Maestría en Ingeniería Electrónica

Dotación de la Capacidad de Localización y Mapeo Simultáneo *Indoor* a un Robot Móvil Terrestre Usando Sensores IMU y LIDAR Sobre el *Framework* ROS

José Daniel Quinche Guerra

Bogotá, D.C., diciembre de 2018

ESCUELA COLOMBIANA DE INGENIERÍA
JULIO GARAVITO

Dotación de la Capacidad de Localización y Mapeo Simultáneo *Indoor* a un Robot Móvil Terrestre Usando Sensores IMU y LIDAR Sobre el *Framework* ROS

Trabajo de grado para optar al título de magíster en Ingeniería Electrónica con énfasis en Control y Automatización Industrial.

Ing. Alexander Pérez Ruíz MSc. PhD.
Director del trabajo

Bogotá, D.C., diciembre de 2018

Dedicatoria

Este trabajo de grado es dedicado a mis padres Graciela y José de Jesús, quienes han sido siempre mi apoyo incondicional a lo largo de toda mi vida.

Agradecimientos

Agradezco a mi tutor el Ingeniero Alexander Pérez por todos sus consejos, preocupación y enseñanzas, a Alexandra Puerto mi compañera de vida por soportar las necesidades y problemas a lo largo de estos años de estudio, a mis amigos, profesores y personal del laboratorio de la Escuela Colombiana de Ingeniería por sus consejos y ayuda durante el desarrollo de este proyecto.

Resumen

La robótica móvil ofrece una gran posibilidad de desarrollar aplicaciones en diversas áreas de la vida cotidiana. Sin embargo, para que sea útil en la mayoría de las aplicaciones, se debe contar con un mapa del entorno sobre el cual planificar trayectorias libres de colisión. Este mapa permite no sólo moverse en el espacio circundante sino identificar la posición del robot basado en las características del entorno.

En los últimos años se han creado gran variedad de algoritmos para la realización del SLAM con la intención de dar la capacidad de posicionar un robot en un entorno desconocido a la vez que se navega en él, dando como resultado técnicas con muy buenos resultados y con un bajo costo computacional que permiten ser implementados en sistemas embebidos como los computadores de placa reducida.

En este trabajo se hizo uso de las técnicas de creación de mapas por métodos probabilísticos que resultan ser un gran aporte a la solución del problema del SLAM y como fue la creación de la plataforma móvil mediante la integración de componentes de hardware y software en especial mediante el uso del *framework* ROS.

Aun cuando no se logró integrar la unidad de medición inercial ni una cámara WEB para el desarrollo de la odometría, se exploraron otras alternativas que permitieron la implementación y ejecución de los experimentos con los métodos *Gmapping* y *HectorSLAM* cubriendo el alcance planteado para el trabajo.

Índice general

1. Introducción 10

2. Localización y mapeo simultáneos 12

2.1 Robot móvil autónomo 12

2.2 El problema del SLAM 12

2.3 SLAM 13

2.3.1 SLAM Probabilísticos. 14

2.3.2 Técnicas de SLAM probabilísticos 15

3. Robot Operating System (ROS) 20

3.1 Estructura básica de ROS 20

3.2 Sistema de archivos de ROS 21

3.3 Linea de ordenes de comando de ROS 22

3.4 Herramientas gráficas de ROS 23

3.5 ROS en sistemas embebidos 24

3.5.1 Unidad de procesamiento 24

4. Plataforma móvil 26

4.1 Diseño del robot móvil 26

4.1.1 Diseño del módulo de potencia y actuadores. 27

4.1.1.1 Selección de baterías. 27

4.1.2 Diseño del módulo sensores. 30

4.1.3 Diseño del módulo sistemas embebidos y software. 32

4.2 Localización y mapeo simultáneos 41

4.2.1 Construcción del robot móvil terrestre. 42

4.2.2 Conexionado 43

4.2.3 Encoder. 44

4.2.4 Pruebas de funcionamiento. 46

5. Experimentos 47

5.1 Odometría 47

5.2 SLAM 50

5.2.1 Gmapping 51
5.2.2 Hector_mapping. 55
5.3 Contribución con la realización del proyecto 57
6. Conclusiones 58
7. Anexos 60
A1. Detección de IMU: 60
A2. Instalación de MPU-9250: 61
A3. Detección de PCA9685: 70
A4. Selección de SBC: 72
A5. Codigo en Python del nodo Velocidad 73
A6. Codigo en Python del nodo Odom 75
Acrónimos 78
Referencias 79

Índice de figuras

Fig. 1. Un modelo grafico del algoritmo del SLAM. *13*
Fig. 2. Diagrama de flujo del proceso de SLAM con EKF. *16*
Fig. 3. Grafo básico de ROS con dos nodos y un tema (topic). *21*
Fig. 4. Sistema de archivos de ROS. *22*
Fig. 5. Firefly RK-3288. ..... *25*
Fig. 6. Módulos básicos que componen el robot móvil. *26*
Fig. 7. Componentes del módulo de potencia y actuadores. *27*
Fig. 8. Controlador PCA9685 encargado del manejo de los actuadores del robot móvil. *28*
Fig. 9.Motor DC de escobillas modelo Saturn 35T original del vehículo utilizado en el proyecto (Amain hobbies, 2018). *29*
Fig. 10. Mecanismo de acople del servomotor para el manejo de la dirección y servomotor junto con el ESC *29*
Fig. 11. Componentes del módulo sensores. *30*
Fig. 12. Cuadro comparativo para seleccionar la IMU adecuada para el proyecto. *31*
Fig. 13. RPLIDAR A1 radar óptico láser usado en el robot. *32*
Fig. 14.Modulo sistemas embebidos y software *32*
Fig. 15. Esquema general de conexiones entre elementos de hardware *33*
Fig. 16. Modo de uso del nodo Teleop_twist_keyboard. *34*
Fig. 17. Datos publicados por el nodo rtimulib_node. *35*
Fig. 18.Nodo ImuFilter correspondiente al filtro de madgwick. *36*
Fig. 19.Nodo Velocidad encargado de publicar el número del actuador y su valor de PWM correspondiente. *37*
Fig. 20.El Nodo i2cpwm_board es el encargado del manejo de la PCA9685. ..... *37*
Fig. 21.El Nodo Odometry realiza la publicación de la información de odometría. *38*
Fig. 22.El robot_pose_ekf estima la posición y orientación 3D del vehículo. *39*
Fig. 23. RplidarNode publica los datos tomados por el escáner láser en el tema /scan. *39*
Fig. 24. Slam_gmapping crea un mapa 2D con los datos tomados por el escáner láser y el árbol de transformaciones. *40*
Fig. 25. Hector_mapping es un programa con la posibilidad de realizar SLAM sin necesidad de odometría. *41*
Fig. 26. Componentes del módulo Localización y mapeo simultaneo. *41*
Fig. 27. Vehículo RC Brama10$_B$ utilizado en el proyecto. Vista del chasis y retiro de componentes no usados del vehículo RC. *42*
Fig. 28. Montaje de robot móvil terrestre y descripción de partes. *42*
Fig. 29. Arquitectura del robot móvil con capacidad de localización y mapeo simultáneos. *43*
Fig. 30. Toma de medidas para el cálculo del ángulo de dirección del robot móvil. *45*
Fig. 31. Caracterización del robot en el árbol de transformaciones. *48*

Fig. 32. Nodos necesarios para odometría y sus interacciones. *48*
Fig. 33. Visualización de la odometría en RViz en el computador base. Robot desplazándose en círculos para la prueba de funcionamiento del encoder virtual y la odometría. *49*
Fig. 34.Ejemplo de los desplazamientos presentados por el robot al moverse en círculos. *49*
Fig. 35.Árbol de transformaciones final del robot para realizar SLAM *50*
Fig. 36.Nodos en operación para ejecutar SLAM con gmapping. *51*
Fig. 37.Localización y mapeo simultáneos SLAM con el robot móvil y gmapping en el área del corredor *52*
Fig. 38. Localización y mapeo simultáneos SLAM con el robot móvil y gmapping en el área del salón de semilleros. *53*
Fig. 39. Localización y mapeo simultáneos SLAM con el robot móvil y gmapping en el área del salón de semilleros. Pruebas disminuyendo los efectos causados por el desplazamiento mecánico del robot. *54*
Fig. 40.Nodos en operación para ejecutar SLAM con hector_mapping. *55*
Fig. 41.Localización y mapeo simultáneos SLAM con el robot móvil y hector_mapping en el área del corredor *56*
Fig. 42. Localización y mapeo simultáneos SLAM con el robot móvil y hector_mapping en el área del salón de semilleros y corredor contiguo. *57*
Fig. A2.43. Cambio de en la configuración de RTIMULib. *62*
Fig. A2.44. Cambio de en la configuración de QTcreator. *65*
Fig. A2.45. Cambio de ruta para el compilador GCC. *65*
Fig. A2.46. Comprobación del compilador GCC. *66*
Fig. A2.47. Ruta para compilación de RTIMULibDemoGL. *67*
Fig. A2.48.Compilación de RTIMULibDemoGL. *67*
Fig. A2.49. Ejecución de RTIMULibDemoGL. *68*
Fig. A2.50. RTIMULibDemoGL. *69*

Capítulo 1

1. Introducción

En el mundo de la robótica, existe un área que se dedica a la investigación y desarrollo de robots con capacidad de desplazamiento y toma de decisiones, mediante algoritmos computacionales y una variada gama de sensores, que se denomina robótica móvil autónoma (Cursos RoboLab, 2014).

Modelar el ambiente y conocer la posición del robot a medida que se mueve en él, son tareas que hacen parte de la investigación de la robótica móvil. Realizar estas tareas resulta complejo para que el robot se desplace de manera autónoma, porque para saber su posición exacta necesita de un mapa del ambiente que lo rodea y, además, para construir un mapa el robot requiere precisar su localización (Llofriu & Andrade, 2014).

El sistema de localización y mapeado simultáneos (*Simultaneous Localization And Mapping* - SLAM), busca precisamente dar solución a ese problema y hacer que el robot por sí mismo genere el mapa a medida que se desplaza en un ambiente desconocido, a la vez que se ubica en él.

El reto en el SLAM está en reducir la incertidumbre del entorno, que se debe a causas como el ruido, la baja precisión en la medida de los sensores y las propias condiciones de modelar el ambiente en el que se encuentra el robot. Para que el robot pueda ser capaz de procesar dicha información acercándose a una representación realista del lugar donde se encuentra, lo debe hacer mediante un proceso probabilístico. En otras palabras, un robot al moverse de manera autónoma no tiene una sola suposición de su ubicación en cualquier punto del tiempo. Más bien, representa la estimación de la posición como una distribución de probabilidad (Kosuru, 2011).

Para hacerle frente a la incertidumbre de los valores obtenidos por los sensores del robot utilizados en la predicción de su ubicación, se añade un elemento de aleatoriedad en el error. La representación más común es en forma de un modelo de espacio de estado con ruido aditivo gaussiano, lo que lleva al uso del filtro extendido de Kalman (EKF) para resolver el problema SLAM (Durrant-Whyte &

Bailey, 2006). El filtro extendido de Kalman es una herramienta muy utilizada en la estimación estocástica del error de sistemas no lineales y obtiene aproximaciones óptimas de las mediciones sucesivas con el objeto de minimizarlo (Welch & Bishop, 2010).

Dotar a un robot móvil con la capacidad de localización y mapeo simultáneos implica la interacción de varios subsistemas tanto de hardware como de software, por lo que este trabajo se enfocó en reproducir y validar técnicas de SLAM 2D, mediante la fusión sensorial de una IMU y un láser de medición de rango (LIDAR) utilizando el *framework* ROS (*Robot Operating System*), en una computadora de placa reducida (*Single Board Computer*), con la que se controla los movimientos de un vehículo terrestre. Es de resaltar que estos algoritmos están probados ampliamente en computadores de escritorio estándar X86 tanto de 32 bits como de 64 bits y el alcance del trabajo fue validar su comportamiento en arquitecturas ARM.

El resultado del trabajo de grado realizado fue la dotación de la capacidad de localización y mapeo simultáneos a un vehículo terrestre mediante el *framework* ROS y la fusión sensorial de una unidad de medición inercial y un LIDAR, lo que le permite al robot móvil, tener la posibilidad de realizar por lo menos dos técnicas de SLAM, específicamente las que proveen *Gmapping* y *HectorSlam*. Aún con la capacidad de realizar SLAM, la arquitectura ARM no permite compilar todas las librería necesarias para la completa implementación, con lo cual se dificultó la incorporación de un sistema de odometría visual y por conflictos constructivos no se incluyó un encoder con mayor resolución, que afecto la precisión de los resultados.

Al entrarse en este documento el lector encontrará en el capítulo 2 los conceptos de la localización y el mapeo simultáneo junto con la explicación de las dos técnicas probadas. En el capítulo 3 se encuentra la descripción del entorno de desarrollo proporcionado por el sistema operativo para robots ROS (por su sigla en idioma inglés *Robot Operating System*) haciendo énfasis en las particularidades de ejecución en un entorno embebido. En el capítulo 4 se describe la plataforma móvil terrestre utilizada para los experimentos tanto en la parte de hardware como de software. En el capítulo 5 se hace la descripción de los experimentos de calibración y ajuste de la plataforma robótica móvil y de desempeño de los algoritmos de SLAM ejecutados y finalmente en el capítulo 6, se hace una discusión sobre los resultados obtenidos y se dan las conclusiones del trabajo.

Capítulo 2

2. Localización y mapeo simultáneos

2.1 Robot móvil autónomo

Los robots móviles autónomos son todos aquellos con la capacidad de desplazarse en un espacio determinado con independencia y en variedad de condiciones. Con esta definición empezamos a hablar de autonomía, lo cual no involucra solamente lo referente a su suministro de energía sino también a la capacidad de este de observar, planificar, actuar y navegar en ambientes desconocidos, sin que en ello se realice intervención humana.

Los robots móviles surgen de la necesidad de aumentar los campos de acción de la robótica (Ollero Baturone, 2010). Su investigación durante los últimos años ha desarrollado múltiples aplicaciones y usos como, por ejemplo: exploración de otros planetas, barrido de desechos peligrosos, actividades recreativas, búsqueda y rescate en espacios confinados (Silva Ortigoza, García Sánchez, Barrientos Sotelo, & Molina Vilchis, 2012).

2.2 El problema del SLAM

Muchas de las veces en que un robot se desplaza a través de un entorno determinado, logra recorrerlo con facilidad debido a que se le ha proporcionado el mapa del lugar donde se moverá con anterioridad. También está la situación en la que no se cuenta con un mapa del lugar donde se moverá el robot, pero este tiene acceso constante y preciso de su ubicación con lo cual podrá montar un mapa del entorno valiéndose del continuo flujo de información suministrado.

Pero más comúnmente, se presenta la situación, donde se desconoce el entorno y no se tiene acceso a ningún tipo de información sobre la ubicación exacta del robot, por lo que este deberá generar un mapa y ubicarse en el mismo de forma combinada. Esto resulta complejo debido a que para poder

localizarse precisa de un mapa y para poder construir un mapa es necesario conocer su ubicación en forma precisa. Esta es la razón fundamental que busca resolver el SLAM (Localización y Mapeo Simultáneos) (Llofriu & Andrade, 2014).

2.3 SLAM

El SLAM es un proceso por el cual un robot móvil puede construir un mapa del ambiente donde se encuentra y al mismo tiempo utilizar ese mapa para estimar su ubicación en él. En SLAM tanto la trayectoria del robot y la ubicación de todos los puntos de referencia se estiman en tiempo real sin la necesidad de ningún conocimiento de la ubicación (Durrant-Whyte & Bailey, 2006).

El robot depende principalmente de la capacidad de sus sensores para extraer información útil para la navegación, pero existen múltiples factores que adicionan incertidumbre a estas mediciones lo que incrementa la dificultad de estimar correctamente el mapa y la ubicación. Algunas de las causas que generan este problema son el ruido de los sensores, errores en la medida del desplazamiento, simetrías del ambiente y entornos dinámicos (Ballesta, Gil, Reinoso, & Úbeda, 2010).

Para enfrentar la dificultad que da la incertidumbre muchas de las técnicas de localización y mapeo simultáneo plantean la solución al problema de la estimación como distribuciones de probabilidades (Ollero Baturone, 2010).

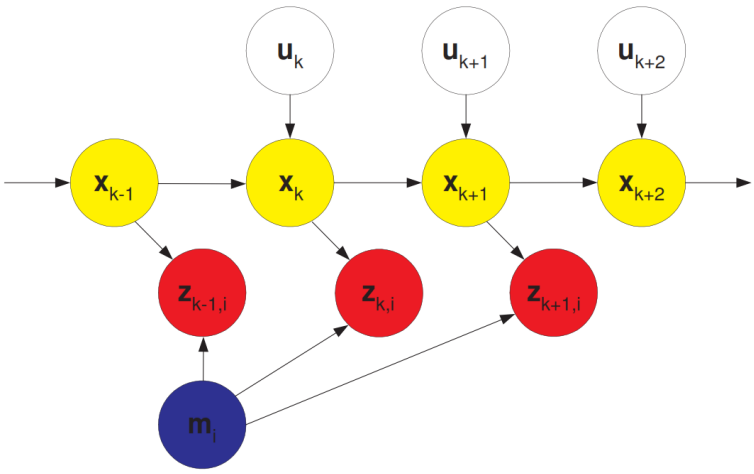


Fig. 1. Un modelo grafico del algoritmo del SLAM. H. Durrant-Whyte y T. Bailey. Ver (Durrant-Whyte & Bailey, 2006).

2.3.1 SLAM Probabilísticos.

Uno de los enfoques para la solución del problema SLAM, son los probabilísticos. Estos métodos se caracterizan principalmente en encontrar la distribución de probabilidad de la posición del robot y del mapa del entorno a través del tiempo.

En la actualidad, este enfoque es uno de los más usados y ha logrado demostrar su funcionabilidad en ambientes grandes y complejos. Entre las variadas técnicas que buscan resolver el problema del SLAM por métodos probabilísticos están: EKF-SLAM, FastSLAM, mapas de ocupación de celdas, triangulación estocástica, filtros de partículas y el visualSLAM (Paniagua Jaramillo, 2014).

2.3.1.1 Procesamiento de la información.

La manera en que se procesa la información cuando se desarrolla alguna solución al problema del SLAM, permite clasificar a los métodos probabilísticos (Llofriu & Andrade, 2014). Algunas de esas clasificaciones son:

- Full SLAM: Consiste en realizar la estimación de la ubicación y del mapa más factible dado un conjunto de observaciones y un conjunto de controles de movimiento en un momento dado.
- Online SLAM: Propone procesar la información de entrada una a una, para encontrar la posición más veraz en cada instante de tiempo (filtros de Kalman, filtros de partículas). El Online SLAM tiene gran popularidad debido a que puede ser ejecutado dentro de un robot operando en tiempo real, lo que hace que sea el método más adecuado en el desarrollo del proyecto.
- Paramétrico: Utiliza distribuciones de probabilidad conocidas que dependan de un conjunto reducido de parámetros para la actualización de su estimado a medida que llega información nueva.
- No paramétrico: Son capaces de representar cualquier función de densidad de probabilidad y mantener máximos locales, lo que equivale a mantener varios estimados completamente diferentes del estado del sistema de forma concurrente.

2.3.2 Técnicas de SLAM probabilísticos

La solución al SLAM probabilístico busca encontrar una representación adecuada para los modelos de observación y movimiento que admita un cálculo eficiente y estable de la estimación y corrección de las distribuciones de probabilidad. Una de las más comunes de estas representaciones, es en la forma de un modelo de espacio de estado con adición de ruido Gaussiano, lo cual conduce a la utilización del filtro de Kalman y filtro de Kalman extendido para resolverlo. Otra alternativa, es representar el modelo de movimiento del vehículo como un conjunto de muestras de una distribución de probabilidad más general no Gaussiana, lo que lleva a la utilización de filtros de partículas o FastSLAM, para resolver el problema de la localización y mapeo simultáneos (Aguilar, 2014).

2.3.2.1 Filtros de Kalman (KF).

El filtro de Kalman[1] es un algoritmo matemático de predicción para sistemas lineales de distribución normal o gaussiana muy utilizado para problemas de *Online* SLAM. Este método busca estimar el estado de un proceso de manera que minimice la media del error al cuadrado cuando al proceso se ve afectado por la adición de ruido blanco (Martínez Gómez, 2015).

El algoritmo funciona como un proceso en dos fases, una primera fase de predicción, en la cual el filtro de Kalman produce estimaciones del estado actual de las variables, junto a algunas incertidumbres. La segunda una vez tomada la salida del siguiente estado, el filtro renueva sus estimaciones utilizando una media ponderada, con mayor peso a las predicciones de las cuales se tiene un mayor nivel de certeza (Velásquez Hernández, 2017).

2.3.2.2 Filtro de Kalman extendido (EKF).

El filtro extendido de Kalman es una extensión del KF para sistemas no lineales, como es el caso en la mayoría de los procesos de localización y navegación (Kosuru, 2011). La idea principal del filtro asume que el estado real está muy cerca del estado estimado, por lo tanto, de acuerdo a como varíe el error puede ser modelado por medio de una expansión en series de Taylor linealizada de primer orden.

[1] Rudolf E. Kalman. https://es.wikipedia.org/wiki/Rudolf_E._Kalman

2.3.2.3 EKF-SLAM.

Uno de los primeros algoritmos usados para resolver el problema del SLAM fue el EKF-SLAM. El EKF-SLAM es un algoritmo del tipo *online*, en donde se calcula la posición y el mapa del entorno del robot en un instante de tiempo determinado.

El EKF-SLAM emplea el uso del filtro de Kalman extendido (EKF), para describir los cambios de los estados y el ruido introducido mediante representaciones gaussianas, para hacer frente y corregir la incertidumbre aditiva en cada uno de los ciclos de predicción y corrección.

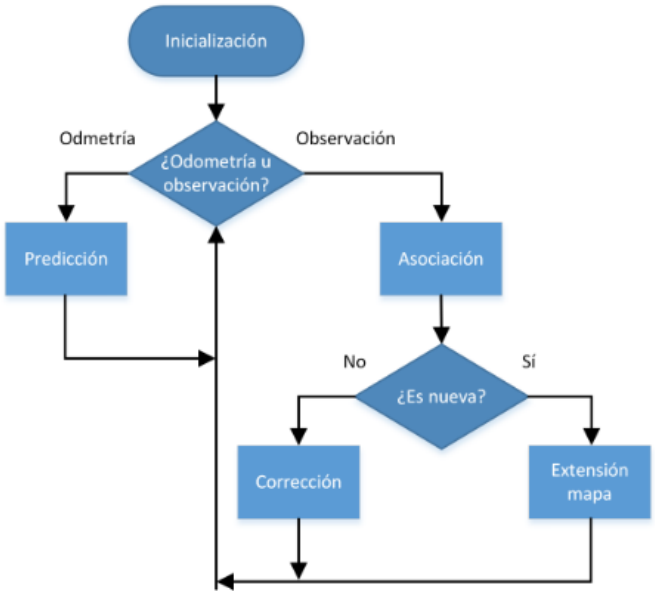


Fig. 2. Diagrama de flujo del proceso de SLAM con EKF. Fuente: (Carvajal Meza, 2015)

EKF-SLAM es un proceso recursivo en el que hay una continua realimentación. En una etapa se actualiza el estado correspondiente a la posición de acuerdo con su odometría. Con la otra etapa, se examinan los puntos de referencia en el entorno, que son comparados con aquellos que ya han sido tomados inicialmente. Los nuevos puntos de referencia se añaden al sistema, mientras que observados anteriormente se utilizan para actualizar el estado del robot y de los propios puntos de referencia, haciendo una corrección con base en la ganancia de Kalman y a la diferencia en las medidas tomadas y las pronosticadas (Martínez Gómez, 2015).

2.3.2.4 FastSLAM.

Una limitación clave de los enfoques basados en EKF es su complejidad computacional. Las actualizaciones de los sensores requieren tiempo cuadrático en el número de puntos de referencia K para calcular. Esta complejidad se debe al hecho de que la matriz de covarianza mantenida por los filtros de Kalman tiene K^2 elementos, los cuales deben actualizarse incluso si se observa un solo punto de referencia. La complejidad cuadrática limita el número de puntos de referencia que pueden manejarse con este enfoque a solo unos pocos cientos, mientras que los modelos de entornos naturales a menudo contienen millones de características.

FastSLAM descompone el problema del SLAM en un problema de localización del robot y una colección de problemas de estimación de puntos de referencia que están condicionados, a la estimación de la postura del robot. Esta representación factorizada es exacta, debido a las independencias condicionales naturales en el problema de SLAM. FastSLAM utiliza un filtro de partículas modificado para estimar la trayectoria posterior sobre el robot. En un filtro de partículas cada partícula posee filtros de Kalman, que estiman las ubicaciones de puntos de referencia condicionadas por la estimación de la trayectoria (Montemerlo, Thrun, Koller, & Wegbreit, 2002). Cada una de estas partículas se le da un peso en relación con lo acertado que este a su estado real y se utiliza este valor para concluir si se usara o no posteriormente (Martínez Gómez, 2015).

El coste computacional de esta técnica es proporcional al número de objetos en el mapa y al número de partículas usado. Sin embargo, el uso de técnicas de representación eficientes para las estimaciones de los objetos puede disminuir el valor computacional hasta crecer solo logarítmicamente con el número de objetos del mapa (Aguilar, 2014).

2.3.2.5 Mapas de ocupación de celdas.

Para realizar la planificación del movimiento, generalmente es necesario definir alguna representación del entorno y tener algún método para determinar la ubicación del robot en ese entorno. El problema del mapeo es determinar su representación, mientras que la localización es el problema de encontrar la posición del robot. Muchas técnicas probabilísticas para la localización dependen de que el mapa se defina como un conjunto de puntos de referencia de tamaño finito que observan los sensores del robot, dando su desplazamiento relativo del robot. Sin embargo, los sensores físicos no suelen detectar puntos de referencia de forma inequívoca. En su lugar, informan la distancia al

obstáculo más cercano o devuelven una imagen del entorno. Para utilizar un algoritmo basado en puntos de referencia, las lecturas del sensor deben pre procesarse en un paso separado para convertir los datos sin procesar del sensor en un conjunto de puntos de referencia detectados. El paso adicional introduce más errores en cualquier algoritmo, además de descartar gran parte de la información del sensor que no detecta ningún punto de referencia.

Uno de los principales inconvenientes de los mapas basados en puntos de referencia es el problema de asociación de datos. Debido a que los datos sin procesar del sensor no están etiquetados con el punto de referencia correcto detectado, el procesamiento del sensor debe determinar de alguna manera exacta qué punto de referencia se observó. Si se cometen errores, los algoritmos de localización y mapeo que dependen de los datos del sensor fallarán. Para compensar el problema de asociación de datos, muchos algoritmos de localización y SLAM incluyen un método para determinar las asociaciones entre los datos del sensor y los puntos de referencia, sin embargo, estas técnicas aumentan significativamente la complejidad de las soluciones. Además, no resuelven el problema de encontrar puntos de referencia en las lecturas sin procesar del sensor. Incluso con estas soluciones integradas, el problema de asociación de datos agrega una cantidad significativa de error.

Una técnica común para la representación de mapas que no sufre asociaciones de datos es utilizar mapas de ocupación de celdas para aproximarse al entorno. Un mapa de cuadrícula de ocupación representa el entorno como un bloque de celdas, cada una de ellas ocupada, de modo que el robot no puede pasar a través de ella, o desocupada, de modo que el robot pueda atravesarla.

Cualquier sensor informará el estado de un conjunto de celdas de cuadrícula que se pueden verificar sin referencia al resto del mapa. Las lecturas de los sensores se comparan con el mapa, lo que altera la probabilidad de que las celdas observadas estén ocupadas. Describe una técnica probabilística para actualizar celdas para varios tipos de sensores y proporciona una técnica para permitir que el mapa se actualice a medida que el robot se mueve.

Desafortunadamente, esta técnica no es realmente la localización y no ayuda al robot a conocer su propia posición. El mapa se mantiene en relación con el robot, en lugar de en un marco de referencia global. En otras palabras, se supone que el robot está en una ubicación fija, mientras que el mapa se mueve a su alrededor. A medida que el robot se mueve, el mapa se ve borroso según el movimiento. Los sensores del robot pueden corregir el mapa en su área inmediata, pero las partes no observadas del mapa deben volverse inútiles. Tampoco hay forma de descubrir la ubicación del robot en referencia a las ubicaciones visitadas anteriormente. Aunque la técnica es problemática como

algoritmo de localización, proporciona una forma muy poderosa de representar el entorno. El uso de un mapa de ocupación celdas permite utilizar los datos sin procesar del sensor sin intentar detectar e identificar puntos de referencia. Además, dado que se utilizan datos sin procesar, no se descarta ninguna información porque no corresponde a un hito. El único problema es que hay una gran cantidad de funciones de mapas, una para cada celda de la cuadrícula (Milstein, 2008).

Capítulo 3

3. Robot Operating System (ROS)

El sistema operativo para robots (ROS) es un *framework* de código abierto que permite crear y compartir software para robots, desarrollado por Willow Garage. ROS proporciona librerías, controladores y herramientas de comunicación para componentes robóticos, lo que lo hace compatible con una gran variedad de robots, actuadores y sensores.

Hay varias ventajas que hacen de ROS adecuado para los robots. El primero es que es fácil de integrar en diferentes robots debido a su abstracción de hardware. Otra ventaja es que Willow Garage está firmemente comprometido con el desarrollo de código abierto y el software reutilizable (Willow Garage, s.f.) y el código producido por universidades, empresas y usuarios está disponible de forma que para realizar un proyecto no hay la necesidad de empezar de cero. Actualmente, ROS tiene software para mapeo, navegación, exploración, reconocimiento de objetos, planificación de movimiento de brazos, simulación y aprendizaje automático. ROS es flexible, ya que ofrece la posibilidad de programar en tres lenguajes: C ++, Python o Lisp (Gallart Del Burgo, 2013).

ROS no es realmente un sistema operativo, funciona sobre Linux principalmente, pero cuenta con un entorno de trabajo y una gran cantidad de repositorios que ofrecen paquetes de software de todo tipo para aplicaciones robóticas. Hay varias versiones de ROS, pero en este proyecto se utilizó ROS-*Indigo Igloo,* que es la versión recomendada de ROS.org por su estabilidad.

3.1 Estructura básica de ROS

Para entender ROS de una manera más fácil se usan grafos. En un grafo podemos hallar nodos, los cuales son programas que se ejecutan en paralelo que se pueden comunicar entre ellos por medio de temas (*topics*), que pueden ser las lecturas de algún sensor, transformación de coordenadas, resultados de operaciones o comandos para actuadores.

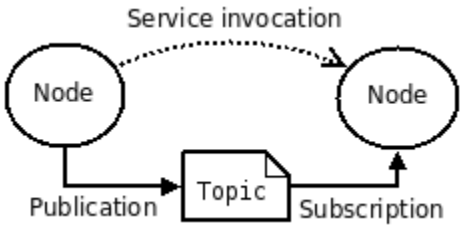


Fig. 3. Grafo básico de ROS con dos nodos y un tema (topic). Fuente: (Local Digital Library II, 2018)

Los nodos pueden publicar o suscribirse a uno o varios temas, pero también pueden comunicarse por medio de servicios, por el cual un nodo puede solicitar información en un momento especifico y no de forma permanente como en un proceso de suscripción (Garcia Montañés, 2017).

3.2 Sistema de archivos de ROS

Como se mencionó anteriormente ROS no es un sistema operativo propiamente, pero posee un sistema de archivos constituido de la siguiente manera:

- **Paquetes**. Los paquetes contienen los procesos de ejecución de ROS que son los nodos, las bibliotecas y archivos de configuración entre otros, que se organizan de manera independiente siendo un elemento constructivo y de lanzamiento del software de ROS.
- **Archivos package.xml.** Estos archivos contienen información sobre autor, licencia, dependencias, flags de compilación, etc, de un paquete. Cuando está dentro de un meta-paquete debe contener la lista de paquetes como dependencias y tener una etiqueta de exportación.
- **Meta-paquetes.** Este término agrupa un grupo de paquetes que tienen un propósito único o especial.
- **Archivos de Mensajes (.msg).** Los archivos de mensaje de ROS definen la estructura de los mensajes que intercambian los nodos entre sí. Cuando se precisa un mensaje para un paquete se guarda en el directorio msg del propio paquete.
- **Archivos de Servicio (.srv).** Los archivos de servicio de ROS definen la estructura de los mensajes de servicio que van a intercambiar los nodos. Cuando se delimita un mensaje propio para un paquete, se guarda en el directorio srv del propio paquete.

- **Repositorios.** La mayoría de los paquetes se mantienen usando un sistema de control de versiones (SCV) como git, subversión (svn), mercurial (hg). El conjunto de paquetes que comparten el mismo SCV se llama repositorio.

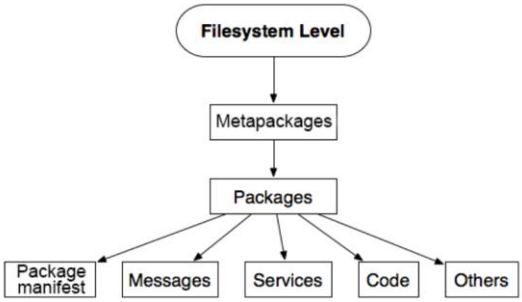


Fig. 4. Sistema de archivos de ROS. Fuente: (Rapado García, 2016)

3.3 Linea de ordenes de comando de ROS

Algunas de las líneas de comendo de ROS son básicas para su operación y desarrollo de software, las cuales se presentan a continuación (Rapado García, 2016).

- **roscore:** Permite poner en funcionamiento al Master de ROS. Internamente lanza el servidor de parámetros y el nodo *rosout* que se encarga de la monitorización del sistema.
- **rosrun:** Lanza un nodo indicando el nombre de su paquete y el nombre del nodo a ejecutar.
- **roslaunch:** Hace posible lanzar archivos con extensión launch. Estos archivos tienen un formato XML y usan el lenguaje de etiquetas para describir lanzamiento de nodos, configuración de parámetros, inclusión de ficheros y otras opciones. Con roslaunch es posible lanzar múltiples nodos de ROS en local o remotamente a través de SSH.
- **rosnode:** Muestra información de depuración de los nodos que se están ejecutando en el sistema ROS, incluyendo subscriptores, publicadores y conexiones.
- **rostopic:** Permite obtener información sobre los topics de ROS, incluyendo publicadores y subscriptores de cada tema (*topic*), su frecuencia de actualización o su tipo de mensaje asociado.

- **rosservice:** Muestra los servicios que están disponibles en todo el sistema ROS y hacer peticiones sobre cualquiera de ellos.
- **rosparam:** Obtiene y establece parámetros del servidor de parámetros de ROS. También se le puede dar valor mediante el uso de un archivo de extensión yaml.
- **catkin:** Es el sistema de compilación de paquetes de ROS. Combina macros de CMAKE y scripts de Python para dotar de funcionalidad sobre el flujo de trabajo de CMAKE.

3.4 Herramientas gráficas de ROS

Estas herramientas graficas permiten entre otras visualizar los resultados de algún proceso, grafos o simulaciones de robots para pruebas virtuales haciéndolas indispensables en el campo de la robótica.

- **RViz:** Es un paquete de ROS que contiene una potente herramienta de visualización 3D. El interés de esta herramienta radica en su capacidad para mostrar *topics* tales como nubes de puntos, trayectorias calculadas, odometría o transformadas junto con el modelo 3D del robot.
- **rqt:** Es un conjunto de herramientas software que implementan diversas herramientas GUI en forma de pluggins. Estos se pueden añadir en una ventana de rqt o pueden ejecutarse individualmente. Los usuarios pueden desarrollar sus propios pluggins para rqt utilizando Python o C++.
- **Gazebo:** Es un simulador ampliamente utilizado por los usuarios de ROS, que permite probar rápidamente algoritmos, diseños de robots, y realizar pruebas de regresión utilizando escenarios realistas. Gazebo ofrece la posibilidad de simular con precisión y eficiencia poblaciones de robots en entornos interiores y exteriores complejos. Integra un robusto motor de físicas, gráficos de alta calidad, e interfaces de programación y gráficos muy útiles. ROS se integra en Gazebo a través de su paquete Gazebo-ROS y ofrece generación de datos de sensores (con o sin ruido), como láseres 2D, cámaras 2D/3D, cámaras RGB-D, sensores de contacto, fuerza o par y así como la posibilidad de usar modelos creados por el usuario.

3.5 ROS en sistemas embebidos

El mercado de las placas embebidas o computadores de placa reducida está en continuo cambio y en completo apogeo, ya se pueden incorporar en una sola placa múltiples periféricos y opciones de comunicación, con un consumo y tamaño muy reducidos teniendo la potencia de un computador personal en un espacio tan pequeño como la palma de la mano (Sánchez Vítores, 2005).

Como se comentó anteriormente, el mercado del hardware libre se ha transformado en un área en plena expansión. Existen en la actualidad multitud de tarjetas microcontroladoras que a causa de su bajo coste y versatilidad han logrado hacerse dentro de la robótica educacional y de investigación. Cada vez es más habitual la elección de este tipo de tarjetas para el adelanto de prototipos hardware o de desarrollo de prácticas de electrónica o programación en laboratorios (Valera, Soriano, & Vallés, 2014).

En el momento de escoger entre todas las opciones disponible hay que tener en cuenta tres aspectos importantes: el sistema operativo, entorno de programación y la posibilidad de actualización o cambio del firmware de la placa a usar con el robot. Debido a que el proyecto requiere del uso de ROS en la placa, era necesario que fuese posible la instalación de un sistema operativo Linux, en especial una distribución como Ubuntu, ya que únicamente es posible trabajar bajo este OS.

3.5.1 Unidad de procesamiento

Para el procesamiento se utiliza comúnmente computadores de placa reducida que son computadores completos embebidos en un solo circuito. La arquitectura de estos computadores los hace atractivos para aplicaciones industriales debido a su tamaño reducido y menor peso que un computador convencional.

Los procesadores montados en estas placas pueden ser de varios tipos y fabricantes como los basados en x86 producidos por Intel y AMD o de diferente arquitectura como los procesadores ARM.

Usualmente los computadores de placa reducida soportan diferentes sistemas operativos siendo el más común los basados en Linux como Ubuntu, Debian y Raspbian, por tratarse de software libre, lo que ha permitido que múltiples desarrolladores trabajar y mejorar esta clase de dispositivos (Paniagua Jaramillo, 2014).

El computador de placa reducida utilizado en este proyecto es la placa Firefly RK-3288. Esta SBC posee un procesador Cortex-A17 de cuatro núcleos con arquitectura ARM, una GPU 3D integrada completamente compatible con OpenGL ES1.1 / 2.0 / 3.0, OpenCL 1.1 y DirectX 11. La RK3288 posee una interfaz de memoria externa de 4GB, doble canal de alto rendimiento (DDR3/DDR3L/LPDDR2/LPDDR3) capaz de mantener un ancho de banda de memoria exigente y 16GB de memoria flash, también proporciona un conjunto completo de interfaces periféricas para admitir aplicaciones muy flexibles, como salida HDMI, VGA, interfaz de audio, GPIOs, puertos seriales, 2 puertos USB 2.0, slot para memorias microSD, puerto Ethernet y WIFI.

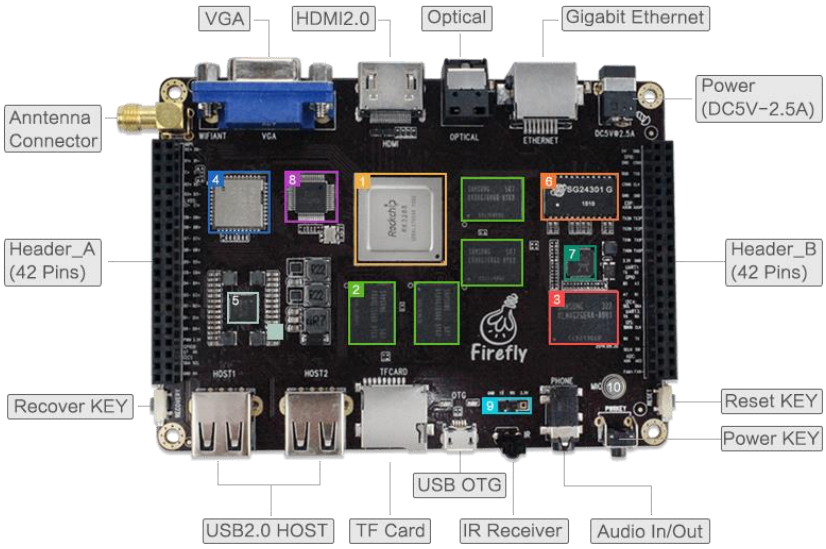


Fig. 5. Firefly RK-3288. Ver (CNXSOFT, s.f.).

Además, la placa posee soporte para Ubuntu Desktop 14.04 el cual soporta la instalación y ejecución de ROS y gracias a sus capacidades graficas es posible ejecutar herramientas graficas como Rviz entre otras.

Capítulo 4

4. Plataforma móvil

Este proyecto sobre robótica móvil tiene como objetivo el diseño y construcción de un prototipo de robot móvil terrestre con la capacidad de localización y mapeo simultáneos. Para lograr esto el robot posee varios módulos que le permiten movilidad, control, percepción y acción. Los diferentes módulos se encargan de una parte básica del funcionamiento del robot haciendo que en conjunto opere de manera correcta. Los módulos que componen el prototipo son los siguientes: potencia y actuadores, sensores, sistemas embebidos y software, cada uno dirigido a dotar al vehículo con la capacidad de localización y mapeo simultáneos.

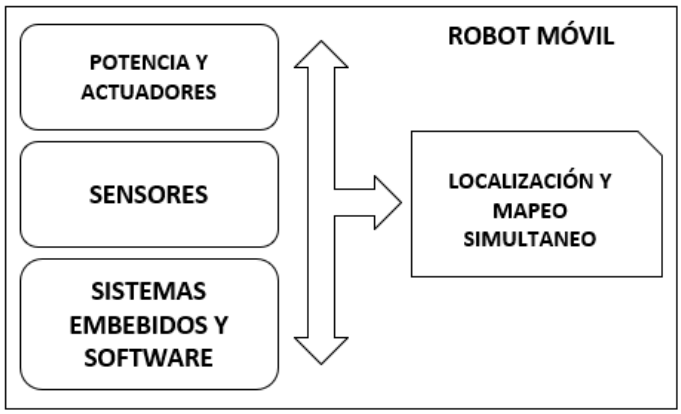


Fig. 6. Módulos básicos que componen el robot móvil.

Para cada módulo se requirió seleccionar o desarrollar según sea el caso, los dispositivos adecuados que los compone, esto se realizó en la siguiente sección que corresponde a la fase de diseño del robot móvil.

4.1 Diseño del robot móvil

La fase de diseño define las diferentes características de cada módulo y selecciona los componentes necesarios de acuerdo con los requerimientos para su funcionamiento teniendo en cuenta los aspectos básicos de operación.

4.1.1 Diseño del módulo de potencia y actuadores.

El módulo de potencia y actuadores lo conformaron todas las fuentes de energía necesarias para la operación del robot, los dispositivos electrónicos para el manejo de motores (*drivers)* y los motores.

Fig. 7. Componentes del módulo de potencia y actuadores.

Para cada uno de estos grupos existe variedad de opciones disponibles para ser utilizadas en el prototipo del robot móvil, por lo que a continuación se describe los criterios de selección de cada uno de ellos.

4.1.1.1 Selección de baterías.

Para el robot móvil se usó un vehículo tipo RC modelo Brama 10_B, que incluye una batería de 7.2 VDC con 1500 mAh de níquel cadmio (Ni-Cd), esta batería tiene un tiempo de uso no muy extenso, pero después de algunas pruebas se evidencio que se descarga rápidamente afectando de manera directa la operación del motor reduciendo sus rpm, razón por la que se decide cambiarla por una que garantice mejores prestaciones.

Aunque como se mencionó anteriormente existen múltiples opciones en la selección de baterías, se utilizó una batería de polímero de litio (LiPo) especialmente porque este tipo de baterías son ligeras, poseen una gran capacidad de almacenamiento de energía y tienen una tasa de descarga alta, lo que las hace muy prácticas para alimentar sistemas eléctricos exigentes.

Se adquirió para el robot una batería con un voltaje de 7.4 VDC y 2200 mAh, esta batería fue utilizada exclusivamente para alimentar el motor del vehículo que opera aproximadamente con una corriente de 2 A.

Para dar suministro de energía a los demás componentes se requirió de una batería adicional que pueda alimentar el computador de placa reducida el cual necesita para su funcionamiento un voltaje de 5VDC y una corriente de 2.5 A, el servomotor encargado de la dirección del vehículo opera a 4.8 VDC y una corriente aproximada de 0.5 A y el controlador de velocidad del motor (ESC) trabaja con un voltaje entre 6 a 8.4 VDC y una corriente de 1 A. Esta batería adicional es un power bank de 16000 mAh con dos salidas USB, una con la capacidad de suministrar un voltaje de 5 VDC y una corriente de 2.4 A y la otra un voltaje de 5 a 6.5 VDC y una corriente de 3 A. Este dispositivo opera de manera correcta proporcionando la energía suficiente para todos los dispositivos incluido el motor del LIDAR.

4.1.1.2 Selección de drivers.

En el mercado se encuentra gran variedad de drivers para el manejo de motores DC y otros dispositivos, pero se requería un tipo de controlador capaz de manejar el ESC del motor del vehículo, el servomotor y el motor del LIDAR además que facilitara el manejo de las diferentes señales de PWM, con estos requerimientos se usó el driver PCA9685 un controlador con interfase I^2C, 12 bit para señales de PWM y disponibilidad de 16 canales.

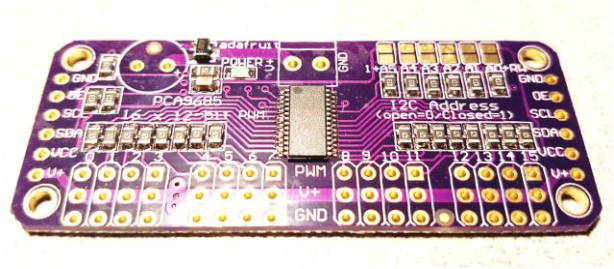


Fig. 8. Controlador PCA9685 encargado del manejo de los actuadores del robot móvil.

Gracias a su interfase I^2C este controlador nos permitió realizar el control de todos los actuadores necesarios en el robot móvil por medio de una conexión de tan solo dos cables y una resolución de hasta 4096 posiciones (2^{12}).

4.1.1.3 Actuadores del robot móvil.

El robot móvil por tratarse de un vehículo originalmente hecho para su manejo por radio control, ya cuenta con un motor *SATURN* 35T DC de escobillas de 12500 RPM. Este motor esta acoplado a una caja de reducción que transmite y distribuye la potencia del motor a las cuatro ruedas del robot.

Fig. 9.Motor DC de escobillas modelo Saturn 35T original del vehículo utilizado en el proyecto (Amain hobbies, 2018).

Para el manejo de la dirección, el vehículo utiliza un servomotor estándar modelo HPI SF-1 acoplado a un mecanismo que al moverse en un rango de 30 y -30 grados permite cambiar el ángulo de giro del robot móvil como se aprecia en la figura 10.

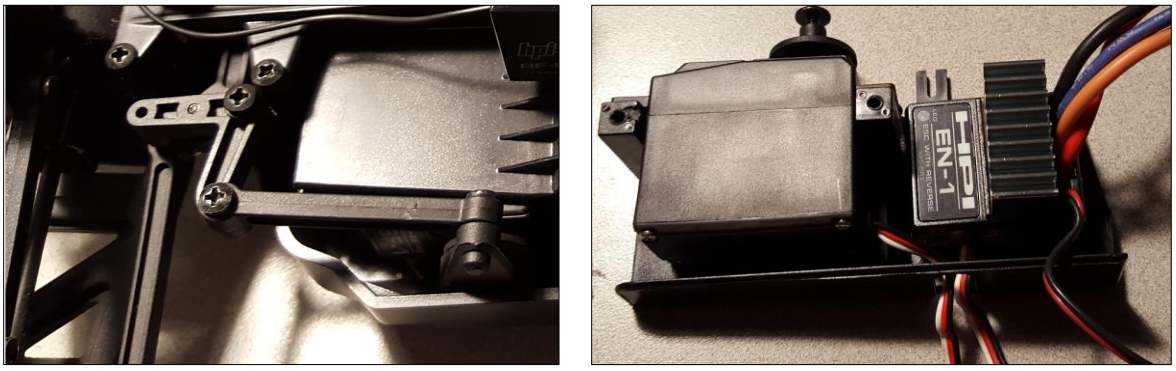


Fig. 10. Mecanismo de acople del servomotor para el manejo de la dirección (izquierda) y servomotor junto con el ESC (derecha).

El tercer actuador que se controla en el proyecto es el correspondiente al motor del LIDAR, este motor opera en un rango de voltaje entre 3.6 y 6 VDC, debiendo girar en sentido horario y se encuentra acoplado mediante una correa y poleas al mecanismo del escáner. La velocidad de operación del

motor se controla por medio de una señal de PWM que se debe ajustar de acuerdo con el voltaje de alimentación que se use.

4.1.2 Diseño del módulo sensores.

El módulo de sensores lo conformaron todos los dispositivos de adquisición de datos que permitieran tomar medidas del entorno y de movimiento del robot, como lo son para este caso la unidad de medición inercial (IMU) y el escáner láser (LIDAR).

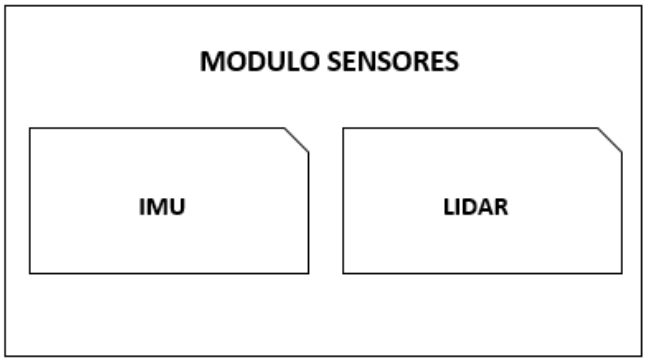


Fig. 11. Componentes del módulo sensores.

Para una adecuada selección de los dispositivos que se usaron para medir una variable determinada, fue primordial conocer bien el proceso en el cual es aplicado. Se debieron tener presente las características y factores del proceso al cual es sometido el instrumento lo que permite una apropiada fusión sensorial y generar los datos necesarios para el desarrollo del proyecto.

4.1.2.1 Selección de unidad de medición inercial.

La selección de los sensores debió tener en cuenta además de sus características (resolución, precisión, margen de medida, tiempo de respuesta, estabilidad), tamaño, precio y facilidad de adquisición. En el cuadro de la figura 12, se muestra un comparativo entre varias IMU, de este paralelo se estableció que la unidad de medición inercial más adecuada para el proyecto es la MPU-9250.

SPECIFICATIONS	MPU-6050		MPU-9250		MPU-9150A	
MOTION TRACKING	6-AXIS	1	9-AXIS	2	9-AXIS	2
GYROSCOPE	3-AXIS	2	3-AXIS	2	3-AXIS	2
	SCALE RANGE: ±250, ±500, ±1000, ±2000°/s	2	SCALE RANGE: ±250, ±500, ±1000, ±2000°/s	2	SCALE RANGE: ±250, ±500, ±1000, ±2000°/s	2
ACELEROMETER	3-AXIS	2	3-AXIS	2	3-AXIS	2
	SCALE RANGE: ±2g, ±4g, ±8g, ±16g	2	SCALE RANGE: ±2g, ±4g, ±8g, ±16g	2	SCALE RANGE: ±2g, ±4g, ±8g, ±16g	2
MAGNETOMETER	-	0	3-AXIS	2	3-AXIS	2
	-	0	HALL-EFFECT MAGNETIC SENSOR	2	HALL-EFFECT MAGNETIC SENSOR	2
	-	0	±4800µT	2	±1200µT	1
DIGITAL MOTION PROCESOR	SI	2	SI	2	SI	2
COMPASS	NO	0	SI	2	SI	2
ADC	16-bit	2	16-bit	2	16-bit	2
COMM BUS	I2C 400kHz	2	I2C 400kHz	2	I2C 400kHz	2
COMM BUS	SPI 1MHz	2	SPI 1MHz	2	-	0
TEMPERATURE SENSOR	SI	1	SI	1	SI	1
POWER SUPPLY	2.375 - 3.46 VDC	2	2.4 - 3.6 VDC	1	2.375 - 3.46 VDC	2
PACKAGE	4x4x0.9 mm	1	3x3x1 mm	2	4x4x1 mm	1
VALOR PESO COMP.	22		31		28	

Fig. 12. Cuadro comparativo para seleccionar la IMU adecuada para el proyecto.

De acuerdo a los criterios de selección mostrados se escogió la MPU-9250, un dispositivo con mayores prestaciones que las otras unidades de medición inercial del mercado, de bajo costo y que de acuerdo su fabricante *InvenSense* tiene un consumo de energía 44% menor que otras unidades similares como la MPU-9150A y un rendimiento 3 a 4 veces mejor en su giroscopio y magnetómetro (Inc., s.f.).

4.1.2.2 RPLIDAR.

El RPLIDAR es el radar óptico láser fabricado por la empresa Robo Peak, el cual fue suministrado por la Escuela Colombiana de Ingeniería Julio Garavito para para el desarrollo del proyecto. El escáner laser específicamente el modelo A1M1-R1 que se usó como parte del robot del proyecto, es un láser 2D con escaneo de 360° también conocido como radar óptico. Mediante el giro mecánico mide la distancia a un objeto en un rango determinado de hasta 6 m, de manera que obtenemos una nube de puntos con los cuales podemos construir un mapa del entorno donde se encuentra el robot (RoboPeak, 2014).

Fig. 13. RPLIDAR A1 radar óptico láser usado en el robot.

El láser trabaja con una frecuencia de muestreo por giro pudiendo ser configurado, lo que hace a este dispositivo ideal para ser usado en múltiples aplicaciones especialmente en lugares interiores.

La resolución angular está determinada por la velocidad de giro del motor, lo que impacta directamente en el número de muestras en cada escaneo. La velocidad de rotación del motor puede ser controlada por el usuario por medio de una señal PWM como se mencionó anteriormente.

4.1.3 Diseño del módulo sistemas embebidos y software.

El módulo de sistemas embebidos y software lo conformaron el computador de placa reducida y el software necesario para la lectura de datos de los sensores, la operación de los actuadores y el mapeo y localización simultáneos. Vale la pena aclarar que el software al que se hace referencia corresponde únicamente a los nodos dentro de ROS.

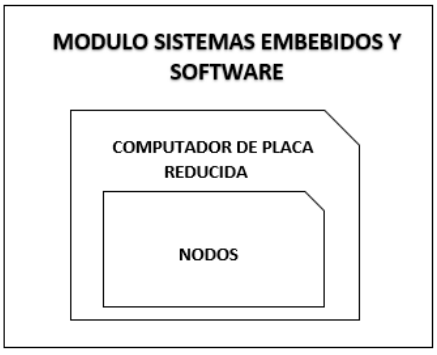


Fig. 14.Modulo sistemas embebidos y software.

En los anexos al final del documento se encuentra de manera detallada el paso a paso y los comandos de instalación de varios de los nodos utilizados en el proyecto.

3.1.3.1 Selección Computador de placa reducida SBC.

Escoger de manera apropiada el computador de placa reducida, da la facilidad en la instalación del software requerido, en la conexión de los sensores y en el fácil acoplamiento a otras plataformas de hardware, las cuales se encargan del manejo de los actuadores del robot móvil.

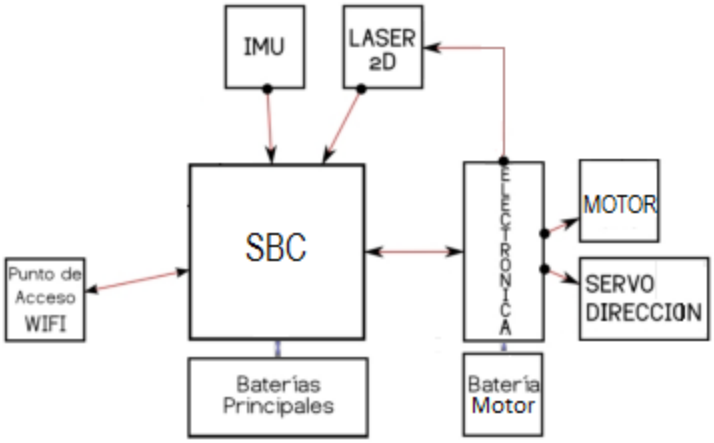


Fig. 15. Esquema general de conexiones entre elementos de hardware (modificado del original). Ver (Jiménez González, 2008).

En el esquema de la figura 15, el procesamiento se concentra en un computador de placa reducida (SBC) al que se conectan los sensores (IMU y LÁSER 2D) y la electrónica que maneja el accionamiento de los motores. Por último, se comunica con el computador base por medio un punto de acceso WIFI.

En la actualidad hay una gran variedad de sistemas embebidos que cuentan con grandes características y funcionalidades por lo que al igual que con la unidad de medición inercial, se realizó un cuadro comparativo para establecer que plataforma era la más adecuada para el proyecto, el cual se puede detallar en el anexo A4.

De esta comparación se seleccionó la Firefly RK-3288 la cual posee las mejores características para la realización del proyecto y en especial por poseer un punto de acceso WIFI necesario para el envío y recepción de datos desde y hacia el computador base.

4.1.3.2 Selección de software.

Esta parte del módulo contiene todos los nodos que se ejecutaron bajo ROS y que hicieron posible el SLAM para el robot móvil del proyecto. Cada nodo cumple una tarea específica como controlar los actuadores del vehículo, filtrar los datos tomados de los sensores o realizar los algoritmos para la localización y mapeo simultáneos. A continuación, se describen cada uno de los nodos utilizados.

Teleop_twist_keyboard

Teleop_twist_keyboard (Graylin Trevor, s.f.) es un nodo que por medio del teclado permite controlar un robot móvil publicando valores de velocidad lineal y angular en el tema (*topic*) /cmd_vel mediante un mensaje tipo *twist*. El mensaje tipo *twist* contiene seis valores uno para cada componte (x, y, z) de las velocidades lineal y angular los cuales se pueden incrementan o disminuir en pasos del 10%.
Estos valores son publicados desde el computador base y al ejecutarse muestra en pantalla la configuración de teclas y los valores de velocidad que están siendo publicados, tal como se ve en la figura 16.

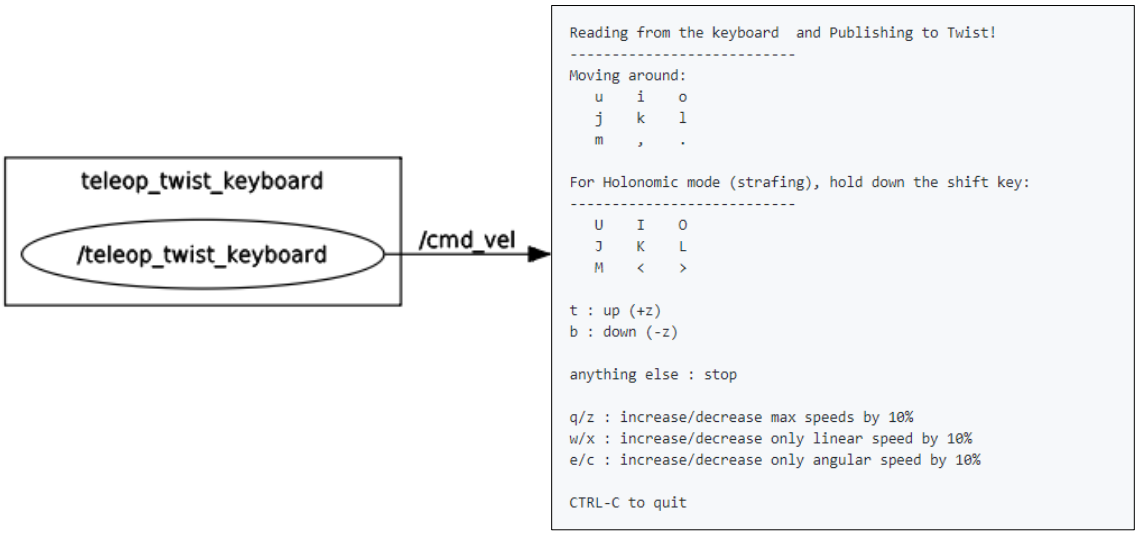


Fig. 16. Modo de uso del nodo Teleop_twist_keyboard.

Rtimulib

Rtimulib (Barnett, s.f.) es una completa librería que facilita la conexión de una gran variedad de IMU de 9 o más grados de libertad y obtener los datos filtrados mediante algoritmos de fusión, todos estos datos necesarios para el desarrollo de la odometría. La librería proporciona también los recursos necesarios para llevar a cabo la calibración del sensor, procedimiento que se puede encontrar de forma detallada en el anexo A2.

Con respecto a ROS la librería proporciona un paquete llamado *rtimulib_ros* que contiene el nodo *rtimulib_node* el cual publica los datos de orientación, aceleración lineal y velocidad angular.

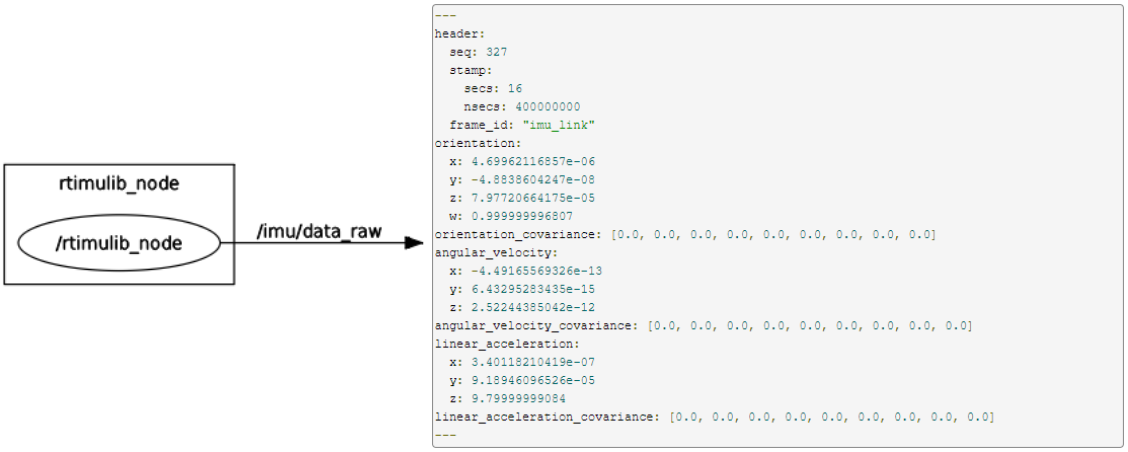


Fig. 17. Datos publicados por el nodo rtimulib_node.

Estos valores son publicados en el tema llamado /imu/data_raw mediante un mensaje tipo imu. Estos datos a pesar de estar filtrados por los algortimos RTQF o Kalman necesitan ser filtrados nuevamente con el fin de disminuir el error en la orientación, para corregir esto es necesario la utilización de uno en especial del conjunto de filtros en el paquete de ROS llamado *imu_tools*.

Imu_filter_madgwick

El paquete para ROS llamado *imu_tools* se compone de una serie de herramientas desarrolladas para unidades de medición inercial, entre esas herramientas se encuentran los nodos *imu_complementary_filter*, *imu_filter_madgwick* y *rviz_imu_plugin*. De este conjunto de herramientas usamos el creado en 2009 por Sebastian Madgwick quien desarrollo un algoritmo de fusión para sensores IMU y AHRS el cual demostró poseer un muy bajo consto computacional y una estabilización muy significativa en los datos suministrados por él dispositivo (x-io Technologies, 2012).

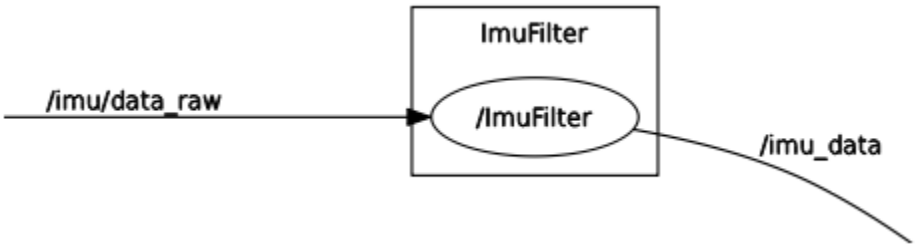


Fig. 18.Nodo ImuFilter correspondiente al filtro de madgwick.

El Filtro fusiona velocidades angulares, aceleraciones y opcionalmente lecturas magnéticas de un dispositivo IMU en una orientación, por esta razón el filtro requiere suscribirse a un tema llamado imu/data_raw que contiene un mensaje con los datos en bruto de la IMU y opcionalmente al tema imu/mag que contiene los valores dados por el magnetómetro. Como respuesta, el nodo publica el tema /imu_data que contiene un mensaje tipo imu fusionado que contiene la orientación.

Si es requerido este posee una serie parámetros configurables que se pueden detallar en el resumen del paquete que proporciona ROS (Dryanovski, ROS.org, 2016).

Velocidad

El nodo velocidad corresponde a un código realizado en Python con el fin de leer los datos publicados por el nodo *teleop_twist_keyboard* y relacionarlos a un valor de PWM, el cual será publicado en dos temas distintos con mensajes tipo *twist* y *ServoArray*. El primer tema es el /servos_absolute que envía

el número del actuador a accionar y el valor de pwm al cual va a operar y el segundo es el tema llamado /pwm que tiene como función enviar el valor de pwm para realizar la odometría.

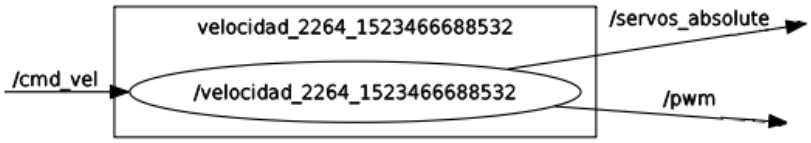


Fig. 19.Nodo Velocidad encargado de publicar el número del actuador y su valor de PWM correspondiente.

El código en Python del nodo se encuentra en el anexos A5 al final del documento.

I2cpwm_board

Este paquete para ROS diseñado por Bradan Lane Studio da la facilidad para el manejo de placas PWM de 16 canales, 12 bit con interfaz I^2C con chip PCA9685 como la usada en el proyecto. Este programa permite el manejo de servos estándar, servos de movimiento continuo, motores de corriente continua o cualquier otro dispositivo capaz de manejar señales PWM o PPM con una frecuencia predeterminada de 50 Hz, que puede ser modificada mediante los servicios públicos propios del paquete (Bradan Lane Studio, s.f.).

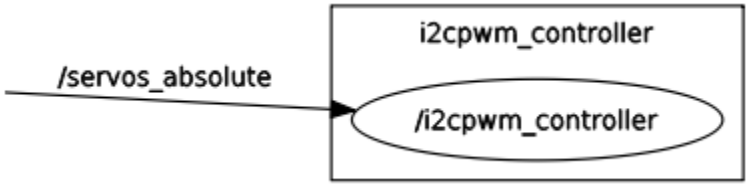


Fig. 20.El Nodo i2cpwm_board es el encargado del manejo de la PCA9685.

Los temas en los que se puede suscribirse son *servos_drive*, *servos_proportional* y *servos_absolute* este último el que se utiliza en el proyecto controla los actuadores con valores de inicio y parada absolutos, permitiendo ajustar los valores de operación de cada actuador.

Odometry

Odometry (Odometria) es el nodo creado en Python para realizar la publicación de información de odometría del vehículo sobre ROS, conjugando los valores de PWM provenientes del nodo velocidad, para crear un encoder virtual que se detalla más adelante y las ecuaciones de movimiento del robot móvil en su configuración Ackerman.

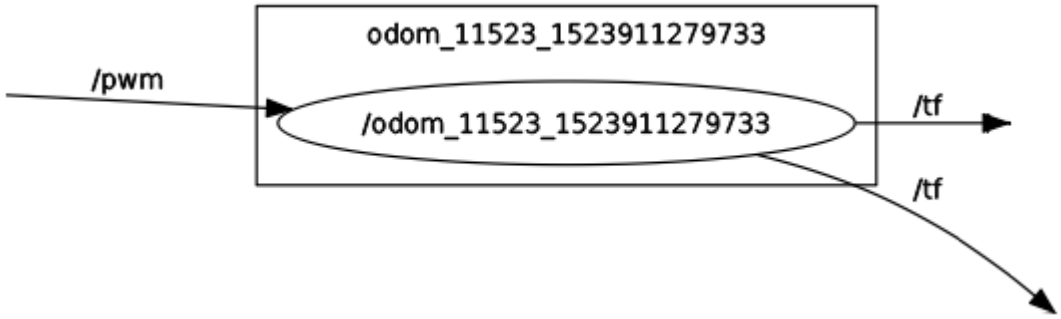


Fig. 21.El Nodo Odometry realiza la publicación de la información de odometría.

El nodo publica estos valores en un árbol de transformación (/tf) utilizando la biblioteca de software TF, realizando las compensaciones necesarias entre dos diferentes marcos de referencia del robot, que son base_link y el marco de referencia para la odometría denominado odom.

Robot_pose_ekf

Es *robot_pose_ekf* es el paquete que se utilizó para estimar la pose 3D del robot, basándose en mediciones parciales de diferentes fuentes. Utiliza un filtro de Kalman extendido con un modelo 6D (posición 3D y orientación 3D) para combinar las mediciones de la odometría, el sensor IMU y la odometría visual.

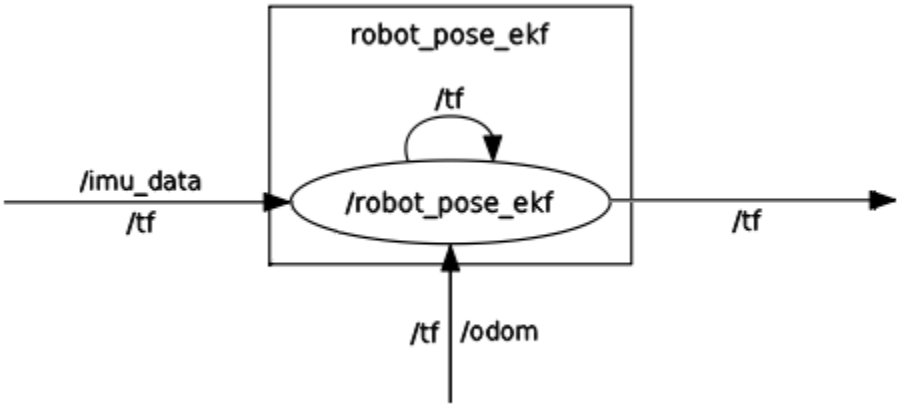


Fig. 22.El robot_pose_ekf estima la posición y orientación 3D del vehículo.

La idea básica es ofrecer una integración débilmente acoplada con diferentes sensores, donde las señales de los sensores se reciben como mensajes de ROS. Con esto el nodo debe suscribirse mínimo a dos temas odom y imu_data y de manera opcional a vo para la odometría visual si se cuenta con una cámara para esta función.

Rplidar

Este paquete para ROS proporciono el manejo de los datos tomados por el escáner láser 2D RPLIDAR A1. El controlador publica datos del tipo sensor_msgs/LaserScan dependientes del dispositivo en el tema /scan. Estos datos luego son tomados por algún otro programa para desarrollar por ejemplo SLAM.

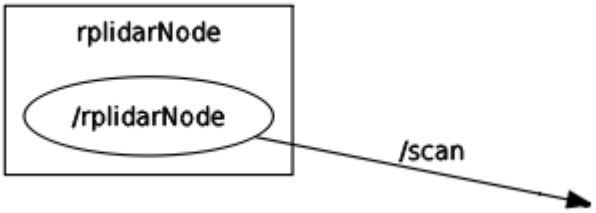


Fig. 23. RplidarNode publica los datos tomados por el escáner láser en el tema /scan.

Este nodo solo publica los datos tomados por el escáner y no se encarga de manejar el actuador, pero sí permite cambiar otros parámetros como el puerto de comunicaciones, la velocidad del puerto, el nombre del *frame* o si este está montado de manera invertida.

Gmapping

El paquete *gmapping* para ROS perteneciente al proyeto OpenSLAM proporciono la capacidad para realizar SLAM (localización y mapeo simultáneos) en el robot móvil, con base en los datos proporcionados por el láser y el árbol de transformaciones de los diferentes marcos de referencia. El nodo de ROS que nos proporciona el paquete *gmapping* llamado *slam_gmapping* puede crear un mapa 2D y presentar la ubicación del robot en un ambiente grafico como por ejemplo rviz.

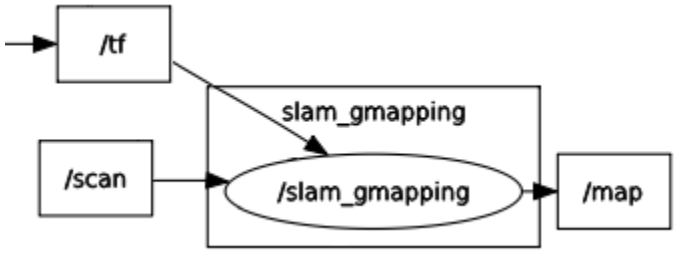


Fig. 24. Slam_gmapping crea un mapa 2D con los datos tomados por el escáner láser y el árbol de transformaciones.

Recientemente los desarrolladores han introducido filtros de partículas Rao-Blackwellized como medios efectivos para resolver el problema de localización y mapeo simultáneos (SLAM). Este enfoque utiliza un filtro de partículas en el que cada partícula lleva un mapa individual del entorno, lo que disminuye drásticamente la incertidumbre sobre la postura del robot en el proceso de predicción del filtro (Grisetti, Stachniss, & Burgard, 2007).

Hector_slam

Hector_slam es un paquete que instala el nodo *hector_mapping* y otros nodos relacionados como *hector_geotiff* y *hector_trajectory_server* que son usados para otras aplicaciones. *Hector_mapping* es un programa de SLAM que se puede usar sin odometría, así como en plataformas que exhiben movimiento de balanceo, aprovecha la rápida tasa de actualización de los escáner tipo LIDAR actuales y proporciona estimaciones de postura 2D (Kohlbrecher, von Stryk, Meyer, & Klingauf, 2011).

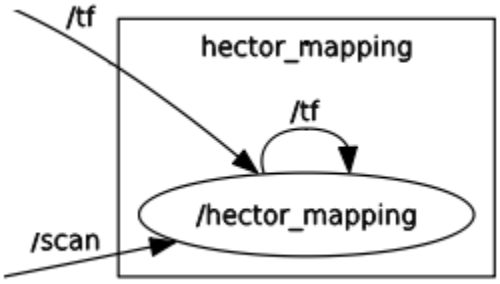


Fig. 25. Hector_mapping es un programa con la posibilidad de realizar SLAM sin necesidad de odometría.

El paquete de *Hector_slam* para ROS se ha utilizado con éxito en diferentes tipos de robots y aplicaciones donde es necesario aprender un mapa de un lugar o ambiente desconocido además de requerir bajos recursos computacionales.

4.2 Localización y mapeo simultáneos

En la fase de localización y mapeo simultáneos se describe la integración de los anteriores módulos en su parte constructiva, conexionado de componentes, encoder para la odometría y pruebas de funcionamiento para poder dotar con la capacidad de SLAM al robot móvil terrestre.

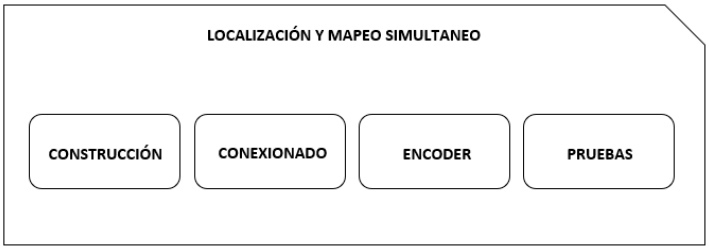


Fig. 26. Componentes del módulo Localización y mapeo simultaneo.

4.2.1 Construcción del robot móvil terrestre.

En este módulo se realizaron las modificaciones necesarias al vehículo RC para transfórmalo en el robot móvil del proyecto. Primero se retiraron los componentes que no son parte del robot como su carrocería y el receptor, quitarlos proporciono mayor espacio para acomodar los dispositivos y soportes necesarios.

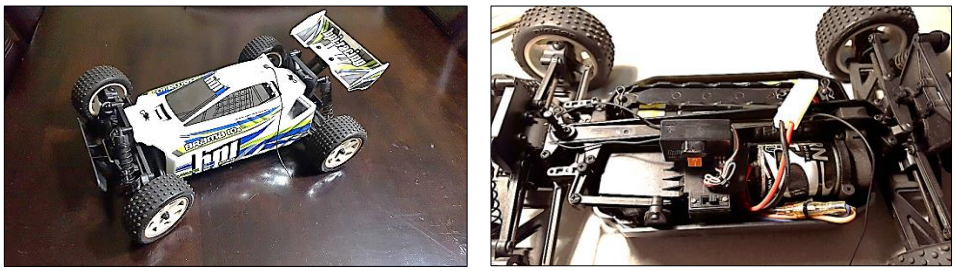


Fig. 27. Vehículo RC Brama10B utilizado en el proyecto (Izquierda). Vista del chasis y retiro de componentes no usados del vehículo RC (Derecha).

Mediante soportes metálicos de aluminio se colocaron láminas de acrílico transparentes cortadas con una CNC láser, previamente diseñadas por computador donde se instalaron el computador de placa reducida, el controlador I^2C de 16-Canales PCA9685, el RPLIDAR y la batería principal (*Power Bank*) tal como se puede ver en la figura 28.

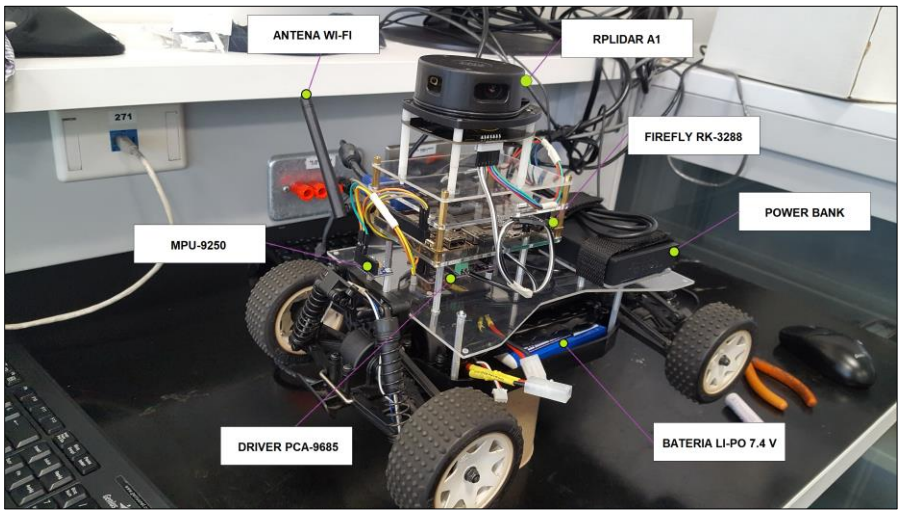


Fig. 28. Montaje de robot móvil terrestre y descripción de partes.

Sobre el centro del eje trasero se montó la IMU, esto porque es siempre perpendicular al centro de rotación del vehículo. En el espacio que provee la carrocería de fábrica para la batería se instaló la de Li-Po de 7.4V y 2000mAh y en la parte superior mediante separadores de nylon se ubicó el láser (LIDAR) para que de esa manera no detectara las diferentes piezas del robot.

Como parte del montaje se instaló en el computador base, la versión ROS-*Indigo Igloo* y se configura asignando una dirección IP fija a este equipo y al computador de placa reducida para que se pueda realizar la comunicación entre los dos equipos a través de un router wifi dedicado exclusivamente para el proyecto.

4.2.2 Conexionado.

Como punto de partida y como base para conectar correctamente todos los dispositivos se estableció la arquitectura del proyecto, donde se aprecia como interactúan los diferentes componentes del robot para dotarlo con la capacidad de localización y mapeo simultáneos.

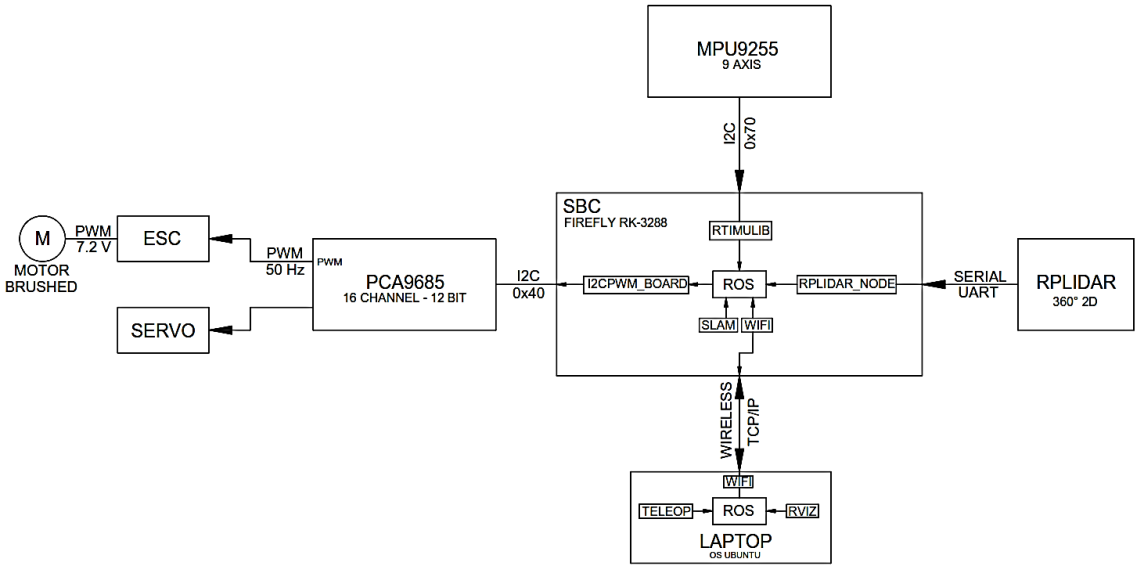


Fig. 29. Arquitectura del robot móvil con capacidad de localización y mapeo simultáneos.

En la arquitectura se muestran los componentes físicos y de software, su tipo de conexión con otros dispositivos y sus protocolos de comunicación y de esta manera poder realizar una descripción de cómo se efectúo el conexionado de todos ellos en el proyecto.

Como lo indica la arquitectura, la unidad de medición inercial envía los datos sensados por medio del protocolo de comunicación I^2C al igual que el controlador PCA-9685, quien recibe los datos de posición del actuador y valor de pwm de la SBC. La MPU-9250 se conecta físicamente a través del controlador el cual que posee puntos de cableado en paralelo para la comunicación y alimentación a un voltaje de 3.3V suministrado por la SBC en su conexión directa. La electrónica del escáner laser se conecta a un voltaje de 5V también suministrado por la SBC y comunicación serial por medio del controlador UART3.

Las conexiones de voltaje y señales PWM para el ESC, servomotor y motor del LIDAR se realizan entre los puertos de 0 al 2 del controlador, teniendo como fuente de poder para estos dispositivos el power bank.

4.2.3 Encoder.

Ante la dificultad presentada en numerosas pruebas de lograr obtener mediciones de velocidad y desplazamiento útiles y estables en el tiempo, a partir de los datos de aceleración lineal suministrados por la IMU, que permitieran implementar un controlador PID y con base en estas mediciones realizar la odometría del robot, fue necesario implementar un encoder "virtual" o por software.

La odometría para un robot móvil se define como la ubicación estimada del robot en un momento determinado en relación con su posición inicial utilizando información sobre su movimiento, para los robots con ruedas por ejemplo sabiendo el número de rotaciones de la rueda y el ángulo de la dirección, se puede hacer una estimación de la ubicación del robot en un momento determinado. Por eso para el robot móvil es necesario usar la información de la odometría que al fusionar con la información de los demás sensores pueda calcular su posición precisa en un lugar determinado (Mansoor, 2018).

La mayoría de los robots móviles cuentan con encoders que proporcionan la información sobre la rotación de sus ruedas, pero como fue el caso del robot del proyecto, este no contó con la facilidad de instalación de encoders adecuados en sus ruedas, sin embargo, no son estrictamente necesarios para crear la odometría. Las ruedas están controladas por motores y si se conoce un valor particular de entrada del motor esto resultara en un numero particular de rotaciones de las ruedas por lo que fue posible con esta información realizar la odometría del robot.

Lo primero que se hizo es dar diferentes valores de pwm al motor durante un tiempo fijo y medir la distancia recorrida por el robot, luego con los datos calcular la velocidad y mediante una regresión lineal se encuentra una ecuación también lineal de la forma:

$$velocidad = a * pwm\_motor + b$$

Ahora para calcular el ángulo de dirección del robot se asumió que los valores de giro son simétricos tanto a la izquierda como a la derecha y se dan 4 valores diferentes al servomotor espaciados de igual manera, y se marca la trayectoria del centro del robot en los diferentes valores mientras este gira un circulo completo. Después en un punto cualquiera se calcula la tangente al círculo y se calcula el ángulo formado entre esta y la línea central del robot.

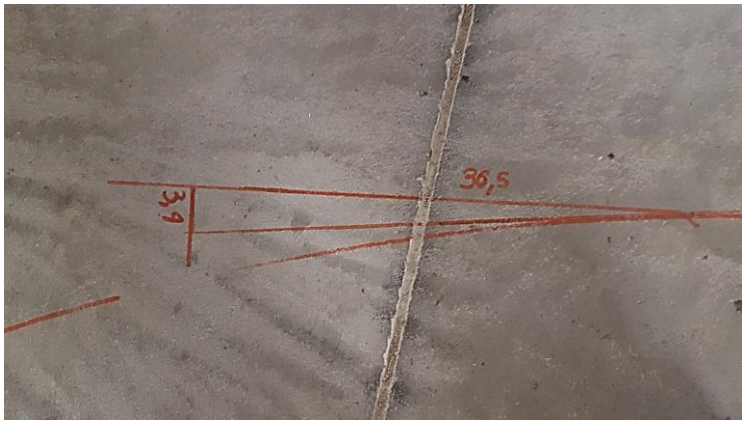


Fig. 30. Toma de medidas para el cálculo del ángulo de dirección del robot móvil.

Entonces, al usar los cuatro valores tomados siendo realmente ocho por simetría, se obtiene la ecuación para el ángulo del robot con respecto al valor de pwm dado al servomotor, resultando ser lineal y de la forma:

$$angulo\_direccion = a * pwm\_servo + b$$

Con esta información y valores de distancia entre las ruedas delanteras y traseras se realizó el código para el nodo *odom* encargado de publicar la odometría al sistema.

4.2.4 Pruebas de funcionamiento.

Las pruebas operacionales consintieron en evaluar el desempeño de la dotación de la capacidad de localización y mapeo simultáneo *indoor* del robot móvil terrestre en un entorno conocido, lo que permitió realizar ajustes y corregir de manera apropiada los errores por programación o por construcción.

A lo largo del proceso de diseño y construcción, se realizaron diversas pruebas de funcionamiento a medida que se instalaban los diferentes dispositivos, pudiendo así ejecutar mejoras con cada una de estas. Las pruebas con el motor permitieron la utilización de un mejor controlador del que se tenía planeado al comienzo del proyecto, y aseguraron realizar una adecuada calibración de la unidad de medición inercial. También evidenciaron algunos problemas en la estructura de soporte y fallas de operación de ciertos programas debido a la arquitectura del procesador del computador de placa reducida.

Capítulo 5

5. Experimentos

Tras el funcionamiento del robot móvil se evaluó su capacidad para desarrollar localización y mapeo simultáneos (SLAM), realizando varias pruebas en espacios *indoor*, empezando primero con el funcionamiento del encoder virtual y luego con los ya mencionados programas Gmapping y HectorSlam, revelando las ventajas y las deficiencias del proyecto, las cuales se mencionan a continuación.

5.1 Odometría

Lo primero que se tuvo que poner a prueba fue el funcionamiento del encoder virtual para la obtención de la odometría, la cual es necesaria como *input* para la mayoría de los programas que hacen SLAM y así poder determinar si era aceptable su uso para proseguir con las demás pruebas.

Para comenzar se debe caracterizar el robot, lo cual consiste en definir las compensaciones en términos de traslación y rotación entre diferentes marcos de coordenadas, como por ejemplo entre el robot y el escáner laser. Cada uno de estos marcos posee un nombre estandarizado en ROS, que define cada uno de estos componentes identificando al robot como *base_link*, la unidad de medición inercial como *imu_link*, el escaner laser como *laser_link*, la odometría como *odom* y el marco de referencia principal es *map*.

Luego de caracterizar el robot podemos ver como se relacionan los diferentes marcos de coordenadas en el árbol de transformación, el cual es publicado por la biblioteca de ROS llamada tf, como se ve en la figura 31.

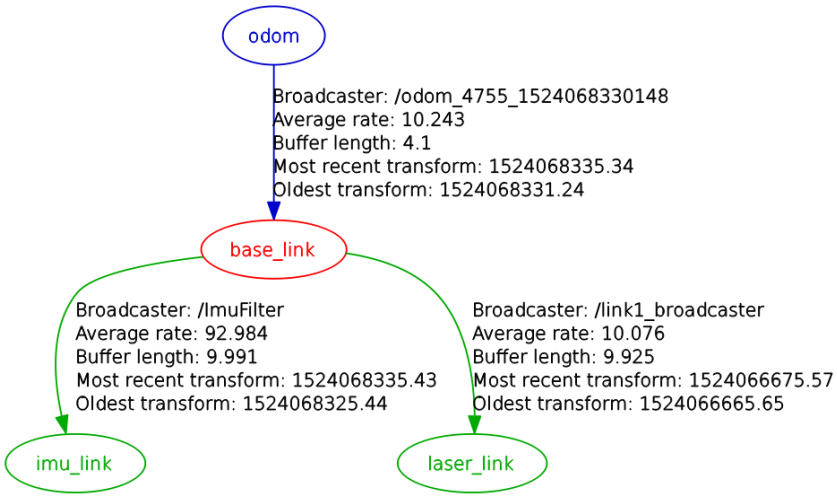


Fig. 31. Caracterización del robot en el árbol de transformaciones.

Ahora, lo siguiente es ejecutar los programas necesarios en ROS para realizar odometría. Los nodos ejecutados y su interacción (publicación - suscripción) para la odometría se pueden ver en la siguiente figura presentada por *rqt_graph*.

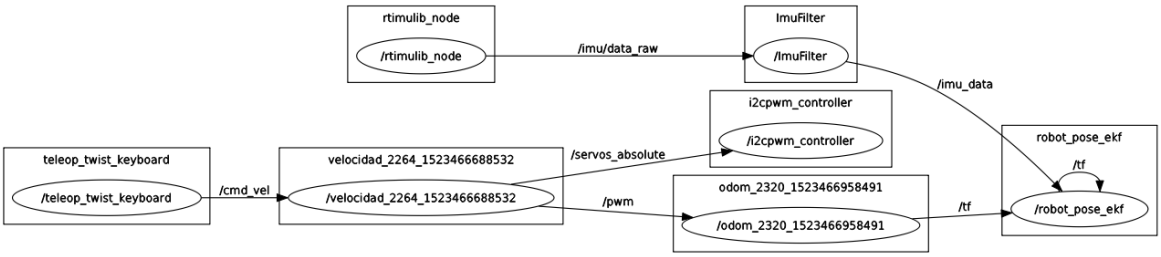


Fig. 32. Nodos necesarios para odometría y sus interacciones.

La visualización se realizó a través del programa en RVIZ, el cual muestra por medio de flechas el desplazamiento del robot y los diferentes marcos de coordenadas a medida que se mueve en círculos en el espacio.

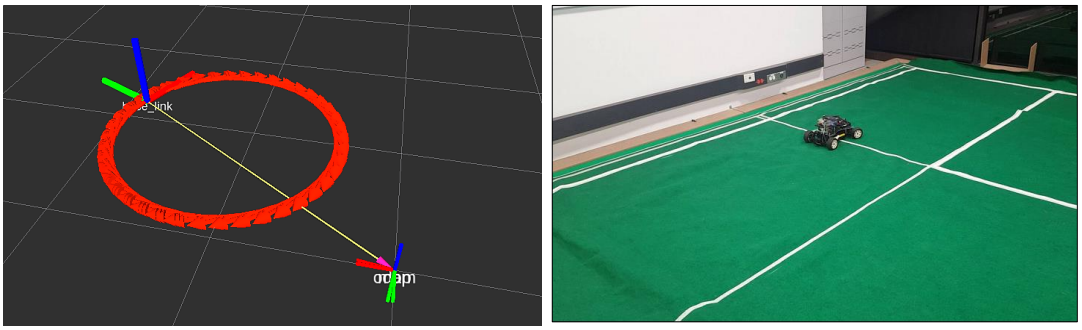


Fig. 33. Visualización de la odometría en RViz en el computador base (izquierda). Robot desplazándose en círculos para la prueba de funcionamiento del encoder virtual y la odometría (derecha).

Como resultado, se obtuvo una buena representación del movimiento del robot al observarse en RViz, pero que no correspondió totalmente a como se mueve el robot en realidad, primero debido a la precisión en la toma de datos para el desarrollo del encoder y segundo que al no poseer una retroalimentación para los cambios en la velocidad debido al terreno, voltaje de la batería, inercia o algún cambio en su peso llegaron a variar la medida que se realizó para cada prueba, además que como consecuencia de las características constructivas propias del vehículo usado, el robot presento desplazamientos que no fueron registrados y causaron un error muy significativo que se vio reflejado en su posición al momento de realizar SLAM.

Fig. 34.Ejemplo de los desplazamientos presentados por el robot al moverse en círculos.

Las causas de estos desplazamientos, presentes al moverse en línea recta o en círculos no son fáciles de corregir mecánicamente y sería necesario de la implementación de varios controles PID para mejorar este problema, sin embargo, se decidió poner a prueba los demás sistemas y realizar pruebas de SLAM.

5.2 SLAM

Para las pruebas del robot al cual se le doto con la capacidad de localización y mapeo simultáneos, se incorporó el marco de coordenadas *map*, el cual es el marco de referencia principal y sobre el cual se crea el mapa y se ubica el robot. Esto se puede observar en el árbol de transformaciones de la siguiente figura.

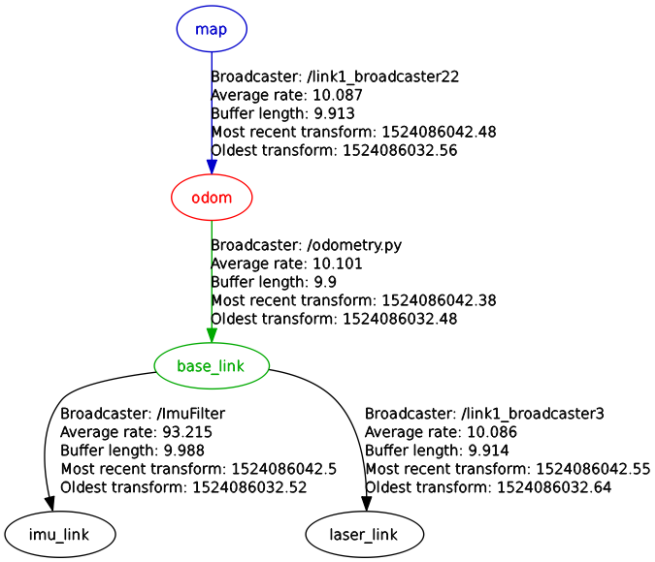


Fig. 35.Árbol de transformaciones final del robot para realizar SLAM.

Hay varios programas desarrollados para correr bajo ROS que realizan localización y mapeo simultáneos, tales como *Cartographer*, *Slam_karto*, *mrpt_ekf_slam_2d*, *Gmapping* y *Hector_slam*, pero solo estos dos últimos se pudieron instalar y ejecutar en el computador de placa reducida, los otros debido a que no funcionan bajo arquitecturas ARM, no se pudieron compilar o sus requerimientos (inputs) no corresponden al alcance del proyecto. A continuación, se muestran los resultados obtenidos de las pruebas del robot después de dotarlo con la capacidad de SLAM con los programas *Gmapping* y *Hector_Slam* y el uso de sensores IMU y LIDAR.

5.2.1 Gmapping

Tras instalar y compilar el paquete *gmapping*, se ejecutaron los nodos necesarios ya descritos como se ve en la figura 36, la cual muestra todos los nodos en operación, luego se posiciono el robot en las áreas de pruebas seleccionadas. El primer lugar de prueba fue el corredor junto al salón I1-204 de semilleros de electrónica del edificio I, que por la sencillez de su geometría y su cercanía con el router wifi proporciono un área óptima para realizar SLAM.

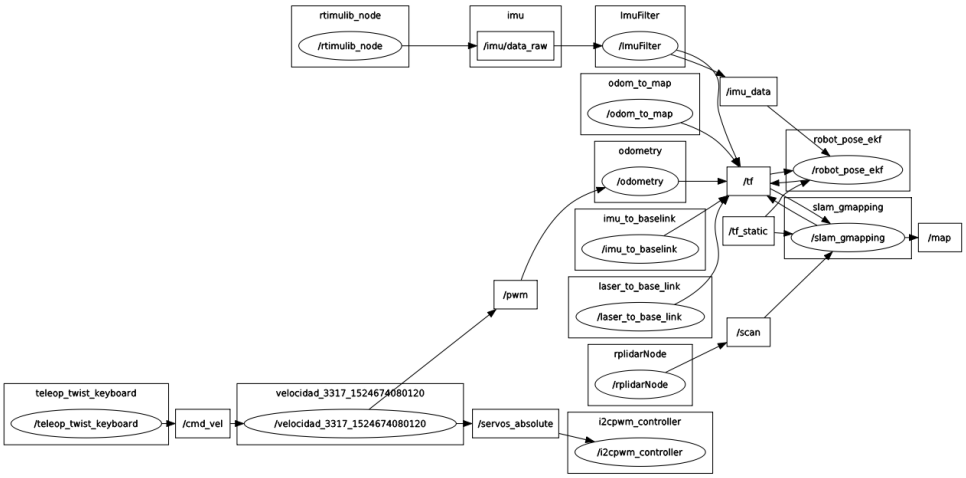


Fig. 36.Nodos en operación para ejecutar SLAM con gmapping.

El segundo lugar de pruebas fue en el salón de semilleros, donde se encontraba la cancha construida para robot futbol, el cual represento un espacio adecuado para el desplazamiento del robot móvil. Los resultados obtenidos por el robot y los hallazgos encontrados se describen a continuación.

5.2.1.1 Corredor.

Comenzando la prueba antes que el robot empiece a moverse como se ve en la figura 37 (a), se observó cómo se genera el mapa del lugar y se aprecian algunos detalles, como la entrada del salón de profesores a la izquierda, la pared a la derecha y un espacio no muy bien definido atrás correspondiente a la puerta de vidrio de la salida de emergencia del costado oriental del edificio. Esto efectivamente debido a que el haz de luz del láser no se refleja muy bien de vuelta hacia el sensor.

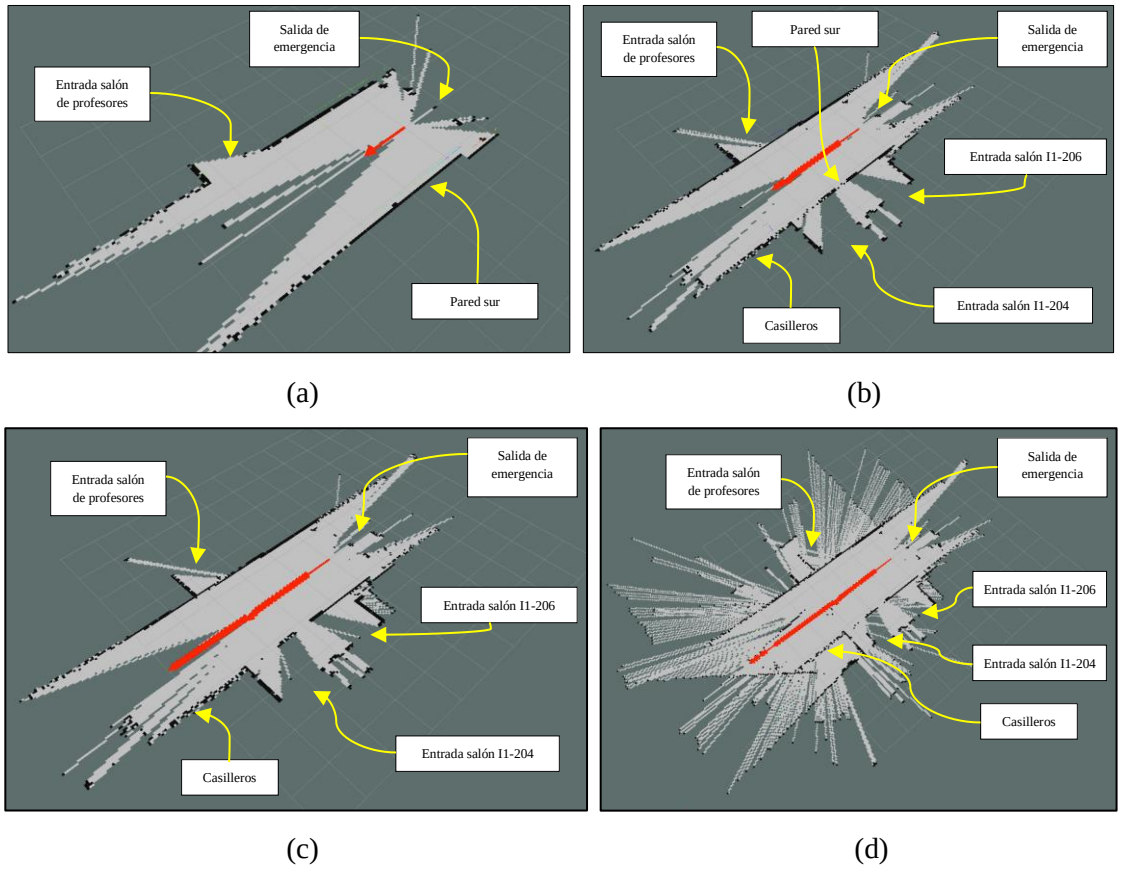


Fig. 37.Localización y mapeo simultáneos SLAM con el robot móvil y gmapping en el área del corredor.

En la figura 37 (b), se aprecia el desplazamiento del robot a través del corredor y como se fue ampliando el mapa en su recorrido, pudiéndose observar, el área de las entradas a los salones (I1-204 y I1-206) y los casilleros. Sin embargo, el mapa presenta un error porque las entradas de los salones se encuentran desplazadas hacia el oriente de su posición real y entrecruzadas con la pared del constado sur del corredor.

Este mapa se mantiene igual a medida que el robot se siguió desplazando en línea recta por el corredor, como se ve en la figura 37 (c), solo mostrando una pequeña corrección de estos mismos elementos hacia él occidente, pero se pierde totalmente la definición al llegar al espacio abierto finalizando los casilleros del costado sur, volviendo a mostrar un corrimiento pronunciado hacia el oriente, sobreponiendo los punto nuevos sobre el mapa ya generado, como se puede ver en la parte (c) de la figura 37.

5.2.1.2 Salón.

En el salón de semilleros se colocó una caja en la mitad de la cancha, con el fin de que sirviera como referencia al poner en marcha el robot móvil, realizando círculos alrededor de esta y así poder definir los espacios generados en el mapa. Al igual que con la prueba en el corredor, se posiciono el robot móvil al costado izquierdo de la caja como en se ve en la figura 38 (b), empezando a verse los detalles muy bien definidos del salón como la pared oriental, los casilleros al lado de los ventanales y el lado occidental de la caja.

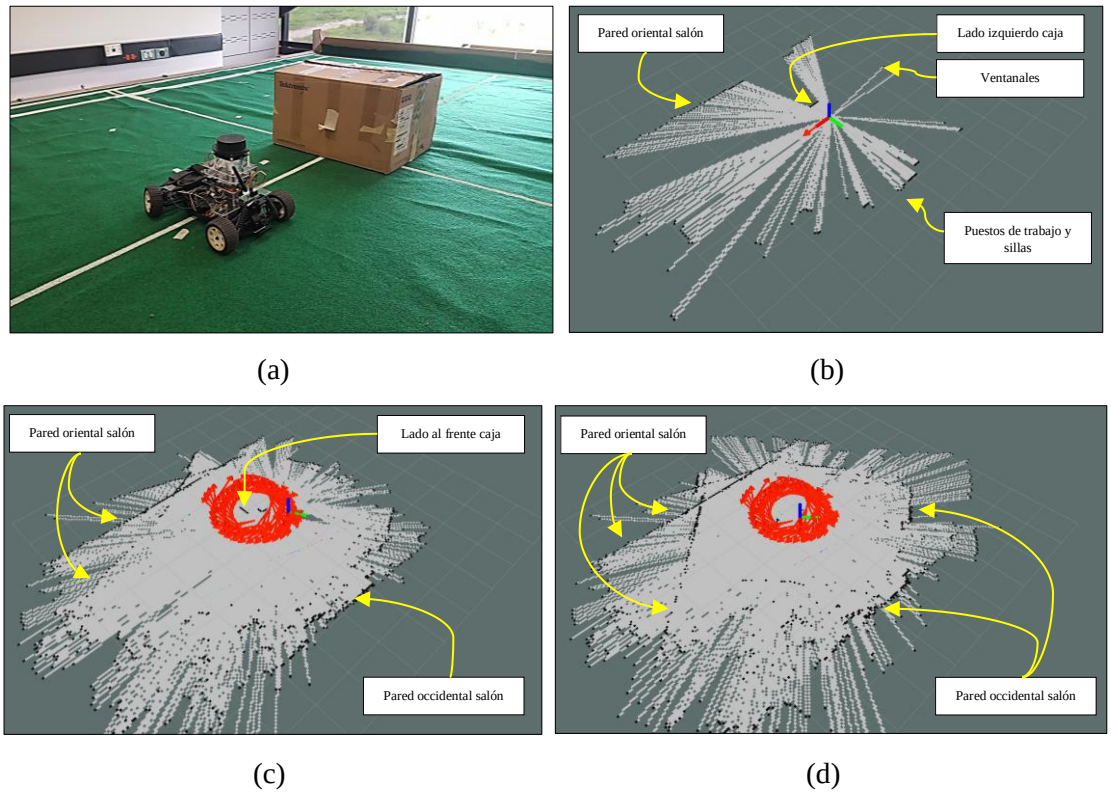
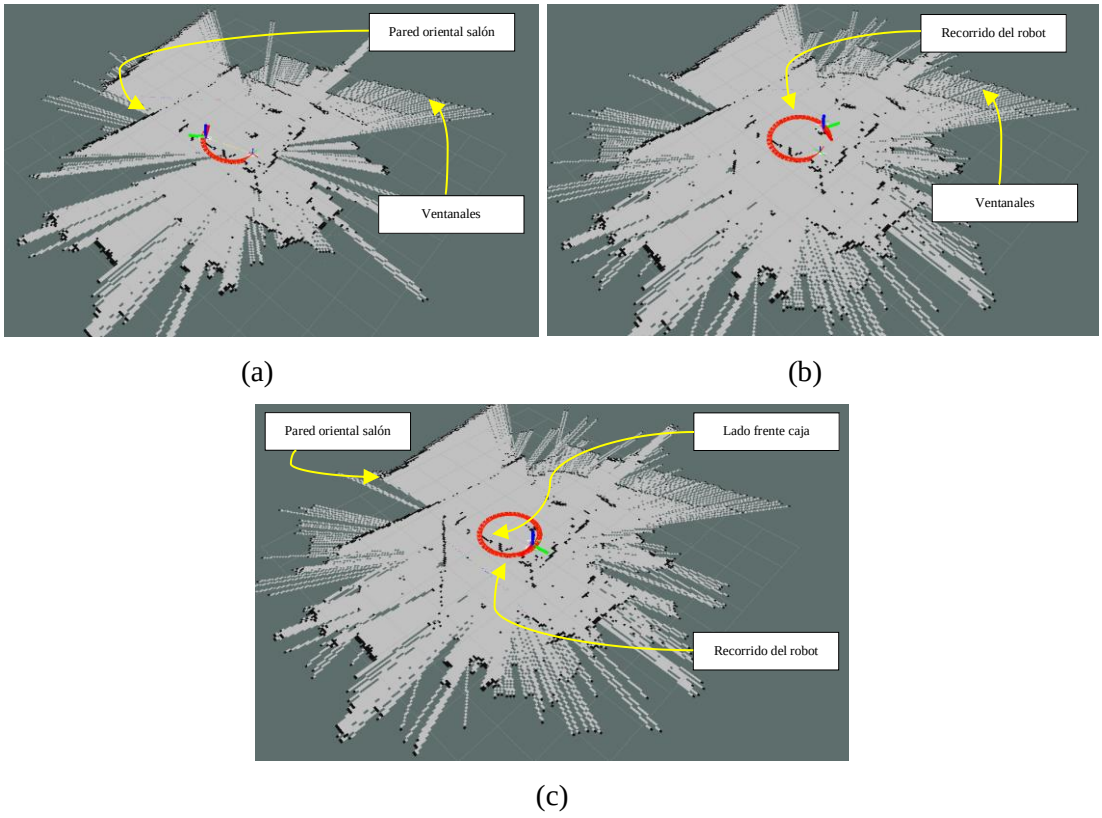


Fig. 38. Localización y mapeo simultáneos SLAM con el robot móvil y gmapping en el área del salón de semilleros.

Cuando el robot entra en marcha moviéndose en círculos, la unidad de medición registra los desplazamientos mostrados en la figura 34 del robot, pero al igual que con las pruebas realizadas en el corredor, el mapa empieza a mostrar un corrimiento girando a la derecha en sentido contrario al movimiento del robot móvil, causando que se muestren varios mapas generados sobrepuestos como se puede observar en la figura 38 (c). A medida que el robot sigue moviéndose estos errores siguen acumulándose uno sobre otro siendo difícil la identificación del área de prueba.

Estos errores de corrimiento en el mapa son generados por la inexactitud de los datos entregados por la odometría, derivados de la implementación del encoder virtual y por la acción de los datos entregados por la imu al filtro de Kalman (robot_pose_ekf), que trata de realizar una mejora en la estimación de la posición provocando el giro del mapa.



(a) (b)

(c)

Fig. 39. Localización y mapeo simultáneos SLAM con el robot móvil y gmapping en el área del salón de semilleros. Pruebas disminuyendo los efectos causados por el desplazamiento mecánico del robot.

Una forma de comprobar esto fue reduciendo los efectos de la IMU, cambiando el filtro en los datos en bruto para que no mostrara los desplazamientos debidos a la mecánica del robot, lo que se obtuvo fue una visualización del recorrido del robot únicamente generado por el encoder virtual mostrando un circulo de giro perfecto como se aprecia en la figura 39 (b) y los datos de la odometría generaron un mapa con un corrimiento menos notorio y desplazado a la izquierda como se ven la figura 39 (c).

5.2.2 Hector_mapping.

El paquete hector_slam nos proporciona el nodo *hector_mapping*, el cual tiene la capacidad de realizar SLAM sin la necesidad de odometría, sin embargo, utiliza las referencias de todos los marcos de coordenadas del árbol de transformaciones si son provistas por el robot móvil, como es el caso del proyecto. En la figura 40, se aprecia la interacción de los nodos utilizados en las pruebas con este programa.

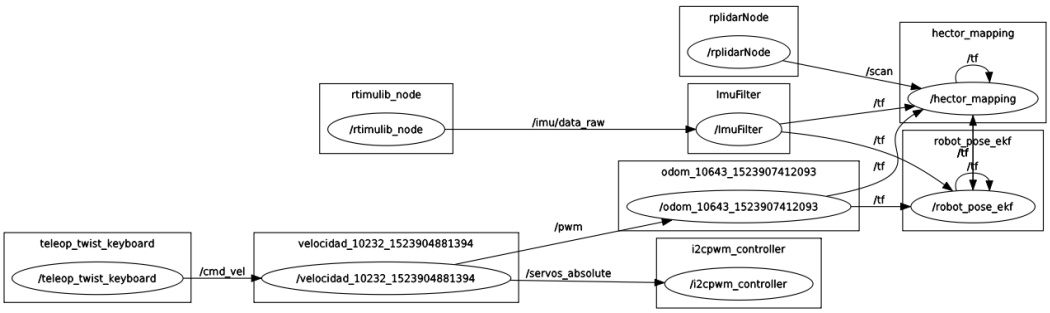


Fig. 40.Nodos en operación para ejecutar SLAM con hector_mapping.

Las pruebas se realizaron en los mismos espacios usados anteriormente con gmapping, lo que permite realizar una comparación de los resultados más adelante y sacar algunas conclusiones del proyecto.

5.2.2.1 Corredor.

En esta prueba se realizó el recorrido a través del corredor, desde la salida de emergencia hasta los casilleros y viceversa. Rápidamente se puedo apreciar en el mapa el corredor, la entrada al salón de profesores, la pared del costado sur y el mismo efecto del láser sobre las puertas de vidrio de la salida de emergencia y la flecha que representa la posición del robot móvil como se puede ver en la figura 41(a).

Al avanzar un poco se expande el espacio del mapa y aparecen los casilleros y el área correspondiente a las entradas de los salones I1_204 y I1-206, aunque esta no se ve muy bien definida en comparación con las realizadas con gmapping, las imágenes permiten identificar el aérea a la que corresponde, pero no se muestra de manera clara en el mapa como lo muestra la figura 41(b).

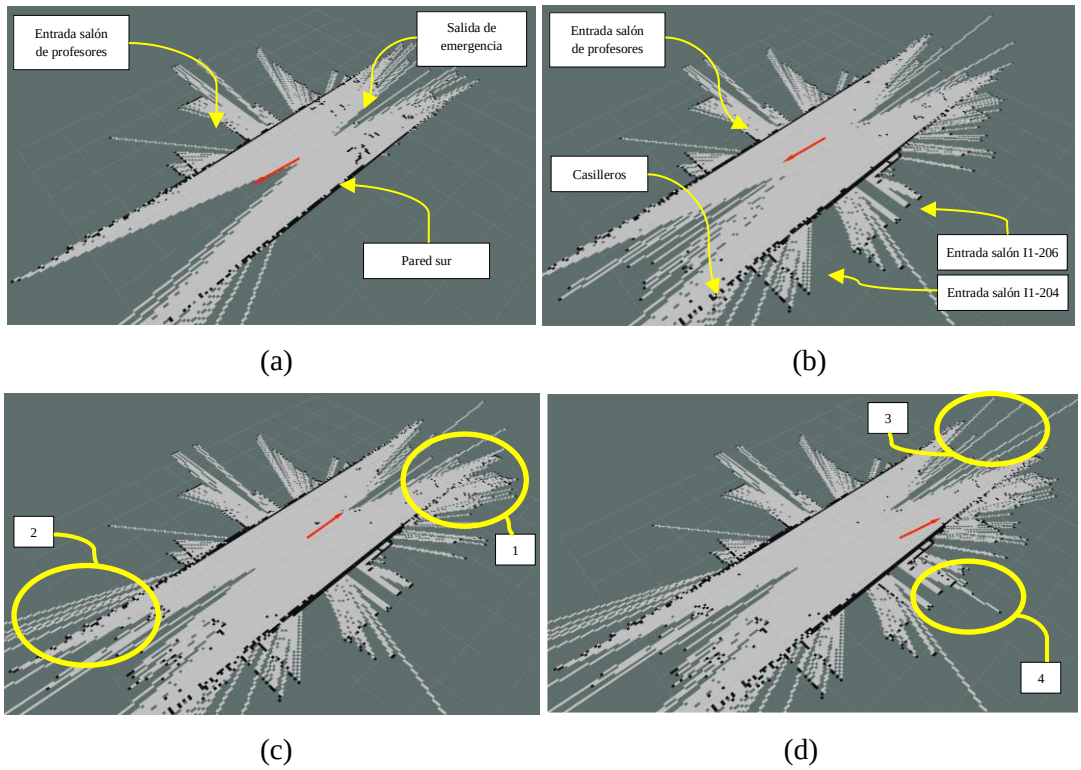
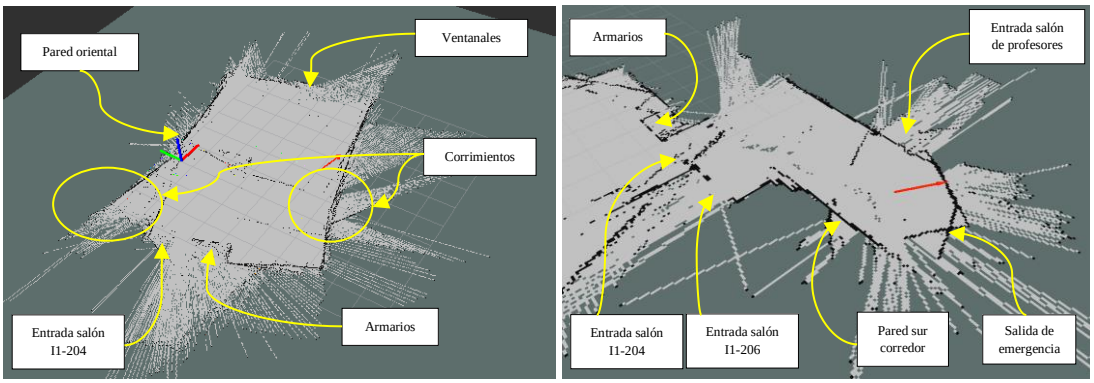


Fig. 41.Localización y mapeo simultáneos SLAM con el robot móvil y hector_mapping en el área del corredor.

Al estar de regreso; figura 41(c), no se ven cambios sustanciales en la generación del mapa, tan solo unas líneas adicionales fuera de él, como se muestra en los sitios señalados con círculos 1 y 2, pero no se ven corrimientos en el mapa como los presentados en las primeras pruebas. Lo mismo sucede al terminar el recorrido como se visualiza en la parte (d), el mapa se mantiene prácticamente sin cambios sin aportar más detalles en el recorrido del robot, solamente mostrando algunas lecturas espurias como lo demarcan los círculos 3 y 4.

5.2.2.2 Salón y corredor.

Las pruebas en el salón con la utilización del nodo *hector_mapping*, resultaron más satisfactorias en cuanto a identificación del área y construcción del mapa, ya que se lo representa de una manera más definida y con una disminución en el corrimiento de las zonas escaneadas. Estos resultados los podemos ver en la figura 42(a), donde se observa de forma más clara el perímetro del salón I1-204, mostrando más detalles como los armarios ubicados a la derecha de la entrada y los corrimientos identificados por los círculos amarillos.



(a) (b)

Fig. 42. Localización y mapeo simultáneos SLAM con el robot móvil y hector_mapping en el área del salón de semilleros y corredor contiguo.

Como se mencionó anteriormente, *hector_mapping* tiene la capacidad de realizar localización y mapeo simultáneos sin la necesidad de datos de odometría, sin embargo, estas pruebas fueron realizadas con la ejecución de los mismos nodos, que proporcionan datos al árbol de transformaciones las cuales impactan directamente en la generación del mapa y es debido a eso que se puede apreciar los mismos corrimientos, aunque menos notorios que los vistos en las pruebas con gmapping.

Al ver estos mejores resultados se realizó un ensayo que involucrara las dos áreas de pruebas, la cual dio como resultado la imagen (b) de la figura 41, en esta imagen de aprecia parte del salón I1-204 específicamente la zona norte donde se ve la entrada y el corredor por donde se desplazó el robot móvil en dirección de las puertas de salida de emergencia. Aquí se obtienen resultados similares con algunos puntos falsos y con corrimiento de algunas aéreas del mapa, pero sin tener datos sobrepuestos con los ya obtenidos por el avance del robot móvil.

5.3 Contribución con la realización del proyecto

Este trabajo de grado pretende ser un punto de partida para cualquier investigación adicional y se puede usar como banco de pruebas para la investigación y comparación de nuevas técnicas o que complementen la validación del desempeño de los dos algoritmos que se usan en este proyecto. Además, permite la realización de trabajos de mayor complejidad que involucren el uso de robots móviles, apropiando la tecnología y el conocimiento plasmado en este documento incluyendo las ventajas y falencias en sobre localización y mapeo simultáneos, el uso de ROS y sensores especializados.

Capítulo 6

6. Conclusiones

Para el proyecto se realizó un conjunto de dos experimentos en cada uno de los entornos, el salón I1-204 de semilleros de electrónica y el corredor contiguo a este, en cada una de las pruebas el robot móvil fue controlado manualmente desplazándolo de manera diferente, con el objetivo de replicar una operación de SLAM automática en un entorno estructurado desconocido, donde un robot normalmente no puede seguir un estilo específico de movimiento debido a la presencia de obstáculos o una estructura de construcción no uniforme.
En los experimentos de SLAM los parámetros utilizados para los algoritmos no fueron modificados de los valores por defecto, esto para permitir el análisis de comparación entre los dos procesos de localización y mapeo simultáneos. Con las pruebas efectuadas en el corredor se puede ver que los algoritmos de *Gmapping* y *Hector_slam* mapean con éxito el área con una precisión razonable, antes que el robot móvil comience a avanzar, es aquí cuando vemos cambios muy notorios espacialmente en el generado por *Gmppaing* que muestra algunas desalineaciones o sobreposiciones debido a errores de la odometría acumulando las estimaciones a lo largo del tiempo, el algoritmo SLAM no pudo escanear coincidencias correctamente y como resultado determinó erróneamente la pose del robot durante el tiempo de ejecución causando estas perturbaciones en el mapa.

En el salón las pruebas fueron más reveladoras, mostrando un mejor desempeño del algoritmo del programa Hector_slam que, aunque mostro una serie de puntos erróneos y desalineaciones en ciertos puntos del mapa, este resulto más acorde con la realidad del área escaneada incluso cuando se propuso ir más allá y escanear el salón junto el corredor. Esto mostro de nuevo el gran impacto de la odometría en el desarrollo del SLAM y en las consecuencias que puede traer la utilización de un encoder o técnica de odometría deficiente.

Es claro con estos resultados que la utilización del encoder virtual o simulado por código de programación, no es la mejor opción al momento de realizar SLAM, pero esta opción no fue la primera que se pensó en utilizar en el proyecto. La primera opción fue la utilización de una cámara web con el fin de desarrollar odometría visual, una técnica ampliamente probada y con buenos resultados en varios experimentos publicados, pero no fue posible su implementación debido al

conflicto que se presenta en el software al momento de realizar la compilación bajo ROS de los procesadores con arquitectura ARM, como es el caso del computador de placa reducida usado en el proyecto la Firefly RK-3288 y su procesador Cortex-A17. En general este inconveniente se presenta en muchos otros programas dispuestos para ejecutarse bajo ROS, ya que su desarrollo se ha realizado para su operación en computadores personales y laptops con procesadores Intel y AMD, lo que representa una limitación al desarrollar este y otros múltiples proyectos sobre placas SBC.

Otra opción para la ejecución de la odometría fue su realización mediante los datos de aceleración lineal y velocidad angular suministrados por la unidad de medición inercial MPU-9250, procurando con estos la implementación de un controlador PID que pudiera mantener velocidades constantes mediante la realimentación de los valores leídos, y calcular medidas de desplazamiento con la doble integración de la aceleración. Los resultados de estas pruebas no fueron satisfactorios, debido a la acumulación del error en el tiempo, donde pequeños errores en la medición de la aceleración y la velocidad angular se integran en errores progresivamente mayores en la velocidad, que se combinan en errores aún mayores en la posición, siendo sumamente difícil su utilización para la odometría del robot móvil.

Es significativo mencionar además de las anteriores conclusiones la importancia seleccionar adecuadamente el tipo de vehículo junto con sus características antes de dotarlo con la capacidad de localización y mapeo simultáneos, puesto que esto representa múltiples condiciones que pueden llegar a limitar los alcances de un proyecto, como es el caso del usado en este trabajo de grado que, debido a varios de esos factores, resultaron influyendo directamente en los resultados obtenidos. De esos factores la velocidad represento un gran inconveniente, debido al peso del montaje fue necesario aumentar los valores de mínimos PWM para entrar en marcha, lo que hizo que el robot móvil entrara en funcionamiento con velocidades altas que dificultaron su maniobrabilidad, lo mismo sucede con su tamaño y partes mecánicas que dificultan el uso locaciones de prueba más pequeños e impidieron mantener estabilidad en el rumbo de los recorridos.

Para finalizar es necesario referirse al limitado alcance de la señal del router WIFI, como se dijo en el párrafo anterior las características del robot móvil requieren de espacios de mapeo más extensos, lo que limito los experimentos a áreas cercanas al router, no solo por el control del robot móvil sino también por la necesidad de la creación del mapa en el computador base a medida que este se fuera desplazando.

7. Anexos

A1. Detección de IMU:

Instalar el paquete i2c-tools. Primero se recomienda realizar una actualización general, antes de realizar los pasos que se describen a continuación:

1. Abrir un nuevo terminal (CTRL+ALT+T)
2. Ejecutar el comando:

```
$ sudo apt-get update
```

3. Ejecutar el comando:

```
$ sudo apt-get upgrade
```

4. Ejecutar para instalar las herramientas y repetir los pasos 2 y 3:

```
$ sudo apt-get install i2c-tools
```

5. Reiniciar
6. Buscar el dispositivo conectado y ver su dirección:

```
$ sudo i2cdetect -y -r 1
```

Para usar I2C con Python se debe ejecutar el siguiente comando:

```
$ sudo apt-get install python-smbus
```

A2. Instalación de MPU-9250:

La idea es instalar la MPU-9250 bajo ROS en la Firefly RK-3288. Para esto usaremos RTIMULib (de richards-tech LLC en Github) como la librería de interfaz y rtimulib_ros (de Romain Reignier) como interfaz para ROS. Con RTIMULib y rtimulib_ros obtenemos los datos de los sensores fusionados lo que es una facilidad para la integración. Antes de comenzar con la instalación cabe mencionar que es necesario que ya se tenga instalado ROS y haber creado un espacio de trabajo *catkin*.

Instalación de RTIMULib

1. Primero instalar cmake, que se utilizará más adelante para editar el archivo cmake. Ejecutar el comando:

```
$ sudo apt-get install cmake-curses-gui
```

2. Descargar RTIMULib:

```
$ git clone https://github.com/jetsonhacks/RTIMULib.git (Jetsonhacks, 2018)
```

3. De forma predeterminada, los dispositivos I^2C son propiedad del root. Para cambiar esto, cree un archivo /etc/udev/rules.d/90-i2c.rules y agregue la línea:

```
$ sudo gedit /etc/udev/rules.d/90-i2c.rules
```

KERNEL==”i2c-[0-7]”,MODE=”0666″

También como alternativa, se puede hacer directamente con la línea de comandos:

```
$ sudo bash -c 'echo KERNEL==\"i2c-[0-7]\",MODE=\"0666\" > /etc/udev/rules.d/90-i2c.rules'
```

4. Preparar la compilación, abriendo la carpeta descargada:

```
$ cd RTIMULib
```

5. Cambiar a la carpeta Linux y ejecutar las siguientes líneas de comandos:

```
$ cd Linux

$ mkdir build

$ cd build

$ cmake ..
```

6. Esto genera el archivo cmake. Los parámetros se pueden cambiar ahora para que Qt no sea necesario y ejecutamos:

```
$ ccmake ..
```

7. Cambiar las opciones relacionadas con 'GL' y 'DRIVE11' a off, luego 'c' para configurar y luego 'g' para generar y salir.

Fig. A2.43. Cambio de en la configuración de RTIMULib.

8. Ahora seguir con las siguientes instrucciones:

```
$ cmake ..

$ make -j4

$ sudo make install
```

```
$ sudo ldconfig
```

Después realizar un reinicio del equipo.

Instalación de rtimulib_ros:

1. Para la instalación se supone que hay un espacio de trabajo con el nombre "Catkin_ws" o de lo contrario debe ser reemplazado por el nombre asignado al área de trabajo catkin. Abrir un terminal nuevo y ejecutar:

```
$ cd ~/catkin_ws/src

$ source devel/setup.bash

$ git clone https://github.com/jetsonhacks/rtimulib_ros.git
```

Configuración:

Para el desarrollo de este proyecto se cambia el tema (topic) de ROS por "/imu/data_raw" ya que en el archivo de origen aparece como "imu/data".

1. Abrir la carpeta rtimulib_ros

```
$ cd rtimulib_ros/src
```

2. Abrir el archivo rtimulib_ros.cpp con el editor

```
$ gedit rtimulib_ros.cpp
```

Y cambiamos la línea

// ros::Publisher imu_pub = n.advertise(“imu/data”, 1); por
ros::Publisher imu_pub = n.advertise(“/imu/data_raw”, 1);

Guardar y construir el nodo:

3. Subir una carpeta

```
$ cd ..
```

4. Subir una carpeta más en el arbol

```
$ cd ..   (sobre la carpeta catkin_ws)
```

5. Ejecutar catkin_make para compilar

```
$ catkin_make
```

Calibración:

Generalmente se proporciona un medio para calibrar el sensor. La calibración habitualmente significa mover la IMU alrededor de cada eje para alcanzar los valores mínimo y máximo del sensor y colocar la información en un archivo de calibración. Para realizar la calibración de la IMU se deben seguir los siguientes pasos:

1. Abrir un nuevo terminal (CTRL+ALT+T) y actualizar:

```
$ sudo apt-get update

$ sudo apt-get upgrade
```

2. Ejecutar:

```
$ sudo apt-get install build-essential qt5-default qtcreator -y
```

3. Una vez que Qt Creator está instalado, es necesario hacer una configuración más. Abrir Qt Creator e ir a:
   Tools->Options

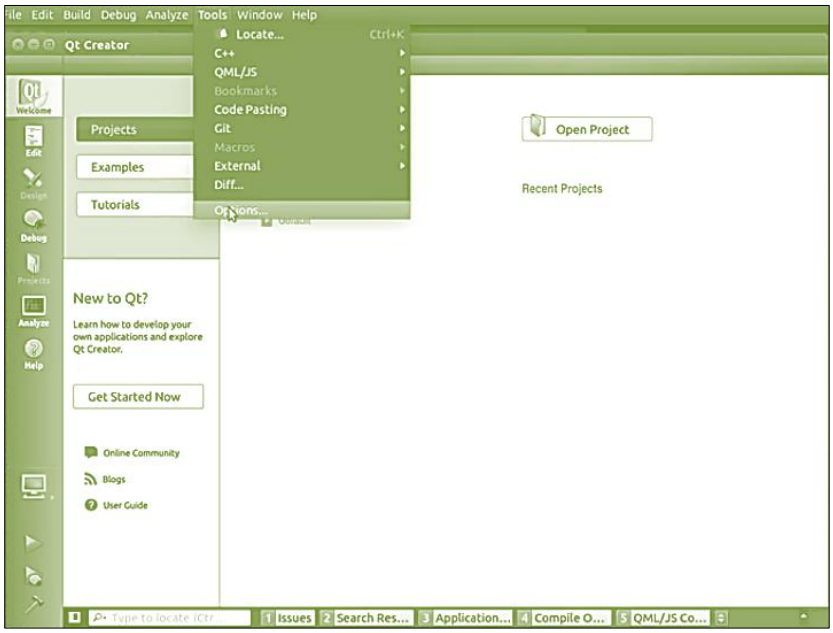


Fig. A2.44. Cambio de en la configuración de QTcreator.

Build & Run->Compilers

Dar clic en el botón "add" y seleccionar "GCC". En el cuadro de texto "Compiler path:", colocar la ruta al compilador gcc. En una instalación estándar, la ruta es: /usr/bin/gcc.

Fig. A2.45. Cambio de ruta para el compilador GCC.

Comprobar la configuración de "Kits", disponible en la pestaña "Kits". Al seleccionar el kit de escritorio (predeterminado), se debería ver en el recuadro – Compiler: GCC. El nombre GCC debe coincidir con el compilador que aparece en la pestaña anterior.

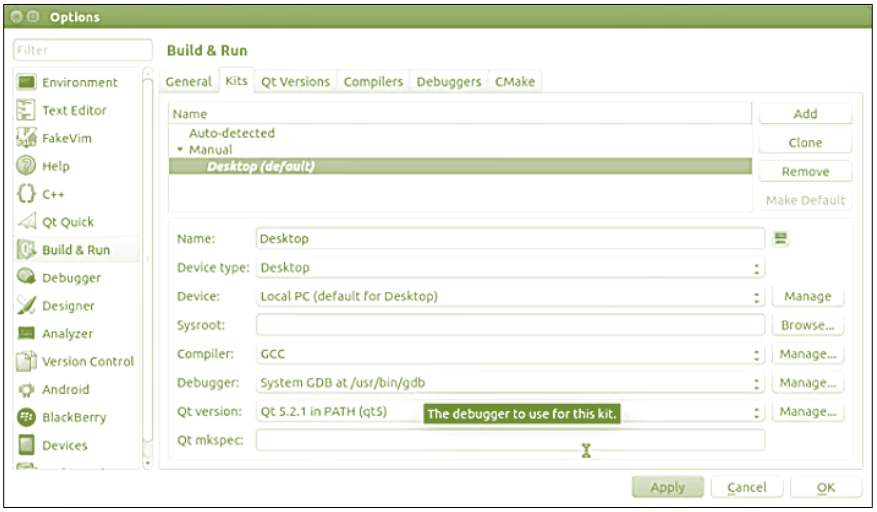


Fig. A2.46. Comprobación del compilador GCC.

Aplicar y salir, quedando lista la instalación. Se recomienda reiniciar el sistema después de la instalación.

4. Ir al directorio RTIMULib/Linux/RTIMULibDemoGL. Allí encontrará el archivo RTIMULibDemoGL.pro.
5. Ejecutar el archivo con Qt Creator haciendo click derecho y escoger a la aplicación. Dar click en ‘Configure Project´.

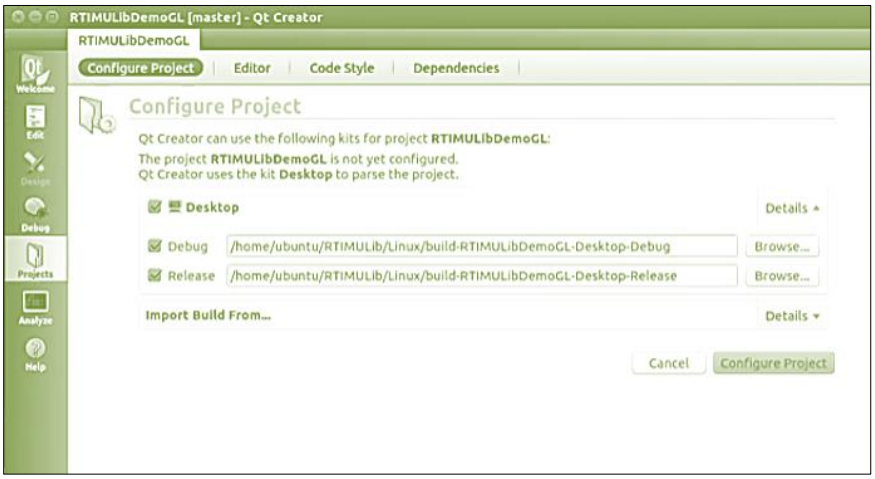


Fig. A2.47. Ruta para compilación de RTIMULibDemoGL.

6. Ir a 'Build' y dar click en 'Build All'.

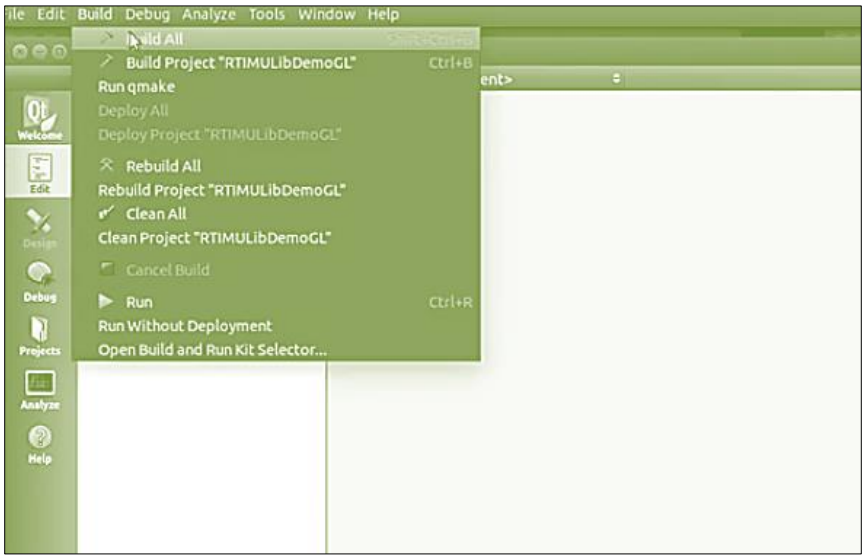


Fig. A2.48.Compilación de RTIMULibDemoGL.

7. Tener en cuenta que para acceder a una IMU en I^2C, el programa debe ejecutarse como superusuario 'sudo', en otras palabras, desde una ventana terminal.
8. Ejecutar el programa click en ‘Debug’, ‘Start Debugging’

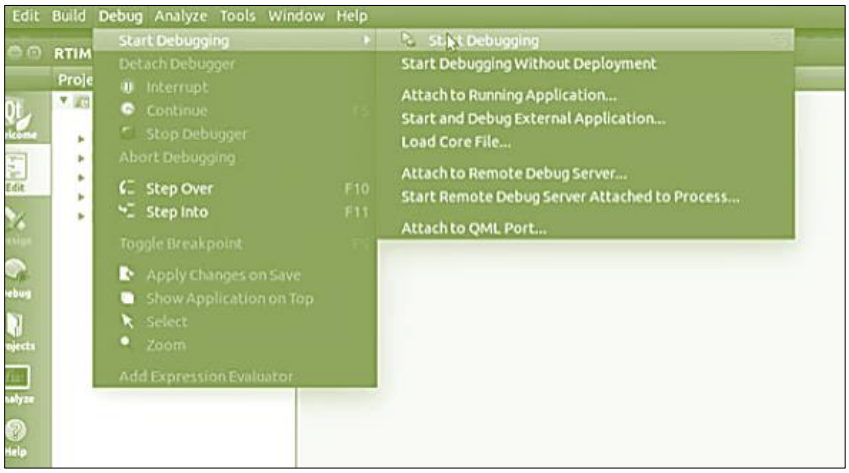


Fig. A2.49. Ejecución de RTIMULibDemoGL.

9. Ahora que ya está instalado cerrar todo, abrir una nueva ventana terminal y ejecute:

```
$ cd '/home/ubuntu/RTIMULib/Linux/build-RTIMULibDemoGL-Desktop-Debug/Output'
```

Luego,

```
$ sudo ./RTIMULibDemoGL
```

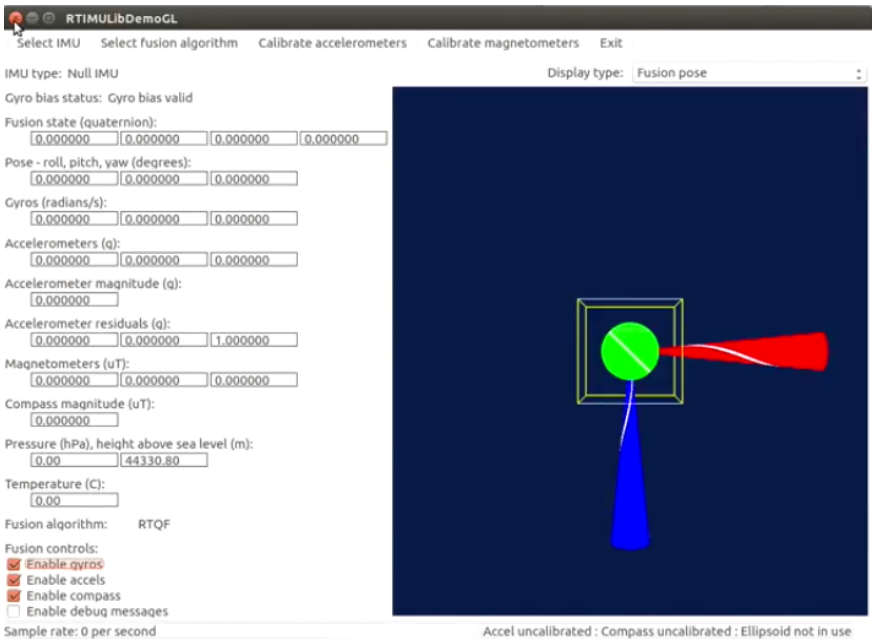


Fig. A2.50. RTIMULibDemoGL.

10. Dar click en ‘Select IMU’ y escoger el modelo y dirección de la IMU.
11. Dar click en ‘Calibrate accelerometers’, seleccionar únicamente el eje X y enseguida mover la IMU en dirección de este eje para alcanzar los valores máximos y mínimos. Repetir el procedimiento con los ejes Y y Z, presionar OK.
12. Dar click en ‘Calibrate magnetometers’ y repetir el procedimiento del paso 11,

A3. Detección de PCA9685:

De la misma forma que se realizó previamente con la IMU es necesario (si no se ha hecho anteriormente) instalar el paquete i2c-tools. Primero se recomienda realizar una actualización general, antes de realizar los pasos que se describen a continuación:

1. Abrir un nuevo terminal (CTRL+ALT+T)
2. Ejecutar el comando:

```
$ sudo apt-get update
```

3. Ejecutar el comando:

```
$ sudo apt-get upgrade
```

4. Ejecutar para instalar las herramientas y repetir los pasos 2 y 3:

```
$ sudo apt-get install i2c-tools

$ sudo apt-get install libi2c-dev i2c-tools
```

5. Reiniciar
6. Buscar el dispositivo conectado y ver su dirección:

```
$ sudo i2cdetect -y -r 1
```

Instalación:

Para controlar el driver PCA-9685 usaremos ros-i2cpwmboard (de Bradan Lane Studio en GitLab) (Studio, s.f.) como interfaz para ROS. El nodo ros-i2cpwmboard publica en tres temas (*topics*) diferentes de acuerdo con la aplicación que se desarrolle, para el proyecto utilizaremos para publicar los diferentes valores de PWM el tema *servos_absolute()*. Antes de comenzar con la instalación cabe mencionar que es necesario que ya se tenga instalado ROS y haber creado un espacio de trabajo *catkin*.

Instalación de ros-i2cpwmboard

1. Primero abrimos el directorio del espacio de trabajo (*catkin_ws*) y la carpeta src, en esta ubicación creamos una nueva carpeta con el nombre i2cpwm_board.
2. Descargamos de la página https://gitlab.com/bradanlane/ros-i2cpwmboard (Studio, s.f.) los archivos, los cuales se descargan en formato .zip.
3. Descomprimimos el archivo y cambiamos el nombre a la carpeta que originalmente se llama ros-i2cpwmboard-master por ros-i2cpwmboard, esto es de manera opcional para reducir un poco el nombre del archivo.
4. Copiamos la totalidad de la carpeta en la carpeta src.
5. Abrimos una nueva ventana terminal (Ctrl+Alt+t).
6. Ejecutamos el siguiente comando

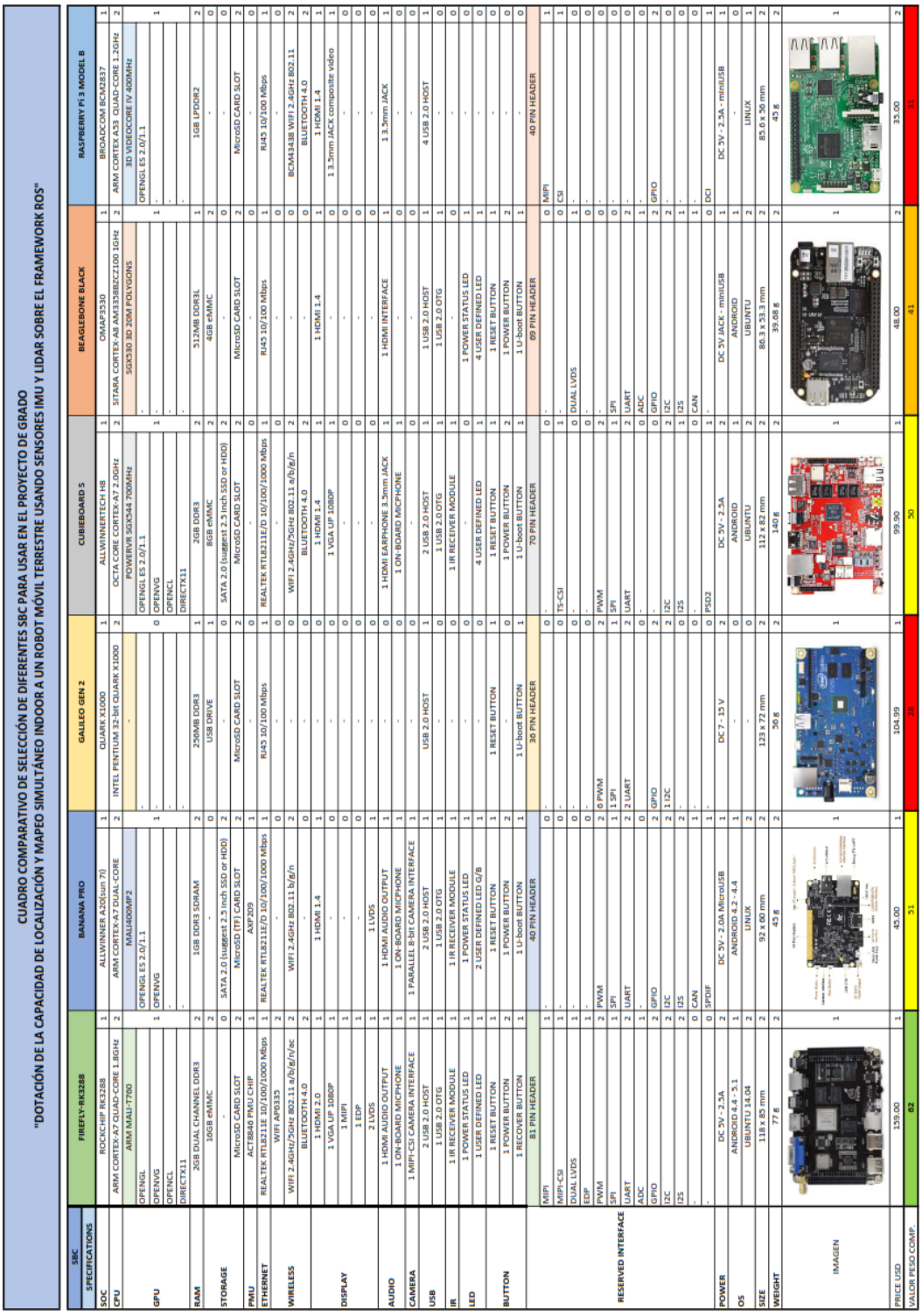
```
$ roscd catkin_ws
```

7. Y luego compilamos sobre la carpeta catkin_ws

```
$ ~/catkin_ws# catkin_make
```

A4. Selección de SBC:

Cuadro comparativo



CUADRO COMPARATIVO DE SELECCIÓN DE DIFERENTES SBC PARA USAR EN EL PROYECTO DE GRADO
"DOTACIÓN DE LA CAPACIDAD DE LOCALIZACIÓN Y MAPEO SIMULTÁNEO INDOOR A UN ROBOT MÓVIL TERRESTRE USANDO SENSORES IMU Y LIDAR SOBRE EL FRAMEWORK ROS"

SBC	FIREFLY-RK3288		BANANA PRO		GALILEO GEN 2		CUBIEBOARD 5		BEAGLEBONE BLACK		RASPBERRY PI 3 MODEL B	
SPECIFICATIONS	ROCKCHIP RK3288	1	ALLWINNER A20(sun 7i)	1	QUARK X1000	1	ALLWINNERTECH H8	1	OMAP3530	1	BROADCOM BCM2837	1
SOC	ARM CORTEX-A7 QUAD-CORE 1.8GHz	2	ARM CORTEX-A7 DUAL-CORE	2	INTEL PENTIUM 32-bit QUARK X1000	2	OCTA CORE CORTEX-A7 2.0GHz	2	SITARA CORTEX-A8 AM3358ZCZ100 1GHz	2	ARM CORTEX A53 QUAD-CORE 1.2GHz	2
CPU	ARM MALI-T760		MALI400MP2		-		POWERVR SGX544 700MHz		SGX530 3D 20M POLYGONS		3D VIDEOCORE IV 400MHz	
GPU	OPENGL	-	OPENGL ES 2.0/ 1.1	-	-	-	OPENGL ES 2.0/1.1	-	-	-	OPENGL ES 2.0/1.1	-
	OPENVG	1	OPENVG	1	-	0	OPENVG	1	-	1	-	1
	OPENCL	-	-	-	-		OPENCL	-	-	-	-	
	DIRECTX11	-	-	-	-		DIRECTX11	-	-	-	-	
RAM	2GB DUAL CHANNEL DDR3	2	1GB DDR3 SDRAM	2	256MB DDR3	1	2GB DDR3	2	512MB DDR3L	1	1GB LPDDR2	2
STORAGE	10GB eMMC	2	-	0	USB DRIVE	1	8GB eMMC	2	4GB eMMC	2	-	0
	-	0	SATA 2.0 (suigest 2.5 inch SSD or HDD)	2	-	0	SATA 2.0 (suigest 2.5 inch SSD or HDD)	2	-	0	-	0
	MicroSD CARD SLOT	2	MicroSD CARD SLOT	2	MicroSD CARD SLOT	2	MicroSD CARD SLOT	2	MicroSD CARD SLOT	2	MicroSD CARD SLOT	2
PMU	AXP 309	1	-	0	-	0	-	0	-	0	-	0
ETHERNET	REALTEK RTL8211E/D 10/100/1000 Mbps	1	REALTEK RTL8211E/D 10/100/1000 Mbps	1	RJ45 10/100 Mbps	1	REALTEK RTL8211E/D 10/100/1000 Mbps	1	RJ45 10/100 Mbps	1	RJ45 10/100 Mbps	1
WIRELESS	WIFI 2.4GHz/5GHz 802.11 b/g/n/ac	2	WIFI AP6335	0	-	0	-	0	-	0	-	0
	BLUETOOTH 4.0	2	-	0	-	0	WIFI 2.4GHz/5GHz 802.11 a/b/g/n	2	-	0	BCM43438 WIFI 2.4GHz 802.11	2
	1 HDMI 2.0	1	1 HDMI 1.4	1	-	0	BLUETOOTH 4.0	2	-	0	BLUETOOTH 4.0	2
DISPLAY	1 VGA UP 1080P	1	-	0	-	0	1 HDMI 1.4	1	1 HDMI 1.4	1	1 HDMI 1.4	1
	1 MIPI	1	-	0	-	0	1 VGA UP 1080P	1	-	0	1 3.5mm JACK composite video	1
	1 EDP	1	-	0	-	0	-	0	-	0	-	0
	2 LVDS	1	-	0	-	0	-	0	-	0	-	0
AUDIO	1 HDMI AUDIO OUTPUT	1	1 HDMI AUDIO OUTPUT	1	-	0	1 HDMI EARPHONE 3.5mm JACK	1	1 HDMI INTERFACE	1	1 3.5mm JACK	1
	1 ON-BOARD MICPHONE	1	1 ON-BOARD MICPHONE	1	-	0	1 ON-BOARD MICPHONE	1	-	0	-	0
CAMERA	1 MIPI-CSI CAMERA INTERFACE	1	1 PARALLEL 8-bit CAMERA INTERFACE	1	-	0	-	0	-	0	-	0
USB	2 USB 2.0 HOST	1	2 USB 2.0 HOST	1	USB 2.0 HOST	1	2 USB 2.0 HOST	1	1 USB 2.0 HOST	1	4 USB 2.0 HOST	1
	1 USB 2.0 OTG	1	1 USB 2.0 OTG	1	-	0	1 USB 2.0 OTG	1	1 USB 2.0 OTG	1	-	0
IR	1 IR RECEIVER MODULE	1	1 IR RECEIVER MODULE	1	-	0	1 IR RECEIVER MODULE	1	-	0	-	0
LED	1 POWER STATUS LED	1	1 POWER STATUS LED	1	-	0	-	0	1 POWER STATUS LED	1	-	0
	1 USER DEFINED LED	1	2 USER DEFINED LED G/B	1	-	0	4 USER DEFINED LED	1	4 USER DEFINED LED	1	-	0
BUTTON	1 RESET BUTTON	1	1 RESET BUTTON	1	1 RESET BUTTON	1	1 RESET BUTTON	1	1 RESET BUTTON	1	-	0
	1 POWER BUTTON	2	1 POWER BUTTON	0	-	0	1 POWER BUTTON	2	1 POWER BUTTON	2	-	0
	1 RECOVER BUTTON	1	1 U-boot BUTTON	1	1 U-boot BUTTON	1	1 U-boot BUTTON	1	1 U-boot BUTTON	1	-	0
	81 PIN HEADER		40 PIN HEADER		36 PIN HEADER		70 PIN HEADER		69 PIN HEADER		40 PIN HEADER	
RESERVED INTERFACE	MIPI	1 -		0 -		0 -		0 -		0 MIPI		1
	MIPI-CSI	1 -		0 -		0 TS-CSI		1 -		0 CSI		1
	DUAL LVDS	1 -		0 -		0 -		0 DUAL LVDS		1 -		0
	EDP	1 -		0 -		0 -		0 -		0 -		0
	PWM	2 PWM		2 PWM		2 PWM		2 -		0 -		0
	SPI	1 SPI		1 SPI		1 SPI		1 SPI		0 -		0
	UART	2 UART		2 UART		2 UART		2 UART		2 -		0
	ADC	1 -		0 -		0 -		0 ADC		1 -		0
	GPIO	2 GPIO		2 GPIO		2 -		0 GPIO		1 GPIO		2
	I2C	2 I2C		2 I2C		2 I2C		2 I2C		2 -		0
	I2S	2 I2S		2 -		0 I2S		1 I2S		1 -		0
	-	0 CAN		1 -		0 -		0 CAN		1 -		0
	-	0 SPDIF		1 -		0 PSD2		1 -		0 DCI		1
POWER	DC 5V - 2.5A	2	DC 5V - 2.0A MicroUSB	1	DC 7 - 15 V	2	DC 5V - 2.5A	2	DC 5V JACK - miniUSB	1	DC 5V - 2.5A - miniUSB	1
OS	ANDROID 4.4 - 5.1	1	ANDROID 4.2 - 4.4	1	-	0	ANDROID	1	ANDROID	1	-	0
	UBUNTU 14.04	2	LINUX	2	-	0	UBUNTU	2	UBUNTU	2	LINUX	1
SIZE	118 x 85 mm	2	92 x 60 mm	2	123 x 72 mm	2	112 x 82 mm	2	86.3 x 53.3 mm	2	85.6 x 56 mm	2
WEIGHT	77 g	2	45 g	2	56 g	2	140 g	2	39.68 g	2	45 g	2
IMAGEN		1		1		1		1		1		1
PRICE USD	159.00	1	45.00	1	104.99	1	99.90	1	48.00	2	35.00	2
VALOR PESO COMP.	62		51		38		50		41		11	

A5. Codigo en Python del nodo Velocidad

```
#!/usr/bin/env python
# license removed for brevity
import message_filters
import rospy
from std_msgs.msg import Float64
from i2cpwm_board.msg import ServoArray
from i2cpwm_board.msg import Servo
from geometry_msgs.msg import Twist
from sensor_msgs.msg import Imu


x1=0.0
x2=0.0
x3=0.0

def callback(data1):
    global x1, x2, x3
    vx=data1.linear.x
    vz=data1.linear.z
    wz=data1.angular.z
    pub = rospy.Publisher('servos_absolute', ServoArray
,queue_size=10)
    r=rospy.Rate(10)              #10Hz
    if vx == 0:
      x1 = (350)             #El 350 es la velocidad 0 o de parada
         srv=Servo(1, x1)
         msg=ServoArray({srv})
         pub.publish(msg)
         r.sleep()
    else:
      x1 = (365)+round(vx)
      if x1 >= 400:
          x1 = 400
      print ("PWM= ", x1)
         srv=Servo(1, x1)
         msg=ServoArray({srv})
         pub.publish(msg)
         r.sleep()

    if wz == 0:
      x2 = (332)
         srv=Servo(2, x2)
         msg=ServoArray({srv})
         pub.publish(msg)
         r.sleep()
    else:
      x2 = (332)+round(wz)
      print ("PWMA= ", x2)
         srv=Servo(2, x2)
```

```
        msg=ServoArray({srv})
        pub.publish(msg)
        r.sleep()

    if vx > 0 or vz>0:
      x3 = (4095)
        srv=Servo(3, x3)
        msg=ServoArray({srv})
        pub.publish(msg)
        r.sleep()
    elif vz<0:
      x3 = (0)
      #print ("PWMLaser= ", x3)
        srv=Servo(3, x3)
        msg=ServoArray({srv})
        pub.publish(msg)
        r.sleep()


def talker():
    rospy.Subscriber("cmd_vel", Twist, callback)
    twist_pub = rospy.Publisher('pwm', Twist ,queue_size=10)
    rospy.init_node('velocidad', anonymous=True)
    rospy.loginfo("Iniciando nodo velocidad")
    rate=rospy.Rate(10)
    while not rospy.is_shutdown():
        twist=Twist()
        twist.linear.x=x1
        twist.angular.z=x2
        twist_pub.publish(twist)
      rate.sleep()



if __name__ == '__main__':
    try:
      talker()
    except rospy.ROSInterruptException:
        pass
```

A6. Codigo en Python del nodo Odom

```
#!/usr/bin/env python

import math
from math import sin, cos, pi

import rospy
import tf
from nav_msgs.msg import Odometry
from geometry_msgs.msg import Point, Pose, Quaternion, Twist, Vector3

x=0.0
y=0.0
yaw=0.0
pwm1=350
pwm2=332


def callback1(data1):
    global x, y, yaw, pwm1, pwm2
    pwm1=data1.linear.x
    pwm2=data1.angular.z
    #print("pwm1:= ", pwm1)
    #print("pwm2:= ", pwm2)

def talker():
    global x, y, yaw, pwm1, pwm2
    rospy.Subscriber("pwm", Twist , callback1)
    odom_pub = rospy.Publisher("odom", Odometry, queue_size=50)
    odom_broadcaster = tf.TransformBroadcaster()
    rospy.init_node('odom', anonymous=True)
    rospy.loginfo("Iniciando Odometria")

    current_time = rospy.Time.now()
    last_time = rospy.Time.now()
    r = rospy.Rate(10)
    while not rospy.is_shutdown():
      current_time = rospy.Time.now()
      if pwm1==350:
               v = 0
           d = 0
           ang = 0
           w = 0
        elif pwm1 !=350 and pwm2==332:
           #v = 0.1449684211*pwm1-52.5397684211
           v = 0.1*pwm1-35.25
           d = 0
           ang = 0
           w = 0
```

```
      else:
            #print("prueba")
            v = 0.1*pwm1-35.25
            d = 0.3156*pwm2-105.568
            ang = math.radians(d)
            w = (v*math.tan(ang))/0.270

      # compute odometry in a typical way given the velocities of the
robot
      dt = (current_time - last_time).to_sec()
      #print("Velocidad:= ",v)
      #print("Direccion:= ",d)
      #print("Dir. Radianes:= ",ang)
      #print("V. angular:= ",w)
      x_dot = v * math.cos(yaw)
      y_dot = v * math.sin(yaw)
      x += x_dot*dt
      print("x:= ", x)
      print("y:= ",y)
      y += y_dot*dt

      yaw += w*dt
      print("yaw:= ",yaw)

      # since all odometry is 6DOF we'll need a quaternion created
from yaw
      odom_quat = tf.transformations.quaternion_from_euler(0, 0, yaw)

      # first, we'll publish the transform over tf
      odom_broadcaster.sendTransform((x, y, 0.), odom_quat,
current_time, "base_link", "odom")

      # next, we'll publish the odometry message over ROS
      odom = Odometry()
      odom.header.stamp = current_time
      odom.header.frame_id = "odom"

      # set the position
      odom.pose.pose = Pose(Point(x, y, 0.), Quaternion(*odom_quat))

      # set the velocity
      odom.child_frame_id = "base_link"
      odom.twist.twist = Twist(Vector3(v, 0, 0), Vector3(0, 0, w))

      # publish the message
      odom_pub.publish(odom)

      last_time = current_time
      r.sleep()
```

```
if __name__ == '__main__':
    try:
        talker()
    except rospy.ROSInterruptException:
        pass
```

Acrónimos

AGV:	Automatic Guided Vehicle (Vehículos de Conducción Automática)
AMD:	Advanced Micro Devices
API:	Application Programming Interface (Interfaz de Programación de Aplicaciones)
EKF:	Extended Kalman Filter (Filtro Extendido de Kalman)
IMU:	Inertial Measurement Unit (Unidad de Medición Inercial)
KF:	Kalman Filter (Filtro de Kalman)
LIDAR:	Light Detection and Ranging (Radar Óptico Láser)
MPU:	Multiple Process Unit (Unidad de Procesos Múltiple)
PWM:	Pulse-Width Modulation (Modulación por Ancho de Pulsos)
PPM:	Pulse Position Modulation (Modulacion de la Posición por Pulsos)
RC:	Radio Control
ROMA:	Robótica Móvil Autónoma
ROS:	Robot Operating System (Sistema Operativo para Robots)
SBC:	Single Board Computer (Computador de Placa Reducida)
SLAM:	Simultaneous Localization And Mapping (Localización y Mapeo Simultáneos)
UGV:	Unmanned Ground Vehicle (Vehículo Terrestre no Tripulado)

Referencias

Aguilar, D. J. (2014). *Construcción de mapas probabilísticos mediante técnicas de SLAM en entorno ROS.* Alcalá de Henares.

Amain hobbies. (13 de 10 de 2018). *Amain hobbies*. Obtenido de https://www.amainhobbies.com/hpi-saturn-35t-brushed-motor-hpi1148/p72755

Araújo, A., Portugal, D., Couceiro, M., Sales, J., & Rocha, R. (2014). Desarrollo de un robot móvil compacto integrado en el middleware ROS. *ScienceDirect, 11*(3), 12.

Ballesta, M., Gil, A., Reinoso, O., & Úbeda, D. (2010). Análisis de Detectores y Descriptores de Características Visuales en SLAM en Entornos Interiores y Exteriores. *Revista Iberoamericana de Automática e Informática Industrial, 7*(2), 68-80.

Bambíno, I. (2008). Una Introducción a los Robots Móviles.

Barnett, R. (s.f.). *GitHub*. Recuperado el 1 de 10 de 2018, de https://github.com/richardstechnotes/RTIMULib2

Bradan Lane Studio. (s.f.). *i2cpwm_board Documentation*. Recuperado el 9 de 10 de 2018, de http://bradanlane.gitlab.io/ros-i2cpwmboard/

Carvajal Meza, B. (2015). Sistema de localización y construcción de mapas. Barcelona: Universidad Politécnica de Cataluña.

CNXSOFT. (s.f.). *Firefly RK-3288*. Recuperado el 03 de 04 de 2018, de https://www.cnx-software.com/2014/09/27/firefly-rk3288-development-board-is-now-available-for-189/

Corke, P. (2011). *Robotics, Vision & Control* (Vol. 73). Brisbane, Queensland, Australia: Springer.

Cruz Ortiz, J., & Vasquez Torrez, C. (2012). Sistema De Navegación Autónoma Para La Plataforma Robótica Móvil (DANI) Del Grupo De Investigación ROMA Basado En Métodos De Control Reactivo. Bogotá, D.C.: UNIVERSIDAD DISTRITAL FRANCISCO JOSE DE CALDAS INGENIERIA.

Cursos RoboLab. (Octubre de 2014). (Universidad de Extremadura) Recuperado el 23 de 04 de 2016, de http://www.cursosrobolab.com/robotica/

Dryanovski, I. (07 de 09 de 2016). *ROS.org*. Recuperado el 1 de 10 de 2018, de http://wiki.ros.org/imu_filter_madgwick?distro=indigo

Dryanovski, I. (s.f.). *wiki.ros.org*. (ROS.org) Recuperado el 29 de 08 de 2018, de http://wiki.ros.org/imu_filter_madgwick

Durrant-Whyte, H., & Bailey, T. (2006). Simultaneous localization and mapping (SLAM): part I The Essential Algorithms. *Robotics & Automation Magazine, 2*, 9.

Ferrer, R. F. (2013). Implementación de algoritmo slam basado en sensor láser hokuyo 04LX - UG01. Valencia: Universidad Politecnica de Valencia.

Gallart Del Burgo, X. (2013). Semantic Mapping in ROS. Estocolmo: The Royal Institute of Technology.

Garcia Montañés, L. (2017). *Navegación sin mapa y mapeado en robótica móvil para entornos no estructurados.* Sevilla.

Graylin Trevor, J. (s.f.). *ROS.org*. Recuperado el 1 de 10 de 2018, de http://wiki.ros.org/teleop_twist_keyboard

Grisetti, G., Stachniss, C., & Burgard, W. (16 de 10 de 2007). *OpenSLAM*. Obtenido de https://openslam-org.github.io/gmapping.html

Inc., I. (s.f.). *www.invensense.com*. (TDK) Recuperado el 19 de 09 de 2018, de https://www.invensense.com/products/motion-tracking/9-axis/mpu-9250/

Jetsonhacks. (2018). *GitHub, Inc*. Recuperado el 10 de 07 de 2018, de https://github.com/jetsonhacks/RTIMULib

Jiménez González, A. (2008). Técnicas de percepción activa para seguimiento de objetos mediante robots móviles en entornos urbanos. Sevilla: Universidad de Sevilla. Escuela Superior de Ingenieros.

Kohlbrecher, S., von Stryk, O., Meyer, J., & Klingauf, U. (2011). A Flexible and Scalable SLAM System with Full 3D Motion Estimation. *IEEE*.

Kosuru, G. (2011). Design and Implementation of an EKF based SLAM algorithm on a mobile robot. Hyderabad, India: Robotics Research Lab.

Llofriu, M., & Andrade, F. (2014). Estudio del estado del arte del SLAM e implementación de una plataforma flexible. Montevideo. Uruguay: Universidad de la República.

Local Digital Library II. (14 de 11 de 2018). *ROS/Concepts*. Obtenido de http://library.isr.ist.utl.pt/docs/roswiki/ROS(2f)Concepts.html

Madgwick, S. O. (2010). An efficient orientation filter for inertial and inertial/magnetic sensor arrays.

Mansoor, W. (17 de 10 de 2018). *Medium.com.* Obtenido de https://medium.com/@waleedmansoor/how-i-built-ros-odometry-for-differential-drive-vehicle-without-encoder-c9f73fe63d87

Martínez Gómez, Á. V. (2015). Desarrollo, implementación y comparación de distintas técnicas de SLAM para robots Pioneer. Albacete: UNIVERSIDAD DE CASTILLA-LA MANCHA.

Martinez, A., & Fernández, E. (2013). *Learning ROS for Robotics Programming.* Birminghan, UK: Packt Publishing Ltd.

Meeussen, W. (s.f.). *wiki.ros.org*. (Ros.org) Recuperado el 29 de 08 de 2018, de http://wiki.ros.org/robot_pose_ekf

Milstein, A. (2008). *Occupancy Grid Maps for Localization and Mapping.*

Montemerlo, M., Thrun, S., Koller, D., & Wegbreit, B. (2002). FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem.

Naminski, M. R. (2013). An Analysis of Simultaneous Localization and Mapping (SLAM) Algorithms. Macalester Math, Statistics, and Computer Science Department.

Ollero Baturone, F. (2010). *Robótica. Manipuladores y robots móviles.* Barcelona: MARCOBO.

Paniagua Jaramillo, J. L. (2014). Diseño e Implementacion de un Sistema de Control que Permita Integrarse con Diferentes Tipos de Robot Moviles Terrestres. Santiago de Cali.

Rapado García, J. M. (2016). *Diseño e implementación de una interfaz gráfica de usuario para mapeado de entornos y navegación en ROS.* Valencia.

Realpe Robalino, M. (2009). Hacia la navegación autónoma de robots a partir de la implementación de un método de localización y mapeo simultáneos (SLAM) mediante el uso de un sistema de visión 3D. Guayaquil: ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL.

RoboPeak. (2014). RPLIDAR Low Cost 360 degree 2D Laser Scanner (LIDAR) System Introduction and Datasheet.

Robopeak. (2018). *GitHub, Inc.* Recuperado el 3 de 7 de 2108, de https://github.com/robopeak/rplidar_ros

ROS , A. (23 de 05 de 2018). *ROS Answers*. Recuperado el 24 de 05 de 2018, de https://answers.ros.org/question/197651/how-to-install-a-driver-like-usb_cam/

Sánchez Vítores, R. (2005). Estado del arte de los sistemas embebidos. *Antena de Telecomunicación*, 20-24.

Silva Ortigoza, R., García Sánchez, J., Barrientos Sotelo, V., & Molina Vilchis, M. (2012). UNA PANORÁMICA DE LOS ROBOTS MÓVILES. *Revista Electrónica de Estudios Telemáticos, 6*(3), 208-222.

Studio, B. L. (s.f.). *GitLab*. Recuperado el 31 de 05 de 2018, de https://gitlab.com/bradanlane/ros-i2cpwmboard

Suáres Marcelo, J. I. (2001). Robot móvil para transporte automatizado. Badajoz: UNIVERSIDAD DE EXTREMADURA.

Valera, A., Soriano, A., & Vallés, M. (2014). Plataformas de Bajo Coste para la Realización de Trabajos Prácticos de Mecatrónica y Robótica. *Revista Iberoamericana de Automática e Informática industrial*, 363-376.

Velásquez Hernández, C. A. (2017). *Desarrollo de Algoritmo de Mezclado de Mapas por Ocupación de Celdas Aplicado a la Navegación y Exploración Colaborativa de Entornos Internos Desconocidos.* Bogotá D.C.

Welch, G., & Bishop, G. (2010). An Introduction to the Kalman Filter. Chapel Hill, NC: University of North Carolina.

Willow Garage. (s.f.). *ROS plataform*. Recuperado el 23 de 04 de 2016, de http://www.willowgarage.com/pages/software/ros-platform

x-io Technologies. (31 de 07 de 2012). *x-io Technologies*. Recuperado el 1 de 10 de 2018, de http://x-io.co.uk/open-source-imu-and-ahrs-algorithms/