Universidad del País Vasco Euskal Herriko Unibertsitatea

BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

INDUSTRIA INGENIARITZA TEKNIKOKO ATALA
SECCIÓN INGENIERÍA TÉCNICA INDUSTRIAL

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TRABAJO FIN DE GRADO

2018/2019

IMPLEMENTACIÓN EN UNA PLATAFORMA MÓVIL DE LA TÉCNICA SLAM MEDIANTE CONTROL INALÁMBRICO

Alumno/Alumna: Merino Martín, Lander
Director/Directora: Oleagordia Aguirre, Iñigo Javier

Curso: 2018-2019

Fecha: 08/07/2019

RESUMEN

En los últimos años, la necesidad de realizar distintas tareas de manera autónoma ha provocado un crecimiento exponencial en el uso de los robots móviles, lo que conlleva a que cada vez se invierta más en la investigación en el campo de la robótica móvil. Gracias a esto se han dado grandes avances en este ámbito, los cuales han sido transferidos a otros campos como la exploración del entorno o la conducción autónoma. Todas las grandes empresas del sector automovilístico trabajan para la creación del coche autónomo, un vehículo que utiliza algoritmos de planificación de trayectoria y evitación de obstáculos, los cuales se han de valer de un sistema de percepción del entorno muy preciso.

El proyecto que se desarrollará en este Trabajo de Fin de Grado (TFG), se basará en la construcción de un robot móvil que, mediante la implementación de la técnica SLAM (Simultaneous Localization and Mapping), genere un mapa 2D del entorno por el que se mueva.

En concreto, el objetivo de este proyecto tratará del diseño y construcción de un robot móvil de bajo coste y de control inalámbrico que explore el entorno mediante el uso de sensores instalados en el robot (encoders, LiDAR, IMU) y el software de ROS Kinetic con el paquete de Hector SLAM, que implementará el algoritmo de la técnica SLAM.

Como objetivo extra del proyecto, se profundizará en el aprendizaje de la tecnología IoT, para lo que se implementarán unos programas que permitan comunicar el sistema ROS con un ordenador con LabVIEW instalado, haciendo uso de la publicación y suscripción de nodos.

LABURPENA

Azkenaldi honetan, era autonomoan ari ezberdinak egiteko beharrak hazkunde exponenziala sortu du robot mugikorren erabilpenean, zeinak eragiten duen gero eta inbertsio gehiago robotika mugikorren arloan. Honi esker, aurrerapen handiak gertatu dira alor honetan. Aurrerakuntza hauek beste eremu batzuetara transferituak izan dira, inguruko ikuskapena edo gidatze autonomoa, besteak beste. Automobilgintza enpresa handi guztiek kotxe autonomoen sorkuntzan dihardute, ibilbide plangintza eta oztopoak ekiditeko algoritmoak erabiltzen dituen ibilgailu bat hain zuzen. Algoritmo guzti horiek inguruaren pertzepzio sistema zehatz batez baliatu behar dira.

Gradu Amaierako Lan (GAL) honetan garatuko den proeiktua, robot mugikor baten eraikuntzan datza, zeinak, SLAM (Simultaneous Localization and Mapping) teknikaren bidez, mugitzen den ingurutik 2D mapa bat sortzen duen.

Zehazki, proiektu honen helburua kostu txikiko robot mugikor bat diseinatzea eta eraikitzea da. Kontrola, haririk gabekoa izango da eta ingurua ikuskatzeko sentsoreak (encoders, LiDAR, IMU) erabiliko dira. ROS kinetik sofwarek, Hector SLAM paketearekin, SLAM teknikaren algoritmoa inplementatuko du.

Aparteko helburu bat, IoT teknologiaren ikasketan sakontzea da. Horretarako, instalaturiko ordenagailu LabVIEW ROS sistemarekin komunikatzeko programa batzuk martxan ezarriko dira, argitalpen eta nodoen harpidetzaren bidez.

ABSTRACT

In recent years, the need to perform tasks autonomously has led to an exponential growth in the use of mobile robots, generating a growing investment in research in the field of mobile robotics. The fast development of this area has been transferred to other fields such as mapping of the environment or autonomous driving. All the major companies in the automotive industry are working on the creation of an autonomous vehicle, a vehicle that uses algorithms to plan the trajectory and avoid obstacles, requiring a very precise perception of the environment.

The project that will be developed in this Final Degree Project (TFG), is based on the construction of a mobile robot that, through the implementation of the SLAM technique (Simultaneous Localization and Mapping), generates a 2D map of its surrounding environment.

Specifically, the objective of this project is the design and construction of a low-cost mobile robot with wireless control, able to explore the environment through the use of sensors installed on the robot (encoders, LiDAR, IMU) and ROS software Kinetic with Hector SLAM package, which will implement the SLAM technique algorithm.

As an additional objective of the project, it is planned to deepen in the knowledge of the IoT technology, working with specific software tools to communicate the ROS system with a computer with LabVIEW, applying the use of the publication and subscription of nodes.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

En la actualidad está en desarrollo la cuarta revolución industrial, también conocida como Industria 4.0. Esta revolución implica la digitalización de los datos relevantes con el fin de conseguir la automatización de procesos para poder predecir, controlar, planear y producir de forma inteligente.

Esta revolución está marcada por la aparición de nuevas tecnologías en la robótica, como las tecnologías cognitivas, la analítica o *Internet of Things* (IoT). Todas estas tecnologías requieren de un flujo continuo y cíclico de información y acciones entre los mundos físicos y digitales, el cual tiene lugar mediante una serie de pasos iterativos conocidos como PDP (*physical to digital to physical*).

La aplicación de estas nuevas tecnologías se fundamenta en el uso intensivo de sensores y dispositivos portátiles, el análisis y la robótica para permitir los procesos y los productos de diversas maneras, así como aportar conocimiento del entorno en el que se mueven. Un claro ejemplo de la importancia de dotar a un robot de un conocimiento fiel y en tiempo real del mundo que le rodea es el desarrollo de los coches autónomos.

Con la intención de introducirse en las nuevas tecnologías de la industria 4.0 se decide hacer este trabajo de fin de grado, para esto se hará uso de sensores innovadores como el LiDAR para obtener un conocimiento en tiempo real del entorno, y de las placas procesadoras Arduino y Raspberry Pi, además del software de ROS (*Robot Operating System*).

Para desarrollar este proyecto se construirá un robot móvil de bajo coste con altas prestaciones para conocer el entorno circundante y su localización en este entorno, permitiendo determinar la trayectoria seguida por el robot. Para este fin, se hará uso de la técnica SLAM (*Simultaneous Localization and Mapping*), un algoritmo que procesa los datos obtenidos por los sensores usados para generar un mapa del entorno y ubicar el robot en él.

Como objetivo extra del trabajo se procederá a comunicar la placa Raspberry Pi con LabVIEW, haciendo uso de la tecnología IoT, para poder visualizar los datos del LiDAR y controlar la trayectoria del robot.

2. OBJETIVOS DEL TRABAJO

Como se ha explicado en el apartado anterior este proyecto está pensado para adquirir conocimientos en las nuevas tecnologías, especialmente en la implementación de la técnica SLAM haciendo uso del sensor LiDAR, y en el uso de la plataforma ROS. Debido a esto el objetivo principal del proyecto es desarrollar un robot móvil de bajo coste con altas prestaciones para la aplicación de estas tecnologías.

Para conseguir este objetivo principal, se han definido los siguientes objetivos de segundo nivel:

- Desarrollo e implementación de un robot móvil.
- Iniciarse y obtener habilidades en el uso de la plataforma ROS.
- Conectar de manera inalámbrica con el robot.
- Aplicar los conceptos de comunicación entre nodos de ROS en LabVIEW.
- Control de los motores DC de manera inalámbrica.
- Control de la trayectoria del robot móvil mediante LabVIEW.
- Elaborar una plataforma en la que se trabaje de forma conjunta con las placas Arduino y Raspberry Pi.
- Desarrollo de odometría mediante la utilización de distintos sensores.
- Analizar el funcionamiento y operatividad de los sensores utilizados.
- Desarrollo de la técnica SLAM en un entorno físico.
- Representar el mapa obtenido por el LiDAR en LabVIEW.

3. DESCRIPCIÓN DE REQUERIMIENTOS

El proyecto a desarrollar está dirigido al aprendizaje de las nuevas tecnologías de una manera práctica. Debido a esto, el requerimiento principal es el de construir un robot móvil capaz de incorporar estas nuevas tecnologías con el fin de ponerlas en práctica, haciendo especial hincapié en la técnica SLAM, la cual tiene como base el sensor LiDAR.

En cuanto a los requerimientos de material y software para la construcción de dicho robot se pueden citar los siguientes:

- Desarrollar un proyecto de bajo presupuesto, con coste de materiales reducido y software libre gratuito.
- Sistema operativo instalado en la placa Raspbery Pi, Lubuntu 16.04.
- Uso de la placa Raspberry Pi 3B+
- Uso de la placa Arduino UNO
- Utilizar la herramienta de ROS para el desarrollo del proyecto.
- Conectar mediante un modo inalámbrico con el robot móvil.
- Definir una IP estática en la Raspberry para poder conectarse de manera inalámbrica siempre que se inicie.
- Ordenador con LabVIEW instalado conectado a la misma red WIFI que la Raspberry.
- Instalaciones de los paquetes indicados en el anexo de instalaciones.
- Seguir las configuraciones del anexo de instalaciones.
- Modificar los archivos mostrados en el anexo de B de manera que queden como se indican.

4. ANÁLISIS DE ALTERNATIVAS

Este proyecto es un primer paso que puede desembocar en distintos proyectos, los cuales están pensados para el desarrollo del Trabajo de Fin de Máster. El uso de la analítica, de las tecnologías cognitivas y de Internet of Things pueden hacer que este proyecto evolucione hacia un robot autónomo que envíe datos en tiempo real a la nube, y sea capaz de actuar de distintas formas gracias al Deep Learning, donde utilizando los datos de la nube el robot aprenda a seguir diferentes criterios frente a distintas situaciones. A continuación se enumeran algunas propuestas de desarrollo de proyectos basados en el trabajo actual:

- Con el fin de conseguir una odometría más fiable y un control de trayectoria mejor, implementar un control de lazo cerrado de las velocidades de los motores gracias a las señales de los encoders hall que tienen incorporados.

- Usando la aplicación de Matlab, implementar distintas técnicas de SLAM, haciendo uso del *Robotic toolbox* entre otros. Esta aplicación también permite implementar mapas de ocupación, los cuales son usados para generar trayectorias desde el punto donde se encuentre el robot hasta el punto deseado.

- Implementar un sistema de detección y evitación de obstáculos haciendo uso de la cámara instalada o el mismo sensor LiDAR.

- Implementar distintas técnicas de SLAM en el mismo entorno de ROS, como Carrtographer ROS, Gmapping …

- En vez de usar el sensor LiDAR, desarrollar el trabajo haciendo uso de la cámara Kinect de Microsoft, la cual es capaz de realizar un mapeo 3D mediante la detección de puntos.

5. ALGORITMO SLAM

5.1. CONCEPTOS PREVIOS PARA LA INTRODUCCIÓN A LA TÉCNICA SLAM

- **ODOMETRÍA:** La odometría es el uso de datos de sensores de movimiento para estimar la posición de un móvil durante la navegación. En la robótica se emplea para conocer la posición relativa del sistema respecto a la posición inicial. La precisión de la odometría es determinante para la correcta estimación de la posición de un móvil basada únicamente en el conocimiento de su trayectoria. Debido a que se trata de un cálculo incremental, inevitablemente los errores son acumulativos.

- **LANDMARKS:** Son los puntos característicos del entorno que pueden ser fácilmente reconocibles y distinguidos por un robot.

  Los requerimientos para usar los *landmarks* son los siguientes:

  - Deben ser estacionarios.
  - Deben ser únicos y distinguibles en el entorno en el que se encuentran.
  - Deben ser abundantes y se tienen que ver desde distintos ángulos.

- **SCAN MATCHING:** Es una herramienta de registro del escaneo láser. Permite escanear la coincidencia entre los mensajes consecutivos de *sensor_msgs/LaserScan* y publicar la posición estimada del láser como un mensaje de tipo *geometry_msgs/Pose2D* o una transformación *tf*.
  No requiere ninguna estimación de odometría proporcionada por otros sensores, por lo que sirve como un estimador de odometría independiente. Es posible incorporar distintos tipos de entrada de odometría (IMU, encoders…) para mejorar la precisión de la estimación de posición.

5.2. CONCEPTO DEL SLAM

En la robótica móvil la mayoría de los robots son programados para desplazarse sobre ambientes controlados o conocidos, si se coloca un robot en un entorno diferente al que fue programado el robot puede llegar a estar completamente perdido.

El SLAM propone una solución a esto. El problema de la Localización y Mapeo Simultáneos busca resolver los problemas que surgen al colocar un robot en un entorno desconocido, y que él mismo sea capaz de construir incrementalmente un mapa consistente del entorno al tiempo que utiliza dicho mapa para determinar su propia localización.

Utiliza una serie de cálculos complejos, algoritmos y entradas de sensores para navegar en un entorno previamente desconocido o para revisar un mapa de un entorno previamente conocido.

5.2.1. Movimiento del robot mediante SLAM

El movimiento del robot mediante SLAM es similar al de una persona tratando de encontrar su posición en un lugar desconocido. Primero la persona observa a su alrededor alguna marca que le resulte familiar. Una vez que la persona reconoce un *landmark*, se puede hacer una idea de

dónde se encuentra respecto al *landmark*. Si la persona no reconoce ningún *landmark* se encontraría perdida. Sin embargo, cuanto más tiempo dedique a la observación del entorno, más *landmarks* podrá reconocer y empezar a generar una imagen mental (mapa) del entorno en el que se encuentra. Esta persona puede que tenga que moverse varias veces antes de hacerse familiar con el entorno desconocido.

Un robot que implemente la técnica SLAM, intenta mapear un entorno desconocido mientras observa donde se encuentra en él. La complejidad aumenta al realizar ambas tareas a la vez. El robot necesita saber su posición antes de poder resolver la pregunta de cómo es el entorno. El robot también tiene que figurarse dónde se encuentra sin la ayuda de tener previamente el mapa del entorno. El SLAM resuelve este problema.

La solución comienza por tener un robot o un vehículo no tripulado. El robot utilizado debe tener datos consistentes para la generación de la odometría. Cuanto mejor sea la odometría mejor podrá estimar el robot su posición a lo largo del tiempo. Esto normalmente se calcula a partir de las posiciones de las ruedas del robot (encoders), y sensor IMU. Hay que tener en cuenta que un pequeño error con las lecturas de la odometría puede provocar que, en un trayecto largo, se produzcan desviaciones importantes respecto a la posición estimada. Estos errores deben de tenerse en cuenta en los algoritmos empleados en el cálculo.

5.2.2. Requerimientos

Uno de los requisitos principales de SLAM es un dispositivo de medición de rango, una forma de observar el entorno alrededor del robot. El dispositivo más usado es el láser scanner como el sensor LiDAR. Los escáneres láser son fáciles de usar y muy precisos, sin embargo, también son muy caros. Debido a su alto coste, se suelen utilizar como alternativa dispositivos como *sonars* o cámaras.

Otro punto clave de SLAM es adquirir datos sobre el entorno del robot. Al igual que los humanos, el robot utiliza *landmarks* (puntos de referencia) para determinar su ubicación mediante sus sensores (láser, sonar…). Un robot utilizara diferentes puntos de referencia para diferentes entornos.

Una vez que el robot ha detectado un *landmark*, puede determinar su propia ubicación extrayendo la información sensorial e identificando los diferentes *landmarks*. Esto puede ser realizado de varias formas, en este proyecto se usa el algoritmo *scan-matching* (coincidencia de escaneo). El robot también puede usar datos de puntos de referencia previamente escaneados (mapa guardado) y compararlos entre sí para determinar su ubicación en el mapa conocido.

En resumen, SLAM es el mapeo de un entorno mediante la interacción continua entre el dispositivo de mapeo, el robot y la ubicación en la que se encuentra. Cuando el robot interactúa con el entorno, no solo mapea el área, sino que también determina su propia posición simultáneamente.

5.2.3. Fundamentos matemáticos

Un componente imprescindible para que el robot realice una navegación correcta es la percepción. Para navegar correctamente el robot necesitará información de su entorno, la cual vendrá dada por los sensores. Los datos de estos sensores estarán corruptos debido al ruido. Esto significa que si el robot solo usa los “raw data” producirá errores que se irán acumulando a lo largo del tiempo. Para eliminar el efecto del ruido en las señales obtenidas por los sensores

es necesario utilizar algún método de filtrado. Un método adecuado es la estimación Bayesiana, que usa modelos probabilísticos para filtrar el ruido.

- Representación en el espacio de estado

Esta representación es conveniente para analizar sistemas con múltiples entradas y salidas, se trata de un modelo matemático de un sistema físico representado en una ecuación diferencial matricial de primer orden. Para sistemas discretos el espacio de estado "*state-space*" se define de la siguiente manera.

$$x_{t+1} = f(x_t, u_t) \quad (1)$$

$$y_t = g(x_t, u_t) \quad (2)$$

Donde x(t) es el vector de estado del sistema, y u(t) y y(t) son las entradas y las salidas del sistema.

El modelo de espacio de estados se formula de la siguiente manera:

$$x_{t+1} = Ax_t + Bu_t \quad (3)$$

$$y_t = Cx_t + Bu_t \quad (4)$$

Donde A es la matriz de estados (relaciona las variables de estado con los estados), B la matriz de entradas, C la matriz de salidas y D la matriz de transferencia (relaciona directamente la entrada con la salida).

Para entender mejor cómo funciona esta representación del sistema en la localización del robot se hará una explicación detallada paso a paso.

Un robot móvil se encuentra en un espacio desconocido empezando desde una localización con coordenadas conocidas x0.

El móvil se desplaza a lo largo del tiempo 't', generando una secuencia de localizaciones del robot 'Xt'. Esta secuencia proporciona su trayectoria (estados).

$$X_T = \{x_0, x_1, x_2, \dots x_T\} \quad (5)$$

Mientras el robot se desplaza adquiere (entradas) mediciones de la odometría (ut) y observaciones del entorno (Zt), que es la información que establece el robot a través de las mediciones entre las características de 'm' y 'xt'.

$$U_T = \{u_0, u_1, u_2, \dots u_T\} \quad (6)$$

$$Z_T = \{z_0, z_1, z_2, \dots z_T\} \quad (7)$$

Se denomina 'm' el mapa real del entorno. El entorno se compone de *landmarks*, objetos, superficies, etc.., y 'm' describe sus localizaciones. El mapa del entorno 'm' se asume que es invariante en el tiempo.

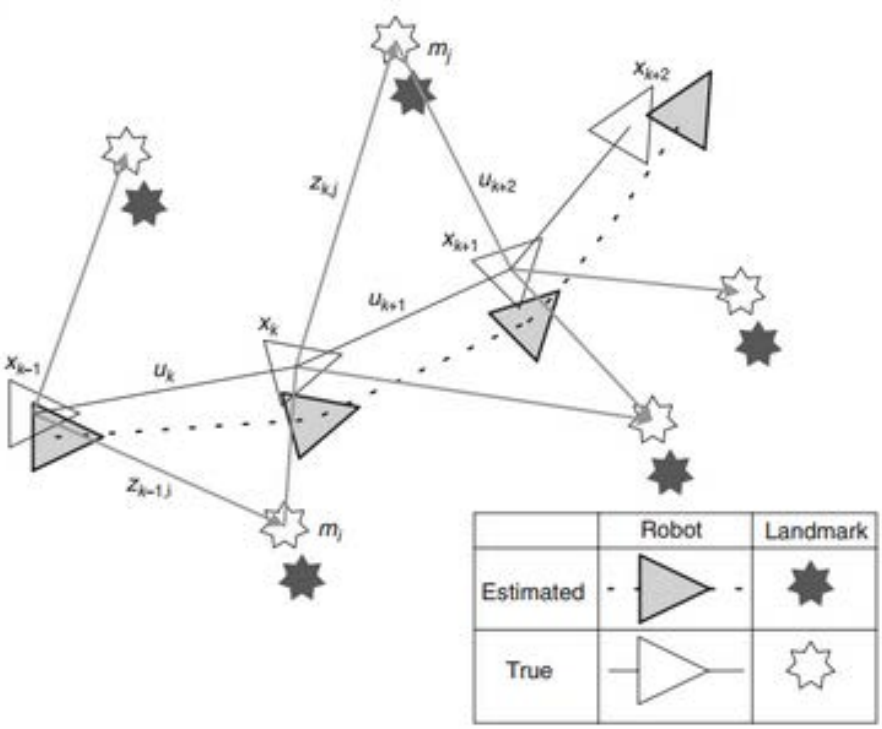

Fig 1. Estimación de la posición del robot mediante landmarks

La literatura describe dos formas del problema del SLAM, ambas de gran importancia. 'full' y 'online' SLAM.

El problema 'full' SLAM consiste en la estimación de la probabilidad a posteriori (probabilidad condicional que es asignada después de que la evidencia o los antecedentes son tomados en cuenta) del recorrido del robot 'Xt' junto al mapa 'm' dada la localización inicial 'x0', las mediciones 'ut' y las observaciones 'Zt'.

$$p(X_T, m \mid Z_T, U_T) \tag{8}$$

El 'online' SLAM intenta recuperar la actual localización del robot en vez de todo el recorrido.

$$p(x_T, m \mid z_{0:T}, u_{0:T}) \tag{9}$$

Dos modelos matemáticos más son necesarios para resolver los problemas del SLAM. Por un lado, un modelo que relacione las mediciones 'zt' al mapa 'm' y a la localización del robot 'xt'.

$$z_t = h(x_t, m) \tag{10}$$

Por otro lado, un modelo que relacione la odometría 'ut' con las localizaciones del robot 'xt-1' y 'xt'.

$$x_t = f(x_{t-1}, u_t) \tag{11}$$

A continuación, se observa un modelo grafico del problema del SLAM.

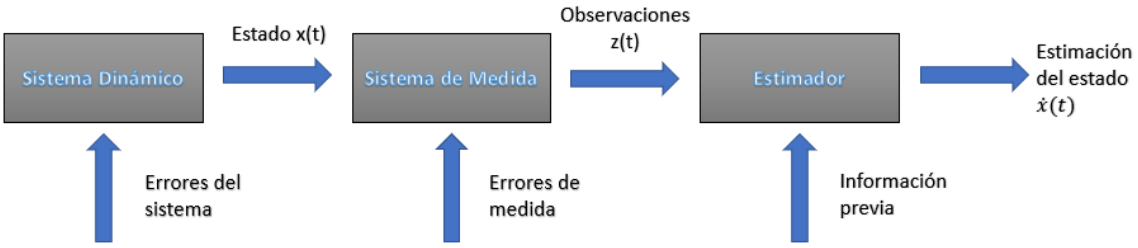

Fig. 2. Formulación matemática en términos del concepto espacio del estado

- Filtro Bayesiano

En la figura anterior se muestra la relación entre controles, estados y mediciones. Sin profundizar en su desarrollo matemático, el filtro Bayesiano es un algoritmo que calcula y representa el impacto que la evidencia tiene en la hipótesis, es decir en la localización del robot.

El filtro extendido de Kalman es una técnica que implementa el filtro Bayesiano, que a diferencia del filtro de Kalman también está destinado a la estimación de sistemas no lineales, debido a que linealiza el sistema mediante series de Taylor.

- Filtro extendido de Kalman (EKF)

El EKF (Extended Kalman Filter) se encarga de estimar la posición del robot a través de la odometría y los *landmarks*. Mediante una etapa de predicción en la que solo se tiene en cuenta la odometría, y una etapa de corrección en la que se utilizan los *landmarks* y las distribuciones de los errores, se haya la nueva posición.

Fig. 3. Funcionamiento del Extended Kalman Filter

A las ecuaciones del apartado de base matemática se le sumará el vector de ruido del proceso (w_t). Mediante la ganancia del filtro de Kalman (K_t) se minimiza la correlación de error a posteriori.

Ecuación de proceso (discreto):

$$A_k = \frac{\partial f}{\partial x}(\hat{x}_{k-1}, u_k) \tag{12}$$

Donde A es la matriz jacobiana de f respecto de X.

Ecuación de medida:

$$H_k = \frac{\partial h}{\partial x}(\hat{x}_{\bar{k}}) \tag{13}$$

Donde H es la matriz jacobiana de h respecto de X.

Ecuación de predicción:

$$P_{\bar{k}} = A_k P_{k-1} A_k^H + Q \tag{14}$$

Donde Q es la matriz de covarianza de ruido del proceso y $P_{\bar{k}}$ es una estimación a priori de la covarianza del error.

Ecuaciones de corrección:

$$K_k = P_{\bar{k}} H_k^H (H_k P_{\bar{k}} H_k^H + R)^{-1} \tag{15}$$

$$P_k = (I - K_k H_k) P_{\bar{k}} \tag{16}$$

Donde K_k se denomina ganancia, P_k es una estimación a posteriori de la covarianza del error, R_k es la matriz de covarianzas de ruido de la medida e I es la matriz identidad.

5.2.4. El proceso de SLAM

El SLAM usa los *landmarks* obtenidos por el sensor láser y la odometría del robot para estimar su posición en el entorno. El encargado de actualizar la estimación de la posición suele ser un EKF, siendo de alta importancia en el algoritmo SLAM. A continuación, se observa el esquema general de funcionamiento de la mayoría de los algoritmos SLAM.

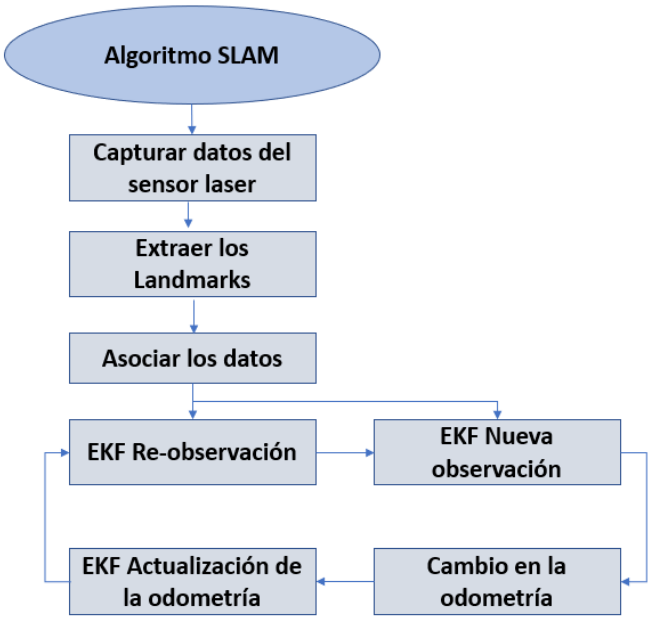

Fig. 4. Funcionamiento del algoritmo SLAM

Los datos del sensor se procesan para extraer los *landmarks*, para posteriormente introducirlos en el filtro Kalman con el fin de actualizar la posición del robot. En caso de no ser un *landmark* y ser simplemente un nuevo punto observación, este se añade al EKF como nueva observación y se empleará más adelante para actualizar la estimación de la posición del robot.

6. DISEÑO DEL ROBOT MÓVIL

6.1. MONTAJE DEL ROBOT MÓVIL

El robot móvil diseñado consta de dos plataformas. En la primera se encuentran los siguientes componentes:

- Una batería recargable de 5 Vdc / 12.000 mAh que alimentará la Raspberry Pi y el LiDAR mediante una conexión USB-microUSB.
- La RaspiCam que se conectara al conector CSI de la Raspberry Pi.
- 2 motores DC de 6V que se conectaran a las salidas A y B del driver L298N.
- Estructura del robot móvil formada por dos ruedas conectadas a los motores y otra rueda loca ubicada en la parte trasera.

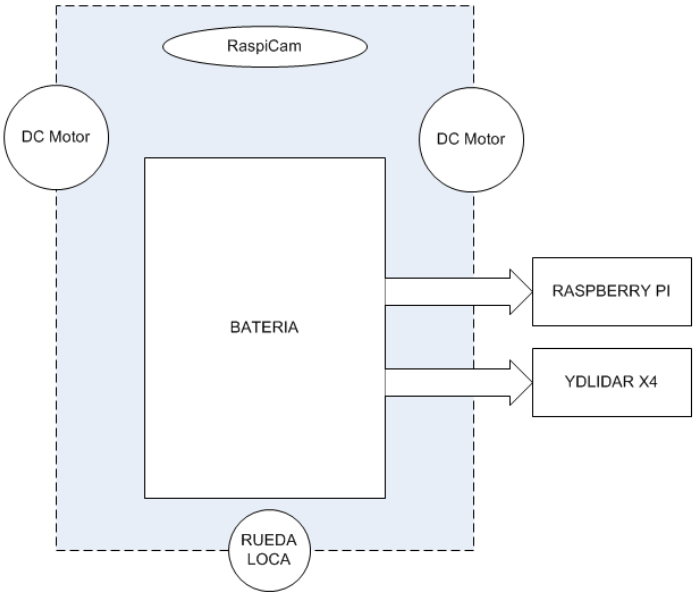

Fig. 5. Estructura de la primera planta del robot móvil

En la siguiente plataforma se encontrarán los siguientes componentes hardware:

- YDLIDAR X4 alimentado con la batería mencionada en el puerto USB_POWER y conectado a la Raspberry Pi mediante conexión USB-microUSB a la entrada USB_DATA.
- Driver L298N alimentado por una pila de 12 V y conectado las salidas A y B a la alimentación de los motores (comprobando que la polaridad de conexión es la correcta para que los dos motores se muevan en la misma dirección en el programa). El control de los estados del driver L298N se hará mediante el Arduino.
- Caja de 8 pilas de tipo AA, total de 12 V.
- Raspberry Pi con sistema operativo Lubuntu 16.04 y ROS Kinetic instalado. Es el encargado de recibir los datos del LiDAR, la cámara y del Arduino, para posteriormente poder procesarlos y usarlos en la implementación de la técnica SLAM. Mediante la conexión WIFI podremos visualizar y editar el contenido de la Raspberry Pi desde el PC y el programa VNC.
- Arduino UNO se usa para el control de los motores y la obtención de datos del sensor IMU. Está conectada vía USB a la Raspberry Pi, la cual cargará el programa al Arduino

con el fin de controlar la trayectoria del robot móvil y obtener mediante los datos del IMU una odometría que reduzca el error en el SLAM.

- IMU conectado al Arduino.

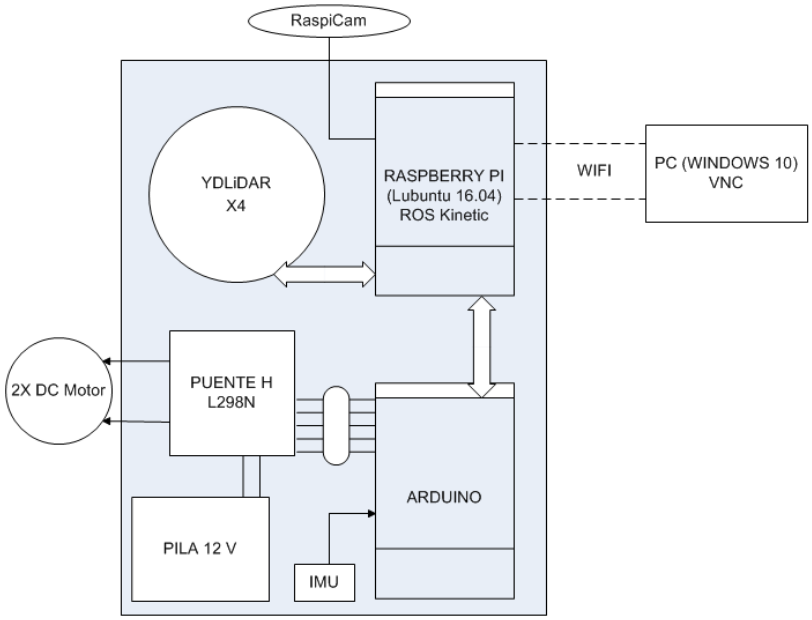

Fig. 6. Estructura de la planta superior del robot móvil

6.1.1. Conexionado

- Conexiones entre los motores DC y el driver L298N

Tabla 1. Conexionado entre motor DC A y driver L298N

MOTOR A	L298N
1- MOTOR -	1- Motor A
6- MOTOR +	2- Motor A

Tabla 2. Conexionado entre motor DC B y driver L298N

MOTOR B	L298N
1- MOTOR -	14- Motor B
6- MOTOR +	13- Motor B

- Conexiones entre el driver L298N y la placa Arduino UNO

Tabla 3. Conexionado entre puente H y Arduino Uno

L298N	ARDUINO UNO
5- GND	GND
7- ENA	3- DO (PWM)
8- IN1	5- DO (PWM)
9- IN2	2- DO
10- IN3	4- DO
11- IN4	6- DO
12- ENB	7- DO

- Alimentación al driver L298N

Tabla 4. Conexionado entre pila y driver L298N

PILA 12 V	L298N
POSITIVO (+)	4- Vin 12 V
NEGATIVO (-)	5- GND

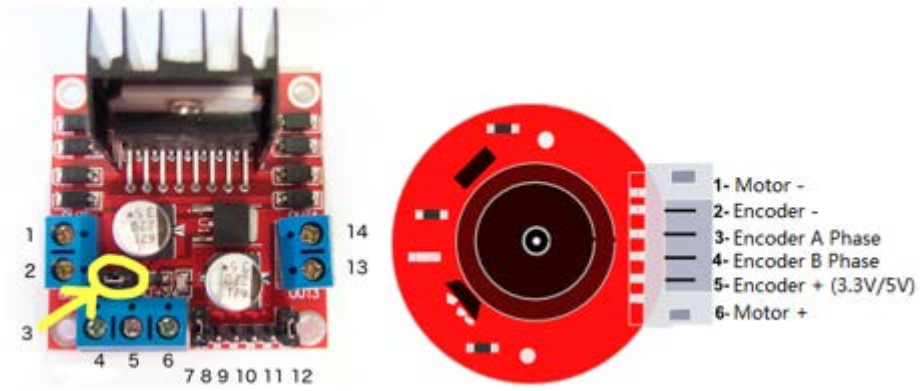

Fig. 7. Pines de conexión de driver L298N y motor DC

6.2. CINEMÁTICA DEL ROBOT MÓVIL

Los dos motores DC que moverán el robot no están directamente conectados al Arduino. Esta tarea es llevada a cabo mediante el driver L298N. No es seguro conectar los motores directamente al Arduino ya que estos requieren de una alta intensidad (hasta 600 mA por motor), y conectarlos directamente podría quemar el chip del Arduino. El Arduino podrá controlar los motores enviando las señales al driver, el cual se encargará de dar la corriente requerida a los motores de manera segura.

El movimiento del robot móvil viene descrito por cinco estados:

- Adelante
- Atrás
- Izquierda
- Derecha
- Parar

No se detallará el modo de operación de los motores DC, pero en caso de querer profundizar se recomienda acudir al link proporcionado en la bibliografía.

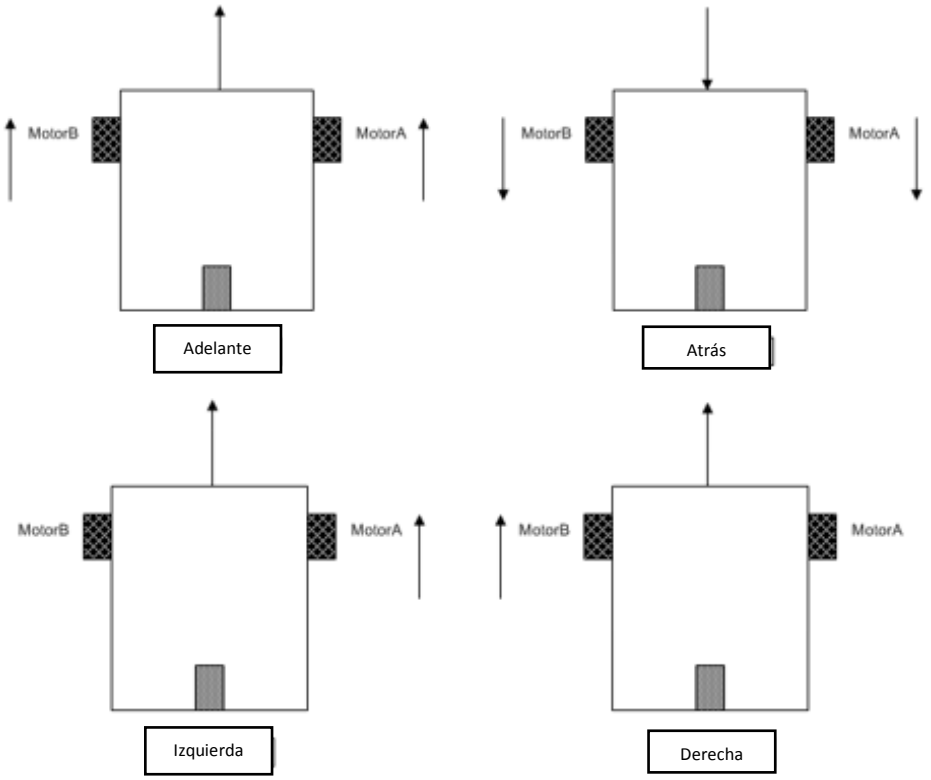

Fig. 8. Cinemática de la trayectoria del robot móvil

El robot móvil se moverá en líneas rectas hacia delante o hacia atrás, y girará accionando solo uno de los motores, dependiendo de la dirección seleccionada. El giro del robot se podría hacer de dos maneras, accionando solo uno de los motores, o accionando ambos motores en sentidos opuestos a la misma velocidad. Esta última opción haría como punto de giro el centro del eje entre las dos ruedas del robot.

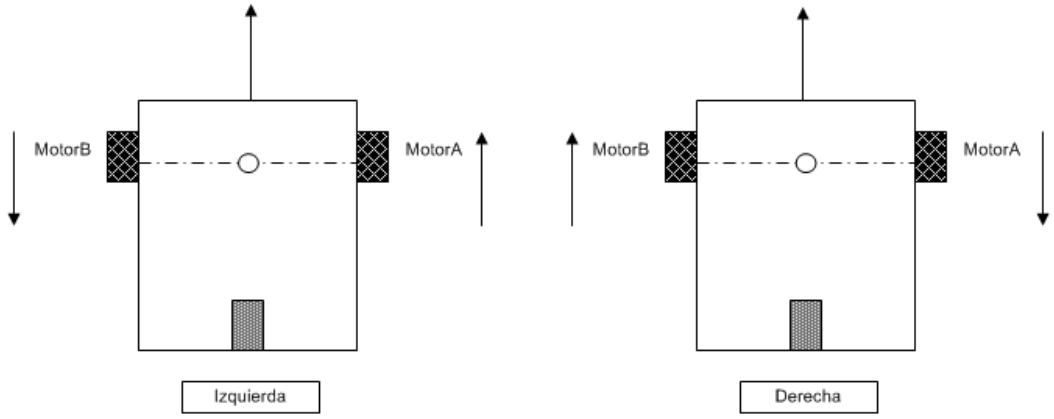

Fig. 9. Alternativa de realización de giros

Para este proyecto en principio se ha seleccionado el accionar solo un motor para la realización de los giros.

Las correspondientes funciones del código Arduino para dirigir el robot tendrán el nombre de los estados mencionados.

7. PLATAFORMA PARA SLAM

7.1. HARDWARE

El "hardware" hace referencia a los componentes físicos que forman parte del sistema. Dentro de estos encontramos componentes eléctricos, electrónicos, mecánicos …

El hardware incluye el chasis del ordenador, los cables, los ventiladores, los periféricos y todos los componentes que se pueden encontrar en un dispositivo electrónico. La Real Academia Española lo define como «Conjunto de los componentes que integran la parte material de una computadora».

En esta sección se describen y analizan los componentes "hardware" que forman el proyecto realizado. Una vez analizado el vehículo se procede a analizar el sistema de procesamiento, donde se expone el proceso de selección del controlador, la valoración de las características y las posibles ventajas y desventajas de este.

Analizada la estructura y el sistema de procesamiento se desglosan los distintos componentes y la función que desempeña cada uno en el proyecto.

Destacar que la placa Arduino usada para el proyecto, en realidad se trata del microcontrolador Elegoo Uno R3, una réplica de menor coste con las mismas características del Arduino Uno, por lo que se explicarán los detalles del Arduino Uno original.

- Descripción y selección de los controladores:

7.1.1. Microcontrolador Arduino Uno

Arduino es una plataforma de creación de electrónica de código abierto, basada en hardware y software libre (no requieren licencias). El software libre son los programas informáticos cuyo código es accesible por cualquier usuario. Arduino ofrece la plataforma Arduino IDE (Entorno de Desarrollo Integrado), un entorno de programación para crear aplicaciones para las placas Arduino.

Por su parte la placa Arduino proporciona todo lo necesario para conectar periféricos a las entradas y salidas del microcontrolador, y promueve la filosofía "*learning by doing*", que viene a decir que la mejor forma de aprender es practicando.

- Funcionamiento

El Arduino es una placa basada en un microcontrolador ATMEL. Los microcontroladores son circuitos integrados en los que se pueden grabar instrucciones, las cuales se programan en el entorno Arduino IDE. Estas instrucciones permiten crear programas que interactúan con los circuitos de la placa.

La placa cuenta con dos interfaces. La interfaz de entrada permite conectar al Arduino diferentes tipos de periféricos, los cuales enviarán datos al microcontrolador para que este pueda procesarlos. La interfaz de salida se encarga de llevar la información que se ha procesado en el Arduino a otros periféricos.

- Tipos de placa

Arduino es un proyecto y no un modelo, por lo que existen diferentes tipos de placas que se ajustan a distintas necesidades del usuario. Entre los distintos modelos más utilizados se encuentran las siguientes placas: Arduino Uno (ATmega328, 14 pines digitales de entrada/salida), Arduino Mega (ATmeg1280, 54 pines digitales de entrada/salida), Arduino Leonardo (ATmega32u4, 20 pines digitales de entrada/salida).

Para la implementación del proyecto se utiliza el Arduino Uno, ya que se disponía de esta placa con anterioridad. Sin embargo, una vez montados los componentes sobre el chasis, la utilización del Arduino Nano se ajustaría mejor al proyecto, debido a su pequeño tamaño y características similares.

- Características del Arduino Uno:

Se trata de la placa más usada y con mayor documentación de toda la familia Arduino. Es una placa basada en el microcontrolador ATmega328P. Tiene 14 pines de entrada/salida digital (6 pueden usarse como PWM), 6 entradas analógicas, un cristal de 16Mhz, conexión USB, conector Jack de alimentación, terminales para conexión ICSP y un botón de reseteo.

Fig. 10. Microcontrolador Arduino Uno

7.1.2. Microprocesador Raspberry Pi

Es una computadora de bajo coste y tamaño reducido. La Raspberry Pi es la placa de un ordenador simple compuesto por un SoC, CPU, memoria RAM, puertos USB, conectividad de red, ranura SD, conexiones para periféricos…

Para ponerlo en marcha hay que conectar periféricos de entrada y salida para poder interactuar como una pantalla, un ratón y un teclado, y grabar un sistema operativo para Raspberry Pi en la tarjeta SD. Una vez hecho esto bastará con alimentarla para poder empezar a usarla.

La Raspberry Pi que se utiliza en el proyecto es la Raspberry Pi 3B+, el último modelo. Entre sus mejoras respecto al anterior modelo destacan una CPU “acelerada” y soporte WiFi de doble banda.

La elección de este dispositivo se basa en la notable mejora de conectividad a internet además de tener un precio similar el de sus predecesoras.

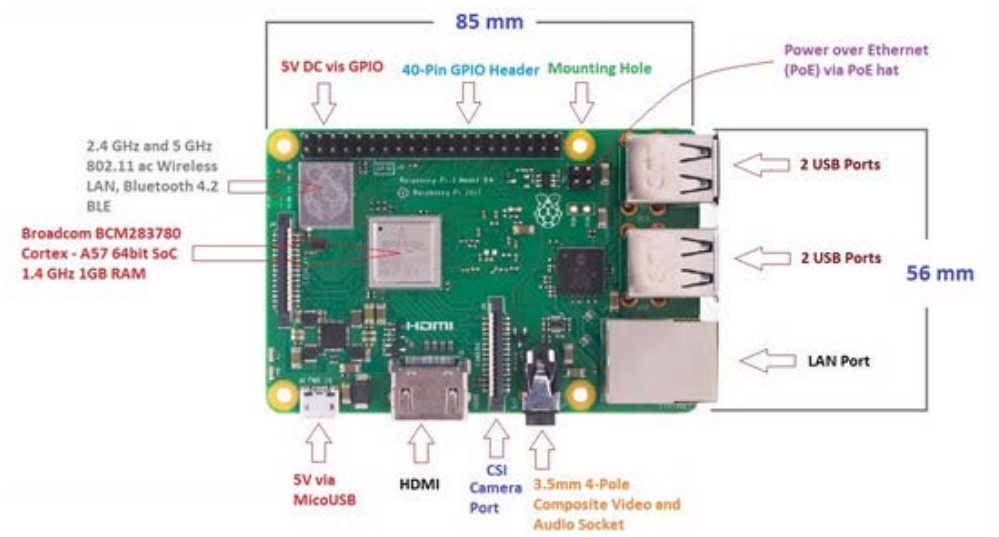

Fig. 11. Microprocesador Raspberry Pi 3B +

Las características del modelo elegido son las siguientes:

- Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC @ 1.4GHz
- 1GB LPDDR2 SDRAM
- 2.4GHz and 5GHz IEEE 802.11.b/g/n/ac wireless LAN, Bluetooth 4.2, BLE
- Gigabit Ethernet over USB 2.0 (maximum throughput 300 Mbps)
- Extended 40-pin GPIO header
- Full-size HDMI
- 4 USB 2.0 ports
- CSI camera port for connecting a Raspberry Pi camera
- DSI display port for connecting a Raspberry Pi touchscreen display
- 4-pole stereo output and composite video port
- Micro SD port for loading your operating system and storing data
- 5V/2.5A DC power input
- Power-over-Ethernet (PoE) support (requires separate PoE HAT)

En este proyecto se utilizarán 2 puertos USB, para conectar el LIDAR y el Arduino para una conexión serial que permita el intercambio de datos entre ambos dispositivos. El micro USB se usará para alimentar la Raspberry (5V/2.5A DC), y el Puerto CSI para la conexión de la Raspberry Pi camera.

o Utilización conjunta de ambas placas

La Raspberry Pi es perfecta para aplicaciones de software mientras que la simplicidad del Arduino lo hace adecuado para proyectos de hardware.

La capacidad analógica y en tiempo real que ofrece Arduino le permite trabajar con casi cualquier tipo de sensor o chip, mientras que la Raspberry Pi y su desarrollo en el software de Linux le permite crear aplicaciones software de alta complejidad y permite procesar varias tareas al mismo tiempo.

Debido a esto, la unión de ambas placas permite desarrollar casi cualquier proyecto electrónico, soportando Arduino la parte sensorial, mientras que la Pi distribuye direcciones.

En el caso del proyecto llevado a cabo, el Arduino se utiliza para accionar motores mediante un puente H y para la lectura del sensor IMU, mientras que la Raspberry se utiliza para tener una conexión a Internet dándola un uso como mini computadora capaz de ejecutar programas y enviar y recibir direcciones del Arduino.

- Descripción de los demás componentes hardware:

7.1.3. IMU

La Unidad de Medición Inercial o IMU es un dispositivo electrónico cuyo objetivo es obtener mediciones de velocidad, rotación y fuerzas gravitacionales, usando una combinación de acelerómetro, giroscopios y a veces magnetómetros.

Los datos recolectados por los sensores de una IMU permiten a un microcontrolador seguir la posición del robot, usando un método conocido como navegación por estima.

- Funcionamiento

Cada uno de los sensores agregados aporta una función para lograr un único resultado. El giroscopio se encarga de medir los giros realizados, mientras que el acelerómetro mide la aceleración lineal que se realiza hacia cualquier lado y por último el magnetómetro obtiene información acerca del norte magnético para estar siempre ubicado con respecto al campo magnético de la tierra.

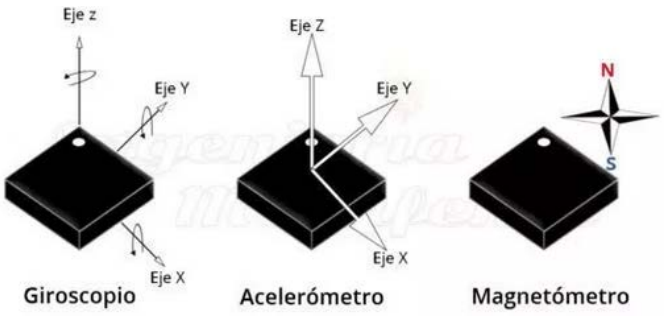

Fig 12. Conjunto de sensores que forman el IMU

- Uso en la navegación

En un sistema de navegación, la información recogida por la IMU son procesados por un ordenador, que calcula su posición actual basado en la velocidad, dirección de trayectoria y tiempo.

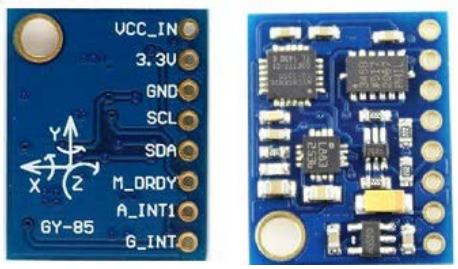

Imagen 12. IMU GY-85

- GY 85

El módulo IMU utilizado en este proyecto es el GY-85, de 9 grados de libertad compuestos por tres sensores. En conjunto, los sensores proporcionan información para determinar el rumbo, inclinación y orientación de un objeto. La comunicación con el procesador o microcontrolador se realiza utilizando el Bus I2C, librería de Arduino *Wire.h*.

o Sensores incorporados en el IMU GY 85:

- ADXL345 – Dirección bus I2C 0x53 – acelerómetro de tres ejes

Detecta las fuerzas de inercia que empujan al sensor y las descompone en las tres direcciones principales de referencia.

Su funcionamiento es similar al de los acelerómetros mecánicos. Cuando el acelerómetro es sometido a una fuerza lateral en una dirección, el sensor interno tiende a retrasarse en la dirección contraria a la fuerza que actúa y al detectar en qué paredes choca y con qué intensidad se calcula la dirección e intensidad de la fuerza aplicada.

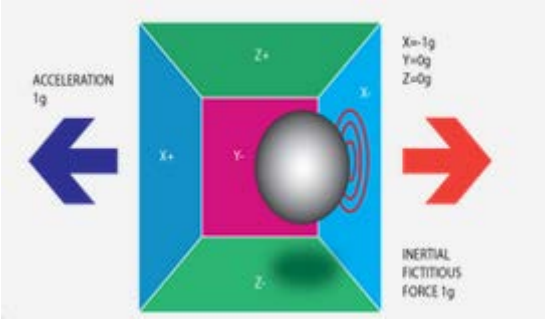

Fig. 13. Funcionamiento acelerómetro

Hoy en día los acelerómetros son dispositivos electrónicos integrados basados en el efecto piezoeléctrico, que provoca pequeñas corrientes inducidas cuando la inercia deforma ligeramente cristales diseñados para ello.

- ITG3205 – Dirección bus I2C 0x69 – giroscopio de tres ejes

Los giroscopios detectan la rotación del sensor mediante la detección de las fuerzas centrifugas que el giro provoca alrededor de los tres ejes principales. Convierte estas fuerzas en velocidad de giro alrededor de los tres ejes principales de referencia y por cálculo, los ángulos de inclinación con respecto a los ejes de referencia para conocer la orientación.

- HMC5883L – Dirección bus I2C 0x1E – magnetómetro de tres ejes

El magnetómetro obtiene lecturas del campo magnético terrestre para calcular la orientación con respecto al norte magnético de la tierra, una especie de brújula magnética. Puede verse afectado si se encuentra cerca de algún campo magnético externo.

7.1.3.1. I2C

Se trata de un protocolo síncrono de comunicación. Usa dos cables, uno para el reloj (SCL) y otro para el dato (SDA). El maestro y el esclavo envían datos por el mismo cable, el cual es controlado

por el maestro, que crea la señal de reloj. Este protocolo es muy usado en la industria, principalmente para comunicar microcontroladores y sus periféricos en sistemas integrados (*Embedded Systems*).

Arduino dispone de soporte I2C por hardware vinculado a ciertos pines. Los pines asociados varían de un modelo a otro, en el caso del Arduino Uno son los siguientes.

Tabla 5. Pines I2C asociados al Arduino Uno

MODELO	**SDA**	**SCK**
Uno	A4	A5

Para usar el bus I2C en Arduino, se proporciona la librería "*Wire.h*", que contiene las funciones necesarias para controlar el hardware integrado.

Funciones de la librería:

- **begin()** – Inicia la librería Wire y especifica si es *master* o *slave*.
- **requestFrom()** – Usado por el maestro para solicitar datos del esclavo.
- **beginTransmission()** – Comenzar transmisión con esclavo.
- **endTransmission()** – Finaliza la transmisión que comenzó con un esclavo y transmite los bytes en cola.
- **write()** – Escribe datos desde un esclavo como respuesta a una petición del maestro o pone en cola la transmisión de un maestro.
- **available()** – Devuelve el número de bytes para leer.
- **read()** – Lee un byte transmitido desde un esclavo a un maestro o viceversa.
- **onReceive()** – Llama a una función cuando un esclavo recibe una transmisión de un maestro. Registra una función de *callback*.
- **onRequest()** – Llama a una función cuando un maestro solicita datos de un maestro. Registra una función de *callback*.

Cada componente que se conecta al bus I2C tiene una dirección única, por lo que cada mensaje que se transmita por el bus debe incorporar esta dirección. Normalmente la dirección de cada componente se encuentra en los datos facilitados por el fabricante.

7.1.3.2. Conexionado

En este caso se utiliza el bus I2C para comunicar la unidad IMU con Arduino. El conexionado será el siguiente:

Tabla 6. Conexionado entre IMU y Arduino Uno

GY-85	**ARDUINO UNO**
3,3 V	3,3 V
GND	GND
SCL	A5
SDA	A4

7.1.4. Driver L298N

El driver L298N básicamente consiste en dos puentes-H, para las dos salidas A y B.

El puente-H sirve para alimentar una carga (un motor) de forma que podamos invertir el sentido de la corriente que le atraviesa.

Activando los transistores opuestos en diagonal de cada rama podemos variar el sentido en el que la corriente atraviesa la carga.

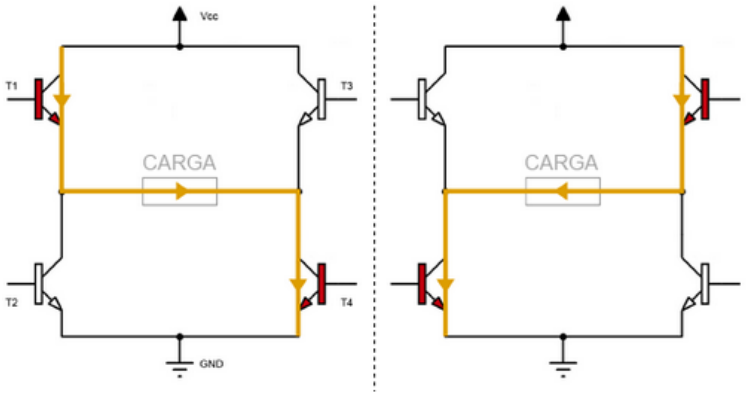

Fig. 14. Inversión del sentido de la corriente que atraviesa el puente H

Nunca se deben activar ambos transistores de un mismo ramal (izquierda o derecha) ya que se produciría un cortocircuito.

El driver L298N incorpora electrónica que simplifica la conexión al puente H, agrupando las conexiones en 3 pines (ENA, IN1 y IN2 para salida A) accesibles por cada salida y eliminando la posibilidad de cortocircuito.

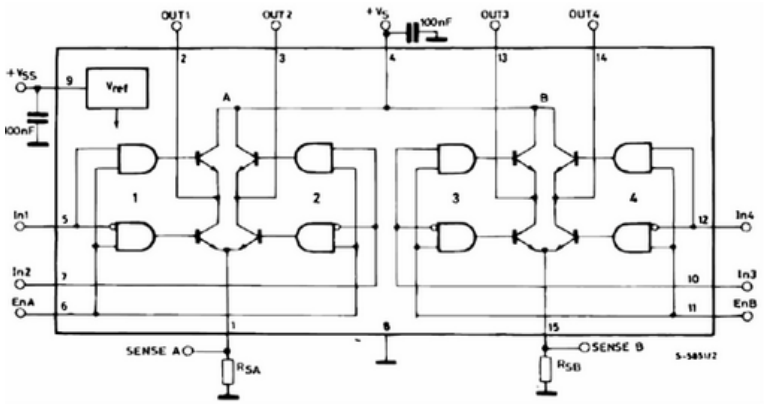

Fig. 15. Electrónica del driver L298N

Este módulo basado en el chip L298N permite controlar dos motores de corriente continua o un motor paso a paso bipolar de hasta 2 amperios. Cuenta con diodos de protección y un regulador LM7805 para suministrar los 5 Vdc de alimentación a la parte lógica del integrado L298N, activable mediante un jumper. Cuenta con otros dos jumpers de selección para habilitar cada una de las salidas del módulo (A y B).

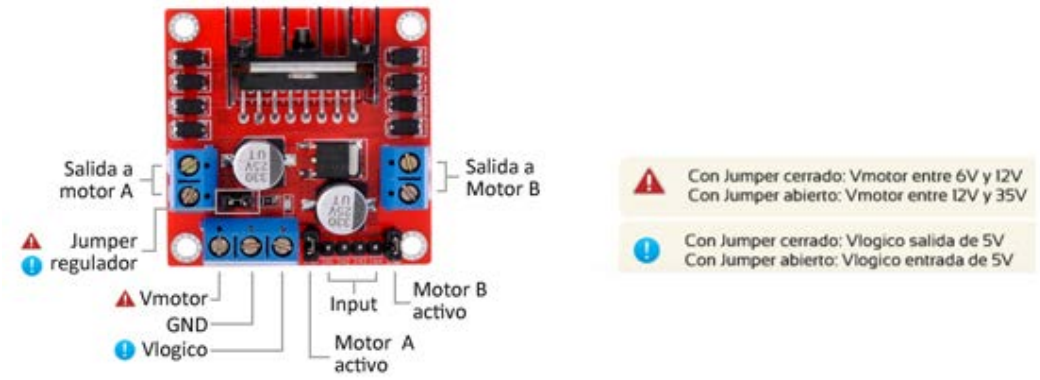

Fig. 16. Pines de conexión del L298N

- Conexión de alimentación

Gracias al regulador integrado LM7805, son posibles 2 maneras de alimentar el módulo.

Cuando el jumper de selección de 5 V se encuentra activo, el módulo permite una alimentación de entre 6 V a 12 V DC, y el pin marcado como +5 V tendrá un voltaje de salida de 5 V DC.

Cuando el jumper de selección de 5 V se encuentra inactivo, el módulo permite una alimentación de entre 12 V a 35 V DC. Como el regulador LM7805 no está funcionando, tendremos que conectar el pin de +5 V a una tensión de 5 V para alimentar la parte lógica del L298N, por lo que harán falta dos fuentes de alimentación.

El módulo dispone de dos salidas a las que se conectarán los motores. La activación de estas salidas está asociadas a los pines ENA y ENB, los cuales sirven para habilitar o deshabilitar sus respectivos motores. Si conectamos los jumpers de selección no se podrá implementar el control de velocidad de los motores, para esto deberemos remover los jumpers de habilitación de las salidas y proporcionar una señal PWM por estos pines (usualmente generada por el Arduino).

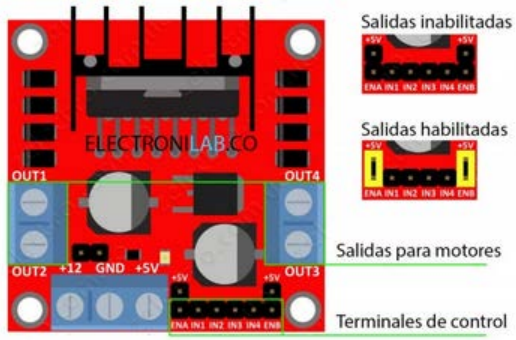

Fig. 17. Configuración de los jumpers de salidas

7.1.4.1. Configuración

Debido a las características comentadas, activaremos el jumper de selección de 5 V, de manera que nos bastará con una fuente de alimentación entre 6 y 12 V para alimentar la parte lógica del integrado y los motores.

Para realizar el control de las velocidades de los motores se deshabilitarán los jumpers de los pines ENA y ENB, de manera que se pueda realizar un control mediante señal PWM.

Para el control de las direcciones de los motores se utilizan los pines de entrada IN1 y IN2 para la salida A, e IN3 y IN4 para la salida B. Mediante estos pines se controla el encendido de los transistores del puente-H, variando la dirección de giro.

Tabla 7. Variación de giro puente H mediante señales de control

	Adelante	Atrás	Parar
IN1 (o IN3)	HIGH	LOW	LOW
IN2 (o IN4)	LOW	HIGH	LOW

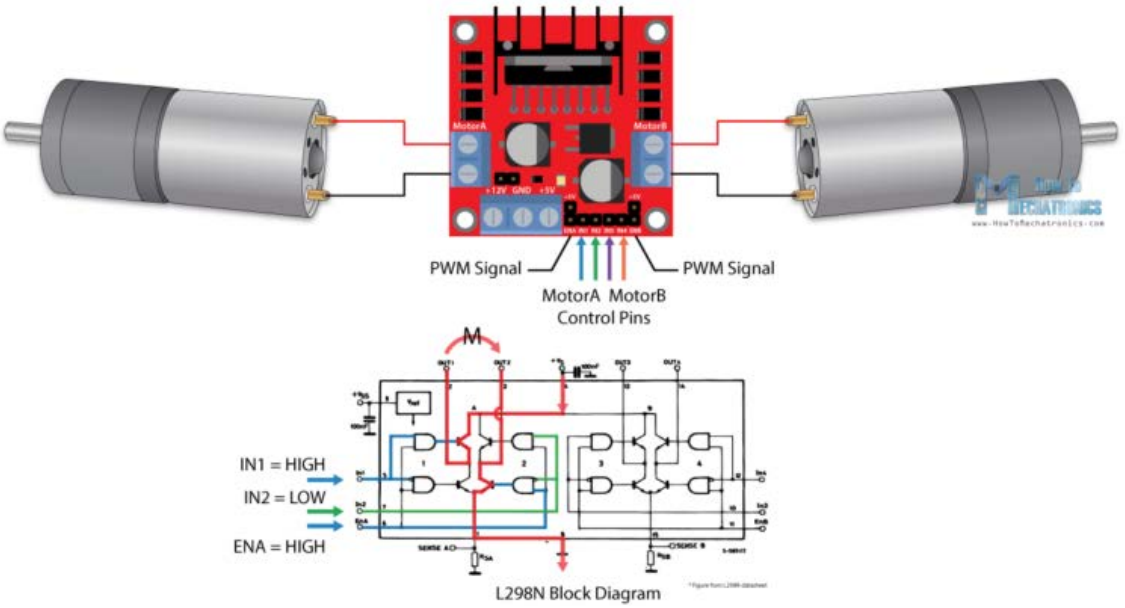

Fig. 18. Conexionado de los motores y su control de giro

Los pines IN1 e IN2 se usarán para controlar la dirección de giro del motor A, IN3 e IN4 harán lo mismo con el motor B. Estos pines controlan la activación de los transistores del puente H tal y como se muestra en la Fig. 18. SI las dos entradas están HIGH el motor se parará.

7.1.5. LiDAR

Es un acrónimo de "Ligth Detection and Ranging", es decir, detección por luz y distancia. Se trata de un sistema láser que permite medir la distancia entre el punto de emisión de ese láser hasta un objeto o superficie. El tiempo que tarda el láser en llegar a su objetivo y volver del mismo, es lo que nos permite calcular la distancia entre los dos puntos. Una de las ventajas de este sensor sobre otros que puedan hacer una función parecida (sonar) es que son mucho más fiables, ya que poseen la capacidad de recibir medidas simultáneas en varias direcciones a la vez con una alta tasa de muestreo. El uso de esta tecnología cada vez se implementa más en los coches autónomos.

Al funcionar mediante láser, la recepción es mucho más rápida y menos propensa a ruidos, por lo que es una opción mucho más precisa que el sonar.

- Ydlidar X4

El LiDAR que se utiliza en este proyecto es el Ydlidar X4. Se trata de un sensor láser que es capaz de escanear en 2D 360° del entorno en el que se encuentra gracias a un motor DC que lleva incorporado al sensor con transmisión por correa.

Tiene un rango de detección de 0,12 m a 10 m con una frecuencia de escaneo ajustable entre 6 y 12 Hz. La resolución angular es de 0,5°.

$$Periodo\ de\ escaneo\ min = \frac{1}{f} * 1000 = \frac{1}{6} * 1000 = 166{,}7\ ms \quad (17)$$

$$Periodo\ de\ escaneo\ max = \frac{1}{f} * 1000 = \frac{1}{12} * 1000 = 83{,}3\ ms \quad (18)$$

Toda la información es transmitida vía USB a la Raspberry Pi. El tipo de salida de los datos es un array que incluye los valores de distancia detectados alrededor del LiDAR. Por lo que el LiDAR publicará cada 167,7 ms en la frecuencia de escaneo mínima y cada 83,3 ms en la frecuencia de escaneo máxima un array de tipo *double*.

´Fig. 19. Dispositivo YDLiDAR X4

7.1.6. Kit RaspiCam + visión nocturna

Se trata de una cámara de bajo coste con fácil conexión a la Raspberry Pi utilizando el conector CSI (Camera Port). Tiene una resolución de imagen de 5MP y permite grabar videos en 1080p a 30fps.

Está diseñada para usarse en la placa Raspberry Pi, y es posible un fácil control mediante el uso de distintos nodos que permitirán visualizar la imagen con el fin de controlar el robot móvil a distancia.

Destacar que incorpora luz infrarroja que permite la visibilidad aun en lugares con baja luz, como los garajes.

7.1.7. Componentes de la estructura

- KIT ROBOT-CAR ARDUINO 2WD

Este kit resulta perfecto para el montaje de la base del proyecto, a pesar de su bajo coste incluye todo lo necesario para la construcción de un coche que se ajuste a las necesidades del proyecto ya que permite añadir otro piso gracias a las ranuras que incorpora.

Componentes del kit:

1. 1x Chasis del coche (metacrilato)
2. 2x Ruedas
3. 2x Motorreductores DC 3-12 (120RPM con tensión de 3Vdc)
4. 2x Encoders de velocidad
5. 1x Caja de batería pilas 4-AA
6. 1x Rueda loca
7. Tornillería

Fig. 19. Kit RobotCar Arduino 2WD

- KIT DE RUEDA DE 65 MM

Debido a que los motores del Robot Car Arduino no tenían la suficiente potencia para mover el robot de una forma controlada se optó por incorporar unos motores de 6Vdc, 210 rpm de velocidad, un par de parada de 10 kg.cm y encoders halls para un posible futuro uso.

Fig 20. Kit de rueda de 65mm

7.2. SOFTWARE

En este apartado se abordarán las herramientas software necesarias para el desarrollo del trabajo. El elemento más importante es la plataforma ROS ya que todos los demás elementos giran en torno a este.

7.2.1. Sistema operativo Lubuntu 16.04

El sistema operativo de la Raspberry Pi sobre el que se ha desarrollado este proyecto es Linux. Se ha elegido la plataforma *Ubiquity Xenial Lxde* para Raspberry Pi, basada en Lubuntu LTS 16.04, ya que es de fácil descarga y tiene preinstalado la plataforma ROS Kinetic.

La instalación del sistema operativo puede seguirse en el apartado de anexos.

La elección de Linux se debe a que ofrece el mejor rendimiento en el uso de la plataforma ROS a diferencia de otros S.O, además de ser el mejor para el desarrollo de sistemas en tiempo real.

Como introducción a Linux, se explicarán los comandos básicos de las ordenes Shell en Linux para familiarizarse con el entorno. Esto será importante para entender cómo se usa el cambio de directorios, ejecución de comandos y compilaciones desde la consola de Linux.

- **cd:** Cambia el directorio actual de trabajo a la ruta que se indique.
- **ls:** Permite ver el contenido de un directorio.
- **cp:** Copia archivos de una ruta a otra.
- **mkdir:** Crea carpetas ejecutando el código e indicando el nombre de la carpeta.
- **source.bash**: Ejecuta archivos .bash. Será necesario siempre que se inicie el proyecto en ROS para recuperar lo último que se realizó.
- **roslaunch:** Ejecuta archivos .launch.
- **catkin_make:** Instala los archivos

7.2.2. ROS (Robot Operating System)

El presente trabajo fin de grado se ha desarrollado dentro del entorno proporcionado por la plataforma ROS, siglas de *Robot Operating System*. ROS es un middleware robótico, es decir, una colección de *frameworks* para el desarrollo de software de robots. Tiene una comunidad muy extendida, con una cantidad de recursos y documentación muy amplia, lo que le convierte en la base de muchos proyectos de robótica compleja.

A pesar de no ser un sistema operativo, ROS provee los servicios estándar de estos tales como la abstracción del hardware, un servidor de parámetros, un sistema de intercambio de mensajes, etc.

Está basado en una arquitectura de grafos, distribuido bajo licencia BSD, donde el procesamiento toma lugar en los nodos que pueden recibir, mandar y multiplexar mensajes de sensores, estados, planificaciones y actuadores, entre otros. La librería está orientada para un sistema operativo UNIX (Ubuntu-Linux).

ROS tiene dos partes básicas: la parte del sistema operativo, ros, y ros-pkg. Esta última consiste en un conjunto de paquetes aportados por la contribución de usuarios que implementan las

funcionalidades tales como localización y mapeo simultaneo (SLAM), planificación, simulación, etc. El objetivo es proveer funcionalidades fácilmente reutilizables.

Los nodos en ROS son, básicamente, procesos que realizan un cómputo, ejecutables dentro de un paquete que utilizan las librerías cliente de ROS para la comunicación con otros nodos.

- Funcionamiento

ROS está concebido como un sistema basado en nodos. En este contexto, un nodo es cualquier hardware o software con el que podamos interactuar. ROS define un estándar para comunicarnos entre los diferentes nodos que compondrán nuestro sistema final.

- Comunicación

ROS tiene un nodo principal, *roscore*, encargado de cargar todos los procesos necesarios para que cualquier nodo pueda ser ejecutado con ROS. Entre ellos se encuentra un nodo *master*, encargado de centralizar las comunicaciones entre todos los nodos presentes en el sistema. Este está basado en un sistema de suscripción/publicación, que utiliza el envío y recepción de mensajes. La publicación o suscripción a estos mensajes se hará en base a temas "*Topic*", que son nombres asociados a un tipo de mensaje concreto, como buses identificativos.

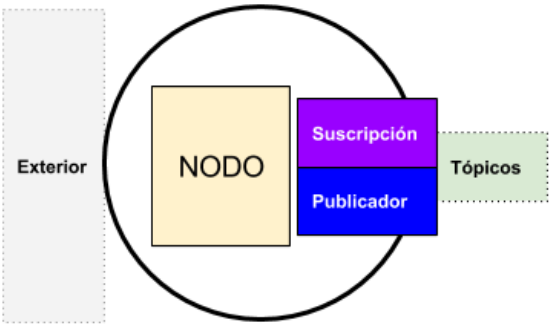

Fig. 21. Funcionamiento general de los nodos

Cuando un nodo comienza a publicar datos en un tema, avisa al nodo master para que en caso de que algún nodo quiera suscribirse a ese tema, le indique dónde debe ir a buscar los datos. En el orden inverso, indica al nodo publicador dónde están esperando los datos de ese tema.

En resumen, al disponer de un nodo principal (nodo *master*), el resto de nodos simplemente han de preguntarle a este qué otros nodos existen y cuáles son los mensajes a enviar o recibir de ellos. Lo que posibilita la incorporación de nodos al sistema en cualquier momento y al instante.

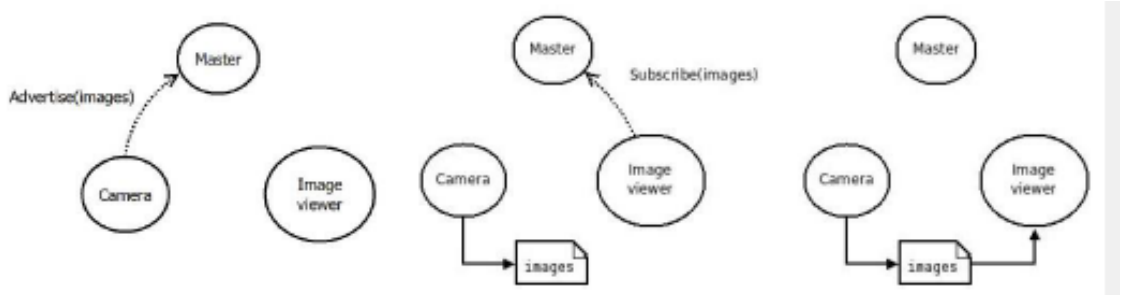

Fig. 22. Funcionamiento de publicación/suscripción del tema images

- Ejemplo de funcionamiento

El nodo "*Camera*" publica un tema "*images*" y le indica al nodo master que los datos de "*image*" están listos para utilizarse. El nodo "*Image viewer*" indica al nodo master que se le suscriba a los datos de "*image*" cuando estos estén disponibles. El nodo master relaciona el tema publicado por la cámara en el nodo "*image viewer*", el cual visualizará los datos (imágenes) que le lleguen del tema "*images*" (Fig. 22).

El protocolo de conexión empleado entre nodos más usual en ROS se llama TCPROS, basado en socket estándar TCP/IP.

- Sistema de Cómputos ROS

En resumen, el sistema de cómputo de ROS está basado en un grafo formando una red bidireccional de procesos que procesan datos compartidos. Los principales conceptos de este grafo son:

- **Nodos**: Explicado anteriormente.
- **Master**: Se encarga de controlar el proceso completo, de forma que gracias a él los nodos pueden comunicarse entre ellos, intercambiar mensajes e invocar servicios.
- **Mensajes**: Es la forma de comunicación de los nodos. Es simplemente una estructura de datos. Pueden incluir tipos de datos primitivos y cadenas de este tipo de datos.
- **Topics:** Los nodos envían mensajes publicándolos en un tema. El tema es un nombre para identificar el mensaje. Los nodos suscritos a dicho tema tendrán acceso a la información publicada por otros nodos en ese tema. Se trata por tanto del medio por el que los nodos pueden enviar mensajes de unos a otros. Cada nodo puede publicar y estar suscrito a varios temas sin ningún problema, ya que lo que se pretende es separar la producción de información de su consumo.
- **Servicios:** Se encargan de las comunicaciones a la hora de hacer peticiones, y entregar respuestas. En los temas, los nodos se encargan de publicar información, que puede ser leída por un suscriptor o no. En el caso de los servicios, lo que se encarga es de hacer peticiones que necesitan de una respuesta, por lo que existe una dependencia real entre los dos nodos a través de servicios. De esta forma, un nodo ofrece un servicio bajo un nombre y será otro nodo el que haga una petición a ese servicio mediante un mensaje determinado y que se queda esperando a la respuesta.
- **Bag:** Se tratan de una forma de guardar los mensajes publicados en un tema y repetirlos si fuera necesario. Son un medio muy importante para guardar datos, como datos de sensores, que puede ser difícil de recoger pero que sean necesarios para realizar pruebas.

- Herramientas de ROS

ROS cuenta con muchas herramientas para el desarrollo de proyectos de robótica. Se analizarán dos fundamentales para el desarrollo de aplicaciones gráficas, como la creación de mapas y localización.

- **Rviz**

Se trata de un paquete de ROS que contiene un visualizador que permite mostrar en tiempo real la posición actual del robot, nubes de puntos o incluso los sistemas de visión del mismo. Esta herramienta se utilizará para visualizar los temas creados por los nodos del sistema SLAM.

- **Rqt**

Permite el desarrollo de GUI en forma de *plugins*. Permite un fácil manejo de las diferentes opciones que ofrece ROS mediante una interfaz gráfica mucho más sencilla. Además, también permite visualizar de forma gráfica las conexiones establecidas entre nodos y temas, así como los servicios requeridos.

- Transformaciones

Los sistemas robóticos necesitan de la existencia de relaciones espaciales, por ejemplo, entre un robot móvil y un marco de referencia fijo para la localización, entre varios marcos de diferentes sensores, etc. Para simplificar y unificar el tratamiento de los marcos espaciales, ROS ha escrito un sistema de transformación llamado *tf*. El cual construye un árbol de transformación dinámica que relaciona todos los marcos de referencia del sistema.

En el caso del problema SLAM se necesita un árbol de transformadas entre el láser, la base del robot, la odometría y el mapa.

base_scan → base_link → odom → map

7.2.3. Hector Slam

Hector SLAM es una implementación *open source* de la técnica de SLAM en 2D. El sensor utilizado por este método es un sensor láser. Se ha seleccionado esta técnica de SLAM 2D debido a que el método Hector SLAM no requiere necesariamente de la información de la odometría del robot móvil, únicamente emplea para calcular el posicionamiento la información obtenida por los barridos láser. Debido a la alta tasa de actualización y a la precisión de los sensores LIDAR se obtiene un algoritmo que se ejecuta rápidamente con gran precisión en el cálculo del posicionamiento del robot.

El tratamiento de los datos obtenidos por el sensor láser se hace a través de las librerías de ROS que se explicarán a continuación.

Hector SLAM es un *metapackage* de ROS que contiene distintas librerías relacionadas con el mapeo, el control de trayectoria, la navegación y localización. Estas librerías mencionadas son los siguientes: *hector_compressed_map_transport, hector_geotiff, hector_geotiff_plugins, hector_imu_attitude_to_tf, hector_map_server, hector_map_tools, hector_mapping, hector_marker_drawing, hector_nav_msgs, hector_slam_launch, hector_trayectory_server*.

En el proyecto desarrollado se usarán en concreto las siguientes:

- **hector_mapping:** Es el nodo encargado del SLAM. Hace uso de las medidas láser para realizar un mapa del entorno en el que se encuentra el robot. Puede utilizar datos de odometría obtenidos de una IMU o de un encoder para complementar al LIDAR, o simplemente puede realizar odometría basándose únicamente en los datos aportados por el sensor láser.
- **hector_geotiff**: Encargado de guardar los mapas y la trayectoria del robot en formato geotiff. Este formato es un estándar de metadatos que permite que información de georreferencia sea encajada en un archivo de imagen de formato TIFF y que pueda ser automáticamente posicionada en un sistema de referencia espacial.

- **hector_trajectory_server:** Encargado de almacenar la trayectoria del robot basada en transformaciones *tf* (una herramienta de ROS que permite el uso de múltiples sistemas de referencia a lo largo del tiempo. Las relaciones entre los sistemas de referencia se mantienen en una estructura de árbol almacenada en buffer en el tiempo, y permite la transformación de puntos, vectores, etc… entre dos sistemas de referencia).

Hector_mapping está subscrito al *topic /scan* y recibe mensajes del tipo *sensor_msgs/LaserScan*. Por lo tanto, se ha compilado un programa en C++ que recibe los datos enviados desde la Raspberry Pi vía socket UDP, y hace de Publisher, enviando las medidas del LIDAR en forma de *sensor_msgs/LaserScan* al tema *scan* para que puedan ser recibidas por el nodo de SLAM. *Hector_mapping* se comunica mediante tema internos con *hector_geotiff* y *hector_trajectory_server*. Tal y como puede verse en la Fig. 23.

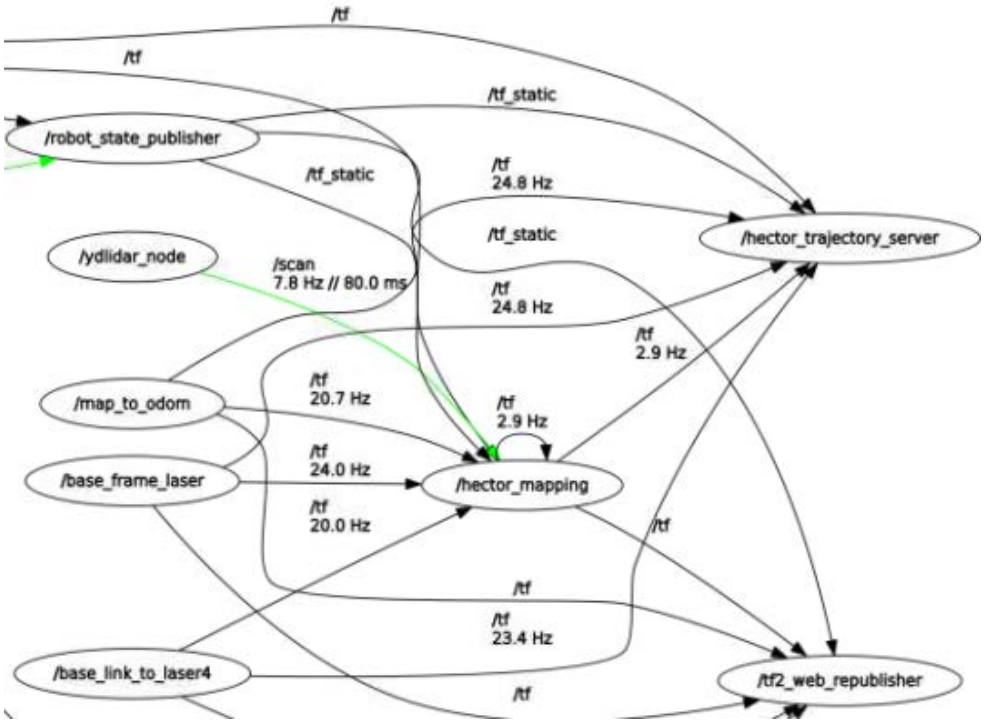

Fig. 23. Diagrama de comunicación paquetes HectorSlam

Para visualizar los mapas creados por estos nodos se utilizará la herramienta de visualización de ROS Rviz.

7.2.4. VNC

VNC o Virtual Network Computing es un sistema de conexión que permite utilizar el teclado y el ratón del ordenador para interactuar con un entorno de escritorio gráfico en un servidor remoto.

En este proyecto se utiliza para controlar la Raspberry desde un ordenador o móvil que estén conectados a la misma red.

VNC consta de dos herramientas. Por una parte, está el VNC Viewer con el que se controla la Raspberry desde el ordenador o el dispositivo que se use para este fin. Por otra parte, está el cliente o VNC Server, el cual viene preinstalado por defecto en los sistemas operativos de Raspbian, pero en Ubuntu se ha de instalar descargando la aplicación desde la web oficial de VNC.

Basta con introducir la dirección IP de la Raspberry y el usuario y contraseña que tenga establecida la Raspberry para poder controlarla desde el ordenador.

7.2.5. ROS for LabVIEW Software

Se trata de una extensión de LabVIEW que es capaz de comunicarse con ROS de la Raspberry Pi. Esta extensión resulta muy sencilla para suscribir y recibir mensajes de los nodos de ROS. Destacar que esta tarea ha sido desarrollada en la versión de LabVIEW 2014.

Para información de la instalación de la extensión, acudir a anexos de instalaciones.

- Funcionamiento

Hay cuatro VIs esenciales para la mayoría de programas de esta extensión.

- **ROS_Topic_Init** inicializa un nodo en ROS y define su nombre, el tema a publicar o suscribir, el tipo de mensaje, el tamaño y la tasa de actualización.

- **ROS_Topic_Write** es el VI que se encarga de publicar los mensajes que son creados en LabVIEW.

- **ROS_Topic_Read** es el VI que se encarga de leer los mensajes publicado en ROS.

- **ROS_Topic_Close** cierra el nodo, es importante acabar con el nodo cada vez que se haya terminado de ejecutar el VI, ya que sino al reiniciar el VI o el ROS Master el nodo puede no inicializarse correctamente.

Aparte de estos VIs se distinguen otros dos tipos para generar o analizar mensajes.

MessageBuilding: sirve para crear mensajes de los tipos que dispone la librería.

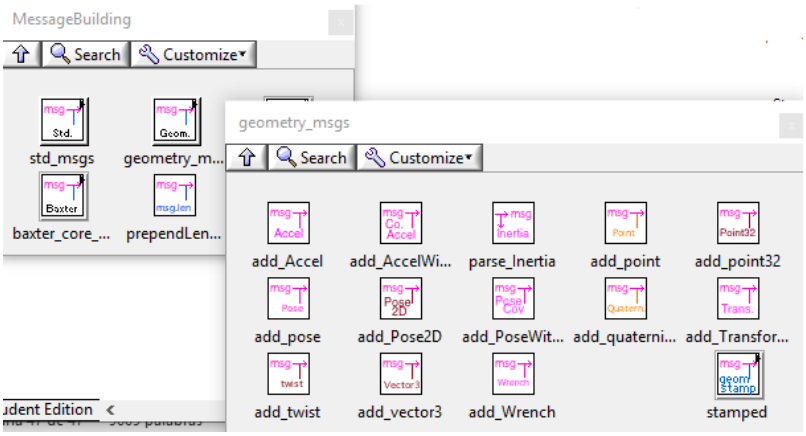

Fig. 24. Mensajes de tipo geometry_msgs que incorpora ROS for LabVIEW Software

Parsing: Analiza los mensajes de los temas leídos.

Haciendo uso de estos VIs se propone como objetivo extra desarrollar un programa que sea capaz de publicar y leer mensajes ROS, entendiendo el funcionamiento del VI y la comunicación entre ROS y LabVIEW. Para esto, se desarrollarán dos tareas:

1- Publicar mensajes de *geometry_msgs/twist* para controlar el robot móvil desde LabVIEW.
2- Leer mensaje del sensor láser, *sensor_msgs/scan*, y graficar la lectura del LiDAR (mapa del entorno actual).

8. IMPLEMENTACIÓN CONTROL TRAYECTORIA DEL ROBOT

8.1. Control mediante teleop_twist_keyboard

En este apartado se explica cómo controlar el robot móvil haciendo uso del teclado del PC desde el que se está controlando la Raspberry Pi.

La instalación de los paquetes de ROS necesarios para la implementación de este apartado está explicada en el apartado de anexos de instalación.

- Algoritmo de funcionamiento

A continuación, se observa el algoritmo de funcionamiento del control de la trayectoria del robot mediante el uso del paquete *teleop_twist_keyboard* y conexión entre las placas Raspberry y Arduino:

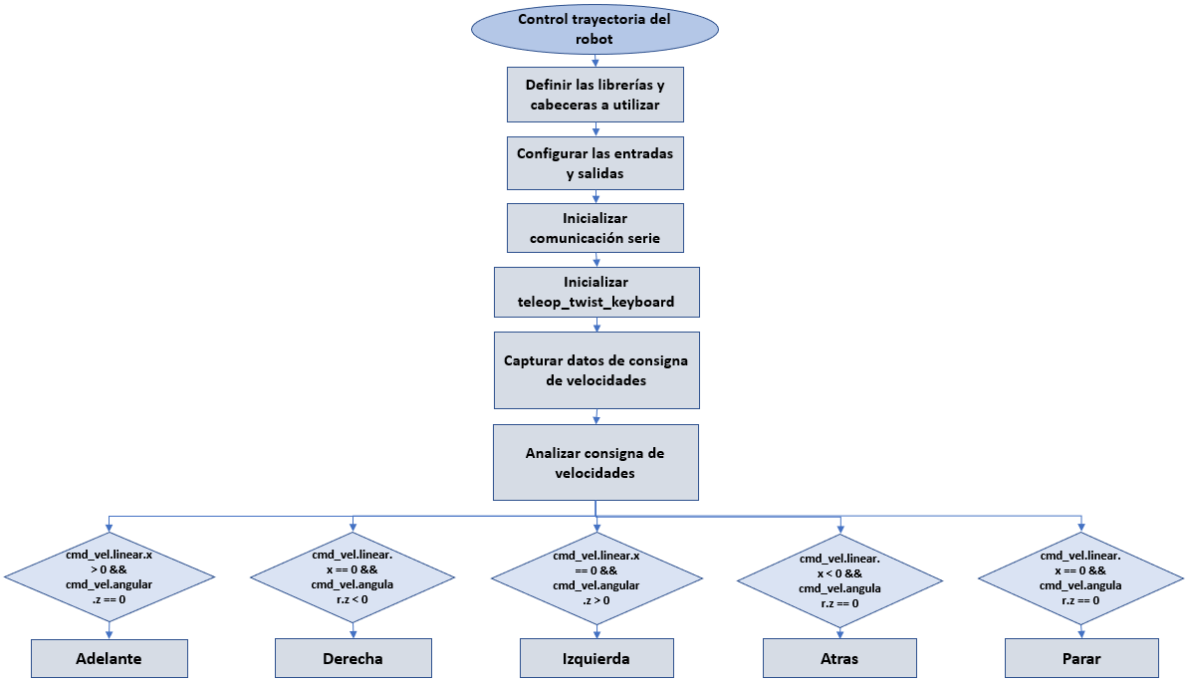

Fig. 25. Algoritmo de funcionamiento de control de trayectoria

El control de la trayectoria se realiza desde el programa de Arduino IDE, por lo que primero será necesario definir las librerías del programa para la comunicación con la Raspberry Pi. La trayectoria del robot es controlada por los dos motores DC que lleva instalados, el control de estos es llevado a cabo por el driver L298N, por lo que también será necesario definir las entradas y salidas de la conexión entre el driver L298N y la placa Arduino. Tras cargar el programa Arduino INO a la placa se efectuará la comunicación serie entre la Raspberry Pi y el Arduino para que puedan comunicarse entre ellos.

- Comunicación puerto serie entre Raspberry Pi y Arduino

Esta conexión se realizará vía serie USB entre ambas placas, y será necesario introducir el siguiente comando.

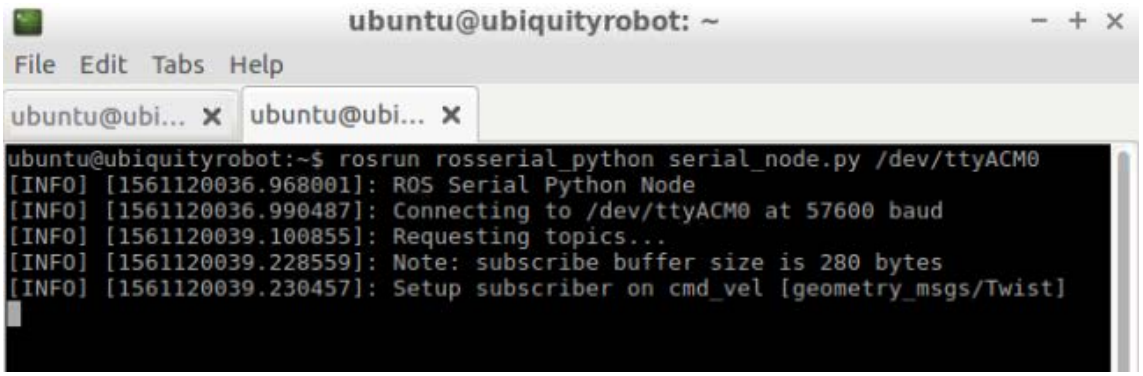

Fig. 26. Conexión serie entre Raspberry Pi y Arduino

Importante: Mirar el anexo de instalación para saber el nombre del puerto serie en el que el Arduino se conecta a la Raspberry Pi, en este caso */dev/ttyACM0*.

Mediante este comando se establece una conexión ROS Serial entre ambas placas que permite a la Raspberry Pi suscribir o publicar mensajes en los temas que indiquemos en el programa de Arduino, en este caso en *cmd_vel*. Esto permitirá suscribir los mensajes del nodo *teleop_twist_keyboard*, que se explicarán a continuación, en el Arduino.

Una vez realizada la conexión entre ambas placas será necesario publicar el tema al que está suscrito el *serial node*. Esto se realizará mediante el paquete de ROS *teleop_twist_keyboard*.

- Teleop_twist_keyboard

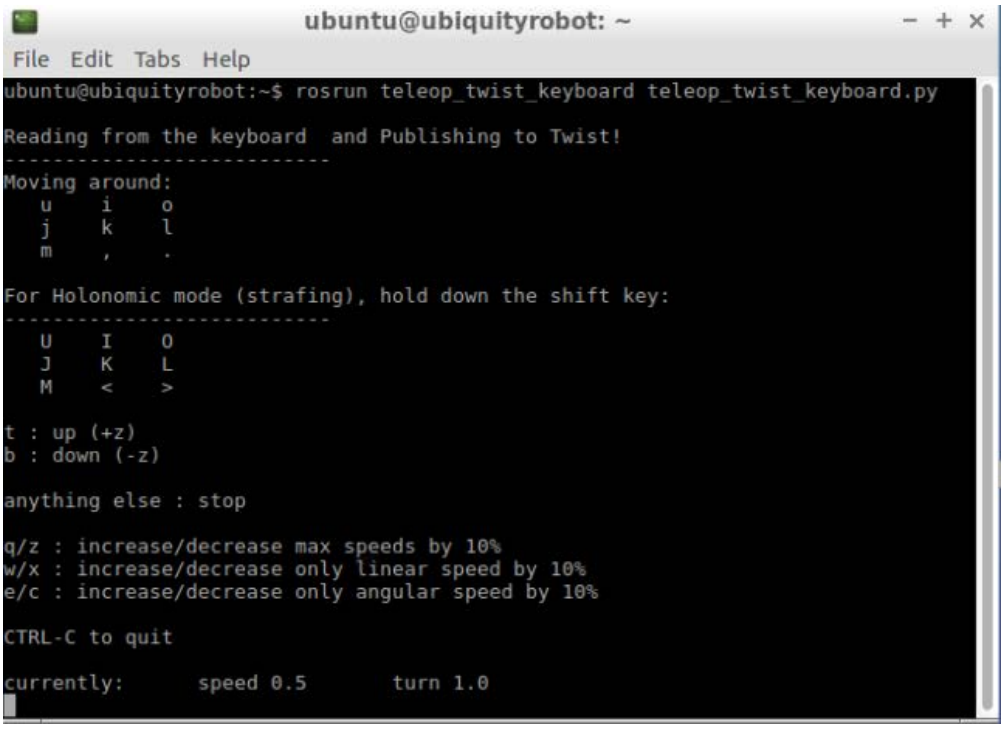

Fig. 27. Interface paquete teleop_twist_keyboard

Este paquete de ROS sirve para controlar la velocidad lineal y angular del robot móvil desde el teclado de un PC. En el proyecto desarrollado, este paquete publica la velocidad lineal en x y la angular en z en un mensaje de tipo *geometry_msgs/Twist*, el cual se controla mediante las teclas indicadas en la Fig. 28.

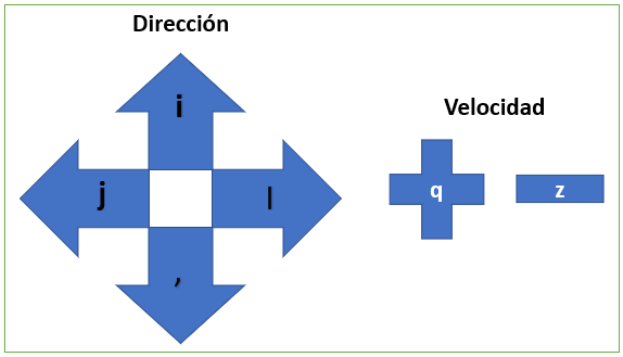

Fig. 28. Comandos para controlar el robot móvil

Esto permitirá al programa de Arduino saber en qué estado se encuentra el robot (adelante, atrás…) y la velocidad a la cual deberán girar los motores, la cual controlará mediante una señal PWM (0 a 255).

A continuación, se observa el programa Arduino para realizar esta tarea, que analizará la velocidad de consigna del tema al que está suscrito. De esta manera el Arduino determinara la dirección y la velocidad a la que se pide que el robot se mueva, actuando en el driver L298N para que este envíe las señales correspondientes a los motores.

- Código Arduino IDE

```
#include <ros.h>              //se añaden las librerías necesarias
#include <geometry_msgs/Twist.h>

#define ENA 3
#define ENB 5
#define IN1 2
#define IN2 4
#define IN3 6
#define IN4 7

int dutycycle;

ros::NodeHandle node;    //crea NodeHandle, un nodo ROS que permite al programa crear publishers y subscribers, y se encarga de las comunicaciones del puerto serie
geometry_msgs::Twist msg;          //expresa la velocidad linear y angular

void adelante()
{
```

```
  analogWrite(ENA, dutycycle);
  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, LOW);
  analogWrite(ENB, dutycycle);
  digitalWrite(IN3, HIGH);
  digitalWrite(IN4, LOW);
}
void izquierda()
{
  analogWrite(ENA, dutycycle);
  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, LOW);
}
void parar()
{
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, LOW);
}
void derecha()
{
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, LOW);
  analogWrite(ENB, dutycycle);
  digitalWrite(IN3, HIGH);
  digitalWrite(IN4, LOW);
}
void atras()
{
  analogWrite(ENA, dutycycle);
```

```
 digitalWrite(IN1, LOW);
 digitalWrite(IN2, HIGH);
 analogWrite(ENB, dutycycle);
 digitalWrite(IN3, LOW);
 digitalWrite(IN4, HIGH);
}
```

void roverCallBack(const geometry_msgs::Twist& cmd_vel) // Crea la función de llamada para el suscriptor, la cual toma constantemente como referencia el mensaje de su argumento. En este caso la devolución de llamada es roverCallBack, el tipo de mensaje es geometry_msgs/Twist y el nombre del mensaje cmd_vel. El software responderá a los mensajes cmd_vel.linear.x y cmd_vel.angular.z y generará una señal PWM correspondiente.

```
{
 if(cmd_vel.linear.x > 0 && cmd_vel.angular.z == 0)
 {
     adelante(); //i
     dutycycle=cmd_vel.linear.x * 255;
 }
 if(cmd_vel.linear.x == 0 && cmd_vel.angular.z < 0)
 {
      derecha(); //l
      dutycycle=cmd_vel.linear.x * 255;
 }
 if(cmd_vel.linear.x == 0 && cmd_vel.angular.z > 0)
   {
      izquierda(); //j
      dutycycle=cmd_vel.linear.x * 255;
   }
 if(cmd_vel.linear.x == 0 && cmd_vel.angular.z == 0)
   {
    parar(); //k
   }
 if(cmd_vel.linear.x < 0 && cmd_vel.angular.z == 0)
   {
    atras(); //,
```

```
      dutycycle=cmd_vel.linear.x * 255;
    }
}
```

ros::Subscriber <geometry_msgs::Twist> sub("cmd_vel", roverCallBack); //publica los mensajes de las velocidades lineales y angulares de tipo geometry/Twist en el topic cmd_vel del callback. Se crea una instancia de un suscriptor con un nombre de tema (topic) "cmd_vel" de tipo geometry_msgs::Twist. Para los suscriptores se ha de hacer una plantilla del suscriptor del mensaje, el topic al que se suscribirá y la función de devolución de llamada

```
void setup()        //en la función de configuración de Arduino se inicializa el identificador del nodo ROS, se anuncia cualquier topic que se publique y se suscriba a cualquier tema
{
 node.initNode();            //inicia el nodo NodeHandle, el nodo ROS para el proceso
 node.subscribe(sub);
  pinMode(ENA, OUTPUT);
 pinMode(ENB, OUTPUT);
 pinMode(IN1, OUTPUT);
 pinMode(IN2, OUTPUT);
 pinMode(IN3, OUTPUT);
 pinMode(IN4, OUTPUT);
}
void loop()         //en la función de bucle se llama a ros::spinOnce()
{
 node.spinOnce();            //Permite llamar al callback
 delay(1);
}
```

8.2. Control mediante LabVIEW

El objetivo de este apartado es similar al anterior, con la diferencia de que para generar el mensaje publicador de velocidad se hará uso del programa LabVIEW instalado en el PC en lugar del paquete *teleop_twist_keyboard*.

Se partirá como base de los programas proporcionados por Clearpath Robotics para el control del robot Husky.

A continuación, se observa el algoritmo de funcionamiento del control de la trayectoria del robot mediante el uso del programa LabVIEW y conexión entre la placa Raspberry y el ordenador con LabVIEW instalado:

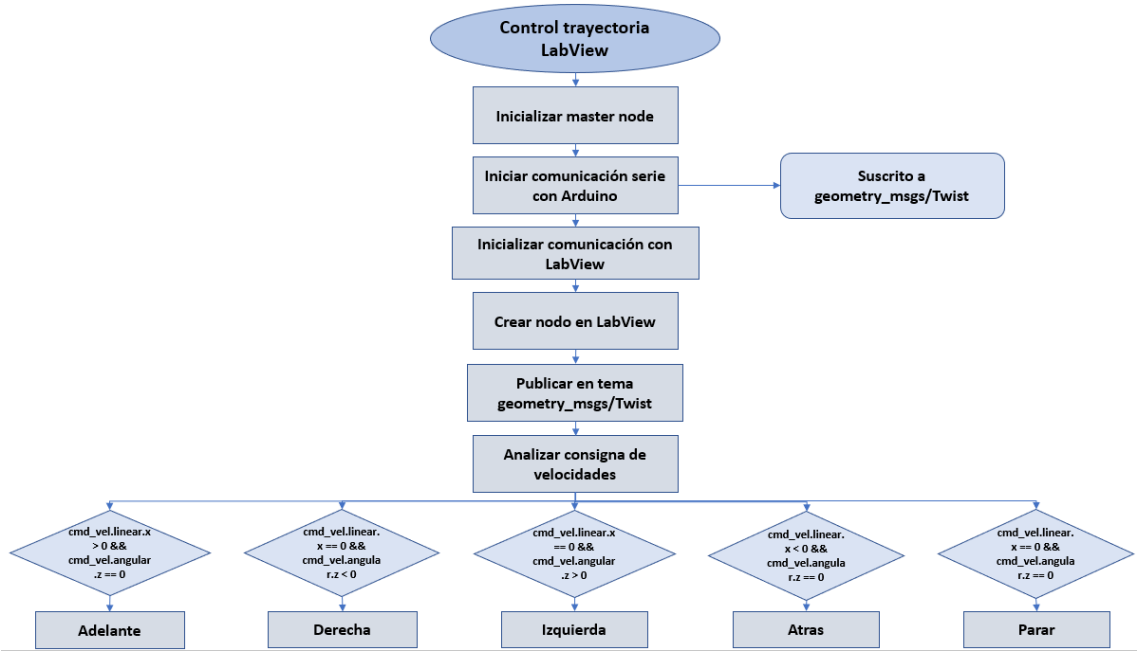

Fig. 29. Algoritmo de funcionamiento control trayectoria mediante LabVIEW

Lo primero será inicializar el nodo master (comando *roscore*), imprescindible para inicializar el resto de nodos y permitir la comunicación entre estos. Como en el anterior apartado, el programa de control de la trayectoria lo realizará el programa de Arduino, por lo que será necesaria inicializar la comunicación serie entre las dos placas. Recordar que el programa Arduino esta suscrito al tema */cmd_vel* del mensaje *geometry_msgs/Twist*. La publicación de este tema se hará mediante el siguiente programa de LabVIEW.

- Programación gráfica

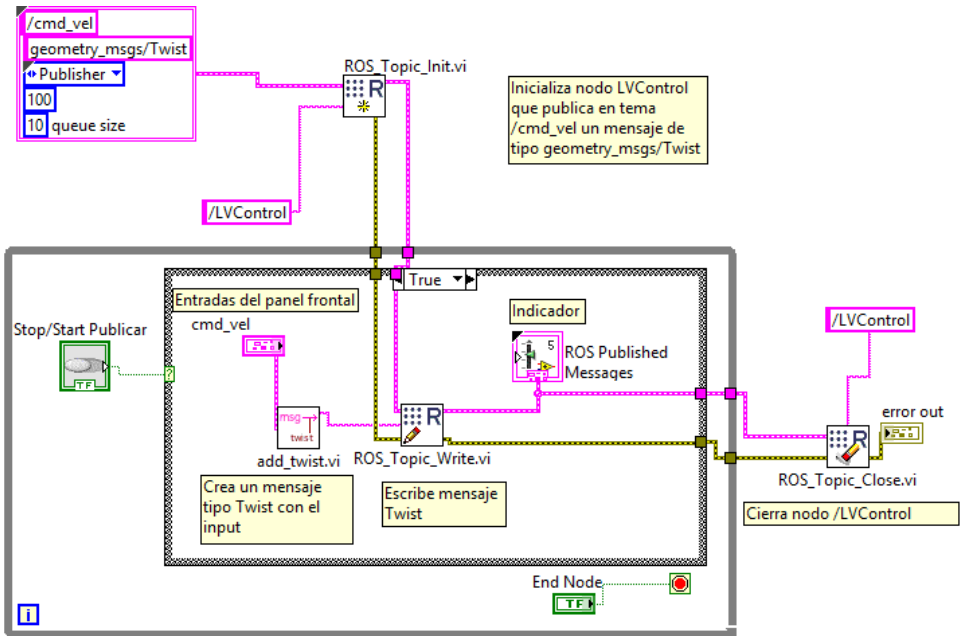

Fig. 29. Control trayectoria diagrama de bloques LabVIEW

- Funcionamiento

Mediante este VI se crea un nodo de nombre */LVControl* que publicará un mensaje de tipo *geometry_msg/Twist* con el tema */cmd_vel* en el sistema ROS.

El programa de Arduino que controla los motores está suscrito a este tema, por lo que se conseguirá el control del robot móvil en sustitución de *teleop_twist_keyboard*.

El subVI *ROS_Topic_Init* crea el nodo mencionado con las características indicadas en su entrada:

- **Nombre del nodo**: /LVControl
- **Tema:** /cmd_vel
- **Tipo de mensaje:** geometry_msgs/Twist
- **Publicador o suscriptor**: Publisher
- **Tiempo de actualización**: 100 ms
- **Tamaño del mensaje:** 10 bit

El subVI *add_twist* se encarga de crear el mensaje *geometry_msgs/Twist*, el cual será controlado con las entradas *cmd_vel* del panel frontal proporcionadas.

El subVI *ROS_Topic_Write* escribe el tema del mensaje creado.

Por último se cierra el nodo creado, para evitar problemas al reiniciar el programa.

- Ejecución del programa

Al ejecutar el VI se deberá introducir la dirección del Master IP de la Raspberry.

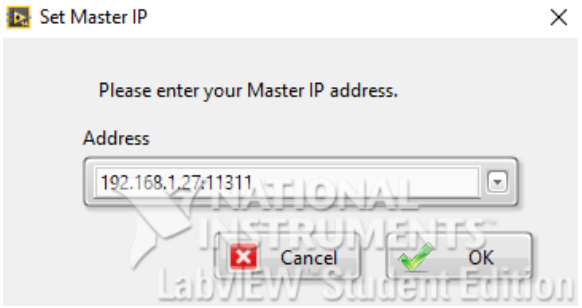

Fig. 30. Dirección IP de ROS Master

En la Fig. 31 se observa la dirección IP del ordenador que está ejecutando el programa de LabVIEW y el tema que publica, haciendo uso del comando *rostopic info*.

```
ubuntu@ubiquityrobot:~/catkin_ws$ rostopic info /cmd_vel
Type: geometry_msgs/Twist

Publishers:
 * /teleop_twist_joy (http://ubiquityrobot.local:44789/)
 * /LVControl (http://192.168.1.12:50315)

Subscribers:
 * /motor_node (http://ubiquityrobot.local:43211/)
 * /serial_node (http://ubiquityrobot.local:37183/)
```

Fig. 31. Publicadores y suscriptores del nodo LabView

Haciendo uso de la herramienta RQT GRAPH, se observa que efectivamente *serial_node* está suscrito al tema que está publicando LabVIEW.

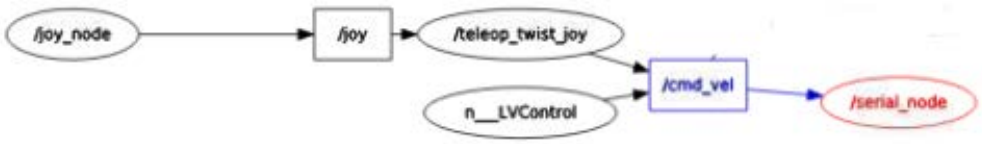

Fig. 32. Diagrama para control de trayectoria LabVIEW

De esta manera, modificando los valores del *cmd_vel* del panel frontal, se podrá controlar la dirección del robot móvil.

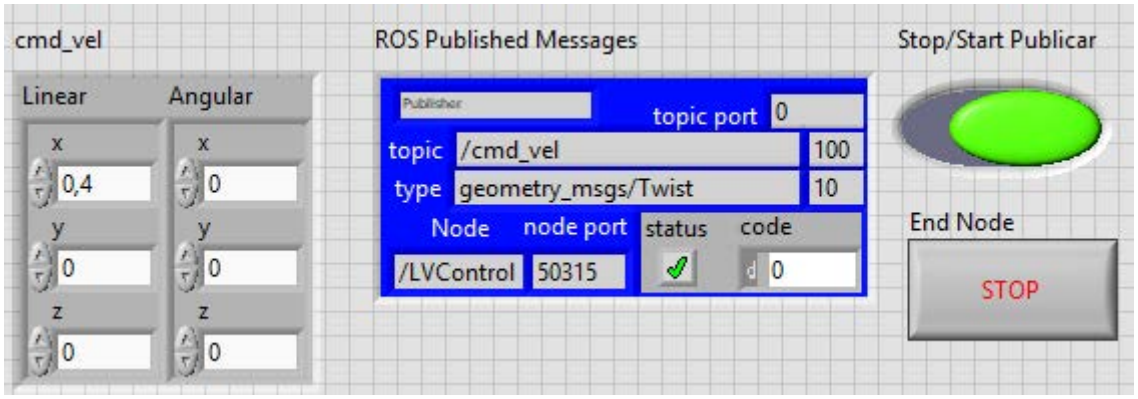

Fig. 33. Panel frontal del control de trayectoria LabVIEW

9. IMPLEMENTACIÓN SISTEMA SLAM

9.1. FUNCIONAMIENTO DEL SISTEMA SLAM

El objetivo de este apartado es crear una interfaz desde la cual se puedan efectuar las siguientes funciones:

- Controlar la trayectoria del robot móvil.
- Visualizar la imagen de la RaspiCam en tiempo real.
- Visualizar los datos obtenidos por el sensor LiDAR.
- Crear y visualizar un mapa de ocupación mediante los datos obtenidos por el LiDAR.
- Visualizar la trayectoria del robot móvil en el mapa creado.

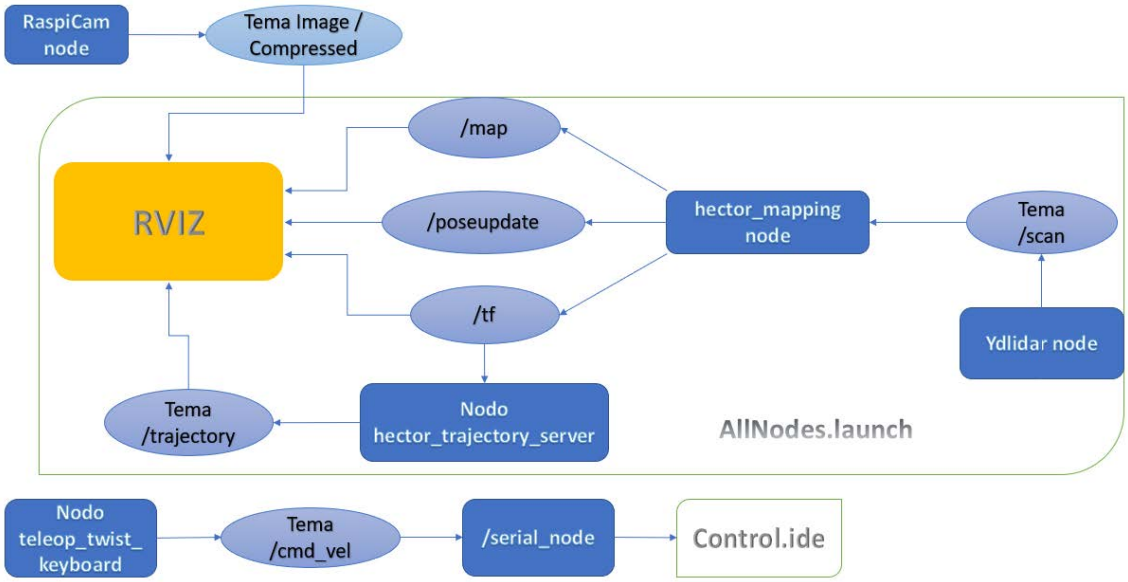

Fig. 34. Diagrama del funcionamiento general del sistema SLAM implementado

- Control trayectoria

Para esta tarea se hará uso de la implementación del control de la trayectoria del apartado 8.1. Se publicará el tema */cmd_vel* del nodo *teleop_twist_keyboard* en el *serial node*.

- Visualizar imagen de la RaspiCam

En esta tarea se ha de crear un nodo que publique el tema de la imagen obtenida por la cámara. Para esto se deberán introducir los comandos de la Fig. 35.

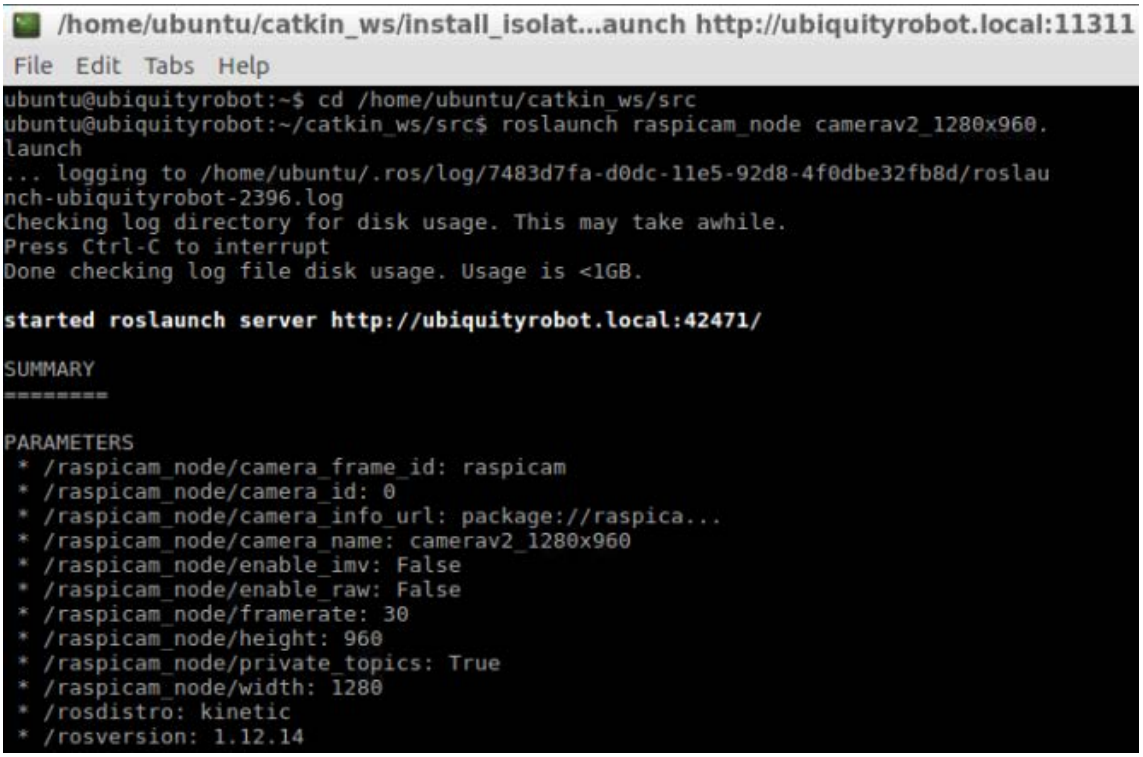

Fig. 35. Ejecución nodo raspicam

Para verificar que la instalación y configuración explicada en el anexo de instalación son correctas, se puede introducir el comando de la Fig. 36 para visualizar la imagen mediante la herramienta RQT IMAGE VIEW:

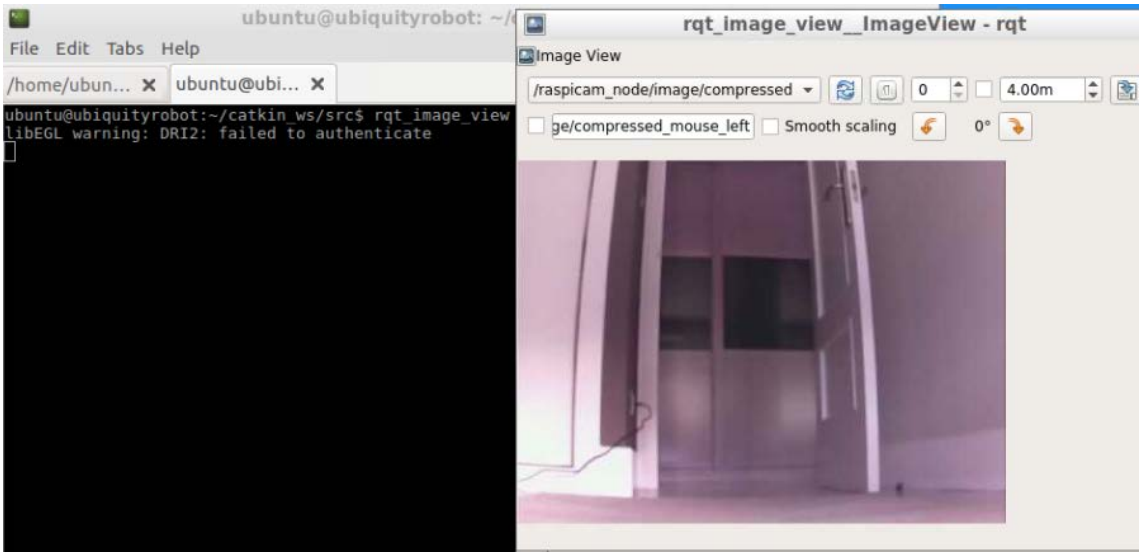

Fig. 36. Visualización raspicam RQT_IMAGE_VIEW

- Creación del mapa de ocupación y visualización de este

Para esta tarea se creará un archivo .launch que ejecutará los nodos necesarios para la implementación de la técnica SLAM y la visualización del mapa y trayectoria en la herramienta de visualización RVIZ.

A continuación se muestra el programa para la ejecución de los nodos necesarios para la implementación del sistema SLAM, además de las correspondientes transformaciones necesarias.

All_nodes.launch

```
<launch>

 <include file="$(find ydlidar)/launch/lidar.launch" />

 <node pkg="tf" type="static_transform_publisher" name="map_to_odom" args="0.0 0.0 0.0 0
0 0.0 /odom /base_link 40" />

<node pkg="tf" type="static_transform_publisher" name="base_frame_laser" args="0 0 0 0 0 0
/base_link /laser_frame 40" />

<!--<node pkg="rviz" type="rviz" name="rviz" args="-d $(find
hector_slam_launch)/rviz_cfg/mapping_demo.rviz"/>-->

 <include file="$(find hector_mapping)/launch/mapping.launch" />

  <node pkg="rviz" type="rviz" name="rviz" args="-d $(find ydlidar)/launch/lidar.rviz" />

  <include file="$(find hector_geotiff)/launch/geotiff_mapper.launch" />

</launch>
```

En las siguientes figuras se observa el funcionamiento del sistema SLAM implementado, en el cual se observan los nodos y los temas creados para el sistema.

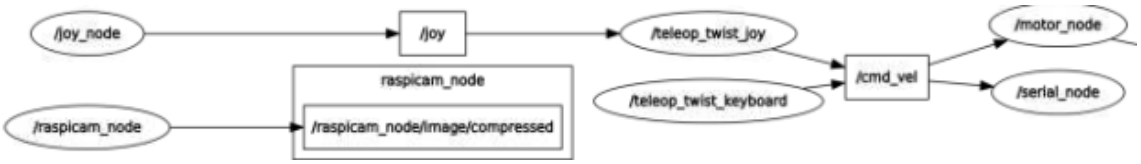

Fig. 37. Gráfica de nodos raspicam, y control trayectoria

La Fig. 37 recoge la gráfica correspondiente al nodo raspicam, este publica el tema /Image/Compressed que será usado por la herramienta RVIZ para la visualización de la imagen en tiempo real, y al nodo /teleop_twist_keyboard que publicará el tema /cmd_vel para el control de la trayectoria.

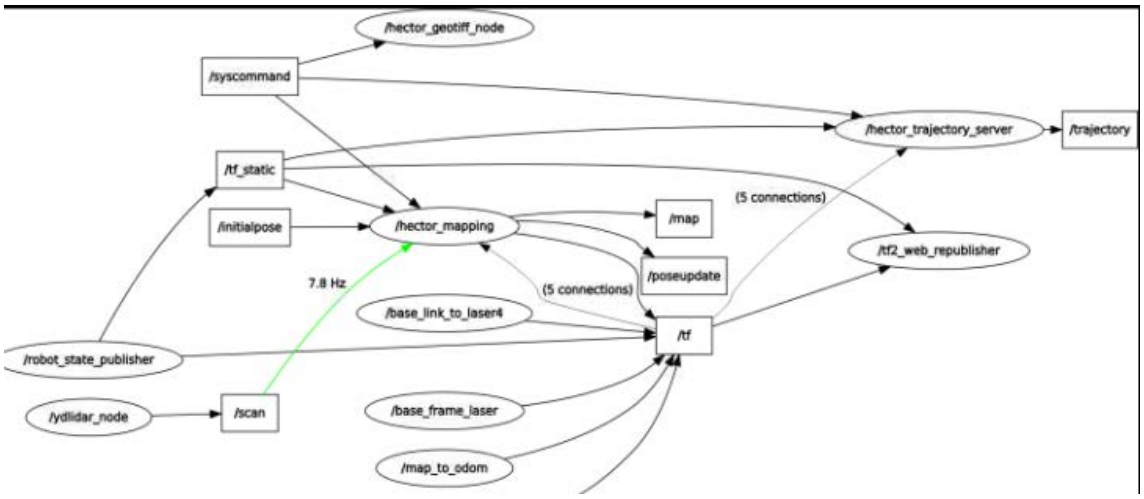

Fig. 38. Gráfica implementación del SLAM

La Fig. 38 representa la parte de la implementación de la técnica SLAM. Haciendo uso de los datos publicados en el tema */scan* por el nodo */ydlidar_node*, el nodo */hector_mapping* genera el mapa del entorno haciendo uso de las transformadas.

```
Serial: 2018101900004271
[YDLidar]: [YDLIDAR INFO] Current Sampling Rate : 4K
process[base_link_to_laser4-2]: started with pid [3177]
process[map_to_odom-3]: started with pid [3178]
[YDLidar]: [YDLIDAR INFO] Now YDLIDAR is scanning ......

process[base_frame_laser-4]: started with pid [3190]
process[hector_mapping-5]: started with pid [3201]
process[rviz-6]: started with pid [3212]
process[hector_trajectory_server-7]: started with pid [3213]
process[hector_geotiff_node-8]: started with pid [3228]
[ INFO] [1561918170.918942988]: Waiting for tf transform data between frames /ma
p and /base_link to become available
HectorSM map lvl 0: cellLength: 0.05 res x:2048 res y: 2048
HectorSM map lvl 1: cellLength: 0.1 res x:1024 res y: 1024
[ INFO] [1561918173.101068126]: HectorSM p_base_frame_: base_link
[ INFO] [1561918173.104477333]: HectorSM p_map_frame_: map
[ INFO] [1561918173.107758727]: HectorSM p_odom_frame_: odom
[ INFO] [1561918173.109313826]: HectorSM p_scan_topic_: scan
[ INFO] [1561918173.111191840]: HectorSM p_use_tf_scan_transformation_: true
[ INFO] [1561918173.113996310]: HectorSM p_pub_map_odom_transform_: true
[ INFO] [1561918173.117109111]: HectorSM p_scan_subscriber_queue_size_: 5
[ INFO] [1561918173.117417860]: HectorSM p_map_pub_period_: 2.000000
[ INFO] [1561918173.117804057]: HectorSM p_update_factor_free_: 0.400000
[ INFO] [1561918173.118216295]: HectorSM p_update_factor_occupied_: 0.900000
[ INFO] [1561918173.118394836]: HectorSM p_map_update_distance_threshold_: 0.400
000
[ INFO] [1561918173.118572960]: HectorSM p_map_update_angle_threshold_: 0.060000
[ INFO] [1561918173.118772959]: HectorSM p_laser_z_min_value_: -1.000000
[ INFO] [1561918173.118986032]: HectorSM p_laser_z_max_value_: 1.000000
libEGL warning: DRI2: failed to authenticate
[ INFO] [1561918175.794427195]: Successfully initialized hector_geotiff MapWrite
r plugin TrajectoryMapWriter.
```

Fig 39. Ejecución AllNodes.launch

Para la visualización de lo implementado como ya se ha mencionado, se usa la herramienta Rviz. En esta herramienta se recogerán todos los temas generados por el sistema, pudiendo visualizarlos seleccionándolos desde la propia interface de Rviz.

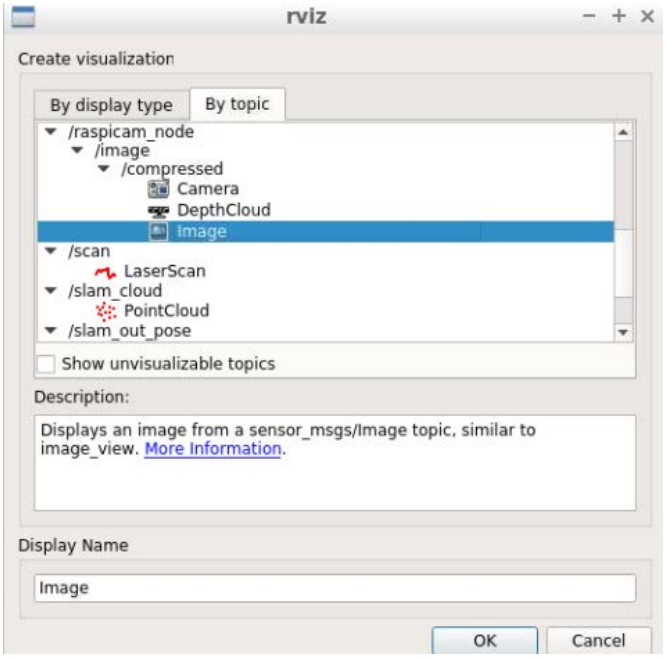

Fig. 40. Añadir temas en la herramienta Rviz

9.2. GRAFICAR LAS MEDICIONES DEL LIDAR EN LABVIEW

Es un caso similar al del apartado 8.2, pero en vez de crear un nodo publicador se crea un nodo al que esté suscrito el tema */scan* que publica el LiDAR.

En la Fig. 41 se muestra el diagrama de flujo de la obtención gráfica de los datos del LiDAR móvil mediante LabVIEW.

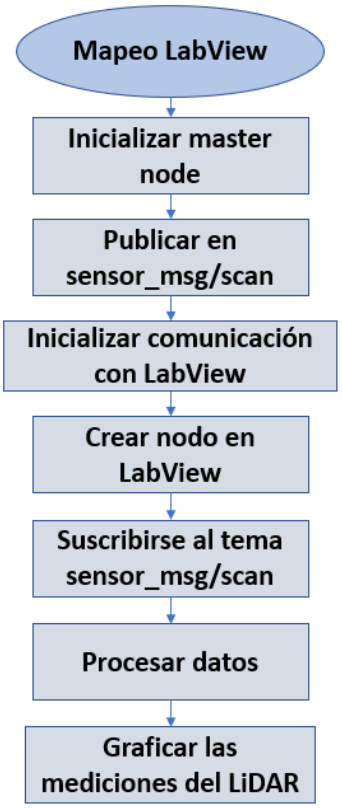

Fig. 41. Algoritmo de mapeo LabVIEW

- Programación gráfica

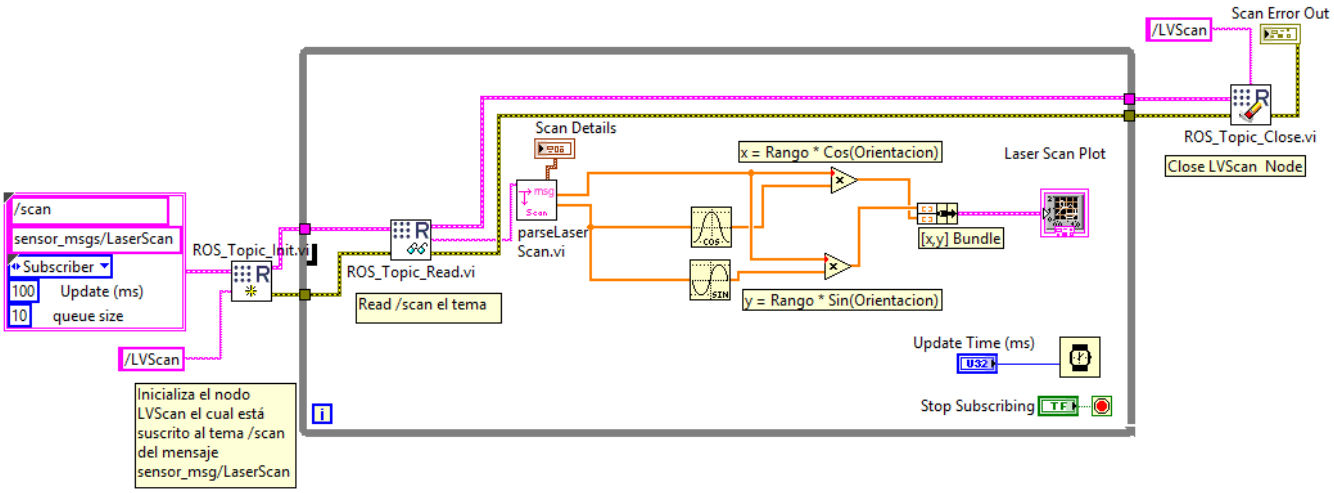

Fig. 42. Diagrama de bloques mapeo LabVIEW

El subVI *ROS_Topic_Init* crea el nodo mencionado con las características indicadas en su entrada:

- **Nombre del nodo**: /LVScan
- **Tema**: /scan
- **Tipo de mensaje**: sensor_msgs/LaserScan
- **Publicador o suscriptor**: Suscriber
- **Tiempo de actualización**: 100 ms
- **Tamaño del mensaje**: 10 bit

El subVI *ROS_Topic_Read* se encarga de leer los datos del nodo creado.

El subVI *parseLaserScan* analiza el tema */scan* al que está suscrito el nodo creado. Este subVI genera una matriz de rango y de rumbo de dimensiones 1 x n donde n es el número de haces (beam) seleccionado. El rango y el rumbo se convierten en las posiciones x e y del pulso de luz.

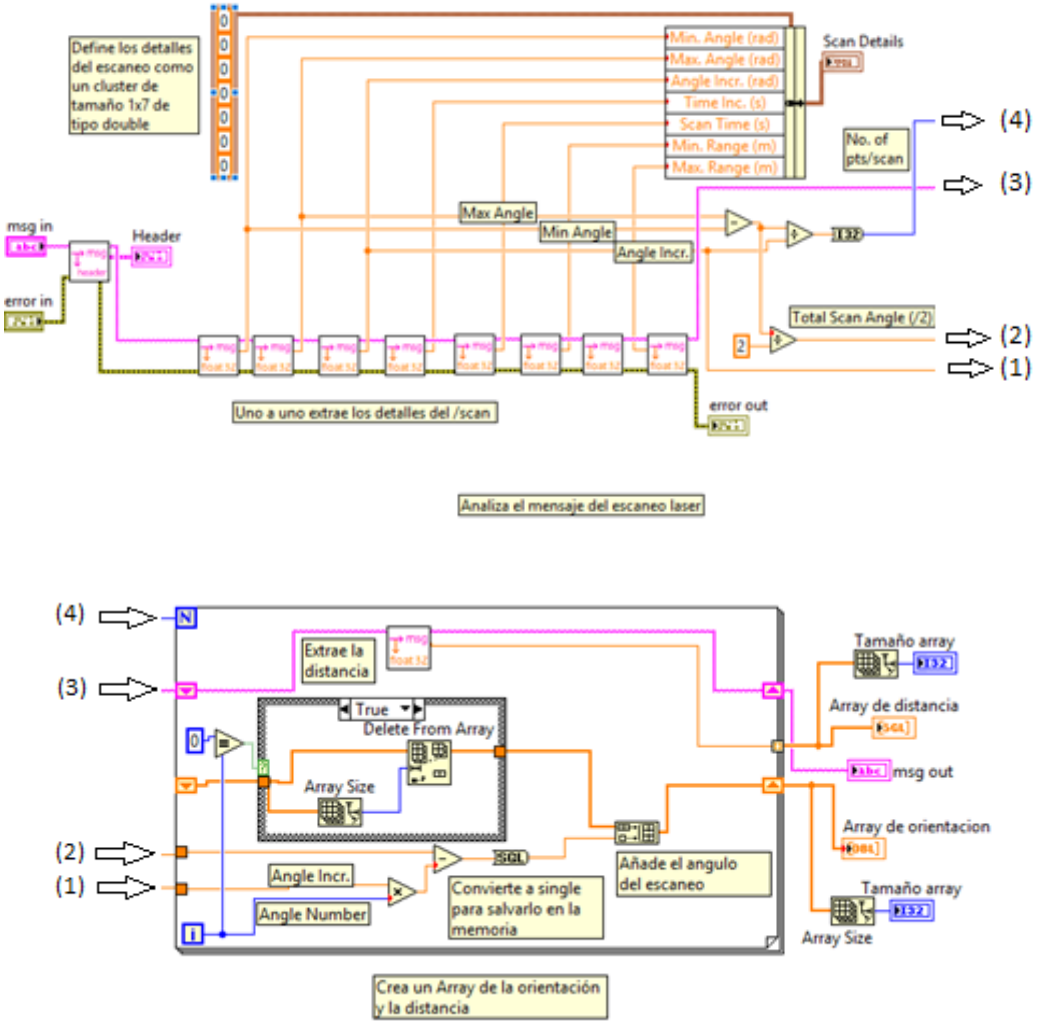

Fig. 43. Diagrama de bloques parseLaserScan

Por último, se cierra el nodo creado, para evitar problemas al reiniciar el programa.

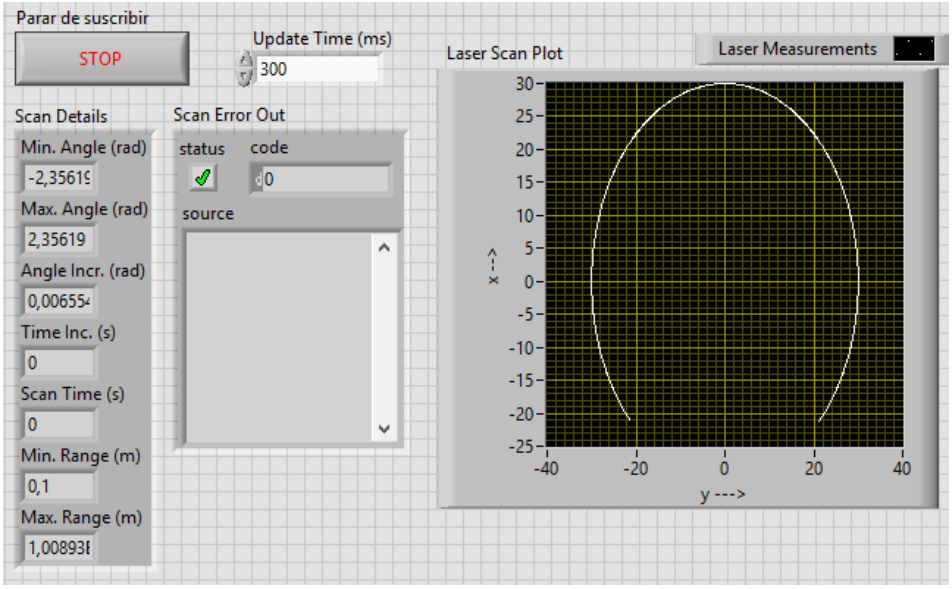

Fig 44. Panel frontal mapeo LabVIEW

10. IMPLEMENTACIÓN DE OBTENCIÓN DE DATOS SENSOR IMU

En este apartado se explica cómo obtener mediciones del sensor IMU mediante un programa Arduino, para posteriormente publicar estas mediciones como tema /imu.

La instalación de los paquetes necesarios para la implementación de este apartado está explicada en el apartado de anexos de instalación.

- Algoritmo de funcionamiento

En la Fig. 45 se observa el algoritmo de funcionamiento de la obtención de datos del sensor IMU:

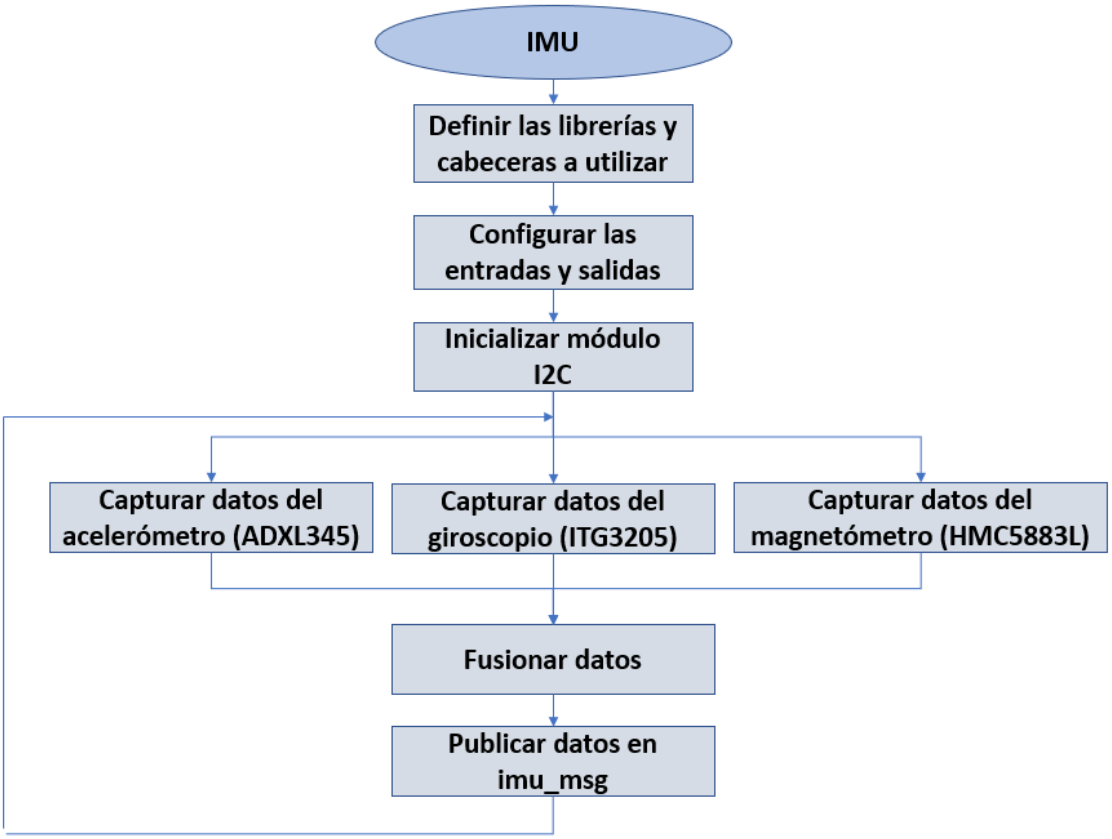

Fig. 45. Algoritmo de implementación del IMU

La obtención de datos se realiza desde el programa de Arduino IDE, por lo que primero será necesario definir las librerías del programa para la comunicación con la Raspberry Pi y las librerías del bus I2C. Tras configurar las entradas y salidas usando las direcciones de los sensores que incorpora el GY-85 se carga el programa en el Arduino, inicializando el módulo I2C. El módulo I2C captura los datos del acelerómetro, giroscopio y magnetómetro. Estos datos se usan conjuntamente generando un mensaje con todas las capturas realizadas. Finalmente se publica este mensaje conjunto como tema /imu, para después procesarlo en la Raspberry con el fin de mejorar la odometría del sistema.

- Código Arduino

```
#include <ros.h>  // Incluye librería para comunicación con ROS
#include <std_msgs/String.h>  // Incluye librería para generación de mensaje string
#include <Wire.h>  // Incluye librería para el bus I2C
const int GY_addr=0x68;  // Dirección I2C del GY-85
int16_t AcX,AcY,AcZ,Tmp,GyX,GyY,GyZ;
// Configura el nodo de ROS y el mensaje publicador
std_msgs::String imu_msg;
ros::Publisher imu("imu", &imu_msg);
ros::NodeHandle node;
// En la función setup, se inicializa en nodo
void setup()
{
 Serial.begin(57600);
 node.initNode();
 node.subscribe(sub);
 node.advertise(imu);
 Wire.begin();
 Wire.beginTransmission(GY_addr);
 Wire.write(0x6B);  // PWR_MGMT_1 register
 Wire.write(0);     // set to zero
 Wire.endTransmission(true);
}
long publisher_timer;
void loop()
{
 Wire.beginTransmission(GY_addr);
 Wire.write(0x3B);  //se inicia con el registro del acelerometro en el eje X
 Wire.endTransmission(false);
 Wire.requestFrom(MPU_addr,14,true);
 AcX=Wire.read()<<8|Wire.read();  // 0x3B (ACCEL_XOUT_H) & 0x3C (ACCEL_XOUT_L)
 AcY=Wire.read()<<8|Wire.read();  // 0x3D (ACCEL_YOUT_H) & 0x3E (ACCEL_YOUT_L)
 AcZ=Wire.read()<<8|Wire.read();  // 0x3F (ACCEL_ZOUT_H) & 0x40 (ACCEL_ZOUT_L)
```

```
  Tmp=Wire.read()<<8|Wire.read();  // 0x41 (TEMP_OUT_H) & 0x42 (TEMP_OUT_L)
  GyX=Wire.read()<<8|Wire.read();  // 0x43 (GYRO_XOUT_H) & 0x44 (GYRO_XOUT_L)
  GyY=Wire.read()<<8|Wire.read();  // 0x45 (GYRO_YOUT_H) & 0x46 (GYRO_YOUT_L)
  GyZ=Wire.read()<<8|Wire.read();  // 0x47 (GYRO_ZOUT_H) & 0x48 (GYRO_ZOUT_L)
  String AX = String(AcX);
  String AY = String(AcY);
  String AZ = String(AcZ);
  String GX = String(GyX);
  String GY = String(GyY);
  String GZ = String(GyZ);
  String tmp = String(Tmp);
  String data = "A" + AX + "B"+ AY + "C" + AZ + "D" + GX + "E" + GY + "F" + GZ + "G" ;
  Serial.println(data);
  int length = data.indexOf("G") +2;
  char data_final[length+1];
  data.toCharArray(data_final, length+1);
  // Se publica el mensaje cada 100 ms
  if (millis() > publisher_timer) {
    imu_msg.data = data_final;
    imu.publish(&imu_msg);
    publisher_timer = millis() + 100;
  }
  node.spinOnce();
  delay(1);
}
```

```
ubuntu@ubiquityrobot:~$ rosrun rosserial_python serial_node.py /dev/ttyACM0
[INFO] [1561392706.052071]: ROS Serial Python Node
[INFO] [1561392706.077327]: Connecting to /dev/ttyACM0 at 57600 baud
[INFO] [1561392708.190512]: Requesting topics...
[INFO] [1561392708.231858]: Note: publish buffer size is 280 bytes
[INFO] [1561392708.239183]: Setup publisher on imu [std_msgs/String]
[INFO] [1561392708.369478]: Note: subscribe buffer size is 280 bytes
[INFO] [1561392708.371818]: Setup subscriber on cmd_vel [geometry_msgs/Twist]
```

Fig. 46. Programa Arduino publicando un tema y suscrito a otro tema a la vez

11. RESULTADOS OBTENIDOS

En este apartado se muestran los resultados obtenidos en las implementaciones llevadas a cabo a lo largo del proyecto.

Se ha sometido el sistema implementado de SLAM, mapeo y localización simultánea, a diferentes pruebas cuyos resultados se mostrarán por la interfaz gráfica RVIZ.s

Sobre la visualización de los datos del escáner láser en LabVIEW se mostrarán en una gráfica, y se analizarán los resultados del control de la trayectoria para valorar si su funcionamiento ha sido el esperado.

- **Construcción del robot móvil:**

El resultado del robot móvil construido es el siguiente:

- Dimensiones: 22 cm de largo, 19 cm de ancho y 14 cm de altura.
- Peso: 1,4 kg con todos los elementos.

En las siguientes imágenes se observa el aspecto del robot móvil:

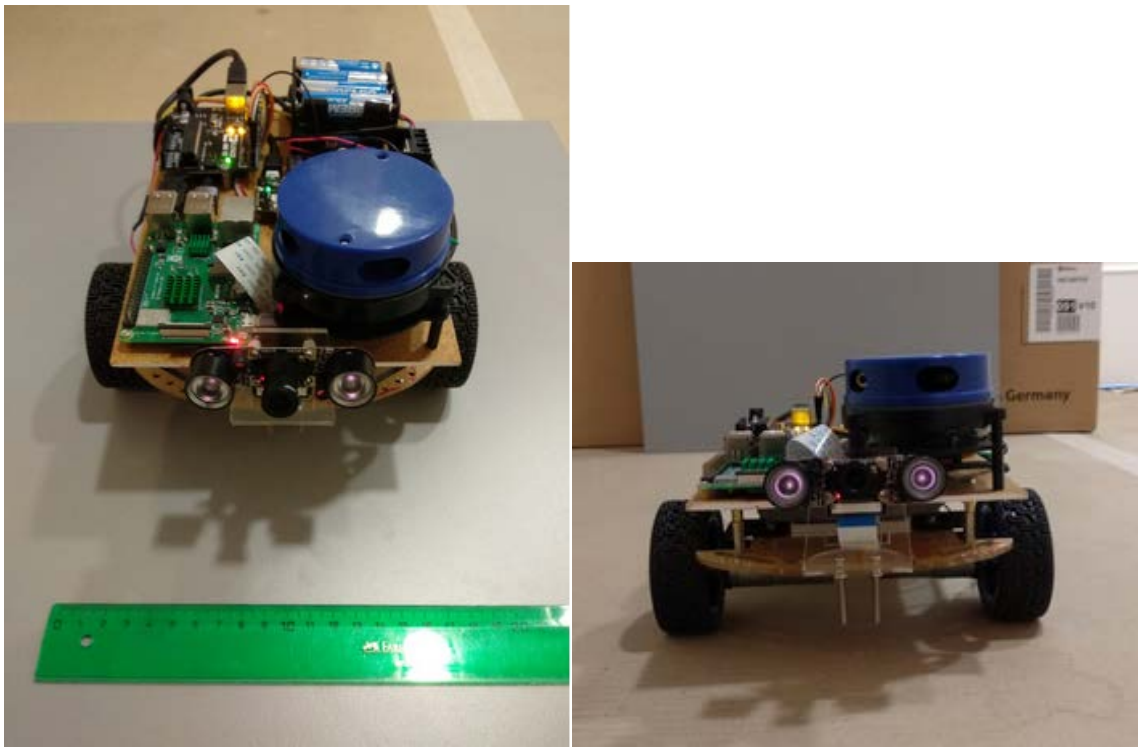

Fig. 47. Vista frontal del robot móvil

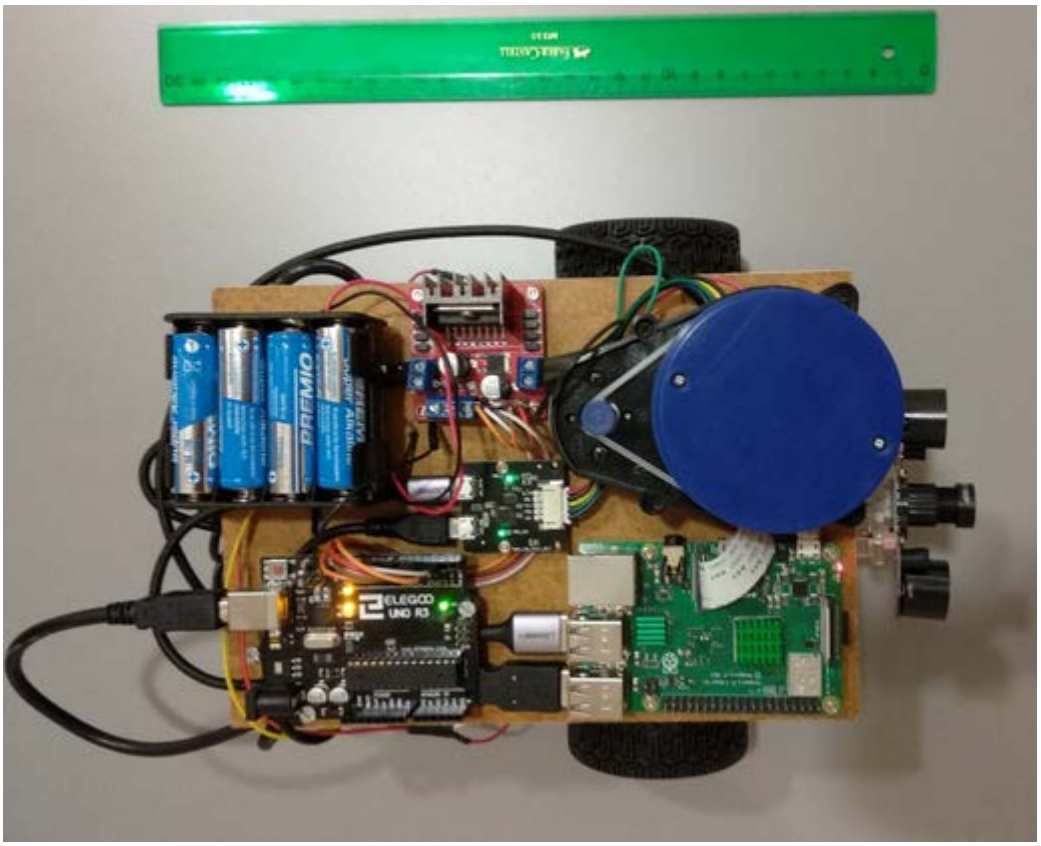

Fig. 48. Vista desde arriba del robot móvil

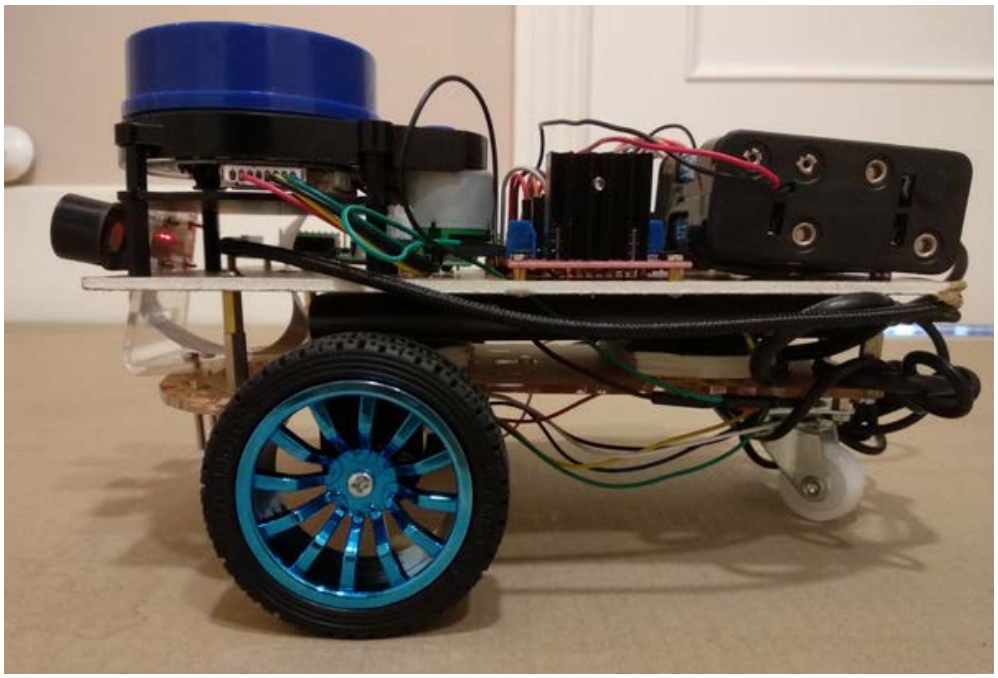

Fig. 49. Vista lateral del robot móvil

El resultado final es una estructura bastante sólida y compacta, pero como posible mejora se podría realizar una mejor organización de los cables de conexión y de los cables conectores USB, de manera que sean más discretos a la vista. Para ello se podrían utilizar guías recogedoras de cable y cables USB de longitudes más cortas.

- **Implementación control trayectoria del robot:**

Los resultados obtenidos en este apartado han sido satisfactorios. Destacar como problema que en primera instancia, debido a que los primeros motores usados en el proyecto no tenían la potencia suficiente para mover el robot hubo que cambiar de motores a unos más potentes, ya que sin estos el control resultaba imposible. Tras realizar el cambio, el control de la trayectoria se lleva a cabo de la manera esperada, pudiendo mejorar la respuesta añadiendo un control de lazo cerrado que con las señales de los encoders puedan ajustar la velocidad de ambos motores, de manera que el robot se desviase menos en las trayectorias rectas.

Destacar que aunque los dos métodos implementados para esta tarea han sido un éxito, resulta más sencillo el control mediante el paquete *teleop_twist_keyboard*, ya que el cambio de dirección se realiza de manera más cómoda, simplemente tecleando. En el control mediante el panel frontal de LabVIEW resulta más complicado el control de trayectoria, ya que desde este hay que realizar cambios con el puntero de una manera más lenta, pudiendo ser un problema en situaciones de riesgo en las que se necesite una respuesta rápida para el robot, como detener el robot frente a algún obstáculo inesperado.

- **Implementación de sistema SLAM:**

En este apartado se somete a distintas pruebas el sistema SLAM implementado, para analizar el comportamiento de este ante distintas trayectorias y condiciones del entorno.

- Interfaz sistema SLAM

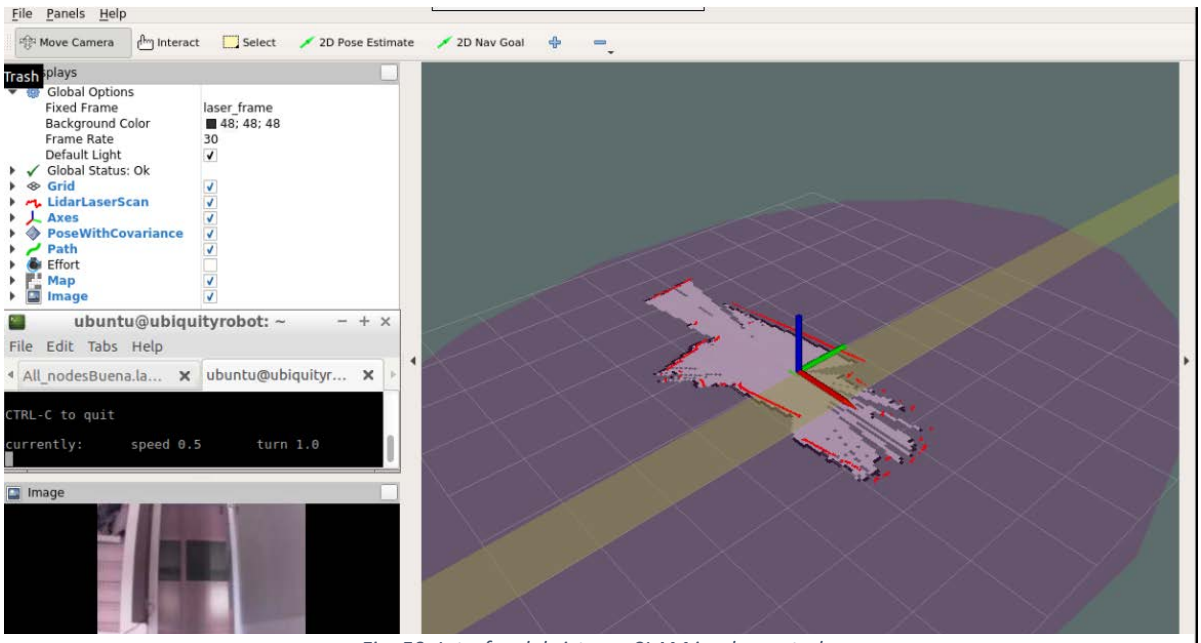

Fig. 50. Interfaz del sistema SLAM implementado

En la Fig. 50 se observa la interfaz desarrollada para la implementación de la técnica SLAM y el control de trayectoria del robot, de manera que se pueda dirigir de manera inalámbrica el robot al mismo tiempo que se observa la imagen de la cámara, el mapeo y la trayectoria seguida por este.

Importante añadir que para controlar la trayectoria del robot será necesario tener el puntero en la ventana de comandos, en la pestaña en la que se haya ejecutado el paquete *teleop_twist_keyboard*.

- Mapeo de una habitación

En la Fig. 51 se observa el resultado del mapeo realizado en una sola habitación, la cual ha sido usada de pruebas para la configuración de los parámetros de la técnica SLAM (Fig. 52). Se observa que con un barrido láser el LiDAR es capaz de mapear con claridad casi en su totalidad toda la habitación. Destacar que los tres ejes indican la posición y orientación actual del robot.

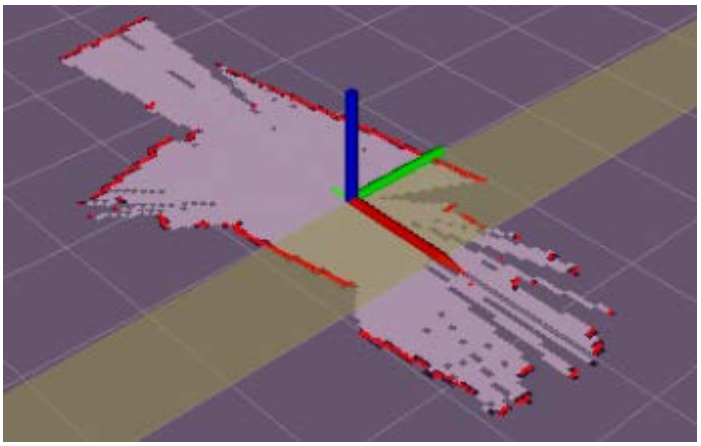

Fig. 51. Mapeo de una habitación

Fig. 52. Entorno de la habitación mapeada

- Mapeo en varias habitaciones

En la Fig. 53 se observan los resultados de la implementación del sistema SLAM haciendo un recorrido de todas las habitaciones de la casa. Los resultados obtenidos son muy poco precisos. Esto puede ser debido a que en el tramo de las puertas, al ser un espacio muy pequeño, el LiDAR puede dar errores de medición, perdiendo la percepción del robot en el entorno. De esta manera al seguir mapeando sin saber su localización, genera un mapa del entorno en una posición distinta a la que debería de estar, creando un mapa confuso.

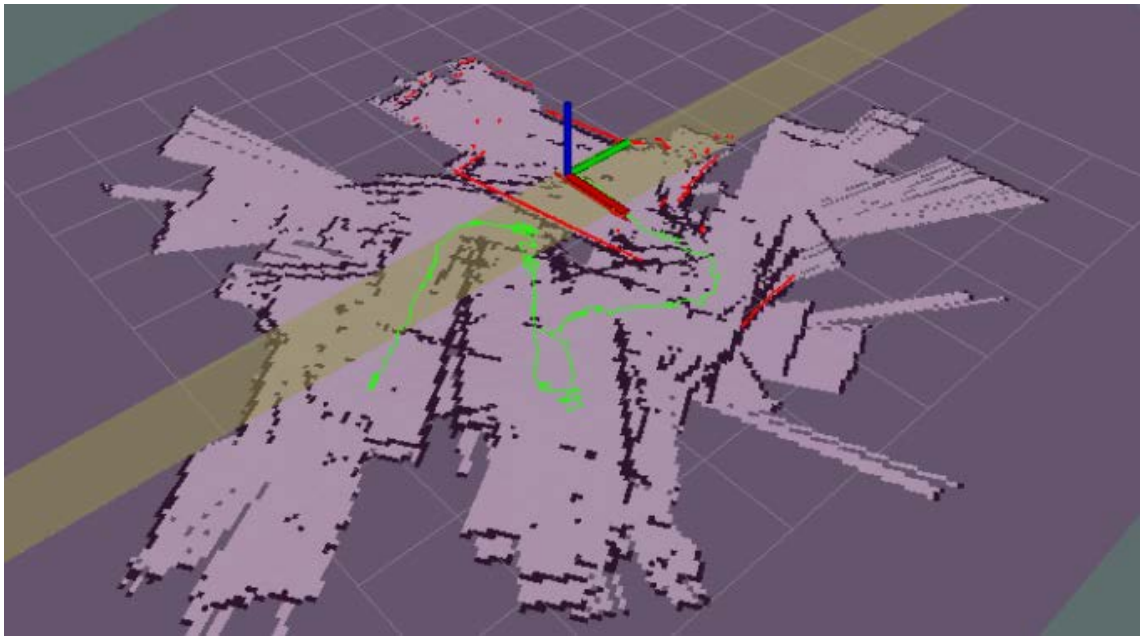

Fig. 53. Resultados del SLAM en varias habitaciones

También se realizará otro recorrido con parámetros más rápidos en el procesamiento de los datos y una velocidad del robot menor para ver si da un mejor resultado.

- Comparación de SLAM con los nuevos parámetros y velocidad más lenta

Aunque el recorrido sea distinto al realizado en el anterior apartado, es suficiente para sacar algunas conclusiones.

Utilizando los valores de mapeo del anterior apartado se ha conseguido el siguiente mapa, a una velocidad media del robot.

Mapping.launch

```
<!-- Map update parameters A -->
  <param name="update_factor_free" value="0.4"/>
  <param name="update_factor_occupied" value="0.7" />
  <param name="map_update_distance_thresh" value="0.2"/>
  <param name="map_update_angle_thresh" value="0.9" />
```

<param name="laser_z_min_value" value = "-1.0" />

<param name="laser_z_max_value" value = "1.0" />

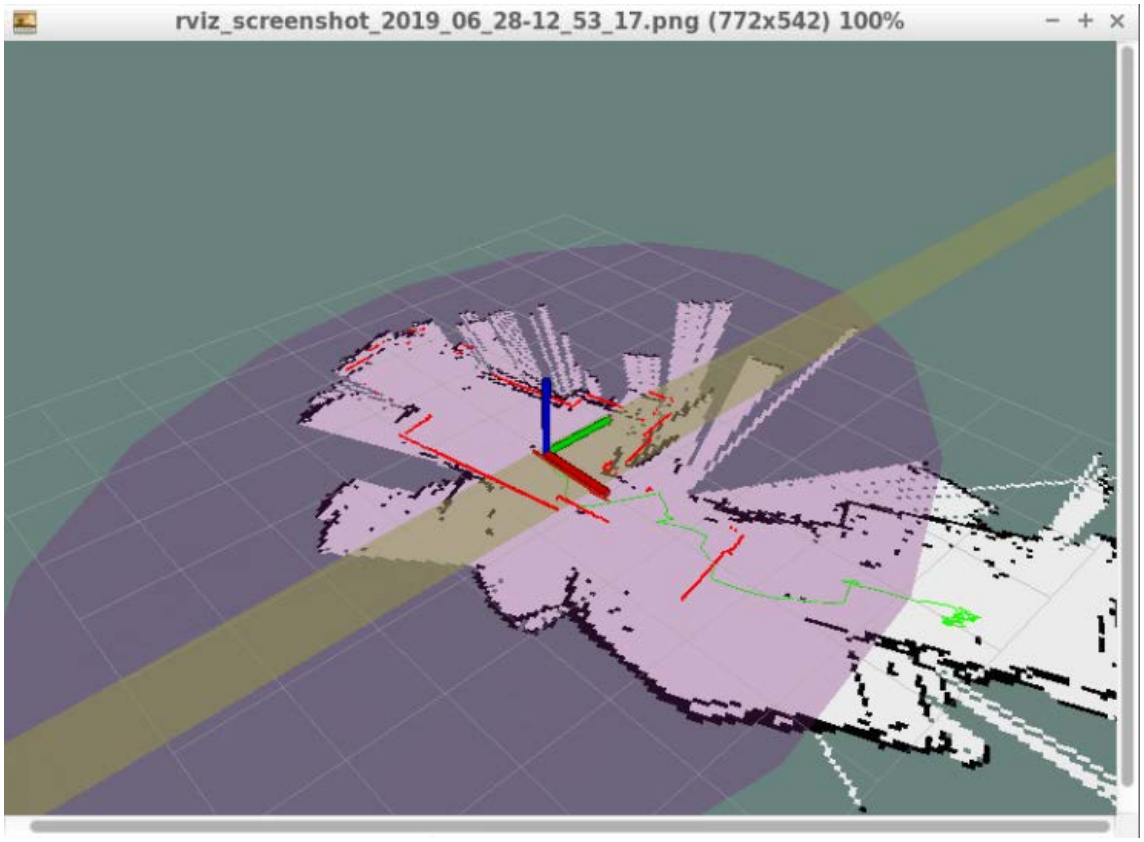

Fig. 54. Resultados del SLAM a velocidad media y parámetros A

Modificando los valores del mapeo para realizar la actualización de ángulo más rápida y la de distancia algo más lenta se han conseguido los siguientes resultados para una velocidad del robot baja.

Mapping.launch

<!-- Map update parameters B -->

<param name="update_factor_free" value="0.4"/>

<param name="update_factor_occupied" value="0.9" />

<param name="map_update_distance_thresh" value="0.4"/>

<param name="map_update_angle_thresh" value="0.06" />

<param name="laser_z_min_value" value = "-1.0" />

<param name="laser_z_max_value" value = "1.0" />

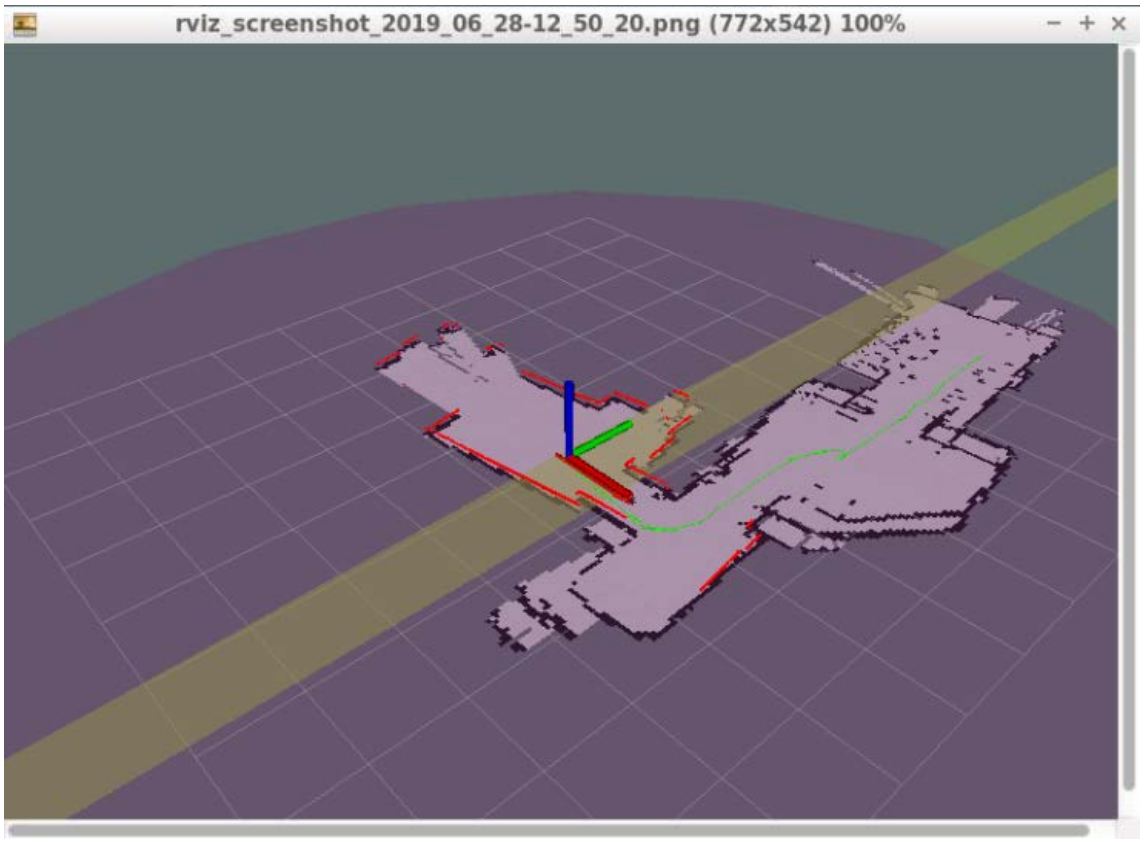

Fig. 55. Resultados del SLAM a baja velocidad y parámetros B

Como puede observarse, los resultados son satisfactorios con algunos errores localizados. La velocidad de transmisión de datos del LiDAR es suficientemente alta para lograr el mapeo en un robot móvil de velocidad baja. Para velocidades más altas será necesario un LiDAR de mejores prestaciones o un microprocesador con mayor capacidad de procesamiento, ya que la Raspberry Pi se puede quedar algo corta a la hora de realizar el SLAM con parámetros de actualización más altos.

Puede observarse en la Fig. 55 la trayectoria seguida marcada por una línea verde, el recorrido ha consistido en ir de una habitación a otra pasando por un corredor estrecho.

Destacar como limitaciones del sensor que cuando el haz láser se refleja en superficies refractarias, tales como cristales, espejos o en este caso los radiadores blancos metálicos, se producen medidas falseadas. Frente a las perturbaciones como baches, el sensor también provoca fallos en el mapeo.

La retícula que se ve de fondo se trata del mapa de celdas, un mapa de ocupación en el que cada celda tiene un metro de lado.

- **Trayectoria circular**

En este apartado se analizan los resultados obtenidos en una trayectoria circular realizada en el entorno reflejado en la Fig. 56:

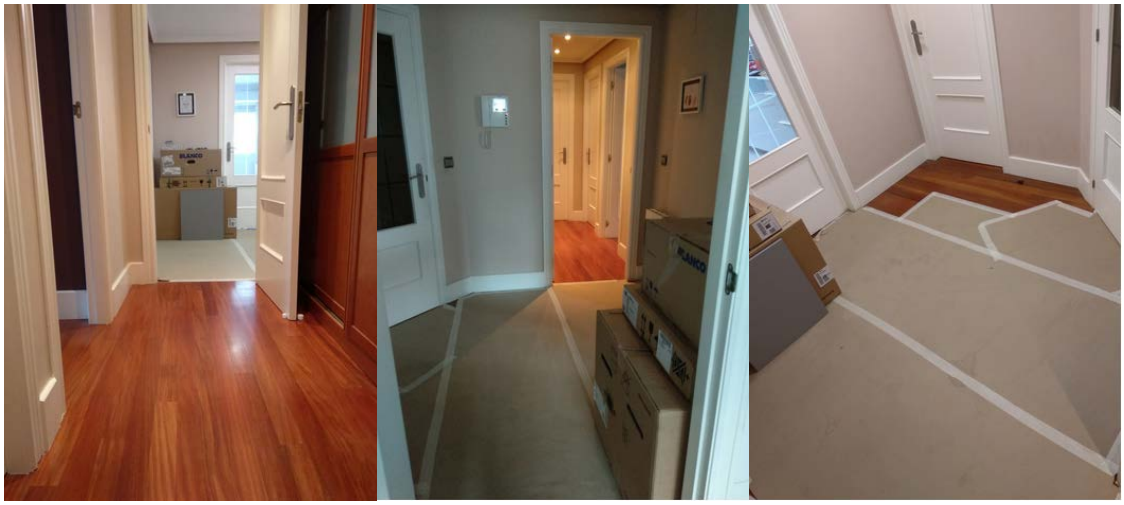

Fig. 56. Entorno de la trayectoria circular

El robot seguirá una trayectoria recta por el primer pasillo, y en el hall donde se encuentran las cajas realizará un giro para volver al punto de partida en el pasillo.

En la Fig. 57 se observa el resultado del SLAM:

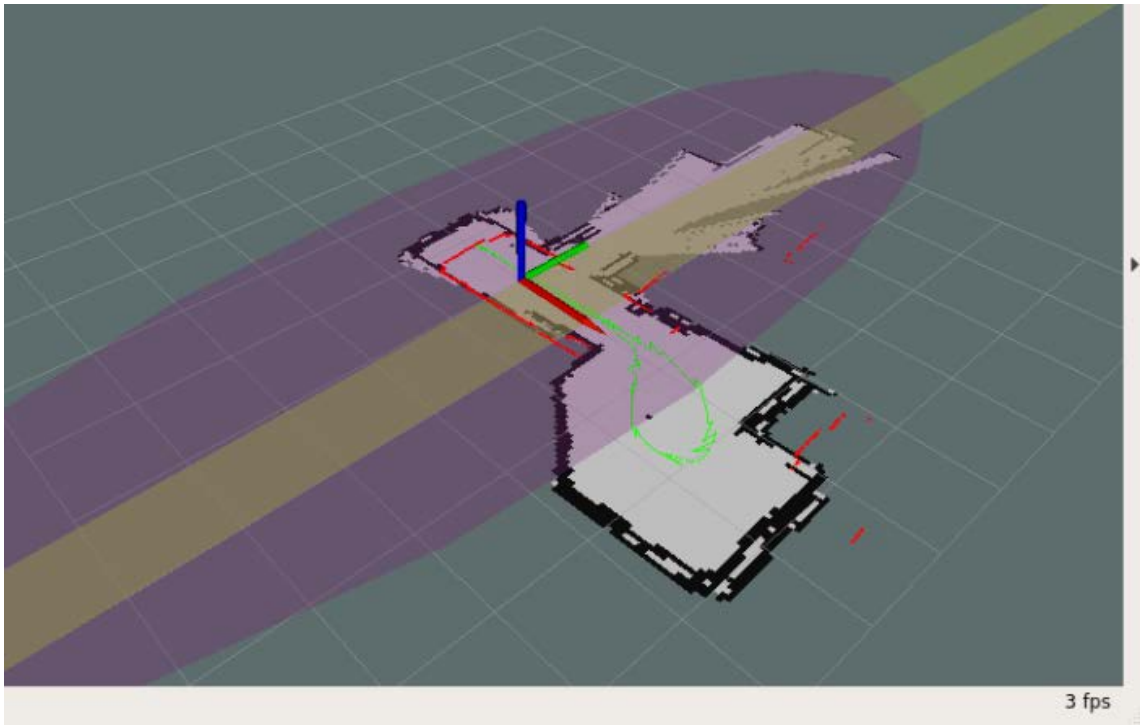

Fig. 57. Resultados SLAM en trayectoria circular

Los resultados son bastante satisfactorios; la trayectoria del robot, indicada por la línea verde, se corresponde con la trayectoria real realizada por el robot móvil. El mapeo también es bastante correcto, teniendo algunos errores sobre todo en el pasillo estrecho.

- **Visualización de mediciones del LiDAR en LabView:**

En este apartado se muestran las mediciones del LiDAR en cada instante de actualización de datos seleccionado. Para la realización de esta tarea se realiza un barrido láser de una habitación, para después usar el tema generado por el escaneo en el nodo suscriptor creado en LabVIEW. En las figuras 58 y 59 se observan los resultados de la visualización del escaneo en LabVIEW:

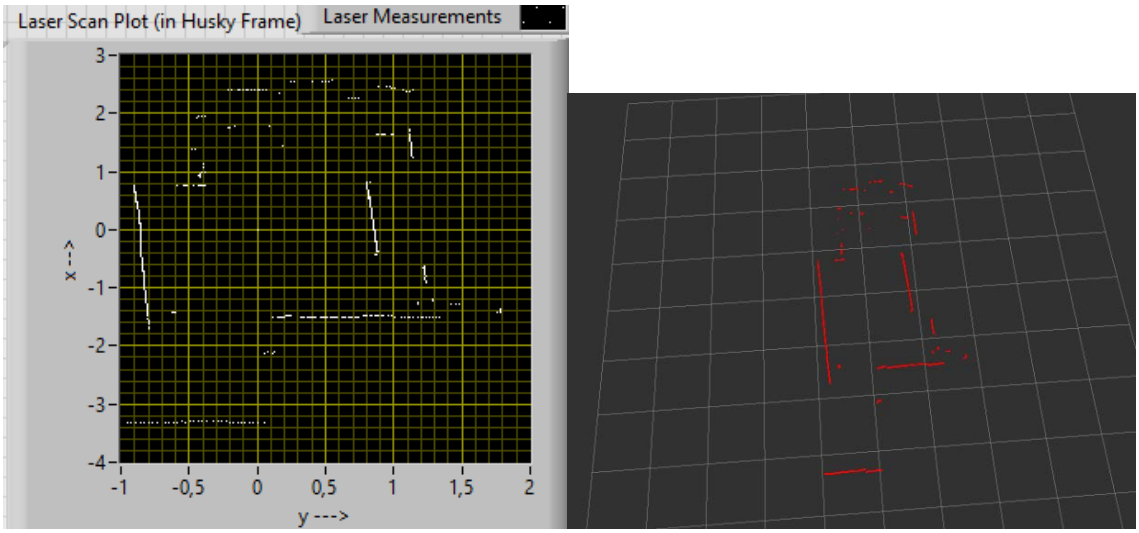

Fig. 58. Visualización mediciones LiDAR LabVIEW vs Rviz

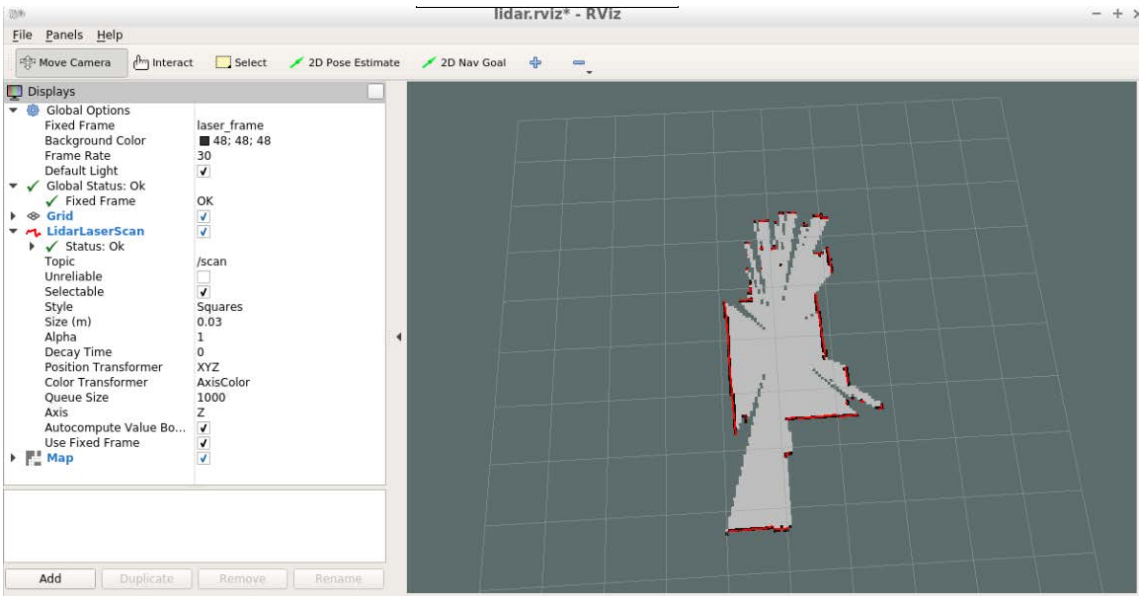

Fig. 59. Mapa de las mediciones obtenidas para LabVIEW

Los resultados obtenidos son bastante satisfactorios, las mediciones visualizadas en el gráfico de LabVIEW se asemejan bastante a mapa real del entorno, teniendo una ligera distorsión en las medidas x del gráfico.

- **Obtención de datos IMU:**

Este apartado ha sido el que más problemas ha dado. Al subir el código de obtención de datos al Arduino, el programa funciona correctamente durante un breve periodo de tiempo, e incluso llega a publicar las mediciones del IMU, como se puede ver en las siguientes figuras.

```
ubuntu@ubiquityrobot:~$ rosrun rosserial_python serial_node.py /dev/ttyACM0
[INFO] [1562609715.225079]: ROS Serial Python Node
[INFO] [1562609715.254601]: Connecting to /dev/ttyACM0 at 57600 baud
[INFO] [1562609717.366056]: Requesting topics...
[INFO] [1562609717.458574]: Note: publish buffer size is 280 bytes
[INFO] [1562609717.459787]: Setup publisher on imu [std_msgs/String]
```

Fig. 60. Publicación serie del tema imu

En la Fig. 66 se observa cómo la comunicación serie se efectúa correctamente y el nodo *serial_node* publica el tema */imu* como un mensaje *std_msgs/String*.

Fig. 61. Gráfico de publicación tema /imu

En la Fig. 62 se observan las lecturas hechas por el sensor, indicando los datos en el siguiente formato: "A" + AX + "B"+ AY + "C" + AZ + "D" + GX + "E" + GY + "F" + GZ + "G

Introduciendo el comando de *rostopic echo imu*, se observan las publicaciones de este tema.

```
---
data: "A0B127C0D0E0F0G"
---
data: "A0B127C0D0E0F0G"
---
data: "A254B0C104D6912E6F0G"
---
data: "A254B0C104D6912E6F0G"
```

Fig. 62. Publicaciones del tema /imu

Pero tras un breve tiempo de funcionamiento, la comunicación serie entre el Arduino y la Raspberry Pi se detiene, generando los errores mostrados en la Fig. 63.

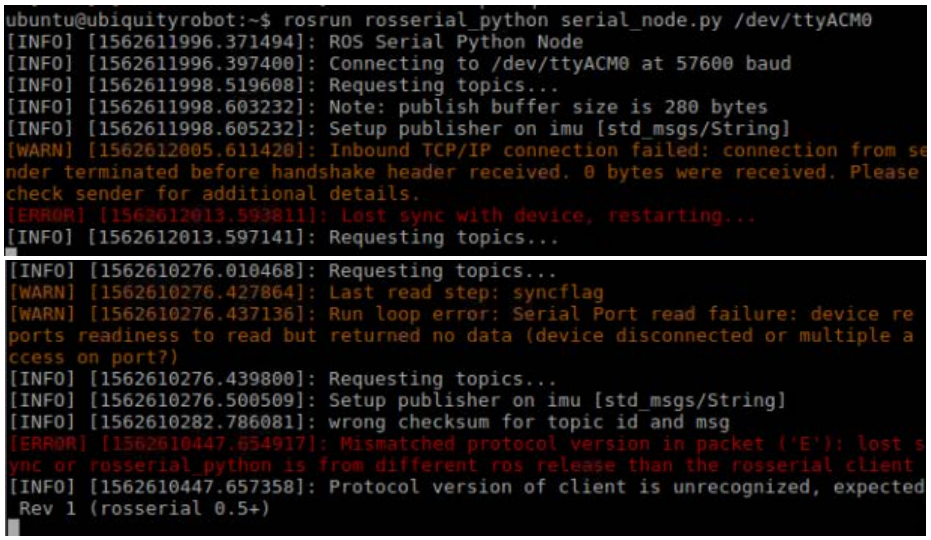

Fig. 63. Errores publicación serie datos IMU

Se cree que es un problema con el buffer del Arduino, que no es capaz de recoger tantos datos en un mensaje y se satura. Aun así, no se sabe concretamente la causa del error. Se probó a conectar el IMU directamente con la Raspberry Pi, pero solo se consiguió la lectura de los resultados, sin lograr con éxito posteriormente publicarlos en el tema */imu*.

- Obtención de datos IMU, conexión directa a la Raspberry

Tabla 8. Conexionado entre IMU y Raspbery Pi

MODELO	**SDA**	**SCL**
Raspberry Pi 3B+	GPIO 0 (SDA)	GPIO 1 (SCL)

En el Anexo de instalaciones se indica cómo hacer la instalación para el uso del bus I2C en la Raspberry Pi. Tras realizar las correspondientes instalaciones de las librerías, introduciendo los comandos indicados en la Fig. 64 se obtienen las lecturas de los sensores del IMU.

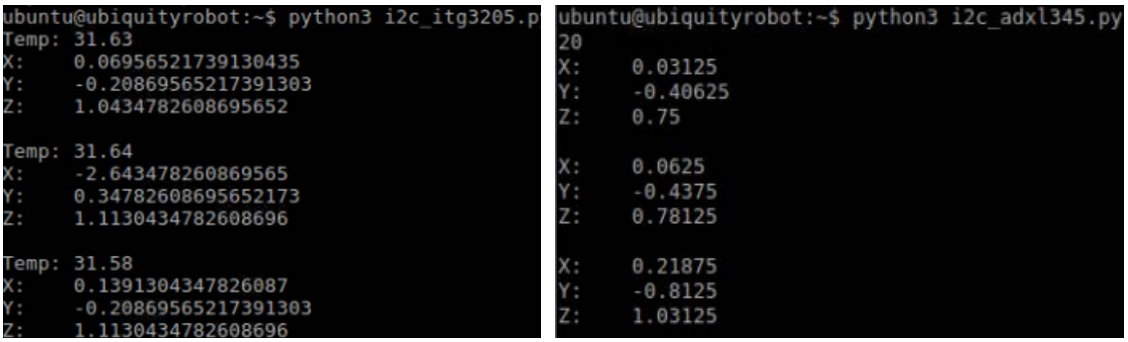

Fig. 64. Lectura mediciones giroscopio (izquierda) y acelerómetro (derecha)

12. PLANIFICACIÓN Y PLAN DE PROYECTO

A continuación se exponen las fases y tareas con las que se han ido cumpliendo los objetivos mencionados:

- Fase 1: Comienzo. Se define el alcance del proyecto de fin de grado a desarrollar.
  - Tarea 1: Estudio de posibilidades del proyecto.
  - Tarea 2: Elección del proyecto y definir su alcance.
- Fase 2: Planificación.
  - Tarea 3: Planificación de los procesos a seguir para conseguir los objetivos, incluyendo la compra de componentes.
- Fase 3: Familiarizarse en el entorno de ROS. Engloba todas las tareas de aprendizaje en este entorno, se hace bastante uso de las clases impartidas en el canal de Youtube, *The Construct*.
  - Tarea 4: Funcionamiento básico de ROS.
  - Tarea 5: Sistemas de navegación.
  - Tarea 6: Técnica SLAM.
- Fase 4: Construcción del robot móvil.
  - Tarea 7: Diseño del robot móvil.
  - Tarea 8: Construcción del robot móvil.
- Fase 5: Implementación en el mundo real de lo aprendido.
  - Tarea 9: Visualizar las mediciones del LiDAR.
  - Tarea 10: Implementación de sistema de navegación.
  - Tarea 11: Implementación de sistema SLAM.
  - Tarea 12: Implementación de odometría mediante sensor IMU
- Fase 6: Pruebas y correcciones.
  - Tarea 13: Reflejar los resultados de las pruebas realizadas.
  - Tarea 14: Análisis de los resultados y posibles correcciones.
  - Tarea 15: Implementación de las correcciones.
- Fase 7: Cierre del proyecto. Realizar la documentación correspondiente de todas las fases del proyecto.

La Tabla 9 recoge la duración de estas tareas:

Tabla 9. Organización temporal de tareas

Fases, Tareas	Duración (días)
Comienzo (fase 1)	**5**
Tarea 1	3
Tarea 2	2
Planificación (fase 2)	**10**
Tarea 3	10
Familiarizarse en el entorno de ROS (fase 3)	**30**
Tarea 4	12
Tarea 5	7
Tarea 6	11
Construcción del robot móvil (fase 4)	**10**
Tarea 7	4
Tarea 8	6
Implementación en el mundo real de lo aprendido (fase 5)	**35**
Tarea 9	5
Tarea 10	7
Tarea 11	15
Tarea 12	8
Pruebas y correcciones (fase 6)	**30**
Tarea 13	7
Tarea 14	5
Tarea 15	18
Cierre del proyecto (fase 7)	**25**
Total	**145**

Para la planificación temporal del proyecto, se tienen en cuenta las siguientes condiciones:

- 2 horas de trabajo al día.

$$Horas\ totales\ de\ trabajo = 145 días * 2\frac{horas}{dias} = 290\ horas \quad (19)$$

- Se trabaja todos los días de la semana, festivos incluidos.

Una vez identificadas todas las tareas que se llevarán a cabo y el tiempo que necesitan cada una de estas, se realiza un diagrama de Gantt que muestra de forma visual y sencilla la planificación temporal para realizar dichas tareas (Fig. 65). Este diagrama se ha realizado con el programa GanttProject.

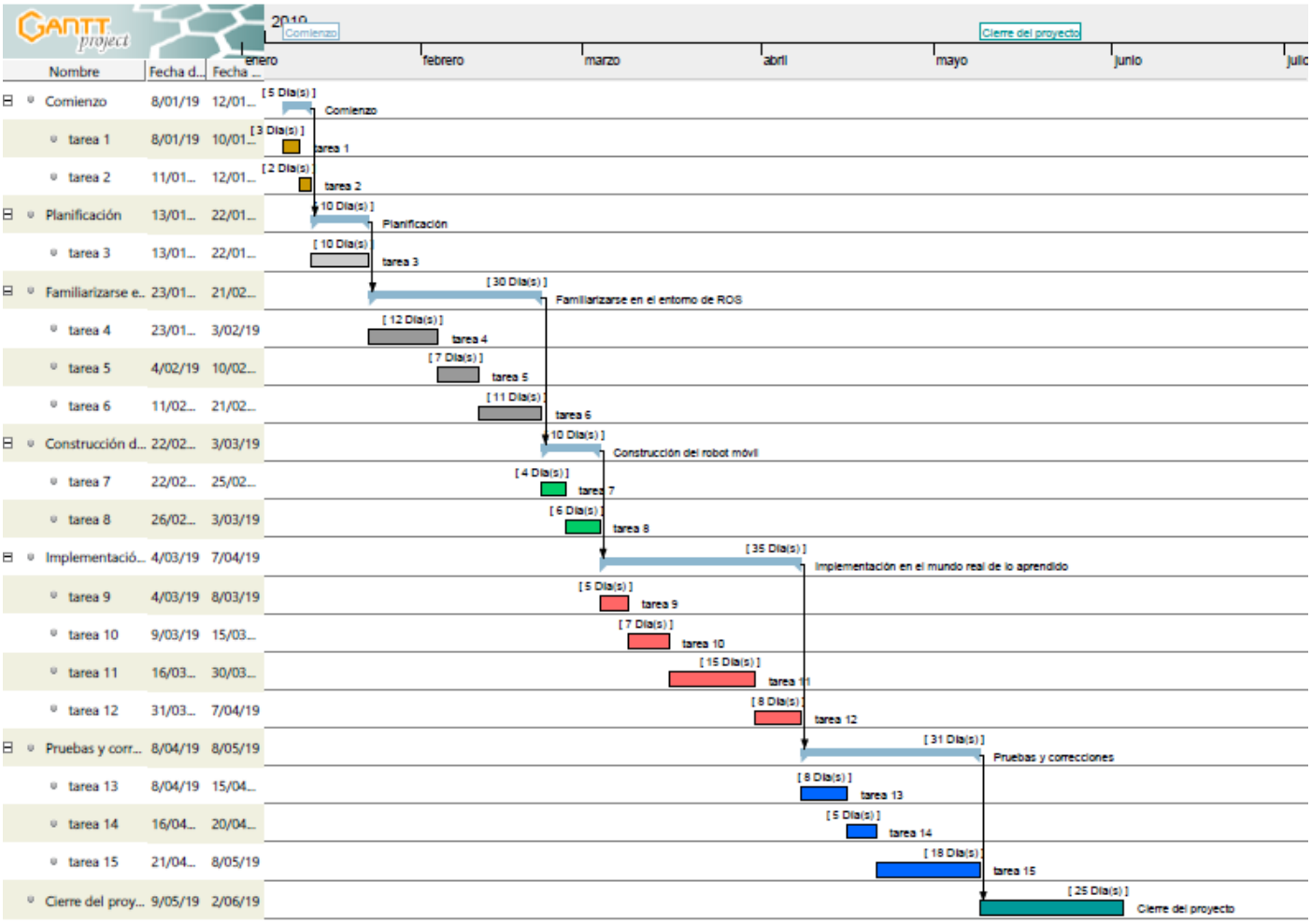

Fig 65. Diagrama de Gantt

Nombre	Fecha de inicio	Fecha de fin
Comienzo	8/01/19	12/01/19
tarea 1	8/01/19	10/01/19
tarea 2	11/01/19	12/01/19
Planificación	13/01/19	22/01/19
tarea 3	13/01/19	22/01/19
Familiarizarse en el entorno de ROS	23/01/19	21/02/19
tarea 4	23/01/19	3/02/19
tarea 5	4/02/19	10/02/19
tarea 6	11/02/19	21/02/19
Construcción del robot móvil	22/02/19	3/03/19
tarea 7	22/02/19	25/02/19
tarea 8	26/02/19	3/03/19
Implementación en el mundo real de lo aprendido	4/03/19	7/04/19
tarea 9	4/03/19	8/03/19
tarea 10	9/03/19	15/03/19
tarea 11	16/03/19	30/03/19
tarea 12	31/03/19	7/04/19
Pruebas y correcciones	8/04/19	8/05/19
tarea 13	8/04/19	15/04/19
tarea 14	16/04/19	20/04/19
tarea 15	21/04/19	8/05/19
Cierre del proyecto	9/05/19	2/06/19

Fig 66. Planificación temporal de las tareas

13. PRESUPUESTO

A continuación, se hará el cálculo del coste total del proyecto de fin de grado realizado. Se tendrán en cuenta los siguientes factores a la hora de realizar el cálculo:

- Horas invertidas: Las horas trabajadas en el proyecto.
- Amortizaciones: El coste proporcional de las herramientas usadas.
- Gastos: La compra de los componentes del robot móvil.

o Componentes del proyecto

En la Tabla 10 se detallan los componentes y el número de estos utilizados en el proyecto:

Tabla 10. Componentes del proyecto

N°	Componentes	N° de unidades
1	Halija 12 V 8 x AA batería	1
2	Kit roscadores y separadores	1
3	Cable Micro USB 2 A	2
4	GY-85	1
5	Kit ruedas 65 mm	1
6	Driver L298N	1
7	Kit Robot inteligente 2WD	1
8	RaspiCam	1
9	Raspberry Pi 3 B+	1
10	YDLIDAR X4	1
11	PowerBank 12000 mAh	1
12	Elegoo Arduino Starter Kit	1

o Precios unitarios

Tabla 11. Precio unitario de los componentes del proyecto

N°	Componentes	Precio unitario (Euros)
1	Halija 12 V 8 x AA batería	7
2	Kit roscadores y separadores	14
3	Cable Micro USB 2 A	2,22
4	GY-85	5,34
5	Kit ruedas 65 mm	20,78
6	Driver L298N	1,38
7	Kit Robot inteligente 2WD	7,42
8	RaspiCam	12,76
9	Raspberry Pi 3 B+	36,01
10	YDLIDAR X4	88,29
11	PowerBank 12000 mAh	18,48
12	Elegoo Arduino Starter Kit	32

En la Tabla 11 se muestran los precios unitarios de cada componente.

- Amortizaciones

En la Tabla 12 se muestra el coste proporcional del equipamiento utilizado en el proyecto.

Tabla 12. Coste de las amortizaciones

	Valor de compra	Vida útil	Tiempo utilizado	Total (€)
ORDENADOR	800 €	5 años	270 h	4,94
			TOTAL	**4,94**

$$\frac{800\ €}{5\text{años}} = 160\ €\ /\text{año} \quad (20)$$

$$\frac{365*24\ \text{horas}}{270\ \text{horas}} = 32{,}4 \quad (21)$$

$$Total\ amortización = \frac{160}{32{,}4} = 4{,}94\ € \quad (22)$$

- Valor total del material utilizado

En la Tabla 13 se muestra el valor total de los componentes utilizados en el proyecto:

Tabla 13. Valor total del material utilizado

N°	Componentes	Precio unitario (Euros)	N° de unidades	Amortización (Euros)	Valor total (Euros)
1	Halija 12 V 8 x AA batería	7	1		7
2	Kit roscadores y separadores	14	1		14
3	Cable Micro USB 2 A	2,22	2		4,44
4	GY-85	5,34	1		5,34
5	Kit ruedas 65 mm	20,78	1		20,78
6	Driver L298N	1,38	1		1,38
7	Kit Robot inteligente 2WD	7,42	1		7,42
8	RaspiCam	12,76	1		12,76
9	Raspberry Pi 3 B+	36,01	1		36,01
10	YDLIDAR X4	88,29	1		88,26
11	PowerBank 12000 mAh	18,48	1		18,48
12	Elegoo Arduino Starter Kit	32	1		32
13	Ordenador		1	4,94	4,94
				TOTAL	**252,51**

- Horas de trabajo invertidas

En la Tabla 14 se muestra el coste total de las horas de trabajo invertidas, del autor y del tutor de este TFG.

Tabla 14. Coste horas de trabajo

	HORAS	COSTE POR HORA	TOTAL (€)
Ingeniero Junior	290	7 €/h	2.030
Director	10	35 €/h	350
		TOTAL	**2.380**

- Presupuesto total del TFG

La Tabla 15 resume el coste total del proyecto de fin de carrera realizado:

Tabla 15. Presupuesto final del TFG

	Precio (Euros)
Valor total del material usado	252,51
Coste total horas de trabajo	2.380
Total	2.632,51
21% IVA	552,83
TOTAL	**3.185,34**

En la tabla anterior se observa que el coste total del trabajo de fin de grado es de 3.185,34 euros. Lo más caro resulta el estudio y la implementación del sistema ROS, por lo que una vez adquiridos estos conocimientos, el coste se vería muy reducido para una posible futura nueva fabricación del robot móvil.

14. CONCLUSIONES Y TRABAJOS FUTUROS

Se realizará una conclusión por cada objetivo planteado al comienzo del proyecto, para luego sacar una conclusión final del trabajo de fin de carrera desarrollado:

- Desarrollo e implementación de un robot móvil.

Se ha conseguido el desarrollado un robot móvil con altas prestaciones, sensores, microprocesador, microcontrolador…

- Iniciarse y obtener habilidades en el uso de la plataforma ROS.

Gracias al proyecto realizado no solo se ha obtenido un nivel intermedio-alto en el uso de la plataforma ROS, sino que también se han obtenido conocimientos en programación Python y del funcionamiento de los nodos de ROS, aplicable a la tecnología IoT.

- Conectar de manera inalámbrica con el robot.

Este objetivo se ha cumplido haciendo uso del programa VNC. Aunque para el proyecto desarrollado proporciona lo necesario, para proyectos de mayor alcance convendría utilizar un método distinto, desde el que se pueda establecer la conexión al robot móvil sin necesidad de estar conectados a la misma red WIFI. Una opción posible sería el uso de la tarjeta GPRS, la cual se acopla a la Raspberry y mediante una tarjeta SIM se podría realizar la comunicación con este sin necesidad de estar conectado a una WIFI.

- Aplicar los conceptos de comunicación entre nodos de ROS en LabVIEW.

Haciendo uso de la tecnología IoT se ha conseguido conectar el sistema ROS con LabVIEW. De esta manera se ha podido establecer un protocolo de comunicación que permita el intercambio de mensajes entre ambos.

- Control de los motores DC de manera inalámbrica.

Aunque se haya logrado el objetivo principal de este apartado, destacar que se podría mejorar el control de la trayectoria haciendo uso de los encoders Hall que incorporan los motores. De esta manera se podría realizar un control de lazo cerrado que hiciese que la velocidad real de los motores correspondiera a la de consigna. Esto conseguiría que los dos motores se movieran a la misma velocidad, dando mayor estabilidad a las trayectorias en línea rectas.

- Control de la trayectoria del robot móvil mediante LabVIEW.

Resultado parecido al del apartado anterior, aunque se haya podido implementar la tarea propuesta, el control de la trayectoria resulta más difícil debido a que cambiar la consigna de velocidad resulta algo más lento que desde el teclado. Aun así, se han puesto a prueba los conocimientos adquiridos en la comunicación entre nodos, creando un nodo publicador mediante LabVIEW que publique el tema de la velocidad de consigna.

- Elaborar una plataforma en la que se trabaje el Arduino y la Raspberry Pi en conjunto.

Gracias a este objetivo se obtiene un sistema que tenga rápida respuesta para los controladores y lectura de sensores, y un procesador con gran capacidad de procesamiento de datos, pudiendo así desarrollar programas de alta envergadura.

- Desarrollo de odometría mediante la funcionalidad de distintos sensores.

Este objetivo no se ha llegado a cumplir debido a que no ha sido posible realizar una odometría con las mediciones de los sensores del IMU, por lo que no han podido usarse estos datos para mejorar la odometría desarrollada con el sensor láser, scan-matching. Aun así, se han conseguido lecturas del IMU desde el Arduino y la Raspberry Pi, entendiendo el uso del bus I2C.

- Analizar el funcionamiento y operatividad de los sensores utilizados.

Esta tarea tenía como finalidad la comprensión de la funcionalidad de los elementos usados en el robot móvil desarrollado, de manera que resulte más fácil trabajar con ellos después.

- Desarrollo de la técnica SLAM en un entorno físico.

Este era el objetivo principal del proyecto, pero para su ejecución era necesaria la comprensión de distintos conceptos adquiridos con los demás objetivos. En cuanto a los resultados del SLAM son mejorables debido a que los mapas generados no son del todo limpios. Esto es debido a que como odometría solo se utiliza la coincidencia de puntos en el escaneo, por lo que a veces falla en la estimación de la posición del robot, creando mapas confusos. Aun así, se obtienen mejores resultados a velocidades bajas del robot móvil. Pero en conclusión este sería uno de los apartados a mejorar en una posible ampliación del proyecto, usando los datos de más sensores para la odometría y configurando los parámetros de actualización de las mediciones a mayor velocidad. Para esto último, el algoritmo del SLAM deberá realizarse en un ordenador de mayor potencia, ya que a la Raspberry Pi puede tener limitaciones en su capacidad de procesamiento de datos, con posibilidad de bloqueos.

- Representar el mapa obtenido por el LiDAR en LabVIEW.

El proceso de este apartado ha sido similar al del control de trayectoria por LabVIEW, pero en este caso el nodo creado se suscribe al tema /scan. De este modo ya se han utilizado las dos funciones posibles de la extensión ROS for LabVIEW Software, la publicación y la suscripción de un tema. Se consideran los resultados muy satisfactorios.

➢ Conclusión final y trabajos futuros

Por lo general el resultado final del proyecto es prometedor, ya que gracias a los conocimientos adquiridos en la plataforma ROS y en las nuevas tecnologías, sumado al robot móvil desarrollado, hace posible muchas formas de desarrollo del proyecto. Teniendo en cuenta para la ampliación del trabajo las posibles alternativas mencionadas en el apartado 4. y las posibles mejoras a realizar en la conclusión de los objetivos, se propone la siguiente actividad para trabajos futuros:

Implementación de un sistema de navegación autónoma que sea capaz de efectuar un recorrido en el mapa de ocupación creado usando un *path planning*. Esto es un algoritmo que traza una trayectoria desde el punto del que se encuentre el robot hasta el punto deseado, evitando lo que el algoritmo entiende como obstáculos, es decir las paredes determinadas por las mediciones del LiDAR. Para el desarrollo de este proyecto la aplicación de Matlab ofrece bastantes facilidades, ya que gracias a distintas toolbox hace posible la realización del *path planning* proporcionando el mapa de ocupación del entorno. Además de esto, haciendo uso de la cámara puede reconocer distintas señales gracias al toolbox de Deep Learning que también incluye Matlab. Se planteó la implementación del *path planning* para este proyecto, pero se descartó debido al alto coste del toolbox Robotic Systems de Matlab.

15. BIBLIOGRAFÍA

[1] Funcionamiento LIDAR. Disponible en:

https://www.nobbot.com/futuro/tecnologia-lidar/

[2] Configuración YDLIDAR. Disponible en:

https://github.com/EAIBOT/ydlidar

http://ydlidar.com/download

[3] Funcionamiento driver L298N. Disponible en:

https://howtomechatronics.com/tutorials/arduino/arduino-dc-motor-control-tutorial-l298n-pwm-h-bridge/

[4] Funcionamiento SLAM. Disponible en:

https://books.google.es/books?id=A5JTp1m9AssC&pg=PA14&lpg=PA14&dq=mathematical+definition+of+slam&source=bl&ots=qIAgWUhTN5&sig=ACfU3U0pTVYeG4XgpnP83zj5wNVqK63sEQ&hl=es&sa=X&ved=2ahUKEwj11qONts_hAhVIPBoKHTs_AuAQ6AEwAnoECAkQAQ#v=onepage&q=mathematical%20definition%20of%20slam&f=false

https://books.google.es/books?id=Xpgi5gSuBxsC&pg=PA872&lpg=PA872&dq=mathematical+definition+of+slam&source=bl&ots=lVllT5n66O&sig=ACfU3U1XCZ3ZI7OdWFcexCf92-svp9M6CA&hl=es&sa=X&ved=2ahUKEwj11qONts_hAhVIPBoKHTs_AuAQ6AEwAXoECAYQAQ#v=onepage&q=mathematical%20definition%20of%20slam&f=false

https://blog.acolyer.org/2015/11/05/simultaneous-localization-and-mapping-part-i-history-of-the-slam-problem/

https://www.gislounge.com/robotic-mapping-simultaneous-localization-and-mapping-slam/

[5] Tutoriales ROS. Disponible en:

https://www.robotigniteacademy.com/en/

https://openwebinars.net/blog/que-es-ros/

http://programacionextrema.es/2017/12/08/ros/

http://answers.ros.org/question/259708/subscribing-and-publishing-geometrytwist-messages-from-turtlesim/

http://wiki.ros.org/roscpp/Overview/Callbacks%20and%20Spinning

http://wiki.ros.org/rosserial_arduino/Tutorials/Blink

https://www.youtube.com/channel/UCt6Lag-vv25fTX3e11mVY1Q

[6] Funcionamiento IMU. Disponible en:

http://ozzmaker.com/guide-to-interfacing-a-gyro-and-accelerometer-with-a-raspberry-pi/

http://www.starlino.com/imu_guide.html

https://maker.pro/arduino/tutorial/how-to-interface-arduino-and-the-mpu-6050-sensor

https://www.instructables.com/id/Interfacing-Digital-Compass-HMC5883L-with-Raspberr/

https://atadiat.com/en/e-ros-imu-and-arduino-how-to-send-to-ros/

https://medium.com/jungletronics/gy-85-a-quick-datasheet-study-79019bb36fbf

http://www.seattlerobotics.org/encoder/200610/Article3/IMU%20Odometry,%20by%20David%20Anderson.htm

https://es.wikipedia.org/wiki/Unidad_de_medici%C3%B3n_inercial

https://www.cnblogs.com/hangxin1940/archive/2013/04/05/3000395.html

[7] Comunicación ROS con LabVIEW. Disponible en:

https://www.clearpathrobotics.com/assets/guides/ros/ROSforLabVIEW.html

https://www.labviewmakerhub.com/doku.php?id=learn:libraries:linx:misc:target-manual-install

http://www.clearpathrobotics.com/assets/guides/ros/ROSforLabVIEW.html#track-a-husky-and-plot-lidar-scans

[8] Arduino. Disponible en:

https://ubunlog.com/instala-arduino-ide-en-tu-ubuntu-para-tus-proyectos-con-arduino/

http://arduino.cl/arduino-uno/

https://www.xataka.com/basics/que-arduino-como-funciona-que-puedes-hacer-uno

[9] Raspberry Pi. Disponible en:

https://www.raspberrypi.org/

https://es.wikipedia.org/wiki/Raspberry_Pi

[10] Funcionamiento motores DC. Disponible en:

https://es.wikipedia.org/wiki/Motor_de_corriente_continua

ANEXO A: INSTALACIÓN DE SOFTWARE

En este apartado se indica el procedimiento de instalación desde cero de todas las herramientas requeridas en el hardware para la implementación de la técnica SLAM. Estas instalaciones se harán en el PC y en la Raspberry Pi. En cada sección se describe qué hacer para que todo usuario pueda repetirlo tras leer este apartado.

- **Raspberry Pi:**

  - Instalación sistema operativo

La Raspberry Pi es un ordenador de pequeño tamaño y menor potencia, por lo que para que funcione se ha de instalar un sistema operativo. Para esto será necesario una tarjeta micro SD vacía donde grabaremos la imagen del sistema operativo seleccionado. En este caso se ha utilizado el sistema operativo de Ubiquity-Xenial-lxde, el cual está basado en Lubuntu 16.04 pero con la ventaja de que tiene ROS Kinetic ya pre-instalado. La imagen está diseñada para trabajar con Raspeberry Pi 3 modelo B y modela B+. Tras descargar el sistema operativo y descomprimirlo se procederá a grabar la imagen en la tarjeta micro SD. Se necesitará el programa Win32DiskImager para grabar la imagen en la tarjeta. Mediante el uso de un adaptador MicroSD a SD se conecta la tarjeta en el ordenador, en el programa Win32 Disk Imager se selecciona la tarjeta insertada y la imagen descomprimida, posteriormente se selecciona “write” para grabar la imagen.

  - Configuración inicial

Una vez grabada la imagen del sistema operativo en la tarjeta, se inserta en la Raspberry Pi. Para la primera configuración de la Raspberry Pi será necesario un monitor con conexión HDMI y un teclado USB. Se conecta la placa al monitor para conectarla a la red WIFI que se vaya a utilizar, y se mira la dirección IP de la placa usando el comando de la Fig. 68.

```
ubuntu@ubiquityrobot:~$ ifconfig
enxb827eb7d7809 Link encap:Ethernet  HWaddr b8:27:eb:7d:78:09
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:7271 errors:0 dropped:0 overruns:0 frame:0
          TX packets:7271 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:1105621 (1.1 MB)  TX bytes:1105621 (1.1 MB)

wlan0     Link encap:Ethernet  HWaddr b8:27:eb:28:2d:5c
          inet addr:192.168.1.27  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::bb9a:3c7a:9079:ffc1/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:464 errors:0 dropped:0 overruns:0 frame:0
          TX packets:489 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:48746 (48.7 KB)  TX bytes:237712 (237.7 KB)
```

Fig. 67. Búsqueda Dirección IP Raspberry

- Control remoto mediante VNC

Una vez obtenida la dirección IP (en este caso 192.168.1.27) se procederá a la instalación de VNC para controlar la Raspberry Pi desde el PC sin necesidad del monitor ni teclado USB. En la Raspberry Pi habrá que instalar VNC Server para Raspberry Pi desde la siguiente página, y seguir las instrucciones que aparecen en ella.

https://www.realvnc.com/es/connect/download/vnc/raspberrypi/

Desde el PC que queramos controlar la placa habrá que descargar VNC Viewer para el sistema operativo que se use (este caso Windows 10) desde la siguiente página.

https://www.realvnc.com/es/connect/download/viewer/

Para conectarse, bastará con introducir la dirección IP de la Raspberry Pi y el nombre de usuario en VNC Viewer.

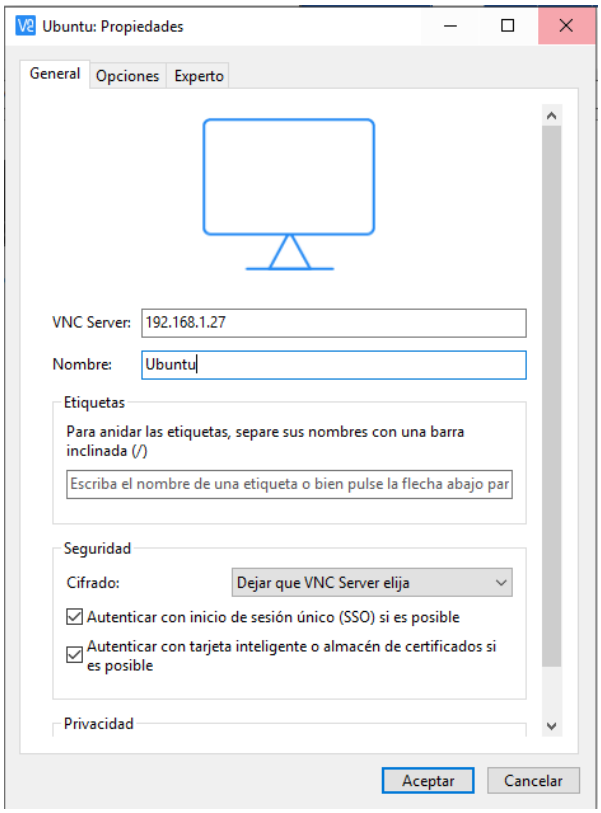

Fig. 68. Acceso a VNC Viewer

- Instalación de ROS

Una vez instalado el sistema operativo y controlando la Raspberry Pi vía remota desde el PC, se procederá a instalar Robotic Operating System (ROS). Aunque en la imagen insertada ya viene preinstalado, se aconseja volver a instalar los paquetes de ROS para su actualización. Para esto habrá que actualizar la source.list (donde están definidos los paquetes a instalar que aceptara el software) introduciendo el siguiente comando:

sudo apt-get update

Para la instalación de ROS kinetic introducir el siguiente comando:

sudo apt-get install ros-kinetic-desktop-full

Saber que los proyectos de ROS serán guardados en la carpeta catkin_ws.

- Instalación de paquete para descargas desde la página Github

Los siguientes paquetes se instalarán desde la página Github, por lo que será necesario instalar el siguiente paquete para habilitar las descargas desde esa página.

sudo apt-get install git

Para copiar e instalar archivos desde GitHub se deberán copiar a la carpeta de trabajo de ROS, la carpeta src dentro de catkin_ws (el directorio de trabajo de ROS).

- Instalación driver YDLIDAR

Se copia el archivo de Ydlidar en el directorio de trabajo de ROS:

cd catkin_ws/src/

git clone https://github.com/EAIBOT/ydlidar.git

Desde el directorio de ydidar, se introducen los siguientes comandos para dar los permisos necesarios para su uso:

roscd ydlidar/startup

*sudo chmod 777 ./**

sudo sh initenv.sh

Para el modelo de LiDAR usado (Ydlidar X4) habrá que cambiar el baudrate que viene como predeterminado a 128000. Para esto será necesario introducir los siguientes comandos y cambiar el valor mencionado.

cd /home/ubuntu/catkin_ws/src/ydlidar/launch
vim x4.launch

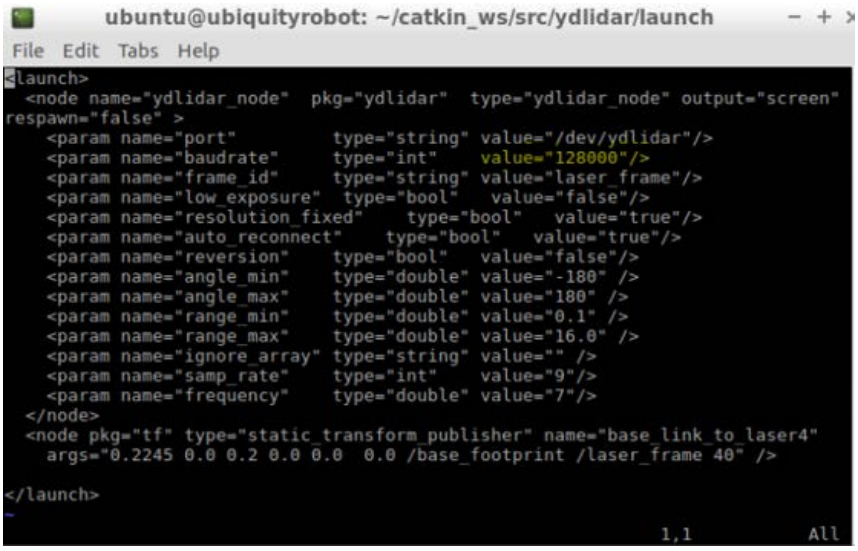

Fig. 69. Selección baudrate

- Instalación paquetes RaspiCam

Copiar el archivo en el directorio de trabajo introduciendo el siguiente comando:

git clone https://github.com/UbiquityRobotics/raspicam_node.git

Debido a que ROS no reconoce unas dependencias de este paquete, habrá que crear el siguiente archivo en la dirección indicada:

/etc/ros/rosdep/sources.list.d/30-ubiquity.list

Y añadir la siguiente línea al archivo.

yaml https://raw.githubusercontent.com/UbiquityRobotics/rosdep/master/raspberry-pi.yaml

Ahora el paquete ya estaría preparado para su instalación.

- Instalación paquetes Arduino

Para instalar el programa Arduino INO se introduce el siguiente comando:

sudo apt-get install arduino arduino-core

Para trabajar con librerías de ROS en el programa del Arduino habrá que instalar rosserial. Este paquete proporciona un protocolo de comunicación de ROS que funciona sobre Arduino, permitiendo al Arduino publicar y suscribir mensajes ROS. Para esto, introducir los siguientes comandos:

sudo apt-get install ros-kinetic-rosserial-arduino
sudo apt-get install ros-kinetic-rosserial

La anterior instalación crea una librería de ROS (ros_lib) la cual ha de ser copiada al entorno de Arduino para poder interactuar con ROS.

Este archivo se copiará en la carpeta <sketchbook> donde el entorno de Linux Arduino guardará los proyectos. Debido a unos errores es necesario eliminar dentro de la carpeta sketchbook/libraries el archivo ros_lib para posteriormente reinstalarlo. Introducir los siguientes comandos:

cd sketchbook/libraries
rm -rf ros_lib
rosrun rosserial_arduino make_libraries.py .

Tras reiniciar el programa de Arduino IDE se observa que se ha añadido la librería de ros_lib.

Fig. 70. Librería de ros_lib

Para dar permiso a la placa Arduino para poder enviar e insertar los programas en la Raspberry Pi, será necesario conocer el nombre asignado al puerto de conexión del Arduino.

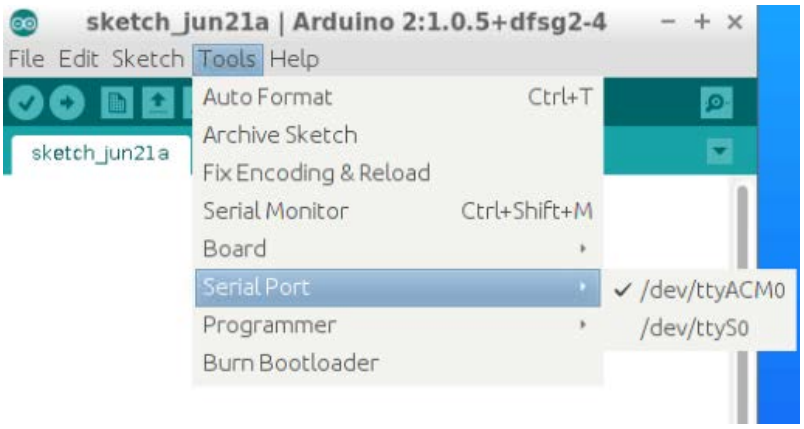

Fig. 71. Selección de Puerto

Una vez conocido el nombre que se le asigna a la conexión del Arduino en el puerto serie se introduce el siguiente comando:

sudo chmod 666 /dev/ttyACM0

- Instalación paquete teleop_twist_keyboard

El paquete se copiará en el directorio de trabajo para su posterior instalación.

git clone https://github.com/ros-teleop/teleop_twist_keyboard.git

- Instalación paquete Hector SLAM

Para copiar el paquete en el directorio de trabajo, introducir el siguiente comando:

git clone https://github.com/tu-darmstadt-ros-pkg/hector_slam.git

- Otras dependencias de los paquetes instalados

Introducir los siguientes comandos para copiar los archivos:

git clone https://github.com/AndreaCensi/csm.git
git clone https://ceres-solver.googlesource.com/ceres-solver
git clone https://github.com/ccny-ros-pkg/scan_tools.git
git clone https://github.com/protocolbuffers/protobuf.git

- Uso de la memoria SWAP

Debido a que la Raspberry Pi no dispone de una memoria RAM lo suficientemente potente como para hacer estas instalaciones de manera rápida se aconseja usar la memoria SWAP.

La memoria SWAP es una partición del disco duro (tarjeta micro SD) que se ha designado como un lugar donde el sistema operativo puede almacenar temporalmente datos que ya no puede contener en la RAM. Este espacio se utilizará cuando ya no haya suficiente espacio en la RAM para mantener los datos de la aplicación en uso, impidiendo así que se congele el sistema.

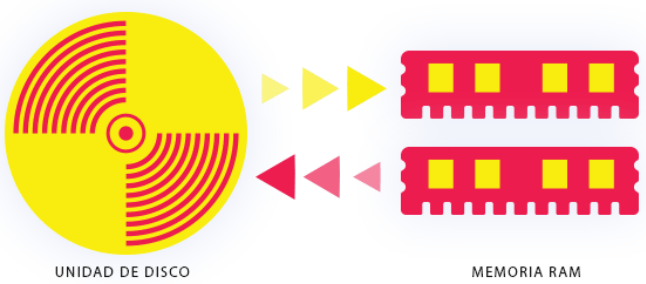

Fig. 72. Memoria SWAP

Para añadir memoria SWAP al sistema se introducen los siguientes comandos (para este proyecto se ha usado una tarjeta micro SD de 36 GB y se ha añadido 1GB de memoria SWAP para la instalación de los paquetes de ROS):

sudo fallocate -l 1G /swapfile %%Se crea la memoria SWAP
sudo chmod 600 /swapfile %%Solo accesible por usuario root
sudo mkswap /swapfile %%Se marca el espacio como memoria de intercambio
sudo swapon /swapfile %%Se habilita la memoria SWAP

La memoria SWAP creada será solo utilizada durante el proceso de instalación de los paquetes, tras reiniciar la Rasberry esta quedará eliminada ya que podría causar problemas en la correcta ejecución de los programas.

- Instalación de los paquetes en el directoria de trabajo de ROS

Tras copiar los archivos necesarios y activar la memoria SWAP, se construye e instala el paquete creado.

Para que la instalación sea más rápida se hace uso de la extensión Ninja, se instala introduciendo el siguiente comando:

sudo apt-get install -y python-wstool python-rosdep ninja-build

Finalmente se procede a instalar los paquetes introduciendo el siguiente comando:

catkin_make_isolated --install --use-ninja

- **PC con LabVIEW instalado:**

Para la implementación de los objetivos en el entorno de LabVIEW, lo primero será descargar una librería que sea capaz de comunicarse con ROS de la Raspberry Pi. En primera instancia se intentó con la librería LINX, pero debido a que esta extensión está más enfocada al control de los pines de la placa, se termina utilizando ROS for LabVIEW Software. Esta extensión resulta más sencilla para suscribir y recibir mensajes de los nodos de ROS. Destacar que esta tarea ha sido desarrollada en la versión de LabVIEW 2014.

Para instalar esta extensión de LINX y ROS for LabVIEW Software habrá que seguir los siguientes pasos:

1. Abrir VI Package Manager y buscar ROS for LabVIEW Software y clicar en instalar.
2. Hacer lo mismo buscando LINX.

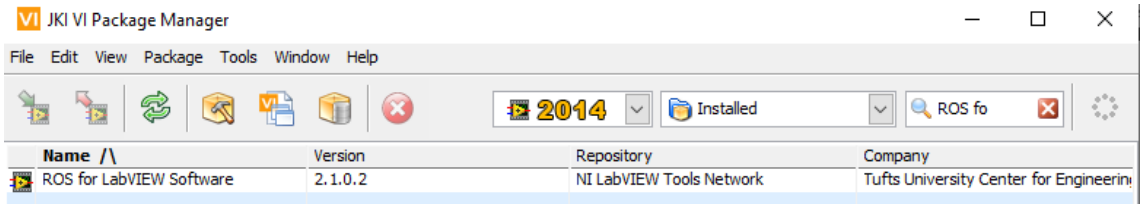

Fig. 73. VI Package Manager

- **Raspberry Pi:**

Una vez instaladas las extensiones habrá que habilitar el SPI y I2C de la Raspberry para poder comunicarse con LabVIEW:

1- Introducir el siguiente comando:

Sudo raspi-config

2- En “Interfacing Options” habilitar SPI e I2C

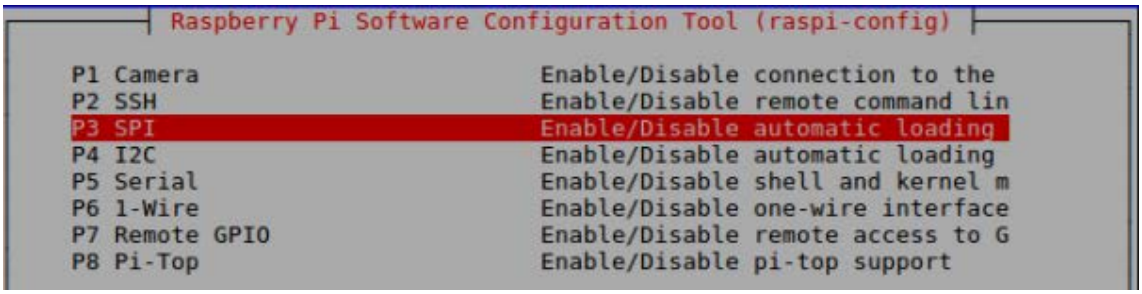

Fig. 74. Configuración Raspberry

3- Reiniciar la Raspberry Pi

Será necesario añadir los directorios del software MakerHub a la Raspberry introduciendo los siguientes comandos:

sudo sh -c 'echo "deb http://feeds.labviewmakerhub.com/debian/ binary/" >> /etc/apt/sources.list'

sudo apt-get update

Para instalar el soporte de Labview, introducir el siguiente comando:

sudo apt-get install -y --force-yes lvrt-schroot

Para solucionar unos errores a la hora de cargar los programas de LabVIEW habrá que dar permiso root al entorno de LabVIEW en la Raspberry:

cd /srv/chroot/labview
sudo schroot --run-session -c lv
cd /usr/lib
*ls liblinx**

En el archivo abierto debería aparecer los siguientes dos archivos:

liblinxdevice_rpi2.so y liblinxdevice_bbb.so

Se crea otro archivo llamado liblinxdevice.so introduciendo el siguiente comando:

ln -s ./liblinxdevice_rpi2.so ./liblinxdevice.so

- **PC con LabVIEW instalado:**

Ahora debería funcionar la conexión entre LabVIEW del PC y la Raspberry Pi. Para comprobarlo en tools seleccionar MakerHub y dentro de LINX, LINX *target configuration*. Introducir la dirección IP de la Raspberry Pi, el nombre de usuario y la contraseña de este para establecer la conexión.

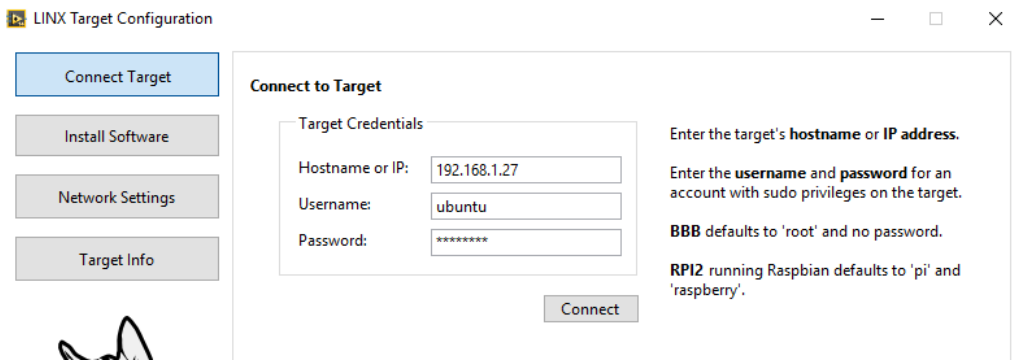

Fig. 75. Conexión LabVIEW-Raspberry

Tras conectarse a la Raspberry habrá que instalar el software de LINX, según los comandos introducidos anteriormente.

- **Raspberry, instalación de librerías I2C para uso del IMU**

Lo primero es instalar la librería quick2wire:

git clone https://github.com/quick2wire/quick2wire-gpio-admin.git

En el directorio donde se haya copiado el archivo, se introducen los siguientes comandos:

make
sudo make install

Agregar al usuario al grupo gpio:

sudo adduser pi gpio

Instalar I2Clibraries:

git clone https://bitbucket.org/thinkbowl/i2clibraries.git

Esta biblioteca contiene las tres interfaces para los sensores del IMU. Es necesario ejecutar i2cdetect para ver las direcciones del dispositivo i2c:

sudo i2cdetect -y 1

```
ubuntu@ubiquityrobot:~$ sudo i2cdetect -y 1
[sudo] password for ubuntu:
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- 0d -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- 53 -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- 68 -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

Fig. 76. Instalación sensor IMU

La dirección 68 corresponde al dispositivo ITG3205, el 53 al ADXL345 y el 0d al dispositivo HMC5883L.

ANEXO B: FICHEROS MODIFICADOS

El algoritmo de SLAM requiere que una gran cantidad de nodos se ejecuten al mismo tiempo, y por lo tanto, trabajar con muchas terminales en las que ejecutar las órdenes necesarias. Para facilitar y simplificar esta labor, ROS dispone de una herramienta llamada launch que permite ejecutar el maestro de la aplicación y al mismo tiempo ejecutar todos los nodos necesarios para el correcto funcionamiento del programa.

En este anexo solo se añadirán los ficheros modificados del sistema SLAM implementado, el resto de ficheros se mantendrá igual a los archivos instalados del paquete HectorSlam.

- **Archivo .launch que ejecuta el nodo del LiDAR (lidar.launch)**

```
<launch>
  <node name="ydlidar_node"  pkg="ydlidar"  type="ydlidar_node" output="screen"
respawn="false" >
    <param name="port"         type="string" value="/dev/ydlidar"/>
    <param name="baudrate"     type="int"    value="128000"/>
    <param name="frame_id"     type="string" value="laser_frame"/>
    <param name="low_exposure"  type="bool"   value="false"/>
    <param name="resolution_fixed"    type="bool"   value="true"/>
    <param name="auto_reconnect"    type="bool"   value="true"/>
    <param name="reversion"    type="bool"   value="false"/>
    <param name="angle_min"    type="double" value="-180" />
    <param name="angle_max"    type="double" value="180" />
    <param name="range_min"    type="double" value="0.1" />
    <param name="range_max"    type="double" value="16.0" />
    <param name="ignore_array" type="string" value="" />
    <param name="samp_rate"    type="int"    value="9"/>
    <param name="frequency"    type="double" value="7"/>
  </node>
  <node pkg="tf" type="static_transform_publisher" name="base_link_to_laser4"
    args="0.2245 0.0 0.2 0.0 0.0  0.0 /base_footprint /laser_frame 40" />
</launch>
```

- **Archivo .launch que ejecuta el mapeo del entorno (mapping.launch)**

```
<launch>
  <arg name="pub_map_scanmatch_transform" default="true"/>
  <arg name="tf_map_scanmatch_transform_frame_name" default="scanmatcher_frame"/>
  <arg name="base_frame" default="base_link"/>
  <arg name="odom_frame" default="odom"/>
  <!--arg name="odom_frame" default="pose2D"/-->
  <!--arg name="odom_frame" default="base_link"/-->
  <arg name="pub_map_odom_transform" default="true"/>
  <arg name="scan_subscriber_queue_size" default="5"/>
  <arg name="scan_topic" default="scan"/>
  <arg name="map_size" default="2048"/>
 <node pkg="hector_mapping" type="hector_mapping" name="hector_mapping"
output="screen">

    <!-- Frame names -->
    <param name="map_frame" value="map" />
    <param name="base_frame" value="$(arg base_frame)" />
    <param name="odom_frame" value="$(arg odom_frame)" />

    <!-- Tf use -->
    <param name="use_tf_scan_transformation" value="true"/>
    <param name="use_tf_pose_start_estimate" value="false"/>
    <param name="pub_map_odom_transform" value="$(arg pub_map_odom_transform)"/>

    <!-- Map size / start point -->
    <param name="map_resolution" value="0.050"/>
    <param name="map_size" value="$(arg map_size)"/>
    <param name="map_start_x" value="0.5"/>
    <param name="map_start_y" value="0.5" />
    <param name="map_multi_res_levels" value="2" />
```

```
<!-- Map update parameters -->
  <param name="update_factor_free" value="0.4"/>
  <param name="update_factor_occupied" value="0.9" />
  <param name="map_update_distance_thresh" value="0.4"/>
  <param name="map_update_angle_thresh" value="0.06" />
  <param name="laser_z_min_value" value = "-1.0" />
  <param name="laser_z_max_value" value = "1.0" />

  <!-- Advertising config -->
  <param name="advertise_map_service" value="true"/>

  <param name="scan_subscriber_queue_size" value="$(arg scan_subscriber_queue_size)"/>
  <param name="scan_topic" value="$(arg scan_topic)"/>
<!-- Debug parameters -->
  <!--
    <param name="output_timing" value="false"/>
    <param name="pub_drawings" value="true"/>
    <param name="pub_debug_output" value="true"/>
  -->
  <param name="tf_map_scanmatch_transform_frame_name" value="$(arg
tf_map_scanmatch_transform_frame_name)" />
 </node>

<!--node pkg ="tf" type="static_transform_publisher" name="map_nav_broadcaster" args="0
0 0 0 0 0 map nav 20"/-->
 <!--node pkg ="tf" type="static_transform_publisher" name
="map_scanmatcher_broadcaster" args ="0 0 0 0 0 0 map base_link 25"/-->
  <node pkg="hector_imu_attitude_to_tf" type="imu_attitude_to_tf_node"
name="imu_attitude_to_tf_node" output="screen">
   <remap from="imu_topic" to="thumper_imu" />
   <param name="base_stabilized_frame" type="string" value="base_stabilized" />
   <param name="base_frame" type="string" value="base_footprint" />
 </node>
</launch>
```

- **Archivo .launch que ejecuta todos los nodos necesarios para el sistema SLAM (All_nodesBuena.launch)**

```
<launch>
 <include file="$(find ydlidar)/launch/lidar.launch" />
 <node pkg="tf" type="static_transform_publisher" name="map_to_odom" args="0.0 0.0 0.0 0 0 0.0 /odom /base_link 40" />
<node pkg="tf" type="static_transform_publisher" name="base_frame_laser" args="0 0 0 0 0 0 /base_link /laser_frame 40" />
<!--<node pkg="rviz" type="rviz" name="rviz" args="-d $(find hector_slam_launch)/rviz_cfg/mapping_demo.rviz"/>-->
 <include file="$(find hector_mapping)/launch/mapping.launch" />
  <node pkg="rviz" type="rviz" name="rviz" args="-d $(find ydlidar)/launch/lidar.rviz" />
  <include file="$(find hector_geotiff)/launch/geotiff_mapper.launch" />
</launch>
```

- **Código del programa de Arduino de control de trayectoria + lectura IMU**

```
#include <ros.h>
#include <geometry_msgs/Twist.h>
#include <std_msgs/String.h>
#include <Wire.h>
#define ENA 3
#define ENB 5
#define IN1 2
#define IN2 4
#define IN3 6
#define IN4 7
int dutycycle;
const int MPU_addr=0x68;
int16_t AcX,AcY,AcZ,Tmp,GyX,GyY,GyZ;
std_msgs::String imu_msg;
ros::Publisher imu("imu", &imu_msg);
```

```
ros::NodeHandle node;
geometry_msgs::Twist msg;
void adelante()
{
  analogWrite(ENA, dutycycle);
  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, LOW);
  analogWrite(ENB, dutycycle);
  digitalWrite(IN3, HIGH);
  digitalWrite(IN4, LOW);
}
void izquierda()
{
  analogWrite(ENA, dutycycle);
  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, LOW);
}
void parar()
{
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, LOW);
}
void derecha()
{
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, LOW);
  analogWrite(ENB, dutycycle);
  digitalWrite(IN3, HIGH);
```

```
  digitalWrite(IN4, LOW);
}
void atras()
{
  analogWrite(ENA, dutycycle);
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, HIGH);
  analogWrite(ENB, dutycycle);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, HIGH);
}
void roverCallBack(const geometry_msgs::Twist& cmd_vel)
{
  if(cmd_vel.linear.x > 0 && cmd_vel.angular.z == 0)
  {
      adelante(); //i
      dutycycle=cmd_vel.linear.x * 255;
  }
  if(cmd_vel.linear.x == 0 && cmd_vel.angular.z < 0)
  {
       derecha(); //l
       dutycycle=cmd_vel.linear.x * 255;
  }
  if(cmd_vel.linear.x == 0 && cmd_vel.angular.z > 0)
     {
       izquierda(); //j
       dutycycle=cmd_vel.linear.x * 255;
     }
  if(cmd_vel.linear.x == 0 && cmd_vel.angular.z == 0)
     {
      parar(); //k
     }
```

```
  if(cmd_vel.linear.x < 0 && cmd_vel.angular.z == 0)
     {
       atras(); //,
       dutycycle=cmd_vel.linear.x * 255;
     }
}
ros::Subscriber <geometry_msgs::Twist> sub("cmd_vel", roverCallBack);
void setup()
{
  Serial.begin(57600);
  node.initNode();
  node.subscribe(sub);
  node.advertise(imu);
  pinMode(ENA, OUTPUT);
  pinMode(ENB, OUTPUT);
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);
  pinMode(IN3, OUTPUT);
  pinMode(IN4, OUTPUT);
  Wire.begin();
  Wire.beginTransmission(MPU_addr);
  Wire.write(0x6B);
  Wire.write(0);
  Wire.endTransmission(true);
}
long publisher_timer;
void loop()
{
  Wire.beginTransmission(MPU_addr);
  Wire.write(0x3B);
  Wire.endTransmission(false);
  Wire.requestFrom(MPU_addr,14,true);
```

```
  AcX=Wire.read()<<8|Wire.read();
  AcY=Wire.read()<<8|Wire.read();
  AcZ=Wire.read()<<8|Wire.read();
  Tmp=Wire.read()<<8|Wire.read();
  GyX=Wire.read()<<8|Wire.read();
  GyY=Wire.read()<<8|Wire.read();
  GyZ=Wire.read()<<8|Wire.read();
  String AX = String(AcX);
  String AY = String(AcY);
  String AZ = String(AcZ);
  String GX = String(GyX);
  String GY = String(GyY);
  String GZ = String(GyZ);
  String tmp = String(Tmp);
  String data = "A" + AX + "B"+ AY + "C" + AZ + "D" + GX + "E" + GY + "F" + GZ + "G" ;
  Serial.println(data);
  int length = data.indexOf("G") +2;
  char data_final[length+1];
  data.toCharArray(data_final, length+1);
  if (millis() > publisher_timer) {
    imu_msg.data = data_final;
    imu.publish(&imu_msg);
    publisher_timer = millis() + 100;
  }
  node.spinOnce();
  delay(1);
}
```