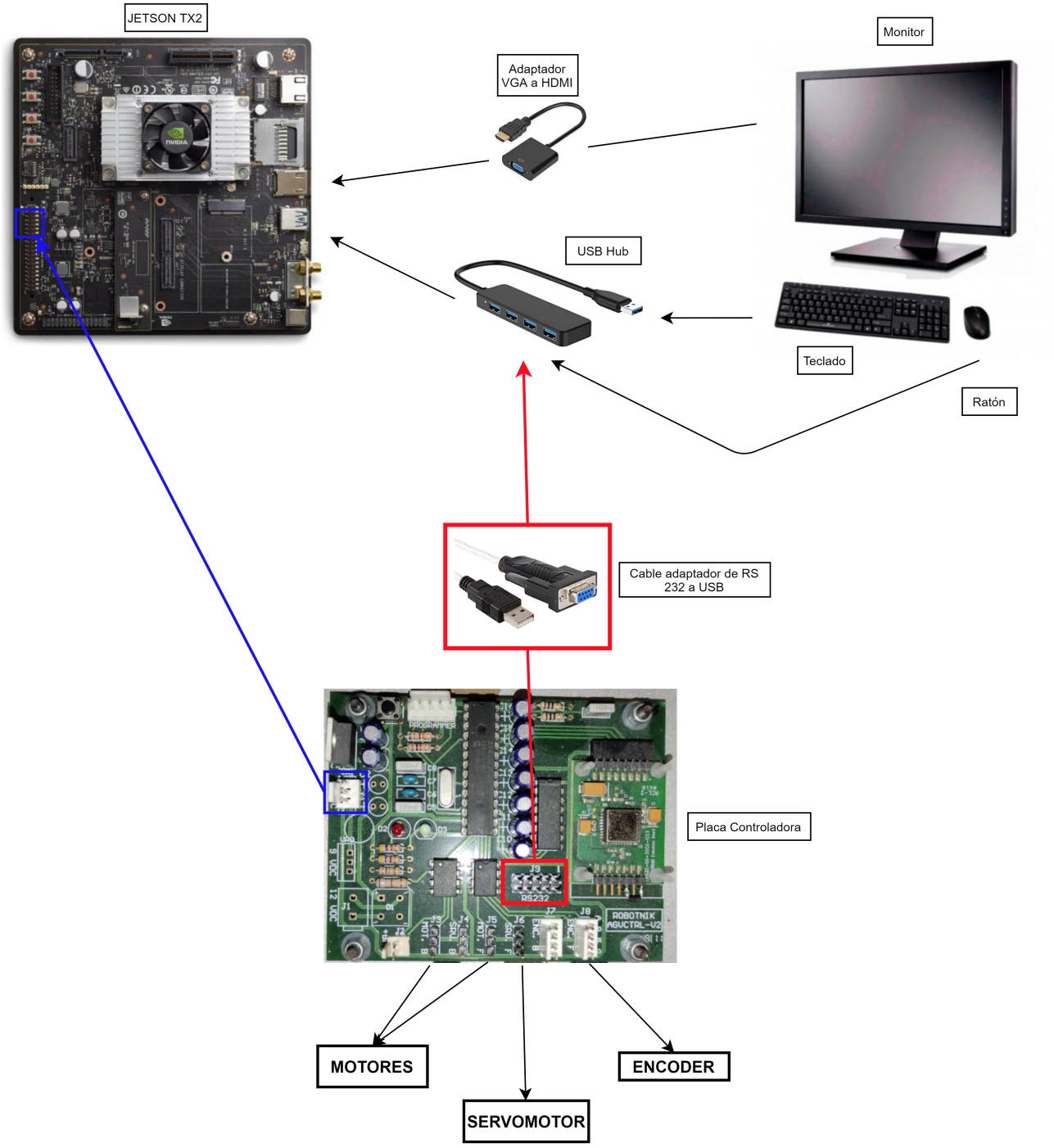

Figura 1: Esquema hardware durante el desarrollo

ESQUEMA DEL HARDWARE FINAL

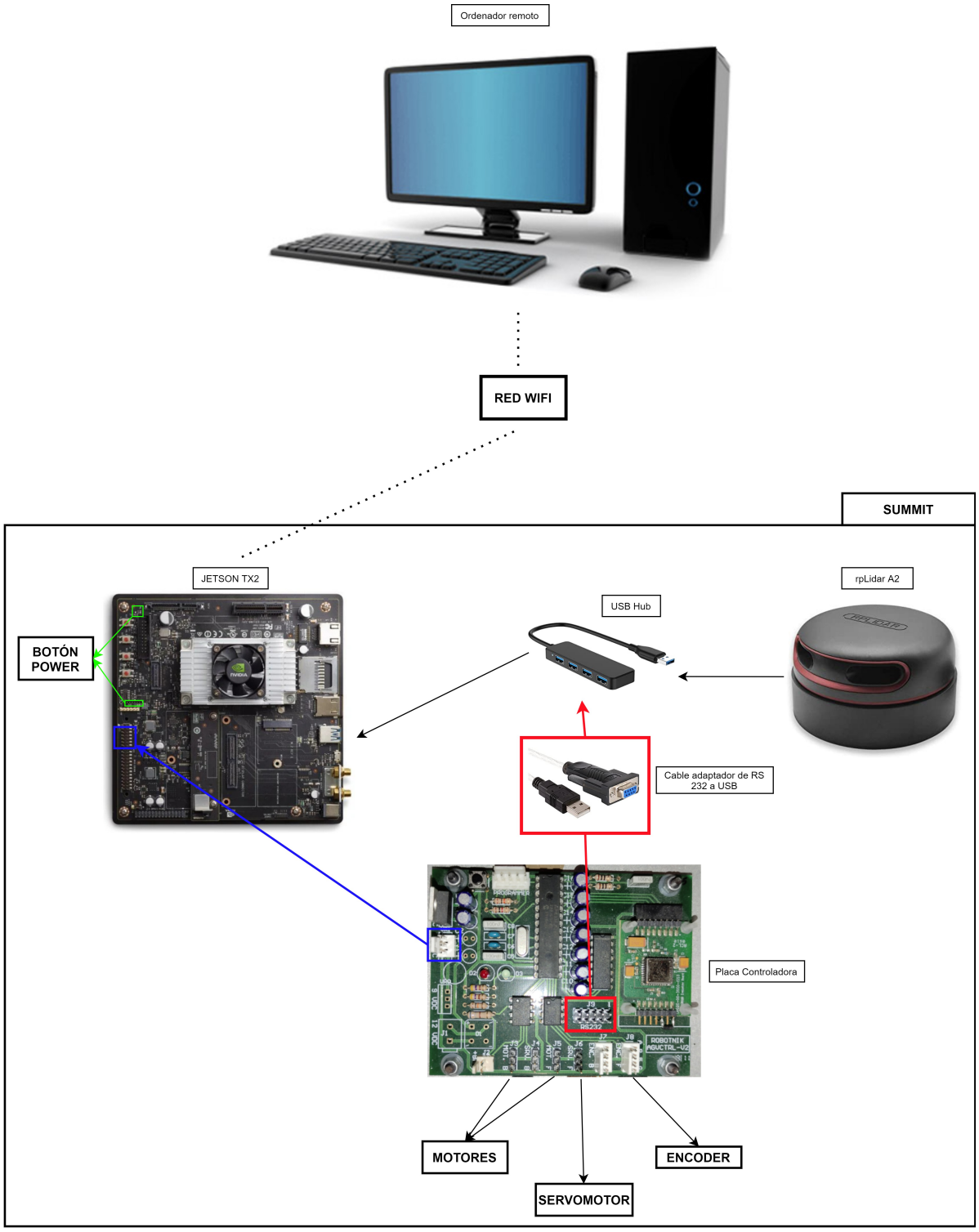

Figura 2: Esquema hardware final

NUEVA EQUIPACIÓN PARA EL ROBOT

PC empotrado: En el nuevo diseño del robot se va a reemplazar el PC empotrado de Intel por la placa Jetson TX2. Este cambio se debe a que el PC de Intel ha quedado obsoleto ya que dispone de un procesador de tan solo 32 bits mientras que la mayoría de procesadores del momento disponen ya de 64 bits. La placa Jetson TX2 de NVIDIA dispone de una velocidad y eficiencia energética excepcionales ya que tiene hasta 8GB de memoria para lograr que el cálculo y procesamiento sean rápidos. Además, contiene 59,7 GB/s de ancho de banda de memoria.

Aun cuando la placa sea reemplazada, se intentará mantener la conexión con todos los elementos, para ello habrá que analizar las conexiones de cada elemento con la placa de Intel y buscar la manera de realizarlas en la nueva placa.

Figura 3: Placa JETSON TX2

Sensorización: En este caso no se hará uso de la cámara PTZ disponible. Para la navegación autónoma se instalará un sensor rpLidar A2 de la empresa Slamtec, con un campo de análisis de 360º. Este elemento irá conectado vía USB al PC empotrado.

7.2 Software

El software a utilizar puede resumirse en un diagrama de bloques con el fin de que sea más visual:

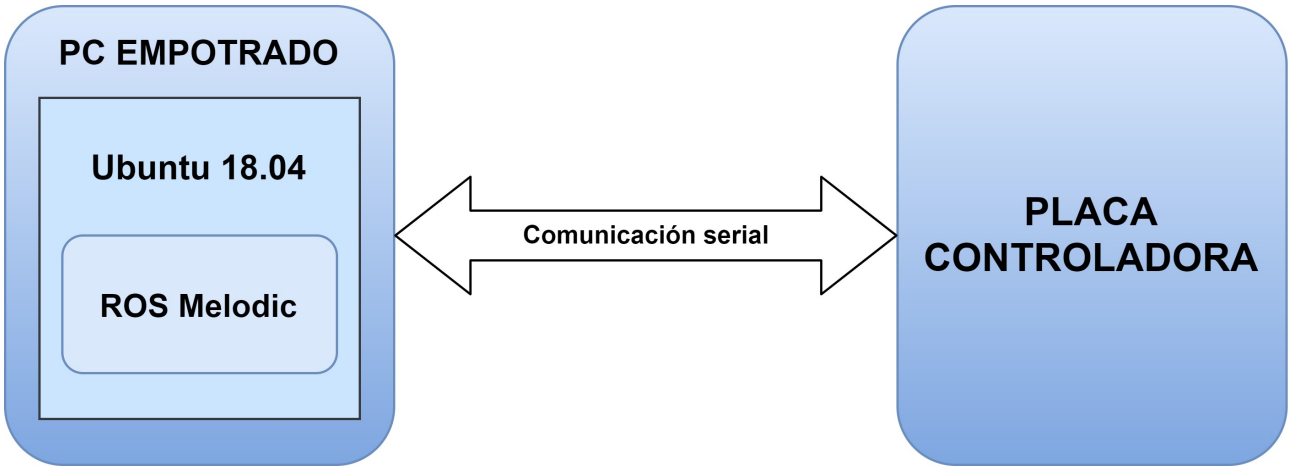

Figura 4: Diagrama de bloques software

En el diagrama pueden apreciarse dos módulos diferentes que corresponden al PC empotrado, en este caso la placa Jetson TX2, y a la placa controladora integrada en el SUMMIT. El PC con sistema operativo Linux, dispondrá de la distribución Ubuntu 18.04 y en esta estará instalada la distribución de ROS Melodic Morenia, sobre la cual se diseñará el software del robot. De esta manera, la Jetson dispondrá de los nodos necesarios a ejecutar para lograr la navegación autónoma deseada.

Mediante la plataforma ROS serán creados los diferentes nodos funcionales. Por una parte, se necesitará un nodo suscriptor que lea la información del entorno que recoge el sensor rpLidar. Esta información será procesada por el ordenador y publicada mediante un nodo publicador para posteriormente transmitirla mediante la conexión serie.

Por otra parte, la controladora irá recibiendo los datos mediante una comunicación serial paralela, procesará tal información, y la enviará tanto a los motores como al servomotor provocando así el movimiento del robot. No se dispone de mucha información sobre esta placa, ya que por parte del fabricante no se han hecho públicos más datos de esta.

8.DISEÑO

Este apartado tratará de describir detalladamente tanto las partes hardware utilizadas para el presente proyecto como la parte software de tales elementos. A su vez será necesario concretar también el tipo de comunicación utilizado entre los elementos principales.

8.1 Hardware.

Por una parte, se hará uso de un **ordenador** para crear una conexión remota al SUMMIT con la cual se podrán lanzar los nodos necesarios para la visualización del mapa y posterior navegación del robot.

El ordenador deberá disponer de un sistema operativo Linux en el que se haya instalado ROS Melodic. Además, tanto el ordenador remoto como el robot deberán estar conectados a la misma red WiFi. De esta manera, el ordenador será capaz de lanzar nodos tanto en la placa JETSON mediante una conexión ssh como en el propio ordenador. El robot, crea un punto WiFi llamado “SUMMIT” sobre el que poder conectarse, la señal de este no es demasiado potente y se ha añadido para su mejora una antena en la parte trasera del robot.

Desde este centro de mando, primeramente se controlará el movimiento del SUMMIT mediante el teclado, con el fin de reconocer el entorno y crear un mapa sobre el que navegar. Finalmente, se cargará dicho mapa y el robot será capaz de navegar autónomamente por la sala mapeada.

Por otra parte, está el **robot terrestre SUMMIT**, principal objeto de este proyecto. Este robot fue creado por la empresa Robotnik, compañía especializada en robots móviles y manipuladores. Dispone de las siguientes especificaciones técnicas: [5]

Tabla 10: Especificaciones técnicas del SUMMIT

ESP. MECÁNICAS		Autonomía	10 h
Dimensiones	720 x 614 x 416 mm	Batería	LiFePO4 15Ah@48V
Peso	65 Kg	Sistemas de tracción	4 ruedas
Capacidad de carga	65 Kg	Motores de tracción	4 x 500W , servomotores brushless
Velocidad	3 m/s	Rango de temperatura	0ºC a 50ºC
Entorno	Interior / Exterior	Máxima pendiente	80 %
ESP. DE CONTROL		Controlador	Arquitectura abierta en ROS PC integrado con Linux
Comunicación	WiFi 802.11n	Conectividad	Interna: USB, RS232 y GPIO Externa: USB, RJ45, 12VDC y batería

Con el kit del robot, viene integrada una **placa controladora** del mismo fabricante, encargada de transmitir señales tanto a los motores, controlando así su velocidad, como al servomotor, haciendo que el robot tome la dirección adecuada.

Primero, es necesario darle voltaje a la placa mediante la conexión de dos pines, uno de ellos conectado a 5V y el otro con conexión a tierra (cables rojo y negro de la imagen). En la configuración anterior esta conexión se realizaba a través del disco duro, pero para este caso se ha decidido conectarlo independientemente. Se ha diseñado conectar los mencionados cables al elemento J21 de la nueva placa, haciendo la conexión de 5V en el pin 4 y la de tierra en el pin 6.

Segundo, los grupos de 3 pines J3 y J5 conectan los 2 motores del SUMMIT. Esa conexión permite que las señales de movimiento se transmitan hasta el motor y que este último realice el movimiento de las ruedas. Además de esta conexión, desde la terminal J8 se conecta la placa controladora al encoder, el cual transforma la energía mecánica en una señal eléctrica para facilitar el control del movimiento. Tercero, mediante el grupo de 3 pines, nombrado J6, se conecta el servomotor, al que se le enviarán las señales de dirección necesarias para la

navegación. Las conexiones mencionadas en este último párrafo, no se han modificado de la configuración anterior.

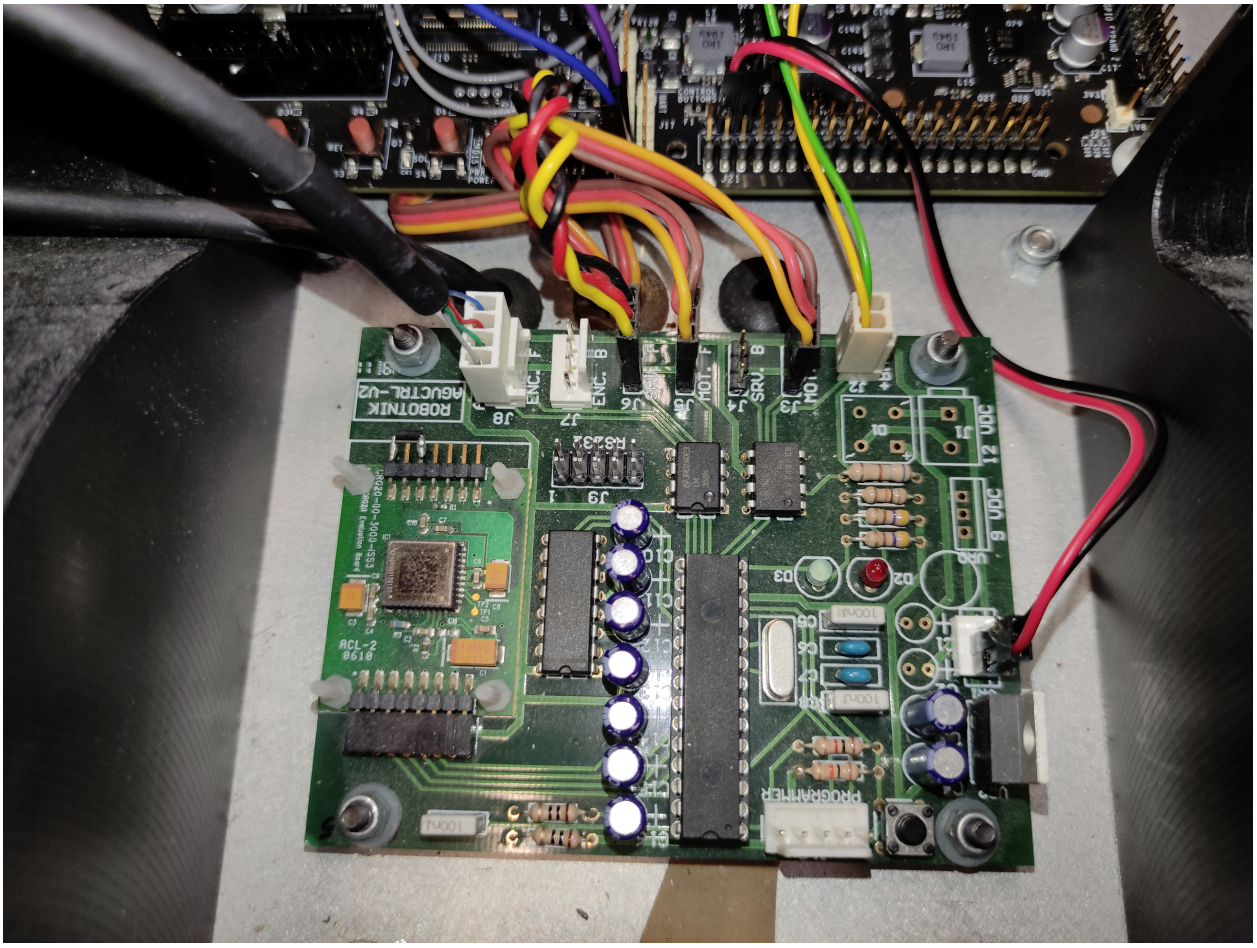

Figura 1: Placa controladora

Por último, el grupo de pines J9 forma una interfaz RS-232 desde la cual viene conectado el PC empotrado Intel D945GSEJT. Este PC, como bien se ha indicado anteriormente en este documento, se va a sustituir por uno más potente, exactamente la JETSON TX2. Será vital diseñar una conexión serie desde la terminal J9 de la controladora a alguna terminal de la JETSON TX2.

El kit de desarrollo **Jetson TX2** es una plataforma de desarrollo funcional para la informática de inteligencia artificial. Está diseñado para ponerlo en funcionamiento en un entorno Linux. La placa portadora, expone muchas interfaces de hardware estándar, lo que permite una plataforma altamente flexible y extensible. Las especificaciones técnicas de la placa pueden resumirse en la próxima tabla: [11]

Tabla 11: Especificaciones técnicas de la JETSON TX2

ESP. TÉCNICAS		Potencia	7,5 W | 15 W
Rendimiento de IA	1.33 TFLOPS	GPU	Arquitectura NVIDIA Pascal con 256 núcleos NVIDIA CUDA
CPU	64 bits, NVIDIA Denver 2 de doble núcleo y complejo de procesadores Arm Cortex-A57 MPCore de cuatro núcleos	Memoria	- LPDDR4 de 8 GB y 128 bits - 59,7 GB/s
Almacenamiento	eMMC 5.1 de 32 GB	Redes	- WiFi Integrado - Ethernet BASE-T 10/100/1000
PCIe	1x4 + 1x1 ó 2x1 + 1x2 PCIe Gen 2, total 50 GT/s	Cámara CSI	- Hasta 6 cámaras (12 a través de canales virtuales) - 12 vías MIPI CSI-2 (3x4 ó 6x2) - D-PHY 1.2 (hasta 30 Gbps)
Codificación de vídeo	- 1x 4Kp60 | 3x 4Kp30 | 4x 1080p60 | 8x 1080p30 (H.265) - 1x 4Kp60 | 3x 4Kp30 | 7x 1080p60 | 14x 1080p30 (H.264)	Decodificación de vídeo	2x 4Kp60 | 4x 4Kp30 | 7x 1080p60 | 14x 1080p30 (H.265 y H.264)
Pantalla	2 DP 1.2/eDP 1.4/HDMI 2.0 multimodo 2 x4 DSI (1,5Gbps/carril)	Mecánicas	- 87 mm x 50mm - Conector de 400 patillas - Placa de transferencia térmica (TTP)

Los conectores de los que dispone pueden verse en el siguiente esquema:

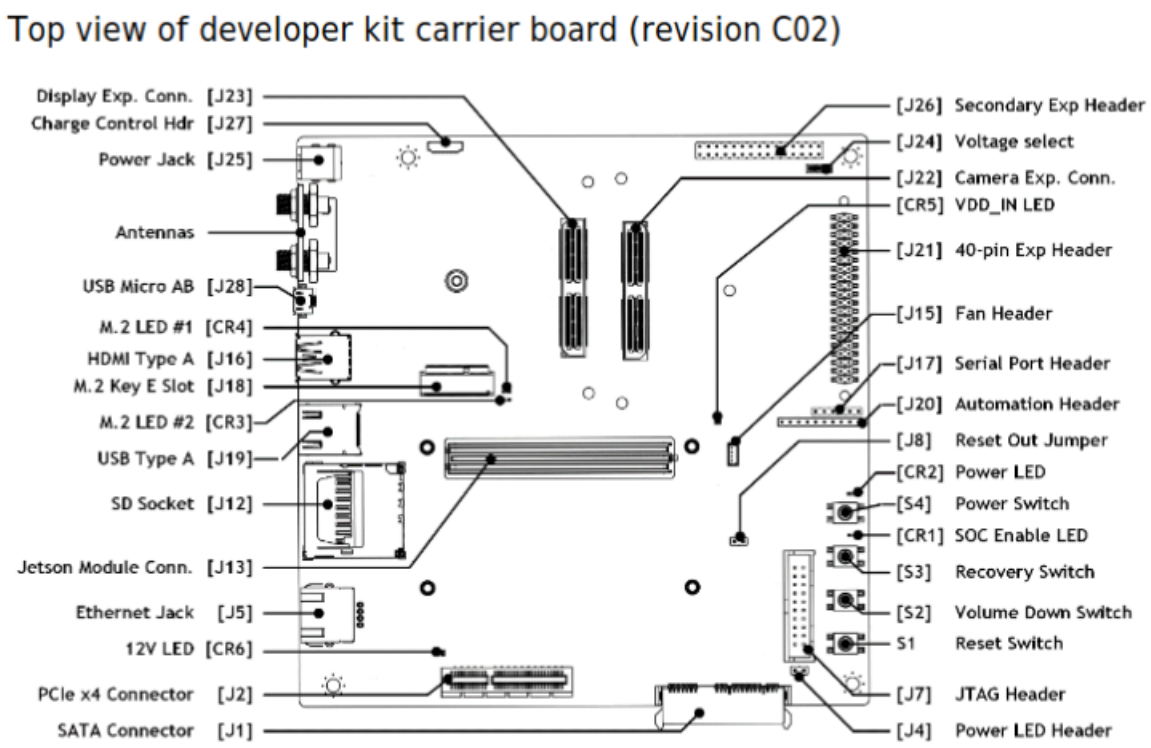

Figura 2: Esquema de la placa JETSON TX2

La **conexión serie** de este nuevo PC con la placa controladora no podrá ser exactamente igual que la anterior ya que la nueva placa tiene una configuración diferente. Primeramente, se estudió crear esta conexión desde la terminal J17 de la JETSON, puesto que en un principio disponía de los pines necesarios para la comunicación serial entre las placas, pero no dió resultado. Finalmente, se ha optado por hacer uso de un cable adaptador USB a RS-232.

Figura 3: Adaptador USB a RS 232

Como complemento para la navegación, se ha añadido el **sensor láser rpLidar A2** fabricado por Slamtec. Dispone de un sistema de medición mediante triangulación láser de bajo costo, rindiendo así sin dificultad tanto en interiores como en exteriores. Sus especificaciones técnicas son las siguientes: [8]

Tabla 12: Especificaciones técnicas del rpLidar A2

ESP. TÉCNICAS		Peso	340 g
Escáner	Rango láser de bajo costo de 360º	Frecuencia de escaneo	- 4000 muestras / s - 10Hz
Distancia	0.15m a 6m	Velocidad de rotación	600 rpm
Resolución angular	0.9º	Alimentación	Requiere al menos 1.5A@5V

Figura 4: Sensor rpLidar A2

La JETSON dispone únicamente de un puerto USB y cómo puede deducirse con los elementos a utilizar se van a necesitar mínimo 2 puertos de este tipo: uno para la conexión serie entre la placa controladora y la placa Jetson TX2, y otro para conectar el sensor rpLidar. Por ello, se hace uso de un **USB hub,** dispositivo que permite concentrar varios puertos USB.

Figura 5: USB hub

Para finalizar con la parte del hardware queda mencionar que en la parte trasera exterior del robot existe un **botón de encendido** para la antigua placa, el cual iba conectado mediante 4 pines al PC antiguo. Dos de los pines corresponden a la activación del led del botón, el cual se enciende cuando el PC está encendido y se apaga cuando está apagado. Los otros dos pines corresponden al circuito de encendido de la placa.

Table 21. Front Panel Header

Pin	Signal	In/ Out	Description	Pin	Signal	In/ Out	Description
Hard Drive Activity LED				Power LED			
1	HD_PWR	Out	Hard disk LED pull-up to +5 V	2	HDR_BLNK_GRN	Out	Front panel green LED
3	HDA#	Out	Hard disk active LED	4	HDR_BLNK_YEL	Out	Front panel yellow LED
Reset Switch				On/Off Switch			
5	Ground		Ground	6	FPBUT_IN	In	Power switch
7	FP_RESET#	In	Reset switch	8	Ground		Ground
Power				Not Connected			
9	+5 V		Power	10	N/C		Not connected

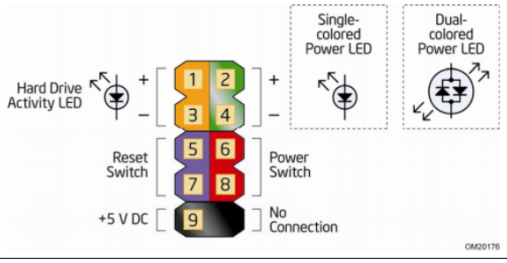

Figure 12. Connection Diagram for Front Panel Header

Figura 6: Esquema de encendido de placa antigua [11]

En el caso de la JETSON TX2, el botón de encendido que se encuentra en la placa quedará dentro del robot una vez este esté montado completamente y no será posible acceder a ello sin abrirlo.

Por tanto, se ha estudiado cómo encender la placa JETSON desde el botón ya existente del SUMMIT. Las conexiones realizadas partiendo de la conexión que existía con el otro PC son:

- Para el encendido del led del botón, se ha conectado el cable que conectaba al pin 2 (+) y el que conectaba al pin 4 (-) respectivamente a los pines 1 y 2 del conector J4 de la placa Jetson TX2.
- Para poder encender la placa desde el botón exterior, se han conectado los cables correspondientes al pin 6 y 8 al conector J20 de la Jetson TX2, a los pines 1 y 2 respectivamente.

8.2 Software

8.2.1 Instalación de la nueva placa JETSON TX2

Antes de empezar con cualquier desarrollo es recomendable comenzar con la configuración básica del sistema para garantizar el funcionamiento adecuado. Hay que tener en cuenta también que es un dispositivo sensible a la electrostática, por lo que será importante desconectar siempre cualquier fuente de energía antes de añadir algún módulo adicional.

Para la configuración de la Jetson TX2, se va a necesitar conectar a esta un USB-hub con el que conectar un teclado y un ratón. Además, se necesitará una pantalla conectada al puerto HDMI de la placa. Haciendo uso de un cable adaptador de Micro-B a USB se conectará la placa a un ordenador externo con el que hacer la instalación pertinente. Una vez logrados estos complementos, se conectará la placa a la corriente, mediante un adaptador de corriente alterna que viene con el kit.

Para encender la placa solo faltaría presionar el botón de power [S4] unos segundos. En cambio, a la hora de configurarlo o cuando se quiera instalar alguna actualización más adelante, se deberá usar el modo llamado “Force Recovery Mode”. Para activar este modo, con el sistema apagado se debe presionar el botón Power [S4], sin dejar de presionar este se presiona el botón Recovery Force [S3] durante un par de segundos. Finalmente se suelta el botón de Recovery Force y

seguidamente el de Power. En este modo, el sistema no estará encendido por lo que no saldrá nada en pantalla.

Una vez aclarado lo anterior se puede comenzar con la instalación. Antes de nada, desde el ordenador con arquitectura Linux, se creará una cuenta en Nvidia y se descargará NVIDIA SDK Manager desde el siguiente enlace: https://developer.nvidia.com/nvidia-sdk-manager

Cuando esté descargado, aparecerá la siguiente ventana desde la cual instalar el SDK Manager:

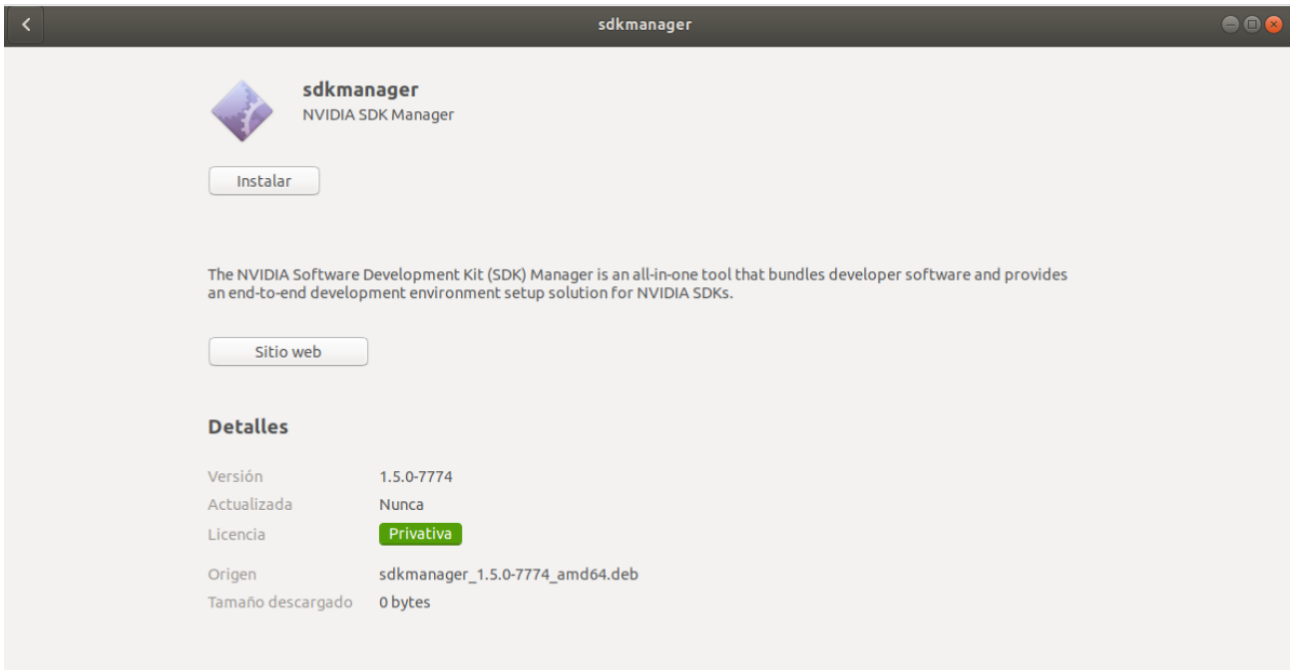

Figura 7: Asistente de instalación SDK Manager

Otra manera de hacer la instalación es utilizando la terminal de Linux y escribiendo lo siguiente:

```
$ sudo apt install ./sdkmanager_[version]-[build#]_amd64.deb
```

Una vez descargado e instalado, se abrirá una nueva terminal y se escribirá el siguiente comando con el que se abrirá el SDK Manager:

```
$ sdkmanager
```

Hecho esto, se deberán seguir los pasos que van apareciendo en pantalla. La placa suele ser automáticamente detectada. Para que esto suceda, la placa deberá estar encendida para lo cual se debe presionar el botón Power unos segundos.

Figura 8: Step 01 en SDK Manager

En el “Step 02” únicamente se aceptan términos y licencias y por último en el “Step 03” se ejecuta la instalación. En mitad de esta, se solicita tener la placa en “Recovery Mode” para poder *flashearla* y es en este momento cuando seguimos los pasos explicados anteriormente para entrar en este modo.

Figura 9: Step 03 en SDK Manager

Después de *flashear*, se deberá encender la pantalla conectada a la placa, donde se solicitará ingresar varios datos como el lugar de residencia y el idioma del nuevo usuario. Entre estos datos habrá que introducir también el nombre del equipo, que se utilizará al conectarse con otros equipos, y el usuario y contraseña para la nueva placa. A continuación, habrá que meter estos nuevos datos de acceso en la siguiente ventana que aparecerá en el ordenador, finalizando con esto la instalación :

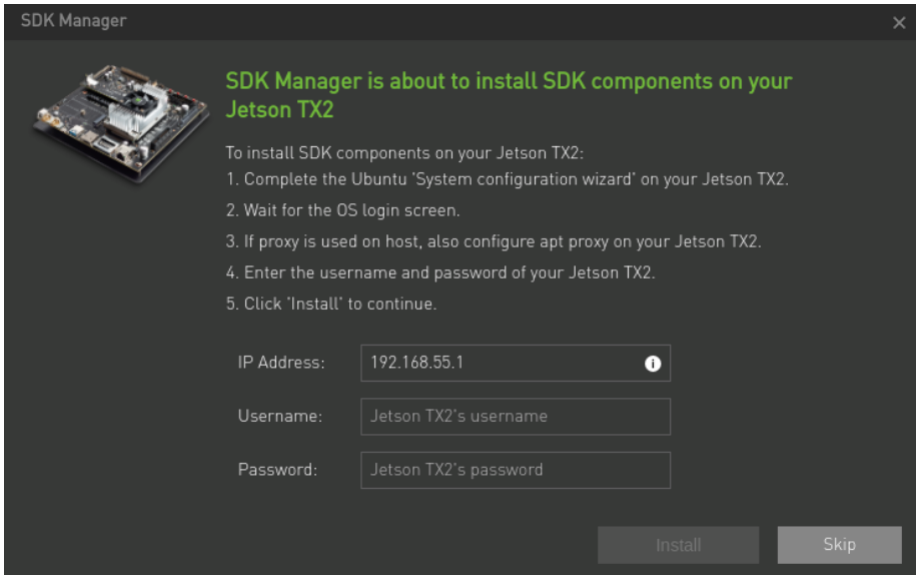

Figura 10: Instalación componentes SDK Manager

8.2.2 Crear un espacio de trabajo

Para configurar el entorno a utilizar será necesario ejecutar el comando “source” en la terminal seguido de la dirección del documento setup.bash. Es recomendable integrar esta acción en el archivo .bashrc para facilitar su configuración. Para ello se ejecuta el siguiente comando en la terminal:

```
$ echo “source /opt/ros/melodic/setup.bash” >> ~/.bashrc
```

Si más adelante fuera necesaria alguna otra configuración se procederá de la misma manera.

Antes de comenzar a trabajar en un proyecto, el primer paso siempre debe ser crear un espacio de trabajo, el cual servirá para todo el proceso. Para ello, en una nueva terminal se escriben los siguientes comandos:

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

De esta manera se habrá creado un directorio llamado “summit_ws”, el cual tiene en su interior 3 carpetas: [13]

- src: Contiene los archivos fuente de los paquetes creados, los paquetes se crean o descargan en esta carpeta.
- devel: Espacio de desarrollo, donde van los ejecutables y bibliotecas antes de instalar paquetes.
- build: Lugar de compilación, dentro de esta carpeta se ejecutan comandos como cmake o make para compilar los paquetes que se han creado en src. Aquí se guardan los archivos caché y otros archivos intermediarios.

Una vez creado el espacio de trabajo, este también habrá que configurarlo como se ha hecho con el entorno de ROS, por lo que añadiremos el archivo correspondiente al “.bashrc”:

```
$ echo “source /home/gici/summit_ws/devel/setup.bash” >> ~/.bashrc
```

8.2.3 Marco teórico de ROS y comandos útiles para este proyecto

Robot Operating System, más conocido como ROS, es un sistema operativo con el que se puede diseñar el software de los robots. Dispone de una amplia gama de librerías y ofrece tutoriales de todo tipo. En este apartado del documento se va a tratar de explicar los conceptos más importantes para comprender el uso de la plataforma. Finalmente, se van a añadir también los comandos más utilizados en el presente proyecto.

Tipos de archivo

- Paquetes: Son la unidad principal de organización del software en ROS. Los elementos que contiene, como pueden ser nodos o librerías, se organizan conjuntamente. El paquete que creado corresponderá al espacio de trabajo.
- Metapaquetes: Son paquetes que representan a un grupo de paquetes. No se van a utilizar.
- Manifiestos del paquete (con terminación .xml): Es la descripción del paquete, contiene datos como el nombre o la versión.
- Repositorios: Se trata de colecciones de paquetes que comparten el mismo sistema VCS (Version Control System) y pueden lanzarse juntos.
- Tipos de mensajes: Los tipos de mensaje definen la estructura de la información enviada. Los mensajes se transmiten entre los nodos mediante los tópicos. Tanto los tópicos cómo los nodos sólo serán capaces de interpretar un único tipo de mensaje, el que se le asigne en su creación.
- Tipos de servicios: Contienen la estructura de la información de los servicios.

Elementos

- Nodos: Son archivos ejecutables que se utilizan para comunicarse con otros nodos, aunque también podrían ejecutarse independientemente. Pueden diferenciarse dos tipos de nodos a la hora de la comunicación entre ellos:
- Nodos publicadores: Son nodos de código secuencial y su función es publicar mensajes en un tópico.

- Nodos suscriptores: Son nodos de código basado en eventos y su función es leer los mensajes que llegan a un tópico.
- Nodo Maestro (ROS Master): Es un nodo proveedor del registro de nombres y de la información de búsqueda de los demás nodos. Gracias a este, los nodos son capaces de identificarse entre ellos e intercambiar mensajes.
- Servicio de parámetros: Es parte del nodo Maestro y almacena los datos en una ubicación central.
- Mensajes: Son estructuras de datos mediante los cuales los nodos se comunican entre sí.
- Tópicos: Cada tópico puede tener nodos publicadores, suscriptores o de los dos tipos unidos a él. De esta manera, los mensajes se transportan en un sistema de publicadores y suscriptores. Estos nodos no son conscientes de la existencia de los demás, sólo del tópico en el que publican o del que leen. Los tipos de mensaje tienen que coincidir para que la comunicación sea exitosa.
- Servicios: Se encargan de interacciones del tipo solicitud - respuesta. No se van a utilizar.
- Bags: Se utilizan para guardar y reproducir mensajes.

Comandos útiles para el proyecto

- Ctrl+Alt+T : Presionando esas tres teclas del teclado se abrirá una nueva terminal.
- Ctrl+Alt+Shift : Con estas tres se abrirá una nueva pestaña en la terminal que ya está abierta.
- Tecla tabuladora: Sirve para recibir opciones desde la terminal, escribiendo los comandos a medias y presionando esa tecla se muestran en la terminal las terminaciones posibles de lo escrito. Es útil cuando no se conoce con exactitud el nombre de un directorio o archivo.
- pwd: Indica el directorio en el que se está trabajando.
- cd: Se utiliza para cambiar de localización.
- ls: Da como resultado los archivos y/o carpetas de los que dispone la localización del momento.

- locate: Se escribe seguido del nombre de un archivo que se quiera localizar y la respuesta es todas las localizaciones en las que aparezca dicho nombre. Por tanto, no haría falta escribir el nombre entero. Aunque si no es algo exacto saldrán demasiadas localizaciones posibles.
- clear: Limpia la terminal sobre la que se escribe, sin parar ejecuciones borra todo lo anterior.
- dmesg: Se utiliza para visualizar los componentes que están conectados a la placa y a que puertos están conectados.
- gedit: Sirve para editar archivos, abriendo los editables desde la terminal. Es necesario disponer de la aplicación para poder usar el comando.
- nano: Hace la misma función que el anterior pero en este caso se abre el archivo en la misma pestaña de la terminal y no es necesaria ninguna aplicación adicional.
- roscore: Encargado de arrancar el ROS Master y el nodo rosout, los cuales estarán en comunicación con los demás nodos. Es importante arrancar este nodo antes que los demás, ya que sino no serán capaces de comunicarse entre sí.
- catkin_make: Compila el espacio de trabajo, en este caso llamado summit_ws. Habrá que ejecutarlo cada vez que se cree un paquete o un nodo nuevo. Hay que ejecutarlo dentro del espacio de trabajo.

```
$ cd /home/usuario/summit_ws
$ catkin_make
```

- rosnode: Funciones diferentes a realizar con los nodos, dependiendo de la palabra que prosiga al comando. Los más utilizados son:
  - rosnode info <Nombre del nodo> : Da la información del nodo que se le indica, como a que tópicos está vinculado o el tipo de mensaje que trata.
  - rosnode list : Da una lista de los nodos que están en ejecución en ese momento.
- rostopic: Funciones diferentes a realizar con los tópicos, dependiendo de la palabra que prosiga al comando. Los más utilizados son:

- rostopic info <Nombre del tópico>: Da la información del tópico que se le indica, cómo que nodos están vinculados a él o el tipo de mensaje que trata.
- rostopic list: Da una lista de los tópicos que están en ejecución en ese momento.
- rostopic type <Nombre del tópico>: Da el tipo de mensaje y por tanto de tópico.

- rosmsg show <Tipo de mensaje>: Provee de las especificaciones del tipo de mensaje que se le indique.
- rosrun <Nombre del paquete> <Nombre del nodo> : Sirve para lanzar un nodo. Para ello hay que indicar el nombre del paquete en el que se encuentra y el nombre del nodo.
- roslaunch <Nombre del paquete> <Nombre del nodo.launch> : Sirve para lanzar nodos. Al contrario que el anterior este puede ejecutar varios nodos al mismo tiempo.
- rqt_graph: Es un nodo que muestra un esquema en el que aparecen tanto los nodos como los tópicos que están en ejecución, facilitando la visualización de las comunicaciones. Para su ejecución se debe escribir:

```
$ rosrun rqt_graph rqt_graph
```

- view_frames: Es una herramienta que sirve para visualizar en un esquema las conexiones de marcos que se crean al lanzar los nodos. Será importante a la hora de relacionar las coordenadas características del robot y del sensor, permitiendo así que trabajen al unísono y lograr la navegación. Para ponerlo en marcha se debe lanzar el siguiente nodo y guardarlo como pdf de la siguiente manera.

```
$ rosrun tf view_frames
$ evince frames.pdf
```

- shutdown : A través de este comando será posible apagar el robot remotamente desde el ordenador. Es necesario que previamente el robot esté conectado mediante una conexión remota y habrá que esperar unos segundos desde que se ejecuta hasta que el robot se apaga.

8.2.4 Paquetes necesarios a instalar para la navegación

Si se conoce el nombre exacto del paquete es posible descargarlo desde un repositorio almacenado en la red y será recomendable hacerlo de esta manera en la medida de lo posible. Para ello se utilizará el siguiente comando:

```
$ sudo apt install <Nombre_paquete>
```

Otra herramienta a utilizar es GitHub, una plataforma colaborativa de alojamiento de código, donde se van actualizando las versiones. Desde la terminal se escribirá el siguiente comando si se quiere descargar un paquete de estos repositorios:

```
$ git clone <https:// … (link del paquete dentro de la plataforma GitHub)>
```

Antes de comenzar con la instalación de paquetes hay que asegurarse de hacerlo dentro de la carpeta src. A fin de ejecutar este proyecto se van instalar los siguientes paquetes:

- El paquete “summit_sim” que contiene todos los datos del robot y diferentes dependencias de este. Entre las cuales se encuentra el archivo llamado “summit_state.launch”, necesario para reconocer las características del robot. Los prerrequisitos para su instalación son “robotnik_purepursuit_planner”, ”robotnik_msgs” y ”robotnik_sensors”.
- Paquete “summit-robot”que contiene “summit_complete.launch” desde el que se carga el modelo del robot al servidor de parámetros, se lanza el nodo “summit_controller” y además es posible lanzar también el nodo correspondiente al sensor láser. En un futuro podría añadirse aquí el lanzamiento de una cámara por ejemplo. Cabe destacar que este paquete se ha descargado desde un repositorio privado de la UPV al que es necesario tener acceso para encontrar el paquete y descargarlo.
- Con el fin de poder mover el robot desde teclado, pudiendo así realizar el mapeo, se instala “teleop_twist_keyboard”.
- Para llevar a cabo la navegación en un mapa será necesario el paquete “summit_navigation”. Al igual que el de “summit-robot” este también ha sido descargado de un repositorio privado. Mediante el ejecutable de este paquete se permitirá cargar el

mapa deseado y navegar sobre él. Los prerrequisitos para instalar este paquete son "summit_sim", "summit_mapping", "summit_localizer" y "teb_local_planner".

- Para la creación del mapa mediante sensores se va a instalar summit_mapping, el cual tiene como prerrequisito los paquetes "gmapping" y "navigation" de ROS.
- El paquete "summit_localizer" contiene el nodo de localización que es utilizado en "summit_navigation" para la navegación del robot. El paquete "navigation" de ROS también es prerrequisito para la instalación de este.
- "robotnik_sensors" aporta las especificaciones necesarias del sensor rpLidar.
- "robotnik_msgs" contiene las definiciones de mensajes y servicios que utilizan los paquetes de Robotnik.
- El paquete correspondiente a la instalación del sensor rpLidar A2 es "rplidar_ros" que se encuentra en el repositorio de GitHub de Slamtec, fabricante del sensor.

Los comandos a ejecutar para la instalación de dichos paquetes son los siguientes:

```
$ cd ~/summit_ws/src

$ sudo apt install ros-melodic-navigation
$ sudo apt install ros-melodic-gmapping
$ sudo apt install ros-melodic-teb-local-planner
$ sudo apt install teleop_twist_keyboard teleop_twist_keyboard.py

$ git clone https://github.com/RobotnikAutomation/robotnik_msgs.git
$ git clone https://github.com/RobotnikAutomation/robotnik_sensors.git
$ git clone https://github.com/GICI-UPV-EHU/robotnik_purepursuit_planner.git
$ git clone https://github.com/GICI-UPV-EHU/summit_sim.git
$ git clone https://github.com/GICI-UPV-EHU/summit-robot.git
$ git clone https://github.com/GICI-UPV-EHU/summit_mapping.git
$ git clone https://github.com/GICI-UPV-EHU/summit_localizer.git
$ git clone https://github.com/GICI-UPV-EHU/summit_navigation.git

$ git clone https://github.com/Slamtec/rplidar_ros.git

$ cd ~/summit_ws
$ catkin_make
```

8.2.5 Cambios en código y código nuevo.

La finalidad de este apartado es la aclaración de los cambios realizados al código descargado en el punto anterior. En la siguiente imagen puede visualizarse el esquema del espacio de trabajo, donde se han instalado todos los paquetes. En rojo se han indicado los ejecutables editados y en verde los que se han creado nuevos.

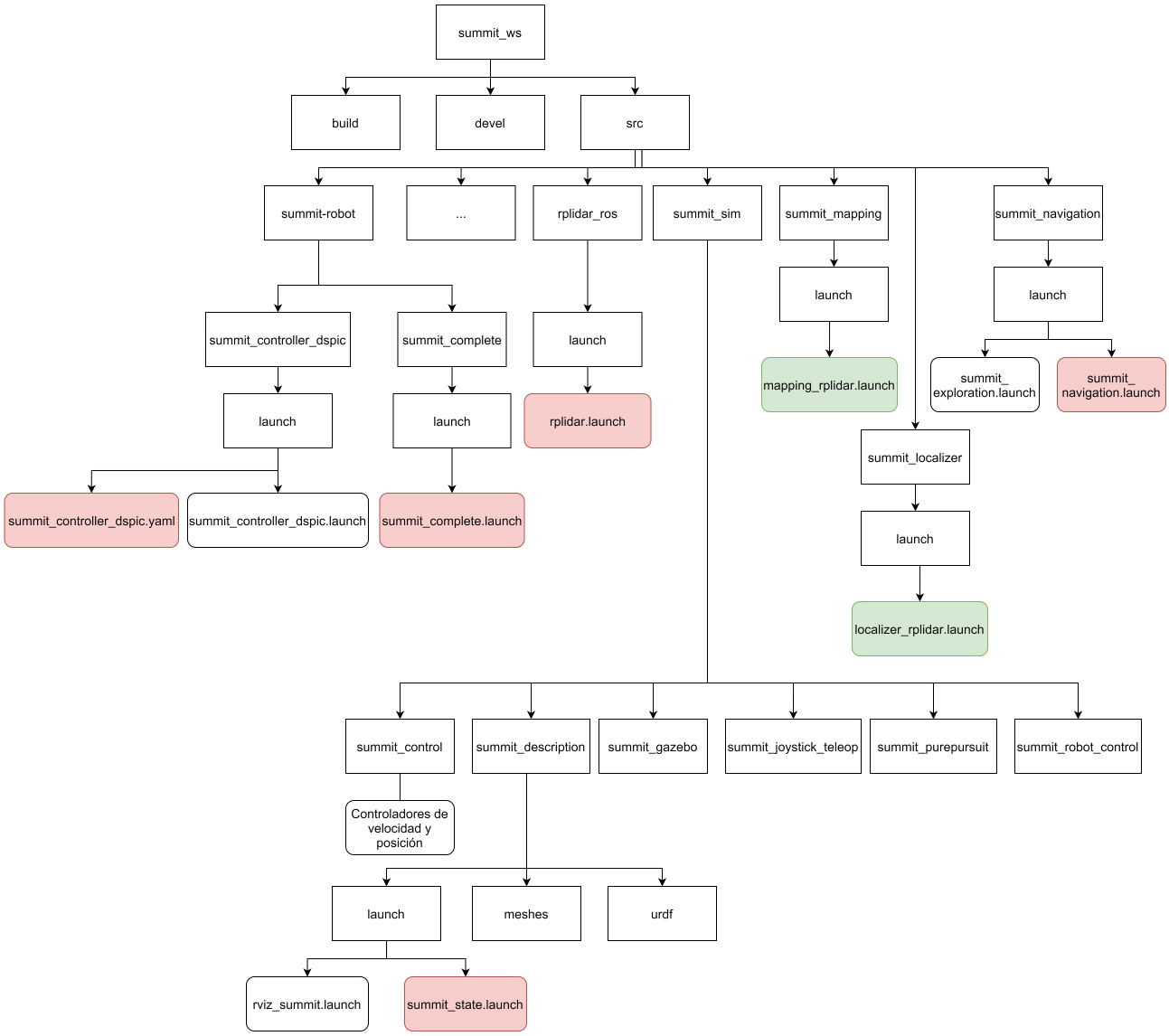

Figura 11: Desglose del directorio summit_ws

summit_controller_dspic.yaml

En este código debe añadirse el nombre del puerto mediante el que se efectua la comunicación serial, en este caso el puerto en el que se haya conectado el adaptador RS-232 a USB.

En un principio ese puerto correspondía a USB0 pero al conectar también mediante USB el sensor rpLidar, uno de ellos se conecta al puerto USB0 y el otro a USB1 aleatoriamente. Para que el código sea válido en todas las ocasiones se ha optado por hacer el siguiente cambio.

Primero se identifican los puertos que corresponden a cada elemento sin tener en cuenta si se le da el nombre de USB0 o USB1. Para ello hay que abrir una terminal y escribir los siguientes comandos:

```
$ cd dev/serial/by-id
$ ls
```

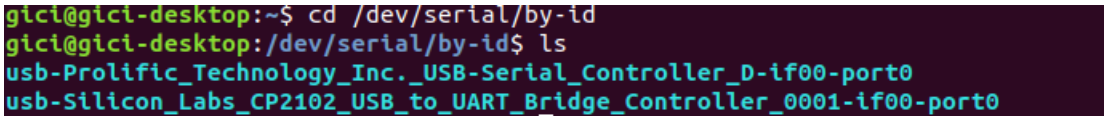

Figura 12: Puertos serie en uso

El primer elemento identificado corresponde al cable RS 232 a USB y por tanto se añadirá en el código "summit_controller_dspic.yaml". Por otra parte, el nombre del segundo, corresponde al sensor láser y se deberá añadir a su código correspondiente, en este caso "rplidar.launch".

El código "summit_controller_dspic.yaml" queda por tanto de la siguiente manera:

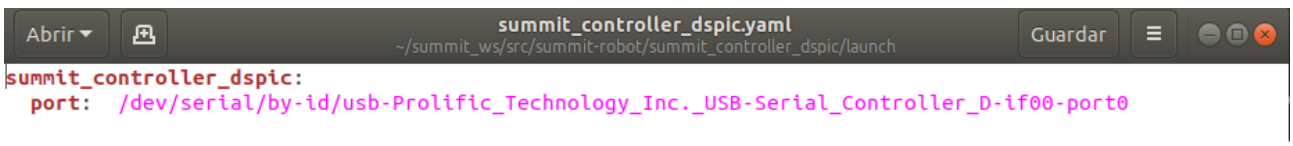

Figura 13: Código summit_controller_dspic.yaml

summit_complete.launch

Por una parte, se ha añadido la línea correspondiente para el lanzamiento del nodo del sensor láser rpLidar A2. Además, se ha comentado la línea que lanza el nodo "summit_state" ya que este se lanzará desde el ordenador remoto mientras que "summit_complete.launch" se debe lanzar desde el robot. Por último, se ha comentado la parte del joystick ya que no va a ser utilizado.

Abrir — summit_complete — ~/summit_ws/src/summit-robot/summit_complete/launch — Guardar

```
<?xml version="1.0"?>
<launch>

  <!-- start rpLidar node -->
  <include file="$(find rplidar_ros)/launch/rplidar.launch" />


  <!-- start hokuyo node -->
  <!-- include file="$(find summit_complete)/launch/hokuyo_laser.launch"/ -->

  <!-- start robot_state_publisher - publish fixed transforms, load robot model to param server -->
  <!--include file="$(find summit_description)/launch/summit_state.launch"/-->

  <!-- start hokuyo node -->
  <!-- node pkg="hokuyo_node" type="hokuyo_node" name="hokuyo_node_front" output="screen"/-->

  <!-- start sound node -->
  <!-- node pkg="sound_play" type="soundplay_node.py" name="sound_play_node" output="screen"/ -->
  <!-- rosrun sound_play say.py 'READY' -->
  <!-- node pkg="sound_play" type="say.py" name="sound_play_say" args="'SUMMIT, READY'"
output="screen"/ -->

  <!-- start axis camera -->
  <!-- include file="$(find axis_camera)/launch/axis_ptz.launch"/ -->

  <!-- start gpsd -->
  <!-- include file="$(find summit_complete)/launch/gpsd_client.launch"/ -->

  <!-- start fotonic camera -->
  <!-- include file="$(find fotonic_3dcamera)/launch/fotonic_3dcamera.launch"/-->

  <!-- start robotnik_arduimu and madgwick filter -->
  <!-- include file="$(find robotnik_arduimu)/launch/robotnik_arduimu_madgwick.launch"/ -->

  <!-- start summit_controller node -->
  <!-- node pkg="summit_controller_dspic" type="summit_controller_dspic"
name="summit_controller_dspic" output="screen"/ -->
  <include file="$(find summit_controller_dspic)/launch/summit_controller_dspic.launch"/>

  <!-- start summit_joystick node -->
  <!-- include file="$(find summit_joystick)/summit_joy.launch" / -->
  <!--include file="$(find summit_pad)/launch/summit_pad.launch"/-->

</launch>
```

XML — Anchura del tabulador: 8 — Ln 1, Col 1 — INS

Figura 14: Código summit_complete

rplidar.launch

En este código se ha añadido el nombre del puerto serie como se ha comentado brevemente en el punto 8.2.5.1. Además, se ha cambiado el valor del frame_id en la línea 7, cambiando “laser” por “rplidar_link” .

Abrir — rplidar.launch ~/Imágenes — Guardar

```
<launch>
  <node name="rplidarNode"          pkg="rplidar_ros"  type="rplidarNode" output="screen">
  <param name="serial_port"         type="string" value="/dev/serial/by-id/usb-
Silicon_Labs_CP2102_USB_to_UART_Bridge_Controller_0001-if00-port0"/>
  <param name="serial_baudrate"     type="int"    value="115200"/><!--A1/A2 -->
  <!--param name="serial_baudrate"     type="int"    value="256000"--><!--A3 -->
  <param name="frame_id"            type="string" value="rplidar_link"/>
  <param name="inverted"            type="bool"   value="false"/>
  <param name="angle_compensate"    type="bool"   value="true"/>
  </node>
</launch>
```

Figura 15: Código rplidar.launch

summit_state.launch

Se han efectuado varios cambios en este editable. En la línea 5, se ha sustituido “summit_xl_description” por “summit_description” y “summit.urdf.xacro” por “summit_rplidar.urdf.xacro”, ya que los nombres que aparecían no correspondían con los paquetes que debían. Por otro lado, en la línea 9, se ha cambiado “state_publisher” por “robot_state_publisher”, puesto que al lanzar el nodo aparecía una advertencia pidiendo tal cambio.

Abrir — summit_state.launch ~/summit_ws/src/summit_sim/summit_description/launch — Guardar

```
<?xml version="1.0"?>
<launch>

  <!-- send summit_xl.urdf.xacro to param server -->
  <param name="robot_description" command="$(find xacro)/xacro.py '$(find summit_description)/urdf/
summit_rplidar.urdf.xacro'" />

  <node name="joint_state_publisher" pkg="joint_state_publisher" type="joint_state_publisher" />

  <node name="robot_state_publisher" pkg="robot_state_publisher" type="robot_state_publisher" />
  <!-- node pkg="robot_state_publisher" type="state_publisher" name="rob_st_pub" >
      <remap from="/robot_description" to="/different_robot_description" />
      <remap from="/joint_states" to="/different_joint_states" />
  </node -->

</launch>
```

Figura 16: Código summit_state.launch

summit_navigation.launch

Este ejecutable, como se indica en el punto 8.2.7, posibilitará la navegación en tantos mapas como se disponga. Aquí es donde corresponde cargar el mapa en el que se desee navegar. Además en este código se efectúa el lanzamiento de "localizer_rplidar.launch".

Abrir — summit_navigation.launch — ~/summit_ws/src/summit_navigation/launch — Guardar

```
<launch>
  <!-- Arguments -->
  <arg name="map_file" default="$(find summit_mapping)/config/mapa_pasillo.yaml"/>
  <arg name="open_rviz" default="true"/>
  <arg name="move_forward_only" default="false"/>


  <!-- Map server -->
  <node pkg="map_server" name="map_server" type="map_server" args="$(arg map_file)"/>

  <!-- AMCL -->
  <!--include file="$(find summit_localizer)/launch/localizer.launch"/-->
  <!--include file="$(find summit_localizer)/launch/localizer_hokuyo.launch"/-->
  <include file="$(find summit_localizer)/launch/localizer_rplidar.launch"/>
  <!--include file="$(find summit_localizer)/launch/localizer_orbbec.launch"/ -->

  <!--GeometryTwist to AckermanDriveStamped-->
  <node name="cmd_vel_to_ackermann_drive" type="cmd_vel_to_ackermann_drive.py"
pkg="summit_navigation" output='screen'/>

  <!-- move_base -->
  <include file="$(find summit_navigation)/launch/move_base.launch">
    <arg name="move_forward_only" value="$(arg move_forward_only)"/>
  </include>
</launch>
```

XML — Anchura del tabulador: 8 — Ln 1, Col 1 — INS

Figura 17: Código summit_navigation.launch

mapping_rplidar.launch

Este código ha sido creado a partir de mapping_hokuyo.launch que es el que se lanzaría con un láser Hokuyo. En la línea 46, se ha sustituido /hokuyo_scan que es el tópico correspondiente al sensor de hokuyo, por /scan que es el tópico del rpLidar. En realidad, con comentar la línea sería suficiente ya que anteriormente se llama "scan".

Abrir — mapping_rplidar.launch — ~/summit_ws/src/summit_mapping/launch — Guardar

```
<launch>
  <arg name="base_frame"  default="base_link"/>
  <arg name="odom_frame"  default="odom" />

  <node pkg="gmapping" type="slam_gmapping" name="slam_gmapping" output="screen">
    <param name="base_frame" value="$(arg base_frame)"/>
    <param name="odom_frame" value="$(arg odom_frame)"/>
    <param name="map_update_interval" value="5.0"/>
    <param name="maxUrange" value="6.0"/>
    <param name="maxRange" value="8.0"/>
    <param name="sigma" value="0.05"/>
    <param name="kernelSize" value="1"/>
    <param name="lstep" value="0.05"/>
    <param name="astep" value="0.05"/>
    <param name="iterations" value="5"/>
    <param name="lsigma" value="0.075"/>
    <param name="ogain" value="3.0"/>
    <param name="lskip" value="0"/>
    <param name="minimumScore" value="200"/>
    <param name="srr" value="0.01"/>
    <param name="srt" value="0.02"/>
    <param name="str" value="0.01"/>
    <param name="stt" value="0.02"/>
    <param name="linearUpdate" value="0.5"/>
    <param name="angularUpdate" value="0.436"/>
    <param name="temporalUpdate" value="-1.0"/>
    <param name="resampleThreshold" value="0.5"/>
    <param name="particles" value="80"/>
  <!--
    <param name="xmin" value="-50.0"/>
    <param name="ymin" value="-50.0"/>
    <param name="xmax" value="50.0"/>
    <param name="ymax" value="50.0"/>
  make the starting size small for the benefit of the Android client's memory...
  -->
    <param name="xmin" value="-1.0"/>
    <param name="ymin" value="-1.0"/>
    <param name="xmax" value="1.0"/>
    <param name="ymax" value="1.0"/>

    <param name="delta" value="0.05"/>
    <param name="llsamplerange" value="0.01"/>
    <param name="llsamplestep" value="0.01"/>
    <param name="lasamplerange" value="0.005"/>
    <param name="lasamplestep" value="0.005"/>
    <remap from="scan" to="/scan"/>
  </node>
</launch>
```

XML — Anchura del tabulador: 8 — Ln 47, Col 10 — INS

Figura 18: Código mapping_rplidar.launch

localizer_rplidar.launch

Al igual que el anterior, se ha creado a partir del editable creado para el láser de hokuyo, por tanto el único cambio necesario ha sido el nombre del tópico, de "/hokuyo_scan" a "/scan".

Abrir — localizer_rplidar.launch — ~/summit_ws/src/summit_localizer/launch — Guardar

```
<launch>
  <arg name="use_map_topic"   default="false"/>
  <arg name="scan_topic"      default="/scan"/>
  <arg name="initial_pose_x"  default="0.0"/>
  <arg name="initial_pose_y"  default="0.0"/>
  <arg name="initial_pose_a"  default="0.0"/>
  <arg name="odom_frame_id"   default="odom"/>
  <arg name="base_frame_id"   default="base_link"/>
  <arg name="global_frame_id" default="map"/>

  <node pkg="amcl" type="amcl" name="amcl">
    <param name="use_map_topic"             value="$(arg use_map_topic)"/>
    <!-- Publish scans from best pose at a max of 10 Hz -->
    <param name="odom_model_type"           value="diff"/>
    <param name="odom_alpha5"               value="0.1"/>
    <param name="gui_publish_rate"          value="10.0"/>
    <param name="laser_max_beams"             value="60"/>
    <param name="laser_max_range"           value="12.0"/>
    <param name="min_particles"             value="500"/>
    <param name="max_particles"             value="2000"/>
    <param name="kld_err"                   value="0.05"/>
    <param name="kld_z"                     value="0.99"/>
    <param name="odom_alpha1"               value="0.2"/>
    <param name="odom_alpha2"               value="0.2"/>
    <!-- translation std dev, m -->
    <param name="odom_alpha3"               value="0.2"/>
    <param name="odom_alpha4"               value="0.2"/>
    <param name="laser_z_hit"               value="0.5"/>
    <param name="laser_z_short"             value="0.05"/>
    <param name="laser_z_max"               value="0.05"/>
    <param name="laser_z_rand"              value="0.5"/>
    <param name="laser_sigma_hit"           value="0.2"/>
    <param name="laser_lambda_short"        value="0.1"/>
    <param name="laser_model_type"          value="likelihood_field"/>
    <!-- <param name="laser_model_type" value="beam"/> -->
    <param name="laser_likelihood_max_dist" value="2.0"/>
    <param name="update_min_d"              value="0.25"/>
    <param name="update_min_a"              value="0.2"/>
    <param name="odom_frame_id"             value="$(arg odom_frame_id)"/>
    <param name="base_frame_id"             value="$(arg base_frame_id)"/>
    <param name="global_frame_id"           value="$(arg global_frame_id)"/>
    <param name="resample_interval"         value="1"/>
    <!-- Increase tolerance because the computer can get quite busy -->
    <param name="transform_tolerance"       value="1.0"/>
    <param name="recovery_alpha_slow"       value="0.0"/>
    <param name="recovery_alpha_fast"       value="0.0"/>
    <param name="initial_pose_x"            value="$(arg initial_pose_x)"/>
    <param name="initial_pose_y"            value="$(arg initial_pose_y)"/>
    <param name="initial_pose_a"            value="$(arg initial_pose_a)"/>
    <remap from="scan"                      to="/scan"/>
  </node>
</launch>
```

XML — Anchura del tabulador: 8 — Ln 1, Col 1 — INS

Figura 19: Código localizer_rplidar.launch

8.2.6 Procedimiento a seguir para lograr un mapa

Habrá dos tipos de ejecutables a lanzar, los que sea necesario lanzar en la placa Jetson TX2, es decir, directamente en el robot, y los que se lancen en el ordenador. Los del primer tipo se lanzarán mediante una conexión remota, para ello en cada nueva terminal que se abra para el robot se deberá ejecutar el siguiente comando:

```
$ ssh gici@192.168.0.2
```

Siendo "gici" el nombre que se le ha asignado al equipo del robot y la serie de números su dirección IP correspondiente.

Antes de ejecutar el comando anterior habrá que seguir un par de pasos previos. Por un lado, para saber la IP de cada dispositivo se utilizará el comando "ifconfig", ejecutándolo tanto en el ordenador como en el robot. En este caso:

- Ordenador remoto: 192.168.0.4
- SUMMIT: 192.168.0.2

Por otro lado, mediante estas IP logradas, se le va a asignar el rol de *master* al PC remoto y el rol de *slave* al SUMMIT. Así, conectando los dos dispositivos a la WiFi generada por el SUMMIT, se logrará la comunicación entre ellos. La asignación de roles se realiza añadiendo las siguientes líneas en los archivos .bashrc. Para la edición del archivo se ejecuta:

```
$ nano ~/.bashrc
```

En el ordenador remoto se añadirán las líneas:

export ROS_MASTER_URI = http://192.168.0.4:11311

export ROS_HOSTNAME = http://192.168.0.4

En el SUMMIT:

export ROS_MASTER_URI = http://192.168.0.4:11311

export ROS_HOSTNAME = http://192.168.0.2

Si se desea acabar con esta conexión no hay más que comentar estas líneas de los archivos .bashrc añadiendo una almohadilla (#) delante de cada línea.

Completada la conexión remota podrán lanzarse los nodos tanto en el ordenador como en el robot desde el mismo ordenador.

Se lanzarán en el robot:

```
$ sudo chown gici /dev/ttyUSB0
$ sudo chown gici /dev/ttyUSB1
$ roslaunch summit_complete summit_complete.launch
$ rosrun teleop_twist_keyboard teleop_twist_keyboard.py cmd_vel:=/summit_controller_dspic/cmd_vel
```

Se lanzarán en el ordenador remoto:

```
$ roslaunch summit_description summit_state.launch
$ roslaunch summit_mapping mapping_rplidar.launch
$ rosrun rviz rviz
```

Cada uno de los nodos se lanzará en una nueva pestaña de la terminal. Además para su correcto funcionamiento se lanzará “summit_state.launch” después de “summit_complete.launch”.

En caso de tener una configuración de rviz guardada, personalizada para este proyecto en concreto, el último comando puede sustituirse por:

```
$ rosrun rviz rviz -d /home/gici/summit_ws/src/summit_navigation/config/summit_mapeo.rviz
```

Siendo “summit_mapeo” el nombre de la configuración y lo que le precede su localización. También es posible abrir rviz sin configuración y posteriormente abrir la configuración guardada desde el menú: File > Open Config.

Para guardar una configuración simplemente hay que añadir los elementos y tópicos que interesen y guardarla desde: File > Save configuration as

A continuación se adjunta la configuración utilizada en este caso:

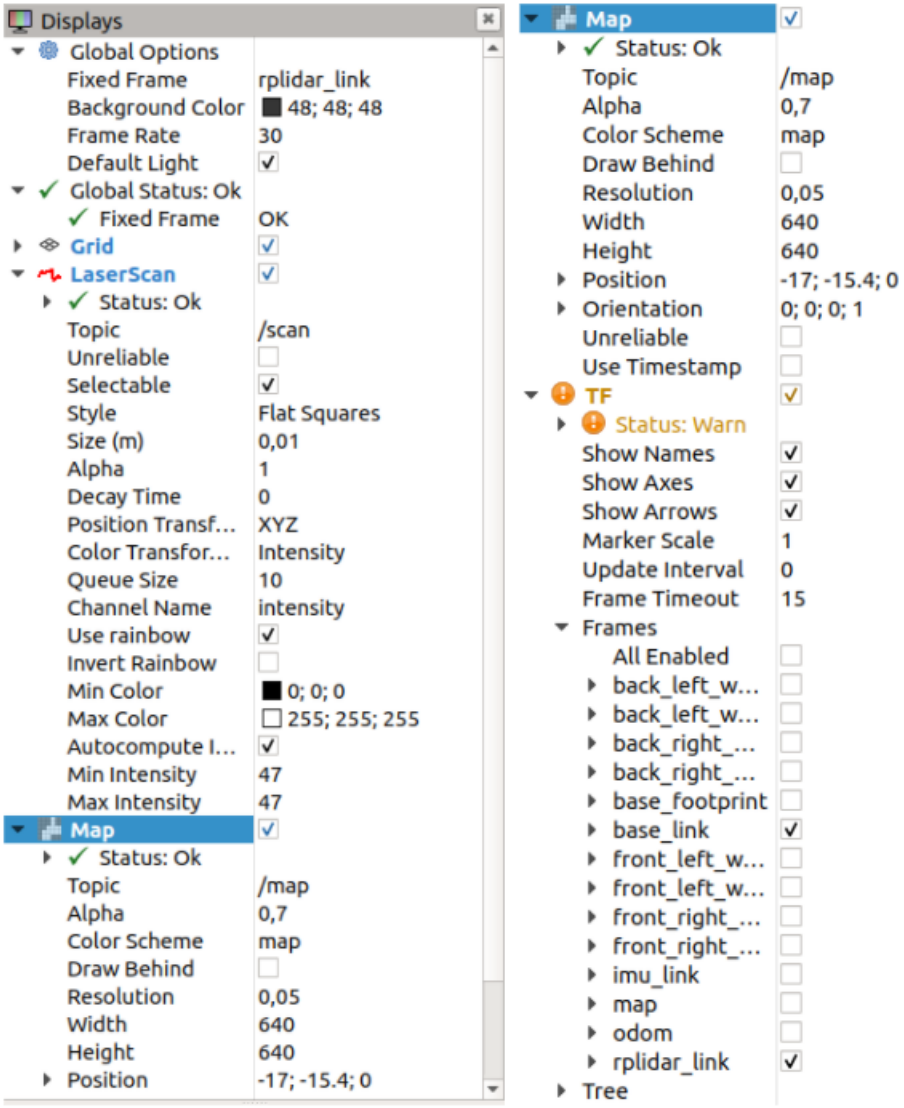

Figura 20: Configuración rviz para mapeo

Con todo en marcha, el robot debería estar listo para ser movido desde el teclado. De esta manera irá moviéndose por la sala dirigido por un responsable hasta completar el mapa deseado. Una vez conseguido el mapa hay que guardarlo para posteriormente poder cargarlo y navegar sobre él.

El procedimiento a seguir es el siguiente:

```
$ roscd summit_mapping/config
$ rosrun map_server map_saver -f <nombre_mapa>
```

Concretando la futura localización del mapa con el primer comando y guardando el mapa con el segundo.

8.2.7 Procedimiento a seguir para lograr la navegación autónoma

En este apartado se van a analizar los pasos que se deben dar para lograr la conducción autónoma del SUMMIT. Por dificultades técnicas no se ha podido desarrollar con éxito dicha función. A pesar de ello, el procedimiento descrito en las siguientes páginas será de utilidad una vez encontrada la solución adecuada.

Habiendo anteriormente creado y guardado el mapa sobre el que se desea navegar, se va a cargar este en el nodo de navegación. Para ello se debe editar el ejecutable “summit_navigation.launch” en el ordenador remoto.

Abrir — summit_navigation.launch — ~/summit_ws/src/summit_navigation/launch — Guardar

```
<launch>
  <!-- Arguments -->
  <arg name="map_file" default="$(find summit_mapping)/config/mapa_pasillo.yaml"/>
  <arg name="open_rviz" default="true"/>
  <arg name="move_forward_only" default="false"/>


  <!-- Map server -->
  <node pkg="map_server" name="map_server" type="map_server" args="$(arg map_file)"/>

  <!-- AMCL -->
  <!--include file="$(find summit_localizer)/launch/localizer.launch"/-->
  <!--include file="$(find summit_localizer)/launch/localizer_hokuyo.launch"/-->
  <include file="$(find summit_localizer)/launch/localizer_rplidar.launch"/>
  <!--include file="$(find summit_localizer)/launch/localizer_orbbec.launch"/ -->

  <!--GeometryTwist to AckermanDriveStamped-->
  <node name="cmd_vel_to_ackermann_drive" type="cmd_vel_to_ackermann_drive.py"
pkg="summit_navigation" output='screen'/>

  <!-- move_base -->
  <include file="$(find summit_navigation)/launch/move_base.launch">
    <arg name="move_forward_only" value="$(arg move_forward_only)"/>
  </include>
</launch>
```

Figura 21: Inclusión del mapa en summit_navigation

En la línea 3, donde se define el “map_file” se debe especificar la localización del archivo, en este caso /summit_mapping/config , y el nombre del archivo, en este caso ‘mapa_pasillo.yaml’ .

En el robot se pone en marcha el nodo “summit_complete” (mediante una conexión ssh como en el caso del mapeo):

```
$ ssh gici@192.168.0.2
$ sudo chown gici /dev/ttyUSB0
$ sudo chown gici /dev/ttyUSB1
$ roslaunch summit_complete summit_complete.launch
```

En el ordenador remoto se pone en marcha el nodo correspondiente a la navegación, donde anteriormente se ha cargado el mapa:

```
$ roslaunch summit_navigation summit_navigation.launch
```

Además, será necesario lanzar en una nueva pestaña el nodo "summit_state" con el que al igual que en el caso anterior se facilitan las caracteristicas del robot:

```
$ roslaunch summit_description summit_state.launch
```

Por último, en el ordenador se ejecuta rviz para visualizar el mapa y ponerle al robot un objetivo al que dirigirse.

```
$ rosrun rviz rviz
```

En este caso, la configuración de rviz será algo diferente, en las opciones globales en la pestaña "fixed_frame" introduciremos "map".

Con la función "2D NavGoal" de rviz es posible marcar un objetivo dentro del mapa. Haciendo click en el botón con ese nombre se facilitará una flecha con la que hacerlo. La dirección de la flecha será la que el robot deba seguir y donde se coloque la flecha marcará hasta qué punto debe llegar.

La aplicación rviz dispone de otra herramienta que puede ser útil antes de la navegación, "2D Pose Estimate". Se trata de otra flecha como la que se usa para la navegación, pero en este caso sirve para indicarle al robot dónde está dentro del mapa, marcando con la dirección de la flecha hacia donde está orientado y donde se coloque la flecha será el lugar exacto en el que se encuentra.

9.DESCRIPCIÓN DE LOS RESULTADOS

En este apartado se van a analizar los resultados obtenidos al finalizar el proyecto, haciendo una comparación con los objetivos planteados al inicio. Cómo se indica en el apartado 3 de esta memoria, el objetivo principal del proyecto era la actualización de los componentes del robot para lograr el reconocimiento del entorno y la posterior navegación autónoma del mismo. En el apartado 8, donde se describe el diseño, puede verse que este objetivo se ha cumplido con éxito.

La sustitución de la placa base ha sido completamente satisfactoria. La conexión de mayor importancia, la conexión serie de la nueva placa base con la placa controladora integrada en el SUMMIT, se ha completado con éxito y funciona a la perfección. En un principio se diseñó conectar estas placas desde otras terminales disponibles en la JETSON TX2, pero por dificultades en el diseño, finalmente se ha optado por hacer uso de un adaptador USB a RS 232. Para completar la conexión, ha sido necesario también añadir dos cables con los que dar voltaje a la placa controladora. Además de esta, se han diseñado varias conexiones más con la placa JETSON TX2, así como la del botón de encendido que se encuentra en la parte exterior trasera del robot, la conexión con el disco duro y la del sensor rpLidar.

Por otra parte, se ha diseñado un software con el que crear un mapa, haciendo uso de la aplicación rviz y de la lectura del entorno que hace el sensor rpLidar integrado. Mediante la puesta en marcha de los nodos mencionados en el apartado 8.2.6 y los comandos adecuados ejecutados desde el teclado, el robot se mueve por la sala deseada mientras el sensor recoge los datos del entorno y rviz va mostrando en pantalla el mapa según este se va creando. En la siguiente imagen puede verse un paso intermedio de este proceso. En la parte izquierda se visualiza el nodo responsable de leer los movimientos indicados desde teclado y transmitirlos al robot. En la parte derecha se observa como se va creando el mapa, correspondiendo los ejes de colores al robot y la línea roja a la lectura del sensor.

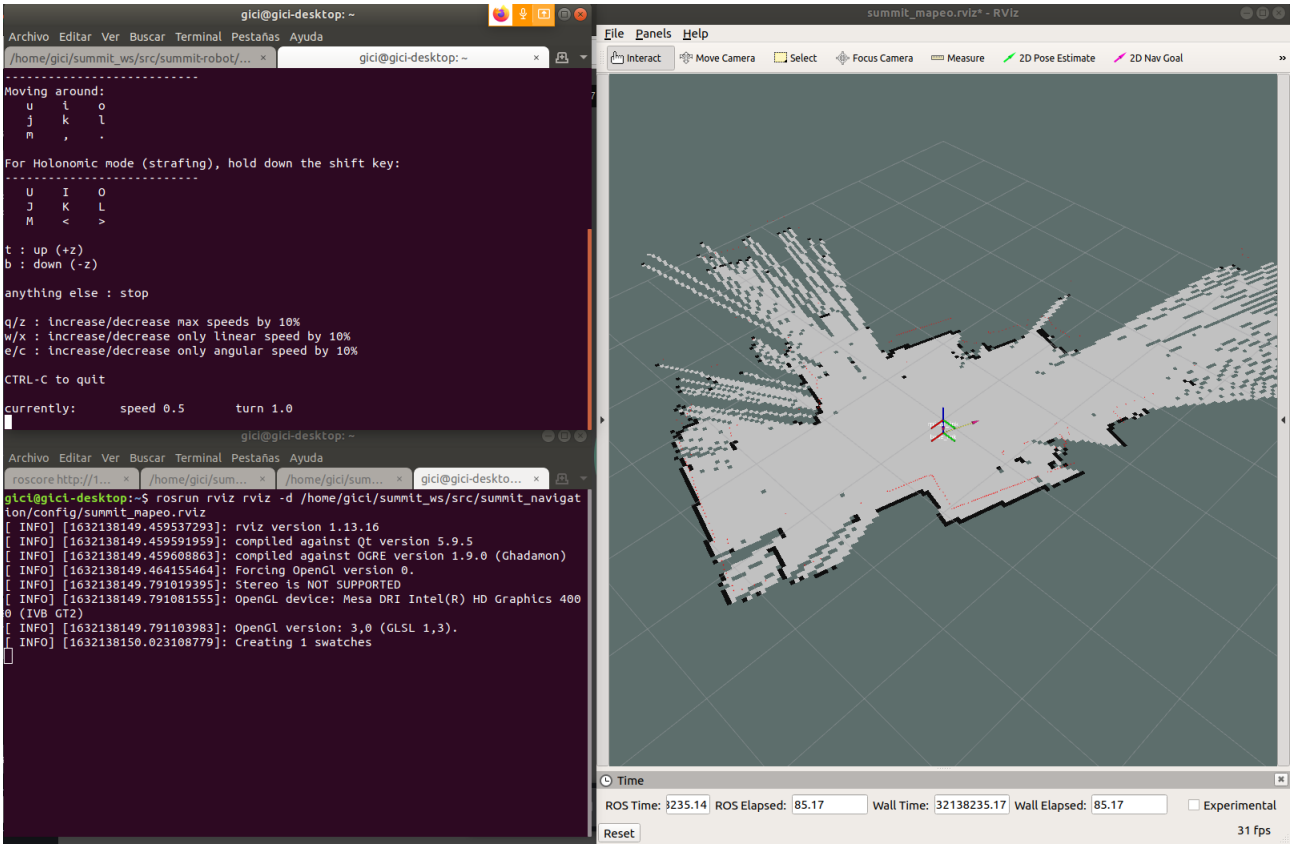

Figura 1: Proceso de mapeo

En el caso que compete, se ha sacado el robot al pasillo del laboratorio y el mapa logrado ha obtenido la siguiente forma:

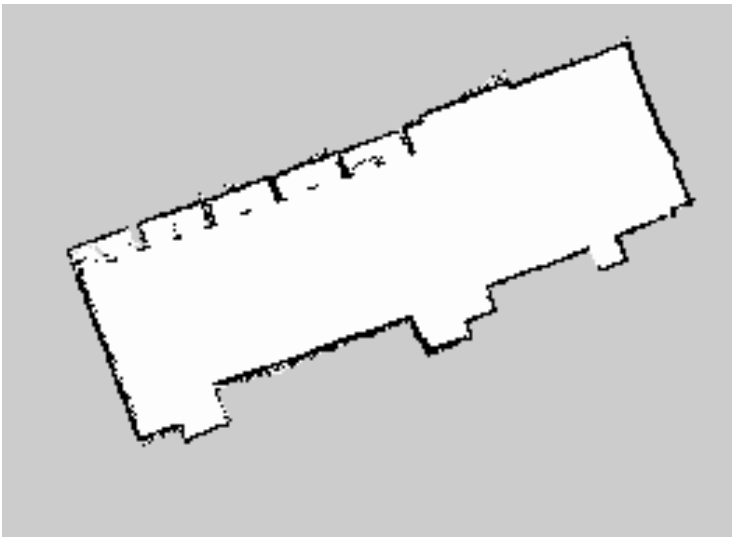

Figura 2: Mapa logrado para posterior navegación

Una vez conseguido el mapa, el objetivo final era introducir este en el código de navegación y situar el robot en él mediante la herramienta "2D Pose Estimate" de la aplicación rviz. A continuación, mediante la herramienta de rviz "2D Nav Goal" que se puede visualizar en la siguiente imagen, al robot se le ordena tomar dirección a un punto exacto del mapa y este avanza hasta tal punto de manera autónoma, evitando los obstáculos que pudiera haber por el camino. El punto al que se le indica desplazarse corresponde con la parte trasera de la flecha.

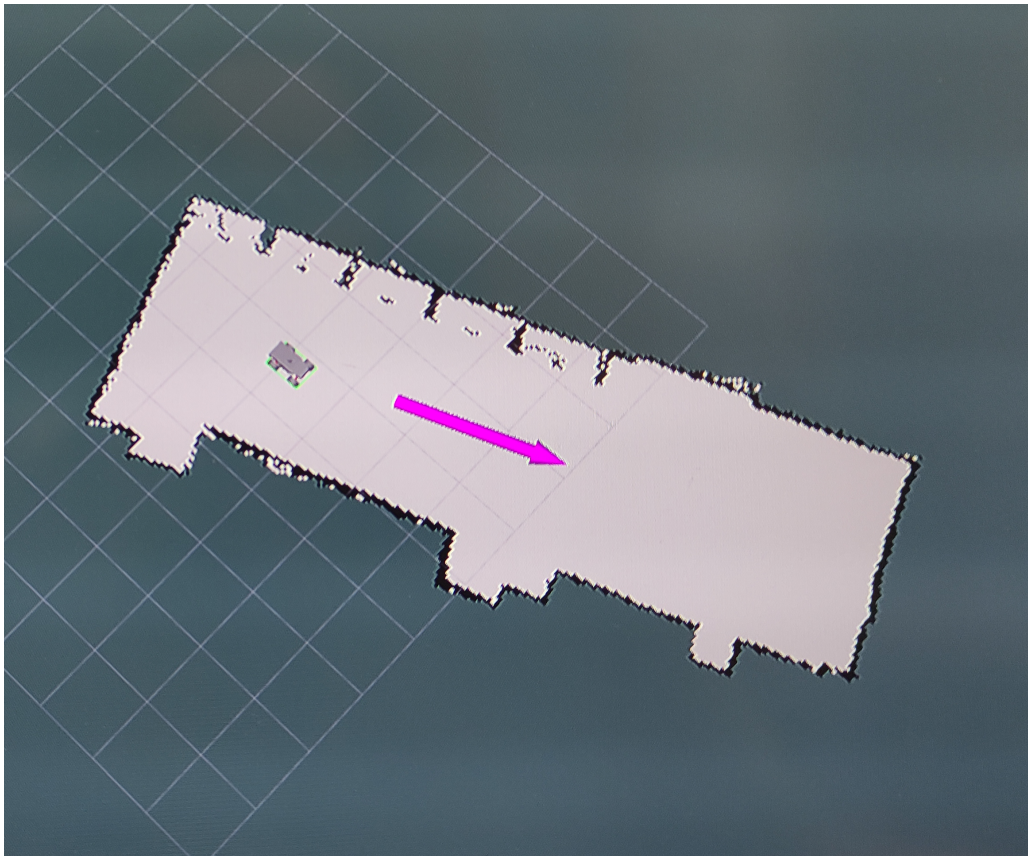

Figura 3: Navegación autónoma en rviz

Para lograr este objetivo principal han tenido que llevarse a cabo los objetivos secundarios que se indicaron. Los resultados conseguidos han sido los siguientes:

- Se ha adquirido fluidez y destreza en la búsqueda bibliográfica y en la localización de fuentes de consulta. Logrando de esta manera la información que ha podido leerse en este documento.

- Se han obtenido conocimientos suficientes para trabajar sobre la plataforma de ROS, habiendo completado los tutoriales de su página oficial y finalmente siendo capaz de comunicarse con el robot.
- La sensorización también ha sido reemplazada. Se ha añadido un sensor rpLidar A2, con el que ha sido necesario estudiar su comunicación. Mediante códigos logrados en los repositorios de GitHub y haciendo las modificaciones necesarias para el proyecto, se ha conseguido ejecutar los nodos tanto que ponen en marcha el sensor para que recoja datos como los que transmiten esos datos a la PC.
- Para la navegación autónoma, se ha utilizado el sensor rpLidar, el cual gracias a los nodos mencionados en el punto anterior crea un mapa del lugar. Posteriormente, este mapa se carga en otro nodo. Ejecutando este último e indicándole al robot su destino se logrará la conducción autónoma.
- La destreza en redactar un informe formal en formato técnico se ha adquirido al escribir la presente memoria.
- Por último, como las comunidades del mundo de la robótica son internacionales, se ha practicado el inglés como método de búsqueda y lenguaje de trabajo.

10.PLAN DE PROYECTO Y PLANIFICACIÓN

Una vez cerrado el diseño del proyecto es conveniente crear un plan de trabajo, dónde se especifican las tareas a realizar y el grupo encargado de cada tarea. A cada una de estas tareas también se le asignará un tiempo a cumplir. Este tiempo se describe en días de trabajo, estimando que al día se trabajan unas 6 horas.

10.1 Descripción del equipo

En la siguiente tabla se describe el grupo de trabajo que se ha encargado del presente proyecto.

Tabla 13: Equipo encargado del proyecto

Nombre	Cargo
Eloy Irigoyen Gordo	Director del proyecto
Nerea Gracia Zarraga	Ingeniera junior

10.2 Descripción de fases y tareas

Se va a dividir este plan en varios paquetes de trabajo a los que se les asignará un plazo y varias tareas. Durante todo el proyecto será necesario ir creando unos entregables para así comprobar que los plazos de ejecución se van cumpliendo. En la siguiente tabla se describen los plazos de los paquetes mencionados:

Tabla 14: Paquetes de trabajo

Paquete de trabajo	Título	Comienzo	Final
WP1	Gestión del proyecto	Día 1	Día 109
WP2	Estudios de alternativas	Día 1	Día 10
WP3	Puesta a punto de los equipos	Día 11	Día 14
WP4	Formación	Día 15	Día 37
WP5	Desarrollo del proyecto	Día 38	Día 89
WP6	Documentación del proyecto	Día 90	Día 109

WP1: Gestión de proyecto

El objetivo de este paquete es que el proyecto disponga de una buena organización y sea ejecutado correctamente. Será importante ir completando a tiempo los entregables fijados inicialmente.

Responsable: Eloy Irigoyen Gordo

Participante: Nerea Gracia Zarraga

Duración: 109 días

WP2: Estudio de alternativas

En este paquete se tratará de identificar las alternativas del equipo a instalar y de elegir entre ellas las más adecuadas para el presente proyecto.

Responsable: Nerea Gracia Zarraga

Participante: Nerea Gracia Zarraga

Duración total: 10 días

- Tarea 2.1: Estudiar las alternativas del software a instalar

  En esta tarea se tratará de decidir cuál de las actuales versiones de ROS es la adecuada para implementar en el proyecto.

  Duración: 3 días

- Tarea 2.2: Estudiar las alternativas en sensorización

  Las variedad de sensorización que puede instalarse es muy amplia, se tratará de reducir las opciones y entre ellas elegir la mejor opción para el proyecto en cuestión.

  Duración: 7 días

WP3: Puesta a punto del equipo.

La puesta a punto trata de la instalación software necesaria para iniciar el proyecto, es decir, la instalación del sistema operativo de Linux, Ubuntu 18.04, y la distribución de ROS Melodic Morenia, en la nueva placa Jetson TX2.

Responsable: Nerea Gracia Zarraga

Participante: Nerea Gracia Zarraga

Duración: 4 días

WP4: Formación

Antes de comenzar a trabajar con el robot, será necesario adquirir los conocimientos suficientes tanto en ROS como en Ubuntu. También se practicará con el simulador Gazebo, probando una simulación creada anteriormente. De esta manera se logrará la familiarización con el robot, lo que será necesario para desarrollar el trabajo de manera fluida.

Responsable: Nerea Gracia Zarraga

Participante: Nerea Gracia Zarraga

Duración: 23 días

- Tarea 4.1: Familiarización con Linux y Ubuntu.

  Para operar con fluidez en el entorno de Ubuntu será necesario conocer los comandos básicos del servidor y practicar con ellos.

  Duración: 4 días

- Tarea 4.2: Familiarización con ROS

  Para conocer la plataforma de ROS y entender sus diferentes herramientas será conveniente practicar mediante los tutoriales de los que se puede disponer mediante su página web.

  Duración: 8 días

- Tarea 4.3: Simulación mediante plataforma Gazebo

  Antes de comenzar a trabajar con el robot SUMMIT se pondrá en marcha una simulación creada anteriormente en otro proyecto. Con esta, se logra la familiarización con el robot, con la sensorización y con la manera de trabajar de los diferentes nodos de los que se partirá para lograr la navegación autónoma. Esta simulación puede ejecutarse en cualquier ordenador que disponga de Ubuntu, ROS y los paquetes necesarios para ello.

  Duración: 11 días

WP5: Desarrollo del proyecto

Este paquete se compone de las tareas necesarias para lograr el objetivo principal del proyecto.

Responsable: Eloy Irigoyen Gordo

Participante: Nerea Gracia Zarraga

Duración: 52 días

- Tarea 5.1: Identificación de las conexiones realizadas con la placa antigua

  Se debe identificar las conexiones existentes con la placa obsoleta para poder realizarlas posteriormente con la placa nueva que se va a instalar.

  Duración: 9 días

- Tarea 5.2: Identificación de los puertos de la placa nueva

  Una vez identificadas las conexiones que se quieren hacer, habrá que investigar en que puertos de la nueva placa pueden hacerse tales conexiones.

  Duración: 7 días

- Tarea 5.3: Instalación de la placa nueva Jetson TX2

  En esta tarea se tratará de realizar las conexiones mencionadas en la tarea 5.2 , reemplazando así la placa que se utilizaba anteriormente.

  Duración: 11 días

- Tarea 5.4: Instalación del sensor rpLidar

  Se trata de instalar el nuevo sensor a la nueva placa ya instalada, para ello descargando los paquetes necesarios.

  Duración: 7 días

- Tarea 5.5: Identificación de tópicos y nodos necesarios

  Aquí se decidirán qué tópicos y nodos se van a utilizar en la navegación autónoma del robot.

  Duración: 8 días

- Tarea 5.6: Diseño de código para añadir la sensorización nueva

  A los paquetes y códigos disponibles será necesario añadirles lo correspondiente al sensor para su uso en la navegación. Para ello se añadirán paquetes en el entorno de trabajo y también líneas de código que llamen a estos paquetes desde los nodos correspondientes.

  Duración: 10 días

WP6: Documentación del proyecto

Esta corresponde a la última fase del proyecto en la que se crea la presente memoria, explicando con detalle el proceso a seguir para lograr los objetivos marcados. El fin del documento será servir de ayuda en un futuro proyecto.

Responsable: Eloy Irigoyen Gordo

Participante: Nerea Gracia Zarraga

Duración: 20 días

10.3 Hitos de la planificación

Con los seis paquetes descritos anteriormente, se deberán ir presentando unos entregables con los que acreditar que los plazos se están cumpliendo. Para ello se definen los siguientes hitos:

Tabla 15: Hitos de la planificación

Hito	Título	Fecha
1	Estudio de alternativas	Día 10
2	Puesta en marcha software	Día 14
3	Simulación ejecutada	Día 37
4	Actualización hardware	Día 71
5	Diseño de código completado	Día 89
7	Entrega de la memoria	Día 109

11.DIAGRAMA DE GANTT

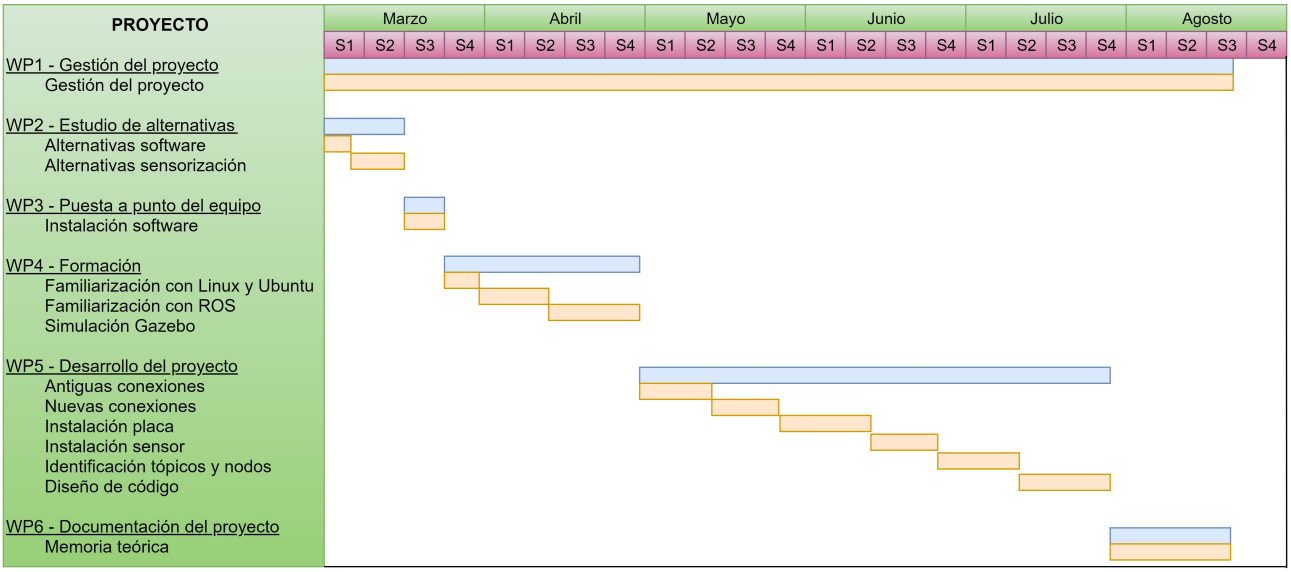

Figura 1: Diagrama de Gantt

12.ASPECTOS ECONÓMICOS

A la hora de valorar los aspectos económicos del proyecto, pueden diferenciarse dos presupuestos diferentes. El primero corresponde al presupuesto de desarrollo que se basa en el tiempo que necesita el equipo para ser capaz de llevar a cabo el proyecto. El segundo en cambio, el presupuesto de ejecución, calcula los costes producidos durante la ejecución del proyecto.

Tabla 16: Presupuesto de desarrollo

CONCEPTO	UNIDAD	NÚMERO DE UNIDADES	COSTE UNITARIO	COSTE TOTAL
Horas internas				14.750,00€
Director	horas	100	60,00 €	6.000,00€
Ingeniera junior	horas	250	35,00 €	8750,00€
Amortizaciones				70,00€
Ordenador	horas	350	0,20 €	70,00€
COSTES DIRECTOS				14.820,00€
COSTES INDIRECTOS (10% de los directos)				1.482,00€
SUBTOTAL				16.302,00€
IMPREVISTOS (10% del subtotal)				1.630,20€
TOTAL				**17.932,20€**

Tabla 17: Presupuesto de ejecución

CONCEPTO	UNIDADES	NÚMERO DE UNIDADES	COSTE UNITARIO	COSTE TOTAL
Horas internas				18.250,00 €
Director	horas	100	60,00 €	6000,00 €
Ingeniero junior	horas	350	35,00 €	12.250,00 €
Amortizaciones				90,00 €
Ordenador	horas	450	0,20 €	90,00 €
Gastos				1.855,78 €
Jetson TX2		1	1.324,90 €	199,00€
rpLidar A2		1	498,68 €	34,50€
Cable RS-232 a USB		1	8,99 €	5,99€
USB hub		1	9,72 €	9,72 €
Adaptador VGA a HDMI		1	13,49 €	13,49 €
COSTES DIRECTOS				20.195,78 €
COSTES INDIRECTOS (10% de los directos)				2.019,58 €
SUBTOTAL				22.215,36 €
IMPREVISTOS (10% del subtotal)				2.221,54 €
TOTAL				**24.436,90 €**

13.CONCLUSIONES

Una vez finalizado el proyecto, en este apartado se describirán las diferentes conclusiones que se han deducido durante la preparación y realización del trabajo.

La primera conclusión a la que se llega es que la implantación de la nueva placa Jetson TX2 aumentará las capacidades del robot, ya que dispone de un procesador más potente que el que se disponía anteriormente. Cuanta mayor potencia, más efectiva será la placa y se conseguirá un mejor rendimiento del robot. Además, es importante disponer de una placa a la altura de las modernidades que van apareciendo en el entorno robótico.

Una segunda conclusión viene de la mano de la anterior, ya que hasta el momento no se había creado la conexión entre la placa controladora del SUMMIT y la Jetson TX2. Crear esta conexión serie no ha resultado nada fácil, ya que además de no haber demasiada información sobre cómo crearla en la Jetson TX2, la información de la que se puede disponer sobre la placa controladora es mínima.

Por último, cabe destacar la importancia de ROS en el mundo de la robótica. Esta plataforma es de gran utilidad para la comunicación y automatización de diferentes dispositivos. Además, resulta de mucha ayuda para la gente que se está iniciando en este mundo, ya que dispone de tutoriales tanto para principiantes como más avanzados con los que poder progresar adecuadamente.

BIBLIOGRAFIA

[1] *Los robots más exitosos de la industria.* (2018, 18 agosto). Master en Industria 4.0 : Universidad de Alcalá-Madrid. https://www.masterindustria40.com/robots-master-robotica/

[2] R. (2019, 21 abril). *Robótica. Qué es la robótica y para qué sirve.* REVISTA DE ROBOTS. https://revistaderobots.com/robots-y-robotica/que-es-la-robotica/

[3] *Robótica móvil. Estudio y caracterización del robot móvil KJunior. Desarrollo de aplicación de robot laberinto.* (2018, septiembre). Universidad politécnica de Cartagena. https://repositorio.upct.es/bitstream/handle/10317/2003/pfc4048.pdf?sequence=1

[4] Gonzalez, D. (2021, 25 enero). *¿Cómo funciona un robot aspirador?* Recuperado 17 de septiembre de 2021, de https://www.euronics.es/blog/como-funciona-un-robot-aspirador/

[5] Robotnik. (2021, 28 julio). *SUMMIT | Robot Móvil Versátil - Robotnik®.* https://robotnik.eu/es/productos/robots-moviles/SUMMIT-es/

[6] México, R. C. (2021, 18 febrero). *Robots Móviles Autónomos (AMR): ventajas, retos y aplicaciones*. CIO MX. https://cio.com.mx/robots-moviles-autonomos-amr-ventajas-retos-y-aplicaciones/

[7] *CLF-1000*. (2018). RobotShop. https://www.robotshop.com

[8] *Venta de productos de robótica*. (2016). ROS Components. https://www.roscomponents.com

[9] Pascual, J. A. (2014, 28 marzo). *Así es Kinect 2.0 para Windows en PC.* ComputerHoy. https://computerhoy.com/noticias/hardware/asi-es-kinect-20-windows-pc-10937

[10] *Intel | Data Center Solutions, IoT, and PC Innovation.* Intel. https://www.intel.com/content/www/us/en/homepage.html

[11] *NVIDIA Jetson TX2: IA de alto rendimiento en primera línea.* (2021). NVIDIA. https://www.nvidia.com/es-es/autonomous-machines/embedded-systems/jetson-tx2/

[12] *Intel Desktop Board D945GSEJT Product Guide.* (2009). INTEL. https://www.intel.com/content/dam/support/us/en/documents/boardsandkits/desktop-boards/945/D945GSEJT/D945GSEJT_ProductGuide02.pdf

[13] *Documentation - ROS Wiki.* (2017). ROS Wiki. http://wiki.ros.org/