AMPLIACIÓN DE ROBÓTICA

Comparativa de técnicas de SLAM empleando un robot movil

Grado en Ingeniería Electrónica, Mecatrónica y Robótica

Autores:
López Gil, Miguel
Montes Grova, Marco Antonio
Osuna Cañas, Alfonso Carlos

Índice

1. Introducción al proyecto **3**

2. Hardware empleado en el proyecto **4**
2.1. Chasis del vehículo 4
2.2. Motores DC 4
2.2.1. Especificaciones 4
2.2.2. Drivers 5
2.3. Encoders 6
2.4. IMU 6
2.5. Raspberry Pi 7
2.6. Arduino Uno 7
2.7. Cámara 8
2.8. Esquema de comunicación del hardware 9

3. Software empleado en el proyecto **10**
3.1. ROS 10
3.1.1. Interfaz con IMU 10
3.1.2. Interfaz con encoders 10
3.1.3. Odometría 11
3.1.4. Interfaz Raspberry-Pi/Arduino 11
3.1.5. Interfaz Arduino/Motores 12
3.1.6. Computación distribuida con ROS-Multimaster Package 12
3.2. Implementación filtro estadístico 12
3.2.1. Filtro de Kalman Extendido 12
3.2.2. ROS-Robot Localization Package 12
3.3. Arquitectura general del sistema 13

4. Puntos destacables de la implementación del proyecto **14**
4.1. Obtención de los parámetros de la cámara 14
4.2. Inicialización del robot 15

5. Análisis e introducción a las técnicas de SLAM **16**
5.1. Clasificación técnicas de SLAM 17
5.2. Técnicas SLAM implementadas en éste proyecto 18
5.2.1. *RTAB-Map SLAM* 18
5.3. Framework de optimización Octomap 19
5.3.1. *ORB-SLAM 2* 20

6. Comparativa de resultados empleando *rosbags* del robot **22**
6.1. Análisis de resultados con el primer *rosbag* 23
6.1.1. RTAB-Map y local path 23
6.1.2. OctoMap 24
6.1.3. ORB-SLAM2 RGB-D 25
6.2. Análisis de resultados con el segundo *rosbag* 26
6.2.1. RTAB-Map y local path 26
6.2.2. OctoMap 26
6.2.3. ORB-SLAM2 RGB-D 28
6.3. Resultados obtenidos de ORB-SLAM2 Monocular 29

1. Introducción al proyecto

En este proyecto se implementan diferentes técnicas de SLAM visuales, así como diversas fuentes de odometría, sobre un robot móvil de bajo presupuesto. El objetivo final ha sido diseñar e implementar una plataforma de desarollo donde poder estudiar los conceptos vistos durante el curso. Para ello, y teniendo en mente el tiempo y los recursos limitados de los que se disponen, se han optado por soluciones baratas o fabricadas *in-house* y de sencilla implementación para favorecer un rápido prototipado.

Figura 1: El robot, también conocido como *Wheel-E*

El robot, o, como ha sido apodado cariñosamente; *Wheel-E*, consta de una *Raspberry Pi 3B+* como ordenador de a bordo, que hace de interfaz con la cámara, lee las medidas de los encoders y establece un bus i2c con una IMU *mpu-6050*. Un microcontrolador ATmega328 de una placa Arduino, comunicado por puerto serie con la Raspberry, ejecuta un PI y actúa sobre los motores.

Como drivers para los motores se han usado tres circuitos integrados *L293D* (dos puentes H por cada chip), que reciben como *input* una señal generada por el micro mediante *Pulse-Width Modulation*. En la sección de hardware de este mismo documento se detallan en más profundidad los sistemas electrónicos empleados, los buses de comunicaciones y se discute la estructura del robot.
La odometría del robot se obtiene a partir de una estimación de la posición a partir de encoders ópticos, y de la velocidad y aceleración medidas por la IMU. La fusión sensorial se realiza con un filtro de Kalman extendido, implementado en el paquete *robot localization* de ROS. El código desarollado para este proyecto y el uso de software externo es expuesto en la seccion4.

Todo el sistema, exceptuándo el control de bajo nivel del micro, corre en ROS Kinetic, en una configuración de multi-masters que relega los procesos computacionalmente costosos (algoritmos de SLAM y fusión sensorial) a un portátil conectado a la Raspberry a través de una red *Wireless* dedicada.
Para obtener información visual del entorno, se empleará un sensor RGB-D, es decir, la cámara *Kinect v1 for Xbox 360*

Por último, una breve explicación teórica del SLAM y se realizará una comparativa de dos técnicas de SLAM, en concreto *RTAB-Map* y *ORB-SLAM 2*.
Además de ello se probará un software de optimización de mapas denominado *Octomap*, el cual ayudará a obtener el mapa de ocupación 2D y 3D.

2. Hardware empleado en el proyecto

2.1. Chasis del vehículo

Fabricado a partir de un tablón plano de edf cortado en una CNC láser de forma gratuita gracias al Fablab de la Universidad de Sevilla en la facultad de Arquitectura.
El robot ha sido diseñado con una topología inspirada en los vehículos de exploración espacial conocidos como *ROVERS*, utilizando materiales y métodos de construcción asequibles y manteniendo como objetivo una posible sencilla sustitución de cada componente.
Algunos elementos sin funcionalidad estructural, como los encoders o el soporte de la cámara, han sido fabricados mediante impresión 3D.
Todos los modelados pueden encontrarse en la sección hardware de *Git-Hub*.

Figura 2: Modelo CAD del robot

La disposición geométrica es la de un vehículo diferencial, con tres pares de ruedas, cada una con su respectiva pareja de motor y encoder.

2.2. Motores DC

Los seis motores utilizados, a pesar de que deberían comportarse de forma parecida, presentan respuestas muy diferenciadas; para una señal de control idéntica cada uno de ellos muestran puntos de operación distintos y se observan respuestas ante escalón muy variadas. Sin embargo, todos presentan un comportamiento común, una gran zona muerta que complicará mucho la operación del robot a velocidades extremadamente bajas. En concreto, con el convesor digital-analógico del microcontrolador seleccionado (más detalles en la seccion2.arduino), se pierde un 40 % de los bits del rango de salida a causa de este incoveniente.

2.2.1. Especificaciones

- Tensión de trabajo: 3-6 V.
- Corriente máxima: 120 mA.
- Dimesiones: 70x22x18 mm.
- Reductora: 48 revoluciones a una.
- Peso (sin ruedas): 29 gramos.

A continuación, se mostrará la caracteristica estática obtenida para uno de los motores, la caracterización para el resto se ha realizado de forma análoga.

Figura 3: Característica estática de un motor

2.2.2. Drivers

Se han empleado tres microchips L293D que integra cada uno dos puentes H. Estos puentes H nos permiten configurar la dirección de giro con dos señales digitales y la velocidad con otras dos señales analógicas.
Por comodidad y robustez, se ha diseñado en *KiCad* una placa de circuito impreso muy simple que facilita el cableado y permite integrar algún componente pasivo (como diodos fly-back o capacidades de by-pass). Gracias al laboratorio de electrónica este diseño ha sido revelado sobre una placa de una sola cara, obteniendo, como se aprecia en las imágenes, un acabado final bastante bueno.

Figura 4: Diseño en KiCad de la PCB.

Los integrados requieren de una referencia de tensión lógica de 5V y una alimentación para los motores, que será a 6V.
Los 5V se sacarán del Arduino y los motores se alimentarán con una fuente externa de 6V y X A.

2.3. Encoders

Los sensores elegidos para medir la velocidad de cada rueda y estimar la posición y velocidad del robot, son unos sencillos encoders ópticos binarios junto a unos discos perforados. Al pasar la luz por una perforación del disco, que gira a la misma velocidad que la rueda, se manda una señal digital al micro. Contabilizando los flancos de subida o de bajada en cierto periódo de tiempo, y conociendo las dimensiones del disco, se obtiene una estimación de la velocidad.

Figura 5: Encoder óptico y disco empleados

2.4. IMU

Como ya se ha mencionado con anterioridad, se ha escogido una IMU MPU-6050 de seis grados de libertad para obtener datos de las velocidades y aceleraciones del robot. El microchip contine varios subsitemas, de los cuales nos interesa resaltar especialmente:

- Giroscópio (tres ejes), con un rango progamable de ±250, ±500, ±1000 y ±2000 °/s (grados / segundos).
- Acelerómetro (tres ejes), con un rango progamable de ±2g, ±4g, ±8g y ±16g.
- Conversores analógico-digitales de 16 bits para las salidas del giroscópio y del acelerómetro.
- Bus i2c para ejecutar operaciones de lectura y escritura en cualquier registro del chip a una frecuencia máxima de 400kHz.
- Bus de comunicación auxiliar que posibilita introducir un magnetómetro en el futuro sin modificar el conexianado de todo el sistema.

Puesto que la IMU consume muy poca potencia en todo momento, se puede alimentar sin problemas desde cualquier salida de referencia de la Raspberry.

A continuación, se mostrará una imagen de la IMU conectada:

Figura 6: Conexionado entre la RPi y la IMU por I^2C

2.5. Raspberry Pi

Este *single-board computer*, como se le denomina en inglés, alberga muchas posibilidades. En el centro de todas ellas se encuentra un procesado ARM Cortex-A53 de 64-bit a 1.4Ghz capaz de correr varias distribuciones de GNU/Linux.
Con la limitada potencia gráfica de la que dispone, y su único gigabyte de memoria RAM resulta necesario procesar los cálculos más complejos en otro sistema más potente.
El modelo utilizado (3B+) incluye WiFi 802.11n a 150Mbit/s máximo. Sin embargo, es conveniente usar una antena WiFi externa más potente para conseguir más rango de señal.
La Raspberry es alimentada a través de un puerto micro USB a 5V con una corriente máxima de 2.5A aunque su lógica funciona a 3.3V.

2.6. Arduino Uno

Por conveniencia se ha optado por una placa arduino, que además de un microcontrolador ATmega328p, integra un puerto usb con un chipset que facilita su programación y la comunicación serie con el mismo. Incluye también un cristal de cuarzo , que provee una fuente prescisa para la señal de reloj y resulta especialmente útil para implementar el control de bajo nivel.
El micro cuenta con una memoria flash progamable de 32 kilobytes y una eeprom de un kilobyte, que es más que suficiente para el uso que se le va a dar. Dos timers de 8 bits y otro de 16 bits generan las interrumpciones necesarias para controlar el tiempo real e implementar el control. De todas las 23 posibles entradas/salidas digitales del micro, se ocupan cuatro salidas para seleccionar el sentido de la marcha de los motores (dos para todos las ruedas de la izquierda y dos para las de la derecha) y seis canales PWM para controlar la velocidad individual de cada uno.

Figura 7: Microcontrolador empleado para la comunicación a bajo nivel

La potencia eléctrica necesaria para que el micro, y sus periféricos, operen correctamente (5V y menos de 100mA) se obtienen por el puerto USB desde la Raspberry Pi, aprovechando el bus de comunicación serie.

2.7. Cámara

La *Kinect for Xbox 360 V1* integra una cámara RGB y un sensor de profundidad de luz infrarroja estructurada. El rango del sensor de profundidad esta comprendido entre los 0.7 y los 6 metros.

Figura 8: Cámara RGB-D Kinect

El campo de visión del dispositivo es de 57º horizontalmente y 43º en la vertical, aunque cuenta con un soporte motorizado que controla el *pitch* de la camara.
La resolución máxima de la cámara es de 1280x1024 píxeles, pero la más usada es 640x480 píxeles puesto que el frame rate depende fuertemente de la resolución con la que se trabaje en los 30Hz.

El calculo de la profundidad lo realiza del siguiente modo a partir de la matriz proyectada:

Figura 9: Cámara RGB-D Kinect

2.8. Esquema de comunicación del hardware

Debido a la multitud de entradas y salidas digitales y analógicas necesarias, al coste computacional de todo el sistema, y a la variedad de las comunicaciones, se plantea una configuración muy modular entre el ordenador de abordo (la *Raspberry Pi*), los periféricos y la unidad de procesamiento para las partes más pesadas del software.

Figura 10: Esquema de conexionado de todos los componentes

3. Software empleado en el proyecto

3.1. ROS

ROS (Robotic Operating System) consite en un framework para la robótica sobre el que desarrolladores de cualquier ambito pueden aportar soluciones modulares a los sistemas de forma general. La modularidad de ROS hace que desarrollar un nuevo sistema o prototipo pueda ser una tarea tan sencilla como interconectar módulos de otros desarrolladores con los propios.

El sistema operativo empleado tanto en el ordenador de abordo del robot,la Raspberry Pi, cómo en en el ordenador principal es Ubuntu 16.04, de tal modo que la distro de ROS en la que ha sido implementado el proyecto es ROS Kinetic Kame[1].

Figura 11: Version de ROS empleada

3.1.1. Interfaz con IMU

Como ya se ha visto en la seccion 2.4, al robot se le ha dotado con una IMU de 9 grados de libertad a fin de obtener medidas de aceleraciones.
La conexión se hace a través de una linea por protocolo I^2C, por lo que ha sido necesario el desarrollo de 2 'wrapers' en Python de funciones de bajo nivel; uno para el acceso a las funciones de comunicación I^2C a nivel de registros hardware y otro a nivel de funciones elementales para el manejo de los datos de la IMU.
Sobre ambos 'wrapers' se ha escrito el nodo de ROS encargado de publicar los datos en 2 'topics', *imu1_rawImu* y *imu1_scaImu*, con los datos tal y como llegan de la IMU en el primero y reescalados en el segundo. El tipo de mensaje en el que se publican los datos es de tipo *sensor_msgs/Imu*.

Como se trata de un sensor de bajo nivel que aporta información a los estratos más bajos del control, para poder disponer depués de mayor ancho de banda, se ha elegido una frecuencia de muestreo del sensor de 50 $[Hz]$; sufucientemente alta para no comprometer la estabildiad del sistema pero tambien lo sufucientemente baja como para no saturar el ordenador de abordo (teniendo en cuenta que este también deberá mantener un flujo constante de imagenes capturadas por la cámara, con el coste que eso conlleva).

3.1.2. Interfaz con encoders

Otro sensor añadido son los encoders mencionados en 2.3. Para su programación se ha desarrollado un 'wraper' en C++ de la librería para Raspberr-Pi 3B *pigpio* [1]. Como los encoders solo pueden ofrecer información de la magnitud de la velocidad y no del sentido es necesario hacer una conversión a nivel de sofware para poder tener esa información.
Esto se hace suscribiendo el nodo creado al 'topic' *cmd_vel*, de forma que de ahí se obtiene la referencia del sentido en el que se mueve el robot. Esta referencia se usa para reajustar el signo de las velocidades leidas directamente de los 6 encoders y publicarlas en el topic *encoders* con el tipo de mensaje *std_msgs/Float64MultiArray*. La medida de los encoders, al tratarse de medidas de baja calidad con valores dispersos, han sido necesaria filtrarlas también.

[1] https://wiki.ros.org/kinetic

En cuanto a la frecuencia de muestreo, se ha seguido la misma filosofia que con la programación del nodo de la IMU y se ha decido muestear a 50 $[Hz]$.

3.1.3. Odometría

A partir del calculo de la velocidad de las ruedas a partir de los encoders se desarrolló un nodo en ROS en el que se calcula la odometría del robot.
Para ello se han hecho una serie de suposiciones y simplificaciones del modelo del robot que después, en base a los resultados obtenidos, se han demostrado bastante acertadas.

- Las ruedas tienen un modelo de rodadura perfecto y no deslizan en el suelo. Esto no es cierto en realidad y supone una fuente de errores importantes por la deriva que acumulan las posiciones estimadas, errores que son corregidos mediante una fusión sensorial.
- El modelo del robot consiste en un modelo diferencial de 2 ruedas. En base a esto a cada rueda 'virtual' se le asigna el promedio de la velocidad de las 3 ruedas de cada bancada.

Las ecuaciones que definen el modelo del robot son las mostradas a continuación:

Figura 12: Representación posición y orientación modelo diferencial

$$v = \frac{v_{dcha} + v_{izqd}}{2} \qquad \theta = \frac{v_{dcha} - v_{izqd}}{b}$$

dónde v es la velocidad lineal, w la velocidad angular y b la distancia entre las ruedas.
Y se definirá la posición y la orientación del robot cómo:

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} cos(\phi) & 0 \\ sin(\phi) & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} v \\ \theta \end{pmatrix} \tag{1}$$

Este nodo se suscribe al desarrollado en 3.1.2 en el 'topic' *encoders*, y cada vez que llega un mensaje nuevo con los datos de los encoders calcula el avance del robot en base al modelo propuesto. La publicación de los datos se hace en el 'topic' *odom* a un ratio de 25 $[Hz]$ con un mensaje del tipo *nav_msgs/Odometry*.

3.1.4. Interfaz Raspberry-Pi/Arduino

La comunicación por puerto serie se implementa haciendo uso de la librería *CmdMessenger*. MIGUEL LAS PUTAS REFERENCIAS ACUÉRDATE., que permite definir comandos (con la posibilidad de asociarles argumentos y callbacks) de forma eficiente y sencilla. Por la parte de la Raspberry, se usan las librerías CmdMessenger (escrita en c) o su alternativa PyCmdMessenger (en Python) en el nodo serial_com.py para transmitir por el puerto serie.
Los siguientes comandos son los más importantes
Cmd_off; Al recibir este comando el micro pasa a no hacer nada, comportándose como un apagado de emergencia.

De esta forma es sencillo publicar un tópico de 'comandos del sistema', como apagar o funciones de debugeo, y otros tópicos para los datos que hayan de retransmitirse. Cuando llegan datos desde el arduino, se ejecuta un callback (de la librería externa, no de ROS) que publica dicha información el tópico correspondiente. Si es el caso contrario, y se busca enviar datos al arduino, el nodo se subscribirá al tópico con ese dato y en el callback asociado (ahora sí de ROS) se envía al arduino.

3.1.5. Interfaz Arduino/Motores

Same old same old

3.1.6. Computación distribuida con ROS-Multimaster Package

Las técnicas de SLAM a testear en el robot desarrollado son computacionalmente costosas. Todo este coste computacional es imposible asumirlo desde una Raspberry-Pi, por lo que surge la necesidad de hacer el procesado en un servidor remoto al robot en tiempo real con las dificultades que eso conlleva, entre ellas y la mas importante, la sincronización de datos.
Bajo estas condiciones de diseño y restricciones aparece una solución de la comunidad: ROS multimaster_fkie. La premisa que lo rige es simple, dado un sistema mutirobot cada elemento mantiene la ejecución de 2 nodos locales, *master_ discovery* y *master_ sync*; el primero es el encargado de mantener una lista del resto de *roscores* disponibles en la red y de publicar el suyo propio como activo. El otro es el encargado de suscribir su *core* los nodos de otros *cores*.
Des esta forma todos tienen acceso de una forma hasta cierto punto 'real-time' a los recursos del resto.

Figura 13: Esquema del funncionamiento básico del paquete Multimaster

En este proyecto su integración se ha llevado a cabo lanzando en la Raspberry-Pi y en otro PC los nodos correspondientes siendo la Raspberry-Pi la suministradora de información y el PC el encargado de procesarla.

3.2. Implementación filtro estadístico

Introducción de porqué es interesante y eso, lo de vender la moto as usual

3.2.1. Filtro de Kalman Extendido

La información que el sistema obtiene de los sensores está sujeta a ruido e incertidumbres de la medida. A fin de minimizar los errores y mejorar la estimación de la posicion y velocidad del robot se ha considerado que era de especial interes y utilidad la aplicación de un filtro recursivo Bayesiano como estimador de estados, en concreto el filtro de Kalman extendido.
Este filtro es una extensión del filtro de Kalman para sistemas no lineales.

3.2.2. ROS-Robot Localization Package

3.3. Arquitectura general del sistema

A continuación, se mostrará la comunicación entre nodos tanto dentro de la RPi cómo en el momento de lanzar las técnicas de SLAM. En primer lugar se mostrará la estructura en el momento en el que se grabó el *bag* y, cómo sería la estructura si se realizará SLAM en tiempo real:

Figura 14: Captura de los nodos corriendo dentro de la Rapberry Pi 3B+

Figura 15: Captura de los nodos ejecutandose cuando se corre SLAM sobre los bags

Figura 16: Captura de los nodos cuando se realiza SLAM en tiempo real

4. Puntos destacables de la implementación del proyecto

4.1. Obtención de los parámetros de la cámara

Destacar que, siempre es necesario conocer los parámetros intrinsecos de la cámara además de la matriz de rotación.
Para obtener la matriz de parámetros de nuestra cámara, se ha hecho uso del paquete de ROS, *Camera_ calibration*.
Este paquete nos proporcionará la herramienta mostrada a continuación, que gracias a un tablero de ajedrez nos dará los parámetros intrínsecos de nuestra cámara:

Las matrices serán dadas en pormato *.ost* y *.yaml* que, despues se le pasará cuando se quiera emplear la cámara para actualizar el topic camera_info,cuyo tipo de mensaje será *sensor_ msgs/CameraInfo*.

Figura 17: GUI proporcionada por el paquete

4.2. Inicialización del robot

Para lanzar todos los nodos de ROS en el ordenador de abordo del robot y en el ordenador principal se han creado los siguientes ficheros *roslaunch*:

Figura 18: Nodos lanzados en el robot

También se lanzarán los drivers de la cámara, aunque no aparezcan en la imagen. Los drivers son proporcionados por el paquete de ROS *Freenect*. Para poder tener control del motor integrado en la cámara, se ha empleado el paquete *Kinect-aux*.

5. Análisis e introducción a las técnicas de SLAM

El SLAM*(Simultaneous Localization and Mapping)* es una tecnica cuyo objetivo es contruir un mapa de un entorno desconocido con un robot movil y conocer la posición de dicho robot dentro del mapa.
El primer planteamiento de la resolución de estos dos problemas cómo son la localización de un robot y la creación de mapas de manera conjunta surgió en 1986 en un congreso del IEEE en San Francisco.
Durante aquellos años, se habia comenzado a introducir métodos probabilistico en la inteligencia artificial y en la robotica. En dicha conferencia, se llegó a la conclusión de que la generación de mapas consistentes y localización de manera probabilistica era un problema fundamental.

Tras ello, se establecieron las bases estadisticas para describir la relaciones entre *landmarks*, objetos de referencia y para manipular la incertidumbre geométrica.
El planteamiento básico del problema de SLAM es el mostrado a continuación:

Figura 19: problema básico de SLAM

Dado un robot que se mueve por un entorno desconocido tomando observaciones relativas de un número desconocido de landmarks empleando un sensor posicionado en el robot. Para el instante k, se definen las siguientes variables:

- $\mathbf{x}_k$: Vector de estados que describe la posición y orientación del vehículo.
- $\mathbf{u}_k$: Vector de control aplicado el instante anterior sobre el vehículo para que se encuentre en el estado x_k en el instante k.
- $\mathbf{m}_i$:Vector que describe posición del i-ésimo landmark cuya verdadera ubicación se supone invariable con el tiempo.
- $\mathbf{z}_{ik}$: Observación tomada desde el vehiculo de la posición del i-ésimo landmark en el instante k.

Por tanto, se deberá computar la siguiente distribución probabilistica en cada instante k:

$$P\Big(\mathbf{x}_k, m|\mathbf{Z}_{0:k}, \mathbf{U}_{0:k}, x_0\Big) \tag{2}$$

La solución se obtendrá de manera recursiva empleando el teorema de Bayes en un proceso en dos pasos: *Time-Update* y *Measurement-Update*. Para ello, será necesario conocer el modelo de observación y el de movimiento.
A continuación se mostrará el modelo de observación y movimiento respectivamente:

$$P\Big(\mathbf{z}_k|\mathbf{x}_k, m\Big) \qquad\qquad P\Big(\mathbf{x}_k|\mathbf{x}_{k-1}, \mathbf{u}_k\Big)$$

Una vez planteado de manera superficial el problema a resolver, el modo de abordarlo se ha desarrollado en muchas direcciones durante los últimos 20 años.
Algunos de estos métodos se muestran a continuación y se hablará de ellos, para posteriormente indagar en dos técnicas concretas.

5.1. Clasificación técnicas de SLAM

La forma en que se toma información de las imagenes que recibe y de su entorno y cómo la procesa permite distinguir las siguientes tipos de sistemas SLAM visual:

- *Denso vs Disperso*
  En función de la cantidad de datos tomada de cada imagen, se realizará ésta clasificación. Las técnicas de SLAM disperso sólo emplean una pequeña región de pixeles, cómo pueden ser puntos característicos. Sin embargo, las técnicas densas emplean la mayoría o todos los píxeles de cada frame que reciben.
  Debido a que la cantidad y tipo de información que toman son diferentes, el tipo de mapa que se obtendrá también lo es. Con las técnicas de SLAM disperso se obtendrá una nube de puntos que será una representación de los puntos característicos de las escena y se usará sobre todo para hacer un tracking de la pose de la cámara. Por otro lado, un mapa denso tendrá muchos más detalles y, requerirá de una mucho más elevada carga computacional.

- *Directo vs Indirecto*
  En función de cómo las técnicas de SLAM empleen y traten la información que reciben, se podrá clasificar en SLAM directo o indirecto.
  El SLAM Indirecto, se basa en extraer primero las features de la imagen, puntos característicos, para posteriormente emplearlos para localizarse y contruir el mapa. Para extraer estos puntos caracteristicos existen muchos descriptores: ORB, SIFT, FAST, etc.
  En contraste, el SLAM Directo,emplean directamente la intensidad de los pixeles, de tal modo que se obtengan features intermedios. Estos metodos tratan de obtener la profundidad y estructura del entorno a partir de una optimización del mapa. Los procesos de extracción de puntos característicos son mucho más pesados computacionalmente si se trabaja al mismo rate que un slam indirecto.
  Por último, destacar, que los metodos indirectos de SLAM son más tolerantes a los cambios de iluminación del entorno.

A continuación, se monstarán una comparativa de una serie de técnicas de SLAM:

Figura 20: Comparativa de técnicas SLAM

5.2. Técnicas SLAM implementadas en éste proyecto

Destacar que, mientras la primera técnica de SLAM es una técnica determinista, la segunda será probabilistica.

5.2.1. *RTAB-Map SLAM*

RTAB-Map *(Real-Time Appearance-Based Mapping)* es una técnica de Graph-SLAM[2] basada en la detección de bucles cerrados incrementales. Es totalmente funcional con sensores RGB-D, Stereo y LIDAR.
El detector de bucles cerrados se basará en la comparativa de cuán semenjantes son la imagenes en una localización y la previa. Cuándo una hipótesis de bucle cerrado es aceptada, se añade una nueva restricción al *graph* del mapa y, tras ello, el optimizador minimiza el error del mapa.

Figura 21: Esquema del Back-End y Front-End de RTAB-Map

Este algoritmo plantea una estrategia de particionado de memoria que pretende asemejarse al funcionamiento de la memoria humana, dónde ésta se estructura en:
Memoria de trabajo del robot *(Working Memory)*, la memoria a largo plazo *(Long Term Memory)*, la memoria a corto plazo *(Short-term memory)* y la memoria sensorial *(Sensory Memory)*.
De ese modo, se mantendrán en la memoria de trabajo del robot aquellas localizaciones que se han visitado recientemente y con más frecuencia, mientras que el resto pasarán a la memoria de largo plazo.

Figura 22: Estructura de memoria de la técnica RTAB-Map

[2]http://robots.stanford.edu/papers/thrun.graphslam.pdf

Se partirá de la premisa de que aquellas localizaciones que son visitadas de forma más frecuente son más propensas a crear bucles cerrados. Por ello, el número de veces que una localización sea visitada será empleado como peso, de esta forma serán transferidas desde la memoria de trabajo a la memoria de largo plazo aquellas observaciones que tengan mayor peso.
La memoria a corto plazo, *STM*, tiene como misión buscar las similitudes que existan entre dos imágenes consecutivas, mientras que la memoria de trabajo, *WM*, es la encargada de detectar los bucles cerrados entre las localizaciones en el espacio. El número de localizaciones almacenadas en la memoria del trabajo del robot es limitado. El tamaño de la memoria a corto plazo, *STM*, está basado en la velocidad del robot y en la frecuencia de adquisición de las localizaciones.

5.3. Framework de optimización Octomap

El paquete de ROS empleado para emplear la ténica RTAB-Map,*rtabmap-ros*, posee la posibilidad de extraer también el mapa de ocupación 2D y 3D gracias al Framework *Octomap*[3].
Este framework es una librería de C++ que permite la obtención de mapas probabilisticos basándose en una estructura de representación jerarquica de datos para subdividir el espacio 3D, esta estructura es denominada *Octotrees*.

Figura 23: Optimización realizará empleando los *Octotrees*

Los *Octotrees* se emplean para almacenar propiedades booleanas del entorno, en el caso de la robotica estas propiedas booleanas es la ocupación de dicha celda, de la cual es posible definir su tamaño minimo en función de la resolución de mapa deseada.

Figura 24: Funcionamiento del *octotree* y su estructura en arbol

[3]https://octomap.github.io/

5.3.1. *ORB-SLAM 2*

ORB-SLAM2 es una técnica de SLAM en tiempo real para cámaras Monocular, Stereo y RGB-D que se engloba dentro de las técnicas de *Sparse-Slam*. Se computa la trayectoria y hace una reconstrucción dispersa 3D del entorno. Al igual que todas las técnicas de SLAM, se basa en la detección de bucles cerrados y se relocaliza en tiempo real.
Se basa en la detección de *keyframes* que emplea para hacer un tracking de los mismos y a partir de ello crear el mapa local que, posteriormente, se optimizará junto al mapa global.
Unos de los puntos destacables de ésta técnica de SLAM son los siguientes:

- Emplea *grafo de covisibilidad*. Tanto el seguimiento como el mapping se focalizan en el área covisible, independientemente del tamaño del mapa completo, consiguiendo así explorar entornos amplios sin aumentar el tiempo y la carga de computación.
- La estrategia para detectar los bucles cerrados de visión en tiempo real se basa en la optimización de un de grafo denominado *Essencial Graph*, lo cuál se desarrollará mas adelante.

A continuación, se tratará un poco el funcionamiento interno del algoritmo, el cuál presenta el siguiente esquema:

(a) System Threads and Modules. (b) Input pre-processing

Figura 25: Estructura interna de ORB-SLAM2

Se observa cómo existen 6 grandes módulos, los cuales de lanzarán en 3 hilos paralelos.
En primer lugar, se tendrá el preprocesamiento de la imagen, de la cuál se obtendrán ORB features *(Oriented Fast and Rotated BRIEF)*.
Los hilos desempeñarán las siguientes funciones:

- El *tracking* se encargará de localizar la cámara en cada frame buscando marches entre las features del mapa local y minimizando la reprojección del error aplicando un *Bundle Adjustment*[4] sólo de movimiento.
- El *Local Mapping* se encargará de gestionar y optimizar el mapa local aplicando un *BA* local.
- El hilo de *Loop Closing* se encargará de detectar bucles cerrados grandes y corregir la deriva acumulada realizando una optimización del grafo del *pose* obtenido. Este hilo, lanza 4 hilos internos que se encagarán de realizar un *Bundle Adjustement* completo tras la optimización del grafo del *pose*, para hayar la solución más optima.

En caso de que el sistema pierda el tracking, lleva integrado un modulo de *Plane recognition* basado en un vocabulario de palabras que básicamente es una especie de preentrenamiento de la técnica, para

[4]https://homes.cs.washington.edu/ sagarwal/bal.pdf

poder obtener y asociar keyframes más fácilmente.

En la imagen a continuación, puede observarse la técnica corriendo en un ordenador. Destacar que ésta técnica de SLAM presenta una carga computacional bastante elevada.

Figura 26: ORB-SLAM2 en funcionamiento

Se observa en la imagen superior izquierda las features que está obteniendo en ese frame, a partir de las cuales se ha creado en mapa local y, por extensión el global. En la imagen inferior izquierda, se puede observar el mapa creado, el cuál se analizará más tarde.
Por último, a la derecha se tienen las terminales a partir de las cuales se ha lanzado el SLAM y se realiza la comunicación con el robot.

6. Comparativa de resultados empleando *rosbags* del robot

Para realizar un análisis y una comparativa entre éstas dos técnicas de SLAM, se ha optado por generar un set de datos del robot y correrlo con ambas técnicas, de tal modo que ambas téngan los mismos datos de entrada y se pueda analizar de mejor modo la salida de la arquitectura.
La implementación de todo el sistema, también es posible correrlo en tiempo real con el robot empleando una o la otra técnica, pero de ese modo sería más complejo realizar una comparativa entre ellas.
Estos datasets, que han sido grabados en forma de *rosbag*, se grabaron desde el ordenador en el que corre el SLAM, de todos los topicos que se recibiaN del ordenador de abordo del robot a través de la comunicación *Multi-master* implementada.
A continuación, se muestra una imagen de los topicos contenidos en ambos datasets:

```
/camera/depth_registered/camera_info
/camera/depth_registered/hw_registered/image_rect_raw
/camera/depth_registered/image_raw
/camera/depth_registered/points
/camera/rgb/camera_info
/camera/rgb/image_raw
/camera/rgb/image_rect_color
/clock
/cmd_vel
/encoders
/imu1_scaImu
/odom
/rosout
/rosout_agg
/tf
/tf_static
```

Figura 27: topicos contenidos en el bag

En ambos datasets el robot recorrerá un camino en torno al salón de la casa de uno de los integrantes del grupo. En el primer dataset, el estado del salón será el comun, de modo que haya una parte del mapa que no contendrán muchas features. Sin embargo, en el segundo se colocaron objetos en zonas dónde la cantidad de features y los contrastes del mapa eran reducidos para así comparar el resultado inicial con uno en un entorno experimental, dónde se ha forzado que el entorno esté lleno de features.

Se realizarán 3 pruebas, las cuales se encuentras recogidas en vídeos existentes en la carpeta del proyecto. En primer lugar se mostrará el mapa creado por *RTAB-Map* y la estimación del *path* del robot. Tras ello, se mostrará el mapa de ocupación generado por *OctoMap* tanto en 2D cómo en 3D y, por último, se mostrará el resultado obtenido con *ORB-SLAM2 RGB-D*.

6.1. Análisis de resultados con el primer *rosbag*

Este dataset será el que contiene el salón en un estado común, sin alterar la cantidad de features existentes en el entorno.
En él, se recorrerá un camino de, aproximadamente, $1m^2$ en torno al salón.

6.1.1. RTAB-Map y local path

En primer lugar, se mostrará el mapa denso obtenido gracias a RTABMAP. Se obsrrva cómo, el loop closure del mapa ha funcionado correctamente de tal modo que la estructura final del mapa será el rectangulo que confirma el salón.

Figura 28: Mapa creado antes y después del *Loop Closure*

El mapa obtenido, será un mapa denso el cual posee toda la información posible del entorno, cómo se muestra a continuación:

Figura 29: Mapa obtenido con RTAB-Map

Gracias a la realización de dicho mapa, se ha optado por medir la tela grande de la pared y el cuadro sobre el sillón naranja tanto en el mapa cómo la distancia real, para comprobar el error que posee ésta técnica.

	Ancho en el mapa creado	Ancho real	Error de RTAB-Map
Tela de la pared	1.75 m	2 m	0.15m
Cuadro sobre sillón	X cm	X cm	

En este caso, se obsrrva que ...

6.1.2. OctoMap

Gracias a *Octomap* se ha obtenido el mapa de ocupación probabilistico del entorno. Se mostrará, al igual que antes, el mapa antes y despues del loop closure, ya que es local destacable de ésta técnica de SLAM.

Figura 30: Mapa de ocupación antes y despues del *Loop Closure*

Además, *Octomap* nos proporciona el mapa de ocupación 3D, cómo el mostrado a continuación:

Figura 31: Mapa obtenido con RTAB-Map y empleando el framework *Octomap*

6.1.3. ORB-SLAM2 RGB-D

En el caso del ORB-SLAM2, con éste dataset los resultados han sido peores, debido a que en cierto momento, los cambios de iluminosidad procedentes del balcón de la casa, hacen que se se pierda dentro del mapa creado. A continuación se muestra el frame anterior a la pérdida de la localización dentro del mapa. Los colores representan la altura en el eje Z.

Figura 32: Momento anterior a la pérdida del robot en el primer dataset

Sin embargo, se observa cómo, cuándo vuelve a ver el inicio del mapa, dónde tomó un número elevado de features, se relocalizará, cómo se muestra a continuación:

Figura 33: Relocalización del robot en el primer dataset

Debido a que el mapa no se ha llegado a completar se ha obtado por no realizar una comparativa de las medidas en éste caso.

6.2. Análisis de resultados con el segundo *rosbag*

En éste segundo dataset, el escenario será más parecido a un escenario experimental debido a que se han colocado objetos por el salón de tal modo que en todo el mapa se obtengan una cantidad elevada de features.

6.2.1. RTAB-Map y local path

Al igual que anteriormente, se destacará el loop closure detectado por el mapa. Se observa cómo, antes de dicho momento, las imagenes de la tela de la pared aparecen superpuestas, sin embargo, posteriormente, se observa cómo ha detectado que ya conocia ese setpoint y cerró el bucle:

Figura 34: Mapa creado antes y después del *Loop Closure* con el segundo dataset

El mapa obtenido, será un mapa denso el cual posee toda la información posible del entorno, cómo se muestra a continuación:

Figura 35: Mapa obtenido con RTAB-Map empleando el segundo dataset

6.2.2. OctoMap

Gracias a *Octomap* se ha obtenido el mapa de ocupación probabilistico del entorno. Se mostrará, al igual que antes, el mapa antes y despues del loop closure, ya que es local destacable de ésta técnica de

SLAM.

Figura 36: Mapa de ocupación antes y despues del *Loop Closure*

Además, *Octomap* nos proporciona el mapa de ocupación 3D, cómo el mostrado a continuación:

Figura 37: Mapa obtenido con RTAB-Map y empleando el framework *Octomap*

6.2.3. ORB-SLAM2 RGB-D

En éste segundo set de datos, se evitó el cambio de iluminación en el punto dónde antes se perdio, de tal modo que en éste caso no pase y, efectivamente, cómo se mostrará a continuación, en ese punto no se perdió:

Figura 38: Modificación del entorno para evitar la perdida de ORB

Por tanto, en éste caso sí se ha logrado obtener un mapa disperso del entorno:

Figura 39: Mapa del entorno empleando ORB-SLAM2

6.3. Resultados obtenidos de ORB-SLAM2 Monocular

Debido a la compleja inicialización que conlleva emplear ésta técnica monocular, no ha sido posible evaluar su funcionamiento con los bags generados, ya que no se logró que se inicializase. Sin embargo, para comprobar su funcionamiento, se ha optado por mover la cámara *Hand-handle*, de tal modo que se asegure la inicialización del SLAM.
El gran punto a favor que posee el empleo de SLAM monocular es que no es sensible a la luz, de tal modo que funciona correctamente en exteriores.
Por ello, la prueba realiza es en el mismo salón, pero saliendo al balcón y grabando al exterior.

Referencias

[1] joan2937, "C,C++ library for control of the raspberry GPIO." https://github.com/joan2937/pigpio, November 2018.

[2] R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: an open-source SLAM system for monocular, stereo and RGB-D cameras," *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, 2017.

[3] IntRoLab, "RTAB-Map(real-time appearance-based mapping): RGB-D, stereo and lidar graph-based SLAM." https://github.com/introlab/rtabmap, 2014.

[4] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: An efficient probabilistic 3D mapping framework based on octrees," *Autonomous Robots*, 2013. Software available at http://octomap.github.com.

[5] T. Moore and D. Stouch, "A generalized extended kalman filter implementation for the robot operating system," in *Proceedings of the 13th International Conference on Intelligent Autonomous Systems (IAS-13)*, Springer, July 2014.

[6] M. Harms, "Python interface for cmdmessenger arduino serial communication library." https://github.com/harmsm/PyCmdMessenger, February 2017.

[7] T. Elenbaas, "Command messenger communication library for arduino & .NET." https://github.com/thijse/Arduino-CmdMessenger, September 2017.

[8] "Gammon software solutions." http://www.gammon.com.au.

[9] Atmel, *Atmel ATmega640 Datasheet*, February 2014.

[10] Invensense, *Invensense MPU-6050 Datasheet Rev 3.4*, August 2013.

[11] Invensense, *Invensense MPU-6050 Register Map Rev 4.2*, August 2013.

[12] H. Durrant-Whyte and T. Bailey, "Simultaneous Localization and Mapping (SLAM) : Part I The Essential Algorithms." https://people.eecs.berkeley.edu/~pabbeel/cs287-fa09/readings/Durrant-Whyte_Bailey_SLAM-tutorial-I.pdf.

[13] T. Bailey and H. Durrant-Whyte, "Simultaneous Localization and Mapping (SLAM):Part II State of the Art." https://pdfs.semanticscholar.org/27d4/6db7ed4e96944080052b761c62102f26b23f.pdf.