ESCUELA SUPERIOR POLITÉCNICA DE CHIMBORAZO

PLATAFORMA ROBÓTICA MÓVIL PARA EXPERIMENTOS DE MAPEO Y LOCALIZACIÓN SIMULTANEA (SLAM) EN BASE A SENSORES DE RANGO

ING. EDISSON IVAN ALDAS SERRANO

Trabajo de Titulación modalidad Proyectos de Investigación y Desarrollo, presentado ante el Instituto de Posgrado y Educación Continua de la ESPOCH, como requisito parcial para la obtención del grado de:

MAGISTER EN SISTEMAS DE CONTROL

Y AUTOMATIZACIÓN INDUSTRIAL

RIOBAMBA - ECUADOR

OCTUBRE 2017

ESCUELA SUPERIOR POLITÉCNICA DE CHIMBORAZO

CERTIFICACIÓN:

EL TRIBUNAL DEL TRABAJO DE TITULACIÓN CERTIFICA QUE:

El Trabajo de Titulación modalidad Proyectos de Investigación y Desarrollo, denominado: ¨PLATAFORMA ROBÓTICA MÓVIL PARA EXPERIMENTOS DE MAPEO Y LOCALIZACIÓN SIMULTANEA (SLAM) EN BASE A SENSORES DE RANGO¨, de responsabilidad del señor Aldás Serrano Edisson Iván, ha sido minuciosamente revisado y se autoriza su presentación.

Tribunal:

Ing. Fredy Proaño Ortiz MsC
PRESIDENTE _____

Ing. Luis Pomaquero Moreno, MsC.
TUTOR PROYECTO DE TITULACIÓN _____

Ing. Patricio Córdova Córdova, Mg.
MIEMBRO DEL TRIBUNAL _____

Ing. Santiago Altamirano Meléndez, Mg.
MIEMBRO DEL TRIBUNAL _____

Riobamba, octubre 2017

DERECHOS INTELECTUALES

Yo, Aldás Serrano Edisson Iván soy responsable de las ideas, doctrinas y resultados expuestos en este Trabajo de Titulación y el patrimonio intelectual del mismo pertenece a la Escuela Superior Politécnica de Chimborazo.

_____

ALDÁS SERRANO EDISSON IVÁN

Nº Cédula: 180327581-5

DERECHOS DE AUTOR

Yo, Aldás Serrano Edisson Iván, declaro que el presente proyecto de investigación, es de mi autoría y que los resultados del mismo son auténticos y originales. Los textos constantes en el documento que provienen de otras fuentes están debidamente citados y referenciados.

Como autor, asumo la responsabilidad legal y académica de los contenidos de este Trabajo de Titulación de Maestría.

_____

ALDÁS SERRANO EDISSON IVÁN

Nº Cédula: 180327581-5

DEDICATORIA

El presente trabajo se lo dedico a mi esposa Jeanette y mis hijos Iván Mauricio y Mathias Isaac, ya que me han brindado la fuerza para que el presente proyecto llegue a su exitosa culminación y quienes les debo el tiempo familiar utilizado en él mismo.

Iván Aldas

AGRADECIMIENTO

Agradezco infinitamente a Dios por darme la oportunidad de poder cruzar una etapa más de mi vida profesional, además a mis padres Edisson y Mercedes que siempre me han apoyado en cada uno de los retos de mi vida, así como también mis herman@s José Luis, María Belén y Mateo Sebastián que siempre han estado apoyándome de una u otra manera.

CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

RESUMEN

El objetivo fue diseñar una plataforma robótica móvil que permita a los estudiantes de la carrera de Mecatrónica de la Universidad Técnica del Norte (UTN), realizar experimentos de Mapeo y Localización Simultanea (SLAM) en dos dimensiones con sensores de rango, para ello se investigó los enfoques y las técnicas utilizadas para dicho fin, destacándose el enfoque probabilístico, con técnicas de estimación y filtros bayesianos. Se investigó los elementos hardware que permiten desarrollar dicho performance. En este punto, el sensor Detección de Luz y Rango (LIDAR) destaca fuertemente y en conjunto con una mini computadora denominada Raspberry Pi 3 como cerebro del Robot y una plataforma robótica comercial modificada como estructura base, se implementó la RasPi_mBot. En la cual, se aplicaron algoritmos computacionales en el entorno Sistema Operativo Robotico (ROS). Se concluye que con la aplicación de los siguientes paquetes de ROS, hector_slam, gmapping y un protocolo de pruebas se desarrolló exitosamente el mapeo y la localización simultánea en entornos controlados similares a los existentes en los laboratorios de la UTN. Se recomienda incluir otros sensores en combinación con técnicas de navegación y planificación de trayectorias para dotar a la RasPi_mBot de autonomía completa.

Palabras Claves:

<TECNOLOGÍA Y CIENCIA DE LA INGENIERÍA> <ROBÓTICA> <MAPEO Y LOCALIZACIÓN SIMULTANEA (SLAM)> <SISTEMA OPERATIVO ROBÓTICO (ROS)> <HECTOR_SLAM (SOFTWARE)> <GMAPPING (SOFTWARE)> <SENSOR LIDAR> <RASPBERRY PI>

SUMMARY

The objective of this research was to design a mobile robotic platform that allows students of the Mechatronics career at Universidad Tecnica del Norte (UTN), to perform Mapping and Simultaneous Location (SLAM) experiments in two dimensions with range sensors. For this, the approaches and techniques used for this purpose were researched, highlighting probabilistic approach with estimation techniques and Bayesian filters. They were researched the hardware elements that allow to develop such performance. At this point, the Sensor of Light Detection and Range (LIDAR) stand out strongly and in conjunction with a mini computer called Raspberry Pi 3 as robot brain and a commercial robotic platform modified as a base structure, it was implemented the Raspi_mBot. In which, computational algorithms were applied in the Robotic Operation System (ROS) environment. It is concluded that with the application of the following packages: ROS, hector_slam, gmapping and test protocol, the Mapping and the Simultaneous Localization were successfully developed in controlled environments similar to those existing in the laboratories at Universidad Tecnica del Norte (UTN). It is recommended to include other sensors in combination with navigation techniques and trajectory planning to provide the Raspi_mBot with complete autonomy.

Key words:

<TECHNOLOGY AND SCIENCE OF ENGINEERING> <ROBOTICS> <SIMULTANEOUS LOCALIZATION AND MAPPING (SLAM)> <ROBOTIC OPERATING SYSTEM (ROS)> <HECTOR_SLAM (SOFTWARE)> <GMAPPING (SOFTWARE)> <LIDAR SENSOR> <RASPBERRY PI>

CAPÍTULO I

1. INTRODUCCIÓN

1.1. Título

Plataforma robótica móvil para experimentos de mapeo y localización simultanea (SLAM) en base a sensores de rango.

1.2. Planteamiento de problema

Uno de los objetivos que persigue la Robótica y la Inteligencia Artificial radica en ampliar el rango de aplicaciones de los robots, sin embargo, esto es posible solo si los dotamos con mayores grados de autonomía.

En función de lo anterior, los robots móviles requieren obtener datos de su entorno y procesarlos rápidamente para tomar decisiones eficaces con la mínima intervención de un operario humano. En este contexto, uno de los mayores problemas que enfrentan este tipo de robots consiste en la localización y mapeo en entornos. En la actualidad, se han desarrollado diferentes técnicas para solucionar dicho problema. En los últimos años, los robots móviles han sido el tema de investigación preferido en muchos laboratorios y universidades. Esto se debe a que son excelentes plataformas de enseñanza y aplicación de teorías y técnicas, teniendo además aplicaciones en gran cantidad de campos de la industria, la milicia, e incluso el hogar.

Por otro lado, el Laboratorio de Robótica y Sistemas Microprocesados de la carrera en Ingeniería Mecatrónica (CIME) de la Universidad Técnica del Norte (UTN) carece de una plataforma robótica móvil que permita experimentar las soluciones existentes hasta la fecha.

Justamente, el presente proyecto implementa una plataforma robótica móvil basada en sensores de rango, para que sea utilizada por los estudiantes de CIME como una herramienta útil de testeo de las soluciones actuales y por qué no, generar nuevas

soluciones en lo referente a la localización y mapeo simultaneo aplicables a robots autónomos móviles.

1.3. Formulación del problema

¿La implementación de una plataforma robótica móvil, permitirá realizar experimentos de localización y mapeo instantáneo en los laboratorios de robótica de la Universidad Técnica del Norte?

1.4. Sistematización del problema

¿Cuáles son los métodos o procedimientos que se emplean actualmente para la localización y mapeo simultaneo 2D en robots autónomos móviles?

¿Qué elementos de hardware y software son necesarios para que la plataforma robótica móvil realice localización y mapeo simultaneo 2D?

¿Cómo se puede integrar el hardware y software de la plataforma robótica móvil?

¿Cuál es la mejor manera de verificar el funcionamiento de la plataforma robótica móvil propuesta?

1.5. Objetivos de la investigación

1.5.1. Objetivo general

Desarrollar una plataforma robótica móvil para experimentos de localización y mapeo instantáneo (SLAM) en base a sensores de rango.

1.5.2. Objetivos específicos.

- Identificar los métodos que se emplean actualmente para la localización y mapeo simultaneo 2D en robots móviles.
- Determinar los requerimientos del hardware y software de la plataforma robótica móvil.
- Realizar el montaje de la plataforma robótica móvil.

- Evaluar el funcionamiento de la plataforma robótica móvil en diferentes ambientes controlados.

1.6. Justificación de la investigación

En las últimas décadas, una de las áreas que ha evolucionado a pasos agigantados es la Robótica y junto a ella, la electrónica y la mecánica. Pero, debido a la carencia de herramientas para adiestramiento en esta rama por su elevado costo tanto en hardware como software ha limítalo a los estudiantes para que se embarquen en proyectos de desarrollo robótico.

Dentro de CIME de la UTN, una fortaleza de los futuros ingenieros es la del diseño e implementación de productos y/o máquinas autónomas, tal como se expresa en su perfil profesional correspondiente: "*El egresado de la Carrera de Ingeniería en Mecatrónica es un profesional competente, crítico, humanista, líder y emprendedor, cuenta con una formación sólida en las diversas áreas del conocimiento lo que le permite involucrarse eficazmente y con responsabilidad social en actividades de investigación, diseño e innovación de productos mecatrónicos, manejo de equipo técnico relacionado con su profesión, transferencia y/o adaptación de tecnología, con cuidado del medio ambiente*" (UTN, 2016), y ellos, al no contar con los instrumentos necesarios en los laboratorios, para que desarrollen diferentes destrezas, representa una desventaja a nivel académico y laborar.

En otro ámbito, el país necesita desarrollar su propia tecnología para contribuir con el cambio de la matriz productiva y dejar de ser tecnológicamente dependientes. En función de lo dicho, al disponer de una plataforma robótica diseñada e implementada localmente, que permita la localización y mapeo simultaneo, puede ser utilizada en una amplia gama de aplicaciones tales como la búsqueda de personas en desastres naturales, la localización de minas personales terrestres, exploración de ambientes peligrosos, por nombrar algunas.

1.7. Hipótesis

Al implementar una plataforma robótica móvil con sensores de rango, se podrá realizar experimentos de SLAM, en los laboratorios de la Mecatrónica de la UTN.

CAPÍTULO II

2. MARCO DE REFERENCIA

En este capítulo se presenta la situación actual en el país en cuanto a desarrollos en robótica móvil, posteriormente se presentan las definiciones de robot móvil autónomo. Luego se expone el problema del SLAM y finalmente las soluciones implementadas basados en paquetes ROS y hardware comercial.

2.1. La Robótica móvil en el país

En el Ecuador, las IES (Instituciones de Educación Superior) donde se ofertan carreras técnicas relacionadas con Robótica son insuficientes y no disponen de equipos específicos para la realización de prácticas. La mayoría de estas apenas cuentan solo con robots industriales, constituidos de brazos articulados sujetos a una base. Acorde a lo anterior, las investigaciones desarrolladas en las universidades del país también son escasas. Prueba de ello, existen pocos antecedentes relacionados a investigaciones dentro del campo de robots autónomos que se comentan a continuación: en la PUCESA se ha implementado una plataforma multipropósito basada en un robot explorador como material didáctico para la Escuela de Sistemas (Hernández, 2015); en la Universidad del Azuay se ha desarrollado un robot para mapeo y exploración de minas subterráneas (Cabrera & Delgado, 2014); y en la ESPOL se diseñó e implemento un equipo de robots autónomos que traman decisiones en tiempo real aplicado al futbol robótico (Villaroel, Calderón, & Carrillo, 2003).

2.2. Robots autónomos móviles

Un robot móvil es una máquina automática que es capaz de moverse en cualquier ambiente para el que fue diseñado. Al contrario de los robots industriales, los cuales generalmente consisten en brazos articulados sujetos a una superficie con actuadores en sus extremos, los robots móviles tienen la capacidad de moverse sin restricciones por cualquier locación (Gopalakrishnan & Tirunellayi, 2014).

El funcionamiento autónomo de un robot, implica la capacidad de adaptarse a los cambios de entorno, en condiciones variadas y sin la supervisión de un humano. (Murphy, 2000)

Esta capacidad es necesaria para que un robot de soluciones a los problemas planteados por la comunidad científica. En la industria está capacidad es evidenciada en los productos comerciales como el Roomba, el vehículo sin conductor de Google, entre otros.

2.2.1. Proyecto Roomba

Los robots de aspiración Roomba son alimentados por un conjunto completo de sensores inteligentes que guían automáticamente al robot alrededor de su hogar. El robot toma 60 decisiones cada segundo, navegando debajo de los muebles y alrededor del desorden para limpiar completamente sus pisos. (iRobot Corporation, 2016)

Figura 1-2: Familia de Robots Roomba

Fuente: iRobot, Inc, 2016

En la figura 1-2 se muestra los modelos actuales de robot Roomba. Las principales características se listan a continuación:

- La tecnología iAdapt Navigation utiliza un conjunto completo de sensores que permiten que su robot trabaje en los rincones y recovecos de su hogar.
- Utiliza sensores acústicos para detectar altas concentraciones de suciedad y luego realiza la limpieza enfocada donde más se necesita.
- El diseño de bajo perfil permite que el robot de la aspiradora Roomba limpie debajo de la mayoría de los muebles y otros menajes de hogar, para que la suciedad no tenga lugar para esconderse.
- Los sensores de detección de acantilados permiten al robot de aspiración Roomba evitar escaleras y otras bajadas.
- La aplicación iRobot HOME le permite limpiar y programar convenientemente, en cualquier momento y en cualquier lugar. Ahora es compatible en dispositivos

con Amazon Alexa y el Asistente de Google. También permite monitorear el estado de los trabajos de limpieza.

- Puede aspirar hasta siete veces por semana mediante configuración.
- Se ajusta automáticamente para limpiar alfombras, azulejos, pisos de madera y laminados mientras se mueve por el hogar.
- Retorno automático a la estación de carga de Home Base.

2.2.2. Vehículo no tripulado de Google

La compañía Google en el 2014 dio a conocer al mundo el prototipo de su vehículo no tripulado, el mismo que carece de pedales y volante. El vehículo utiliza una combinación de sensores de radar y láser, así como una cámara, para conducir de manera autónoma. Para garantizar la seguridad, la velocidad inicial máxima fue de 40 km/h. (BBC, 2014)

En la figura siguiente se muestra dicho prototipo:

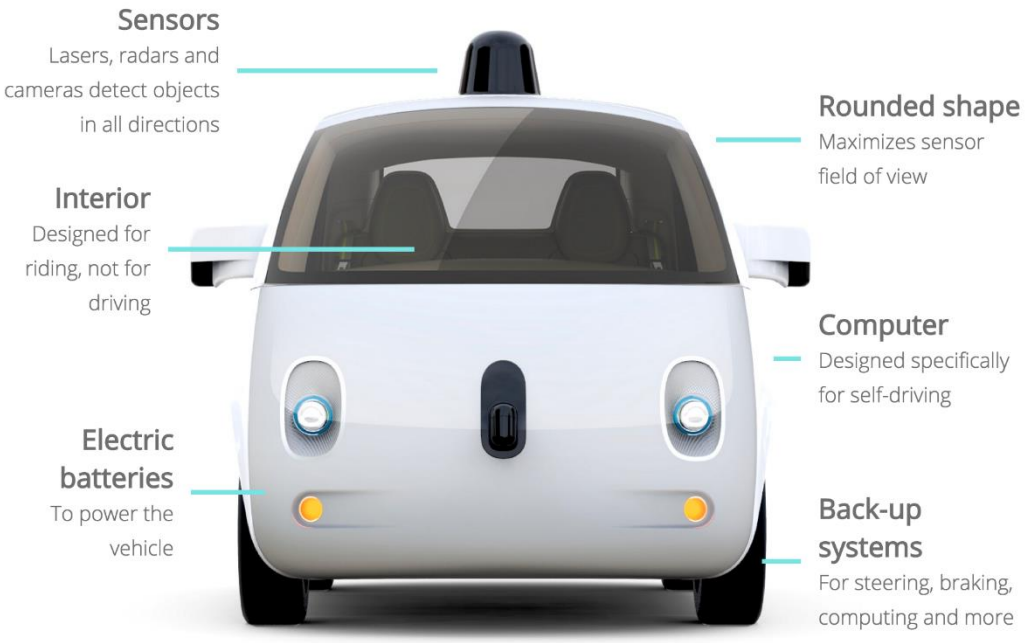


Figura 2-2: Vehiculó no tripulado de Google

Fuente: BBC, Noticias, 2016

Actualmente Google trabaja para fabricar automóviles totalmente autodirigidos de forma segura y fácil para todos. El proyecto comenzó en 2009, y en 2016 se convirtió en WAYMO.

Ahora el vehículo luce así:

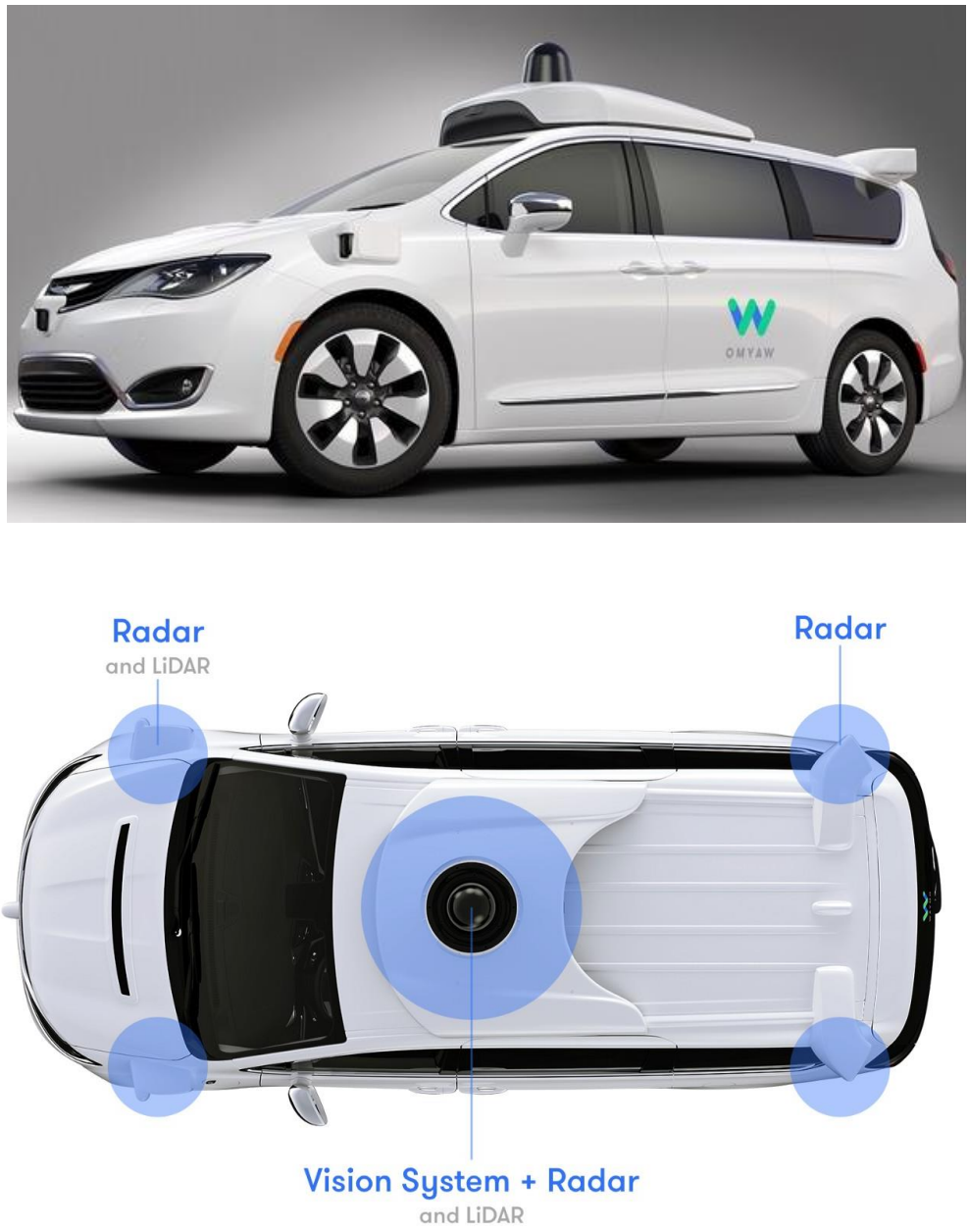


Figura 3-2: WAYMO implementado en una Chrysler Pacific Hybrid

Fuente: WAYMO, Inc, 2017

Estos automóviles tienen sensores y software que están diseñados para detectar peatones, ciclistas, vehículos, trabajos de carretera y más desde una distancia de hasta dos campos de fútbol en todas direcciones. (WAYMO, 2017)

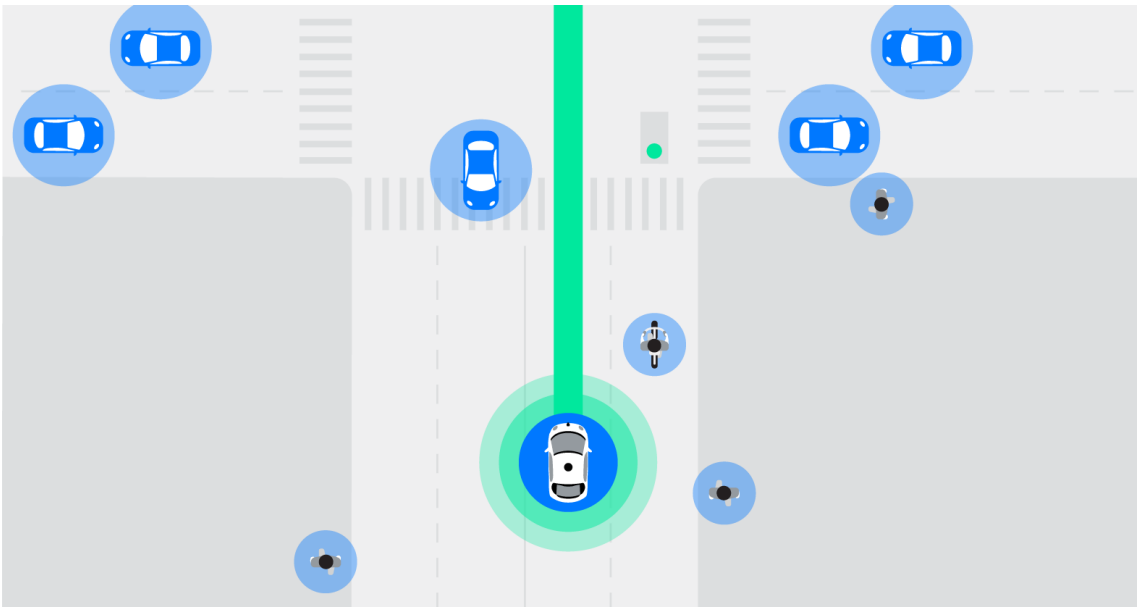


Figura 4-2: Detección de los sensores del WAYMO

Fuente: WAYMO, Inc, 2017

Los sensores y software detectan y predicen el comportamiento no sólo del ciclista, sino de todos los usuarios de la carretera que rodean al vehículo. La figura siguiente ejemplariza lo dicho:

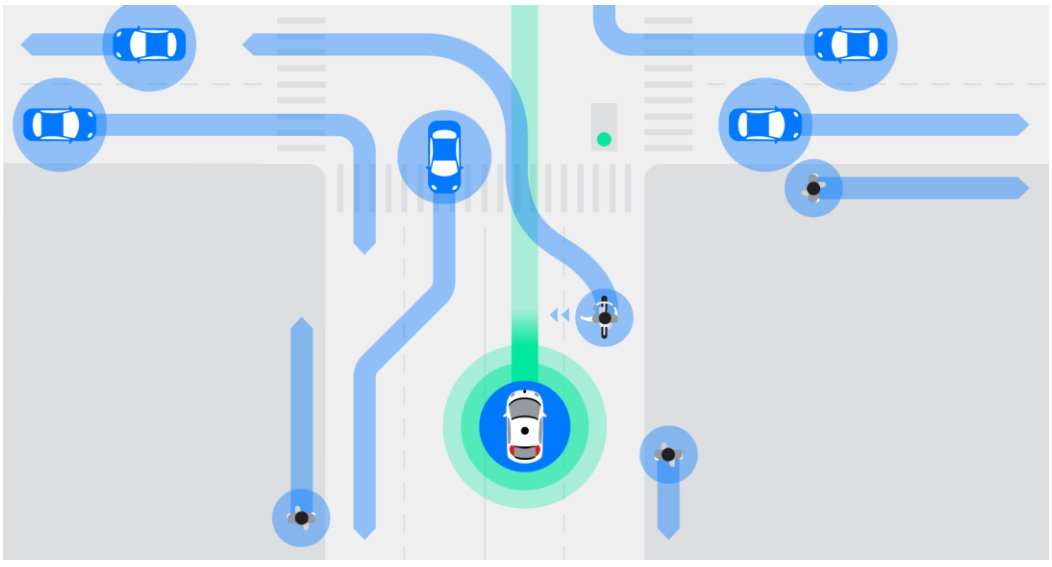


Figura 5-2: Predicción de comportamiento externo del WAYMO

Fuente: WAYMO, Inc, 2017

El WAYMO cuenta con 3 millones de millas de experiencia en el mundo real, este bagaje permitió enseñar al coche a navegar de manera segura y cómoda a través del tráfico diario. La figura 2-6 evidencia todo el poder del WAYMO.

Figura 6-2: WAYMO en acción

Fuente: WAYMO, Inc, 2017

2.3. Plataformas robóticas para investigación

Existen muchas plataformas robóticas móviles disponibles para investigación y desarrollo, las mas importantes se describen a continuación:

2.3.1. TurtleBot

Sin lugar a duda, es la plataforma robótica más conocida. TurtleBot es un kit de robot personal de bajo costo con software de código abierto. TurtleBot fue creado en Willow Garage por Melonee Wise y Tully Foote en noviembre de 2010 (Open Source Robotics Foundation, 2017). En la siguiente figura se muestra los modelos disponibles hasta la fecha con sus características principales.

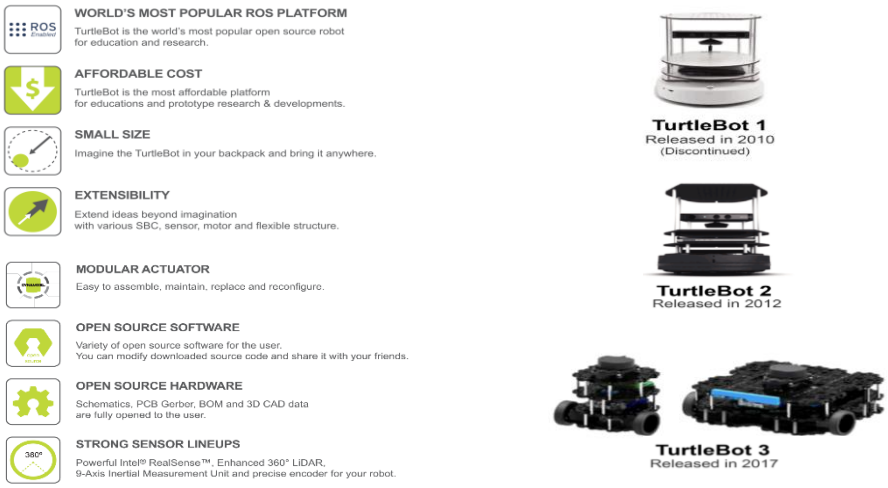


Figura 7-2: Modelos de la TurtleBot

Fuente: TurtleBot, Inc, 2017

2.3.2. Parallax Eddie

El robot Eddie es un robot móvil autónomo que lleva una computadora portátil y un sensor Kinect. El robot Eddie es la plataforma de referencia elegida por Microsoft para el uso del *Microsoft Robotics Developer Studio* que integra el SDK del Kinect con poderosos algoritmos proporcionados por Microsoft que permiten reconocimiento de formas 3D humanas (Generation_Robots, 2017). En la siguiente figura se muestra la plataforma EddieBot.

Figura 8-2: Parallax Eddie

Fuente**:** Parallax, Inc, 2017

Las principales características son:

- ✓ Laptop
- ✓ Sensor Microsoft Kinect
- ✓ Sensores IR
- ✓ Sensores Ultrasónicos
- ✓ Motores 12 VDC

2.3.3. Tetra-DS IV

La plataforma Tetra DS-IV es una plataforma avanzada utilizada para desarrollo y testeo de tecnologías de locomoción y programación de robots autónomos. Está compuesto por una base y tarjetas de control. El diseño modular de las tarjetas de control permite un fácil mantenimiento y la incorporación de actualizaciones futuras (RobotShop, 2017). En la siguiente figura se muestra la plataforma Tetra DS-IV.

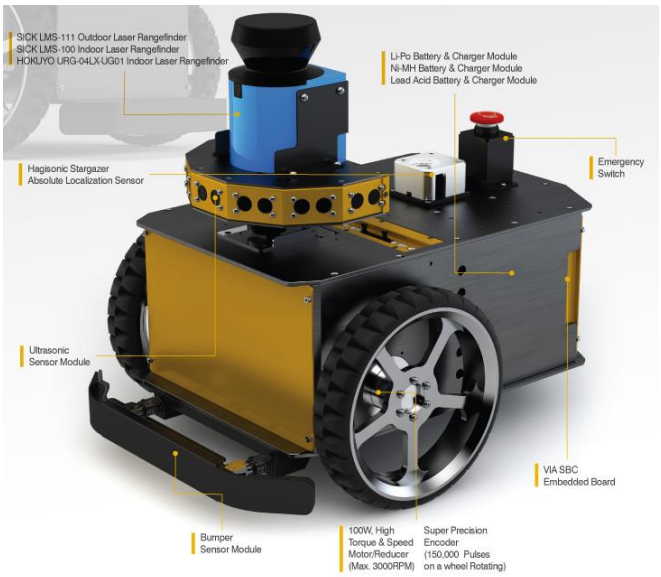


Figura 9-2: Tetra-DS IV

Fuente: Tetra, Inc, 2017

Las principales características son:

- ✓ Sensor Laser
- ✓ Sensores Ultrasónicos
- ✓ Sensor de localización absoluta
- ✓ Motores con encoders
- ✓ Batería Li-Po/Ni-MH

2.3.4. Summit XL

La plataforma móvil Summit XL diseñado por ROBOTNIK, gracias a la excelente estructura mecánica, modularidad y versatilidad permiten utilizarlo para investigación, vigilancia, monitorización remota o aplicaciones militares. Summit XL tiene cinemática diferencial (skid-steering) basada en 4 motores de alto rendimiento. Cada rueda integra un motor sin escobillas y encoder. Además, tiene 2 posibles configuraciones cinemáticas. La configuración omnidireccional dispone de ruedas mecanum montadas sobre un sistema de suspensión independiente. Las ruedas mecanum se pueden cambiar por ruedas convencionales (montaje en llanta), consiguiendo de esta forma cambiar rápidamente de una configuración omnidireccional de interiores a una configuración skid-steering versátil tanto en interiores como en exteriores (Robotnik, 2017). En la siguiente figura se muestra la plataforma Summit XL.

Figura 10-2: Summit XL

Fuente**:** Robotnik, Inc, 2017

Las principales características son:

- ✓ Sensor Laser
- ✓ Sensor IMU
- ✓ Cámaras de rango
- ✓ GPS
- ✓ Soporte ROS

2.3.5. Pioneer 3-AT

El PIONEER 3-AT es una plataforma robótica de cuatro ruedas altamente versátil, desarrollado por ADEPT Mobile robots. Potente, pero fácil de usar, confiable, pero flexible, P3-AT es la elección popular para los proyectos en exteriores o de terrenos accidentados (MOBILEROBOTS, 2017). En la siguiente figura se muestra la plataforma P3-AT.

Figura 11-2: Pioneer 3-AT

Fuente: ADEPT, Mobile robots, 2017

Las principales características son:

- ✓ Sensor Laser
- ✓ Sensor IMU
- ✓ Sensores ultrasónicos
- ✓ Cámaras de visión estéreo
- ✓ GPS
- ✓ Brazos robóticos y pinzas

2.4. El problema del SLAM

SLAM proviene del inglés Simultaneous Localization And Mapping, o en español: Localización y Mapeado Simultáneos o también Localización y Modelado Simultáneos. Es una técnica usada por robots y vehículos autónomos para construir un mapa de un entorno desconocido en el que se encuentra, a la vez que estima su trayectoria al desplazarse dentro de este entorno (Montemerlo, 2003).

Existen ocasiones en las que se conoce de antemano el medio en el cual se moverá el robot y es posible proporcionarle un mapa del mismo. Por otro lado, existen escenarios en los que un robot debe moverse en un entorno desconocido, pero cuenta con información precisa sobre su ubicación. Dicha ubicación puede ser proporcionada por visión global o posicionamiento satelital. En esta situación, el robot se apoya

acérrimamente en la información de su ubicación para armar un mapa del entorno (Andrade & Llofriu, 2013).

Sin embargo, escenarios más complejos existen, donde no se conoce el entorno y tampoco se posee información sobre la ubicación exacta del robot. En este caso, el robot deberá generar un mapa y mantener su ubicación en el mismo de forma concurrente. Esta tarea resulta compleja por el hecho que para poder localizarse de forma precisa se necesita un mapa, y, por otro lado, para poder crear un mapa es indispensable estar localizado en forma precisa. Esta es la tarea que estudia el SLAM (Tejada & Benavides, 2014).

Algunas aplicaciones donde el SLAM es necesario son:

- Exploración del espacio
- Rescate en zonas de catástrofes
- Robots para trabajo doméstico
- Vehículos dirigidos de forma autónoma

2.4.1. Dificultades del SLAM

Si en si el problema del SLAM no era lo suficientemente complejo, a ello se suman factores o fuentes de incertidumbre que incrementan la dificultad de estimar de manera precisa la ubicación y el mapa de un entorno. Algunas de estas fuentes son:

- Ruido de los sensores
- Desplazamiento impreciso del robot
- Simetrías del entorno / Ciclos cerrados
- Observabilidad parcial
- Entorno dinámico
- Capacidad de computo

2.4.2. Enfoques

Para dar una solución al problema del SLAM, existen los enfoques que se listan a continuación:

- Bioinspirados
- Probabilísticos.

La forma en que procesan la información de entrada es la principal diferencia entre los dos enfoques. Actualmente, el enfoque probabilístico domina el campo y ha logrado implementaciones en entornos grandes y complejos (Milford & Wyeth, Spatial mapping and map exploitation: A bioinspired engineering perspective, 2007).

2.4.2.1. *SLAM Bioinspirados*

Este tipo de soluciones buscar imitar los sistemas neurológicos de algunos animales de laboratorio como simios y ratas, pero la mayoría no se pueden operar en entornos reales (Sanderhauf & Protzel, 2010).

Recientemente se ha implementado un sistema denominado Rat-SLAM a gran escala, capaz de confeccionar el mapa de un suburbio entero. Este sistema toma como inspiración las células grid y spatial view. Para calcular el movimiento se utilizan encoders y la estimación del mismo se logra con una cámara mediante comparación de imágenes sucesivas (Milford & Wyeth, Mapping a suburb with a single camera using a biologically inspired slam system, 2008).

Por otro lado, el concepto de Neurorobótica está en auge, aquí los robots o mejor dicho neurorobots tienen sistemas de control basados en el sistema nervioso, es decir el comportamiento del sistema de control simula la arquitectura del cerebro y su dinámica.

2.4.2.2. *SLAM Probabilísticos*

La clave de estos métodos radica en determinar la distribución de probabilidad de la posición del robot y del mapa del entorno a través del tiempo. La figura 2-7 ilustra el concepto de estimación como una distribución de probabilidad (Thrun, Burgard, & Fox, 2005).

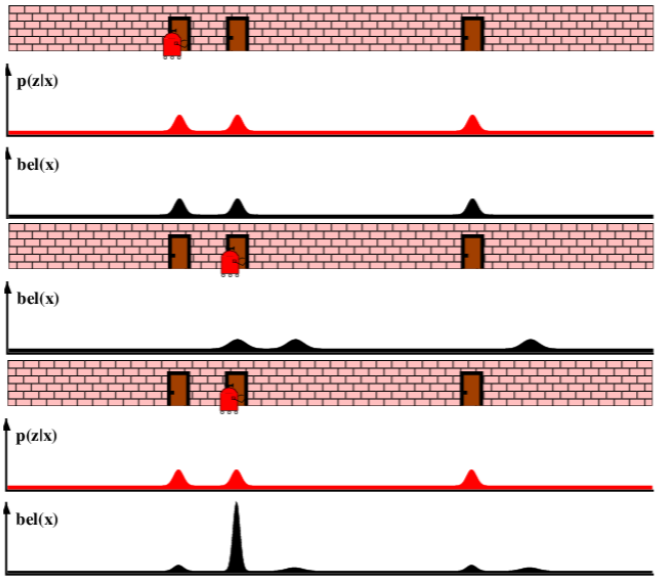


Figura 12-2: Estimación como distribución de probabilidad

Fuente: Thrun, Sebastian, 2005

La estimación de la probabilidad puede formularse con variables estocásticas como:

$$p(x_{1:t}|z_{1:t}, u_{1:t})$$

donde:

- $x_{1:t}$ es la posición a estimar en el instante de tiempo i
- $u_{1:t}$ es la información de movimiento propio
- $z_{1:t}$ es la información sensada

En la literatura científica la probabilidad puede estar representada por:

$$bel(x_{1:t}|z_{1:t}, u_{1:t})$$

Por lo tanto, se puede concluir, que para solucionar el problema de SLAM, se debe estimar la distribución de probabilidad para la variable que representa el estado del sistema x_i en cada instante de tiempo discreto de 1 al t.

2.4.3. Técnicas

Entre las principales técnicas se encuentras las siguientes:

- Filtros de Kalman
- Filtros de Partículas

- Optimización sobre grafos

2.4.3.1. *Filtros de Kalman*

El filtro de Kalman es un conjunto de ecuaciones que proporcionan una solución eficiente al problema de mínimos cuadrados. El filtro es muy potente en diferentes aspectos ya que proporciona una estimación para instantes pasados, presentes y futuros, incluso cuando la naturaleza exacta del sistema es desconocida a priori. El FKE (Filtro de Kalman extendido) es una técnica de linealización de modelos dinámicos no lineales para poder aplicar sobre ellos el filtro de Kalman. Con estas condiciones el FKE es utilizado para crear un mapa del entorno que simultáneamente se utilizará para localizar el robot (Verdero, 2017).

2.4.3.2. *Filtros de partículas*

El algoritmo de Filtro de Partículas proporciona una forma simple y efectiva de modelar procesos estocásticos con funciones de distribución de probabilidad y modelos de propagación arbitrarios. Se basan para esto en métodos secuenciales de Monte Carlo y para representar la densidad de probabilidades se utilizan puntos másicos "partículas" que son estados posibles del proceso, distribuidas sobre su espacio de estados (Carpenter & Fearnhead, 1999).

2.4.3.3. *Optimización sobre grafos*

En lugar de filtros, el SLAM de Grafos o denominado GraphSLAM resuelve el problema utilizado un algoritmo basado en marcas. Para esto, se modela el problema de SLAM como un grafo, donde las posiciones del robot x_i y marcas m^i son representadas por nodos (Frese, 2006).

2.5. Herramientas utilizadas para SLAM

En esta sección se describe en primera instancia las herramientas basadas en ROS para resolver el problema del SLAM y posteriormente el hardware más utilizado para obtener información del entorno donde la plataforma robótica móvil se desenvolverá.

2.5.1. Sistema Operativo Robótico (ROS)

ROS, el sistema operativo del robot, es un marco de código abierto para conseguir que los robots hagan cosas. ROS está destinado a servir como una plataforma de software común para las personas que están construyendo y utilizando robots. Esta plataforma común permite a las personas compartir código e ideas más fácilmente y, quizás más importante, significa que no tiene que pasar años escribiendo la infraestructura de software antes de que sus robots empiecen a moverse (Quingley, Gerkey, & Smart, 2016).

ROS ha sido notablemente exitoso. Al momento de escribir estas líneas, en la distribución oficial de ROS, hay más de 2.000 paquetes de software, escritos y mantenidos por casi 600 personas. Aproximadamente 80 robots comercialmente disponibles son soportados, y podemos encontrar por lo menos 1.850 documentos académicos que mencionan ROS. Ya no es necesario escribir todo desde cero, especialmente si se está trabajando con uno de los muchos robots que soportan ROS, y se puede pasar más tiempo pensando en la robótica, en lugar de manipular los bits y los controladores de dispositivos.

ROS fue desarrollado originalmente en el 2007 por el Laboratorio de Inteligencia Artificial de Stanford (SAIL) con el apoyo del proyecto Stanford AI Robot. A partir de 2008, el desarrollo continúa principalmente en Willow Garage, un instituto de investigación en robótica, con más de 20 instituciones que colaboran dentro de un modelo de desarrollo federado (Martinez & Fernández, 2013).

Los componentes básicos del framework ROS son paquetes ROS y vienen con capacidades listas para usar, como, por ejemplo, SLAM (Simultaneous Localization and Mapping) y AMCL (Adaptive Monte Carlo Localization). Estos paquetes en ROS se utilizan para realizar la navegación autónoma en robots móviles y el paquete MoveIt para la planificación de movimiento de robots manipuladores. Estas capacidades pueden ser utilizadas directamente en el software de un robot sin ningún tipo de molestia. Estas capacidades son su mejor forma de implementación, por lo que escribir un nuevo código para las capacidades existentes es como reinventar las ruedas. Además, estas capacidades son altamente configurables (Lentin, 2015).

Así como en Linux, las distribuciones de ROS se ajustan a las necesidades de cada proyecto. En la figura 2-8 se muestran como las distribuciones han surgido en el tiempo.

Distro	Release date	Poster	*Tuturtle*, turtle in tutorial	EOL date
ROS Lunar Loggerhead	May 23rd, 2017			May, 2019
ROS Kinetic Kame (**Recommended**)	May 23rd, 2016			April, 2021 (Xenial EOL)
ROS Jade Turtle	May 23rd, 2015			May, 2017
ROS Indigo Igloo	July 22nd, 2014			April, 2019 (Trusty EOL)
ROS Hydro Medusa	September 4th, 2013			May, 2015
ROS Groovy Galapagos	December 31, 2012			July, 2014
ROS Fuerte Turtle	April 23, 2012			--
ROS Electric Emys	August 30, 2011			--
ROS Diamondback	March 2, 2011			--
ROS C Turtle	August 2, 2010			--
ROS Box Turtle	March 2, 2010			--

Figura 13-2: Versiones de ROS

Fuente**:** ROS, Distributions, 2017

La estructura de un sistema ROS se puede interpretar mejor mediante niveles, y estos son:

- Sistema de archivos
- Grafo de Computación
- Comunidad

2.5.1.1. *Sistema de archivos ROS*

Similar a un sistema operativo, un programa ROS se divide en carpetas, y estas carpetas tienen archivos que describen sus funcionalidades:

- Paquetes: Los paquetes forman el nivel atómico de ROS. Un paquete tiene la estructura y el contenido mínimos para crear un programa dentro de ROS. Puede tener procesos de tiempo de ejecución ROS (nodos), archivos de configuración, etc.
- Manifiesto del paquete: Los manifiestos del paquete proporcionan información sobre un paquete, licencias, dependencias, indicadores de compilación, etc. Un manifiesto de paquetes se gestiona con un archivo denominado package.xml.
- Metapaquetes: Cuando desea agregar varios paquetes en un grupo, se utilizará Metapaquetes. En ROS Fuerte, este formulario para ordenar paquetes se llamaba pilas. Para mantener la simplicidad de ROS, se quitaron las pilas, y ahora, los metapaquetes realizan esta función. En ROS, existen muchos de estos metapaquetes, una de ellos es la pila de navegación.
- Manifiesto del Metapaquete: Los manifiestos de Metapaquete (package.xml) son similares a un paquete normal, pero con una etiqueta de exportación en XML. También tiene ciertas restricciones en su estructura.
- Tipos de mensajes (msg): Un mensaje es la información que un proceso envía a otros procesos. ROS tiene muchos tipos estándar de mensajes. Las descripciones de los mensajes se almacenan en my_package/msg/MyMessageType.msg.
- Tipos de servicio (srv): Las descripciones de servicio, almacenadas en my_package/srv/MyServiceType.srv, definen las estructuras de datos de solicitud y respuesta para los servicios proporcionados por cada proceso en ROS.

En la figura 9-2 se grafica lo descrito anteriormente.

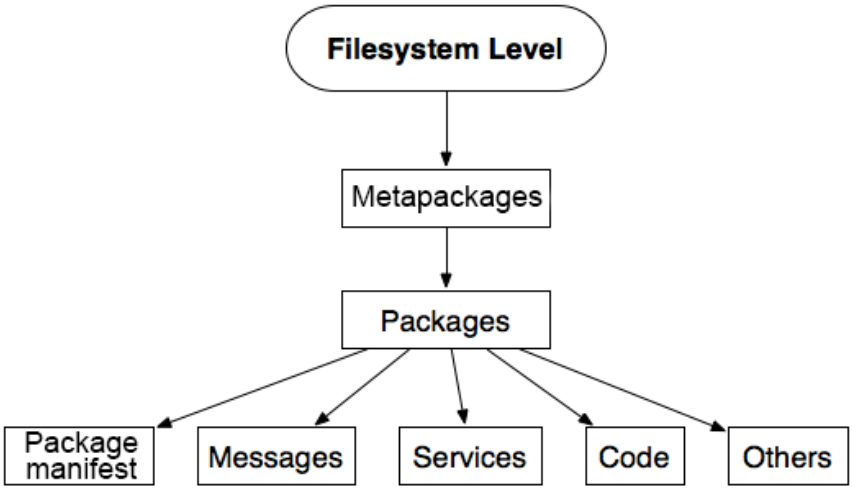


Figura 14-2: Sistema de archivos de ROS

Fuente: Fernández, Aaron, 2015

2.5.1.2. *Grafo de Computación de ROS*

El cálculo en ROS se realiza utilizando una red de procesos llamada nodos ROS. Esta red de cálculo se denomina 'computation graph'. Los conceptos principales son Nodos ROS, Maestro, Servidor de Parámetros, Mensajes, Tópicos, Servicios y Bolsas. Cada concepto contribuye al 'computation graph' de diferentes maneras. La figura 10-2 ilustra estos conceptos.

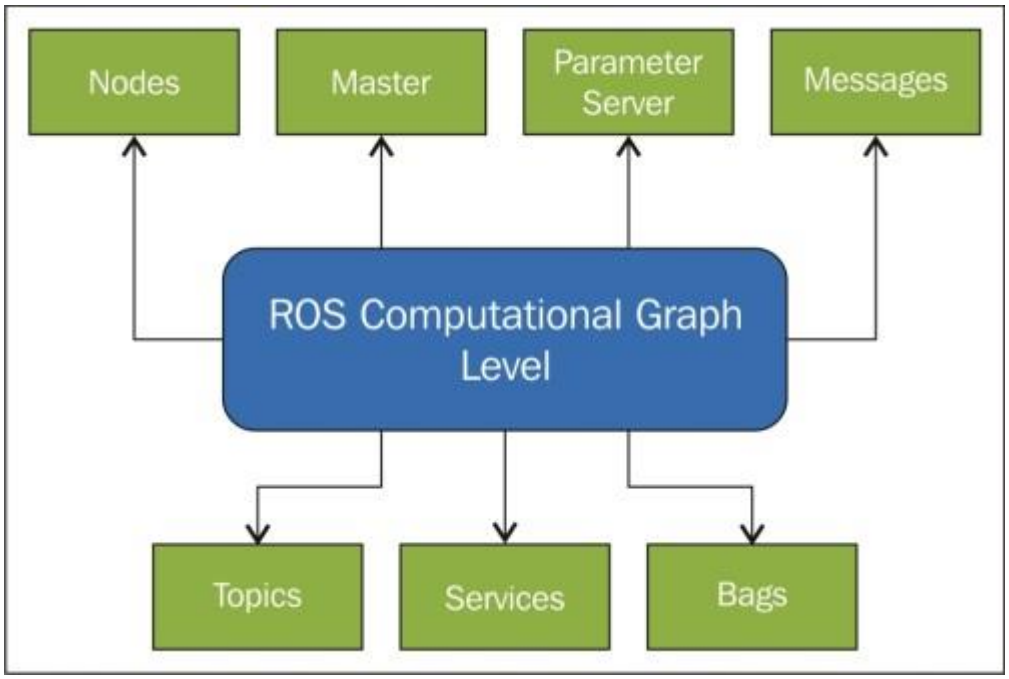


Figura 15-2: Computational graph de ROS

Fuente: Fernández, Aaron, 2015

A continuación, se describen cada uno de los conceptos del grafo de computación:

- Nodos: Los nodos son los procesos que realiza la computación. Cada nodo ROS se escribe utilizando las bibliotecas cliente de ROS como roscpp y rospy. Utilizando librerías cliente API, se puede implementar diferentes tipos de métodos de comunicación en nodos ROS. En un robot, habrá muchos nodos para realizar diferentes tipos de tareas. Utilizando los métodos de comunicación ROS, pueden comunicarse entre sí e intercambiar datos. Uno de los objetivos de los nodos ROS es construir procesos simples en lugar de un gran proceso con toda la funcionalidad.
- Maestro: El ROS Master proporciona registro de nombres y de búsqueda al resto de los nodos. Los nodos no podrán encontrarse, intercambiar mensajes o invocar servicios sin un maestro ROS. En un sistema distribuido, se ejecuta el nodo maestro en una computadora, y otros nodos remotos pueden comunicarse a través de él.
- Servidor de parámetros: El servidor de parámetros le permite mantener los datos almacenados en una ubicación central. Todos los nodos pueden acceder y modificar estos valores. El servidor de parámetros es una parte de ROS Master.
- Mensajes: Los nodos se comunican entre sí mediante mensajes. Los mensajes son simplemente una estructura de datos con el tipo de dato correspondiente, que puede contener un conjunto de datos y pueden ser enviados a otros nodos. Hay tipos primitivos estándar (entero, punto flotante, booleano, etc.) que son soportados por mensajes ROS. También podemos construir nuestros propios tipos de mensajes utilizando estos tipos estándar.
- Tópicos: Cada mensaje en ROS se transporta utilizando buses con nombre denominados tópicos. Cuando un nodo envía un mensaje a través de un tópico, entonces podemos decir que el nodo está publicando el tópico. Cuando un nodo recibe un mensaje a través de un tópico, podemos decir que el nodo está suscrito a este tópico. El nodo de publicación y el nodo de suscripción no son conscientes de la existencia del otro. Incluso podemos suscribir un tópico que podría no tener ningún editor. En resumen, la producción de información y su consumo se desacoplan. Cada tópico tiene un nombre único y cualquier nodo puede acceder a este tema y enviar datos a través de él siempre y cuando tengan el tipo de mensaje correcto.

- Servicios: En algunas aplicaciones de robot, un modelo de publicación/suscripción no será suficiente si se necesita una interacción solicitud/respuesta. El modelo de publicación / suscripción es un tipo de sistema de transporte unidireccional y cuando trabajamos con un sistema distribuido, se puede necesitar una interacción tipo solicitud/respuesta. Los servicios ROS se utilizan en estos casos. Podemos definir un servicio que contenga dos partes; uno es para las solicitudes y el otro es para las respuestas. Con ROS Services, se puede escribir un nodo servidor y un nodo cliente. El nodo servidor proporciona el servicio bajo un nombre y cuando el nodo cliente envía un mensaje de solicitud a este servidor, responderá y enviará el resultado al cliente. Es posible que el cliente necesite esperar hasta que el servidor responda. La interacción del servicio ROS es como una llamada de procedimiento remoto.
- Bolsas: Las bolsas son un formato para guardar y reproducir datos de mensajes ROS. Las bolsas son un mecanismo importante para almacenar datos, tales como datos de sensores, que pueden ser difíciles de recopilar pero que son necesarios para desarrollar y probar algoritmos de robots. Las bolsas son características muy útiles cuando trabajamos con mecanismos complejos de robots.

En resumen, el siguiente diagrama muestra una ilustración de cómo interactúa ROS Master con un nodo de publicación y suscripción, el nodo editor publica un tópico de tipo de cadena con un mensaje "Hello World" y el nodo suscriptor se suscribe a este tópico.

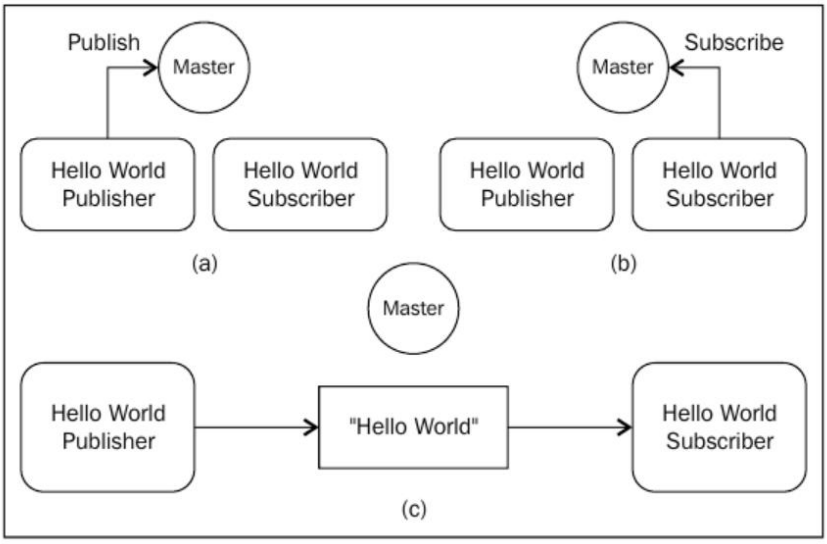


Figura 16-2: Comunicación entre nodos en ROS

Fuente: Fernández, Aaron, 2015

2.5.1.3. *Comunidad de ROS*

Estos son recursos de ROS que permiten a una nueva comunidad ROS intercambiar software y conocimiento. Los diversos recursos en estas comunidades son los siguientes:

- Distribuciones: Similar a la distribución de Linux, las distribuciones ROS son una colección versionada de meta paquetes que podemos instalar. La distribución ROS facilita la instalación y la recopilación del software ROS. Las distribuciones ROS mantienen versiones consistentes a través de un conjunto de software.
- Repositorios: ROS se basa en una red federada de repositorios de código, donde diferentes instituciones pueden desarrollar y liberar sus propios componentes de software de robot.
- ROS Wiki: La comunidad ROS Wiki es el principal foro para documentar información sobre ROS. Cualquiera puede crear una cuenta y contribuir con su propia documentación, proporcionar correcciones o actualizaciones, escribir tutoriales y más.
- Bug ticket: Si encontramos un error existente en el software o necesitamos agregar una nueva característica, podemos usar este recurso
- Listas de correo: La lista de correo de los usuarios de ROS es el principal canal de comunicación sobre las nuevas actualizaciones de ROS, así como un foro para hacer preguntas sobre el software ROS.
- Respuestas de ROS: Este recurso de sitio web ayuda a hacer preguntas relacionadas con ROS. Si se publica alguna duda en este sitio, otros usuarios de ROS pueden verla y dar soluciones.
- Blog: El blog de ROS se actualiza con noticias, fotos y videos relacionados con la comunidad ROS (http://www.ros.org/news).

2.5.2. Paquetes ROS para SLAM

A continuación, se narran los paquetes ROS que implementan las técnicas descritas previamente para dar solución al problema del SLAM.

ROS dispone de varios paquetes para realizar SLAM, ya sea usando visión mediante cámaras o utilizando sensores de rango como radares y láseres, entre ellos podemos mencionar los siguientes:

- vslam
- gmapping
- karto
- hector_slam

2.5.2.1. *Visual SLAM*

Vslam es código de investigación en estado experimental, no está soportado activamente, y se debe utilizar bajo propio riesgo. Visual SLAM utiliza la técnica de ajuste de haz disperso y cámaras como recolector de datos (ROS.org, 2017) . La figura 2-12 muestra este paquete en acción.

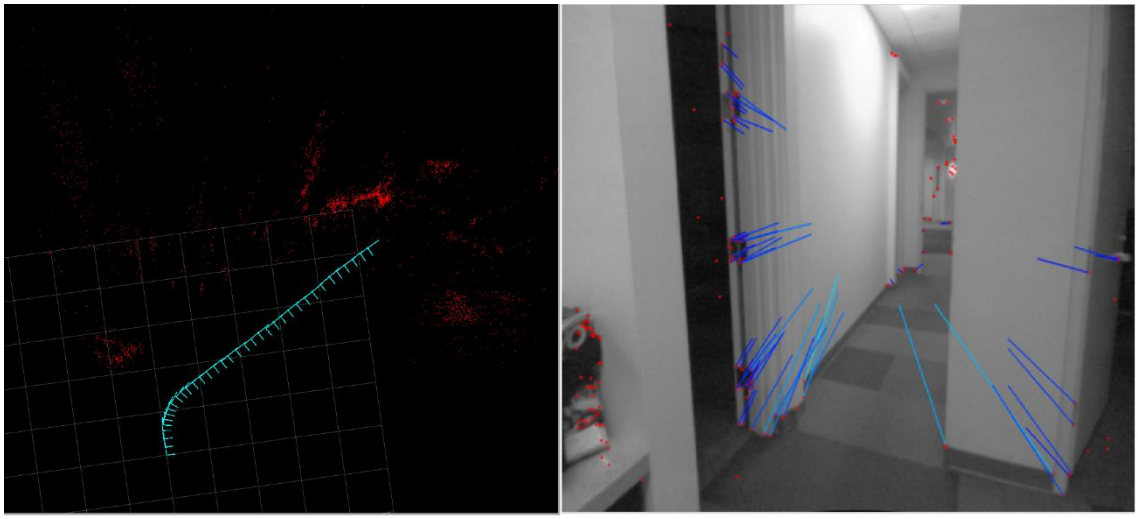


Figura 17-2: Vslam con datos estéreo

Fuente**:** ROS.org, 2017

2.5.2.2. *Gmapping*

Este paquete contiene un envoltorio de ROS para el paquete externo OpenSlam Gmapping. El paquete proporciona localización simultánea y mapeado basándose en la información de un láser 2D montado sobre el robot. Esto permite crear un mapa de rejilla de ocupación 2D (parecido a un plano de la planta de un edificio) a partir de los datos que provienen el láser y la información de posición recogida por el robot móvil (Rapado, 2016).

Recientemente se han introducido filtros de partículas de Rao-Blackwellized como medios efectivos para resolver el problema de SLAM. Este enfoque utiliza un filtro de partículas en el que cada partícula lleva un mapa individual del entorno. Se utilizan

técnicas adaptativas para reducir el número de partículas para el aprendizaje de mapas de cuadrícula. Esto reduce drásticamente la incertidumbre sobre la posición del robot en el paso de predicción del filtro (OpenSLAM.org, 2017). La figura 2-13 evidencia la potencia del paquete.

Figura 18-2: Mapa del campus Freiburg generado con gmapping

Fuente**:** OpenSLAM.org, 2017

2.5.2.3. *Karto*

La versión preliminar de este paquete era de código abierto, pero en junio de 2010, SRI International, una organización independiente de investigación y desarrollo tecnológico originalmente parte de la Universidad de Stanford, anunció Karto 2.0. SRI describe a Karto 2.0 como el sistema de SLAM con el más alto rendimiento disponible. Vendido comercialmente, Karto es un proyecto de desarrollo avanzado dentro del AI Center en SRI International (RBR, 2017).

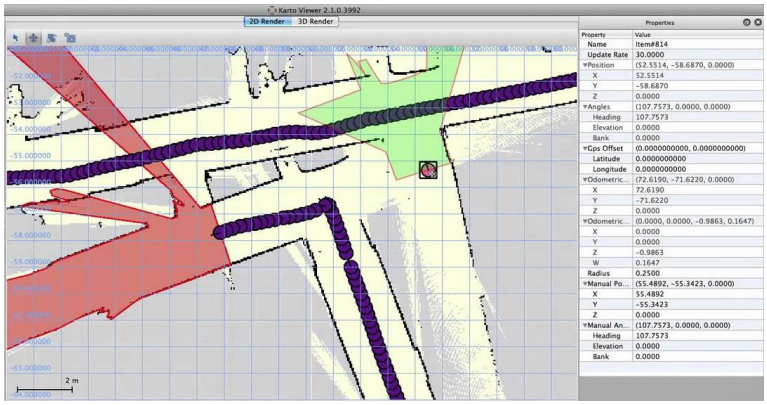


Figura 19-2: Mapa generado con Karto SDK

Fuente**:** kartorobotics.com, 2017

2.5.2.4. *Hector SLAM*

Hector_slam contiene paquetes de ROS relacionados con la ejecución de SLAM en entornos no estructurados como los encontrados en los escenarios de búsqueda y rescate urbano (USAR) del concurso RoboCup Rescue. Fue inicialmente desarrollado por el Team Hector. Este Equipo surgió como un esfuerzo interdisciplinario de investigadores de los Departamentos de Informática y Ingeniería Mecánica de TU Darmstadt dentro del programa de doctorado GRK 1362 en 2009. El nombre del equipo 'Hector' es acrónimo de Heterogeneous Cooperating Team Of Robots o Equipo Cooperativo Heterogéneo de Robots en español. La misión del equipo es investigar y desarrollar robots heterogéneos de búsqueda y rescate que cooperen entre sí y con un supervisor humano remoto para lograr una misión común (Team_Hector, 2017).

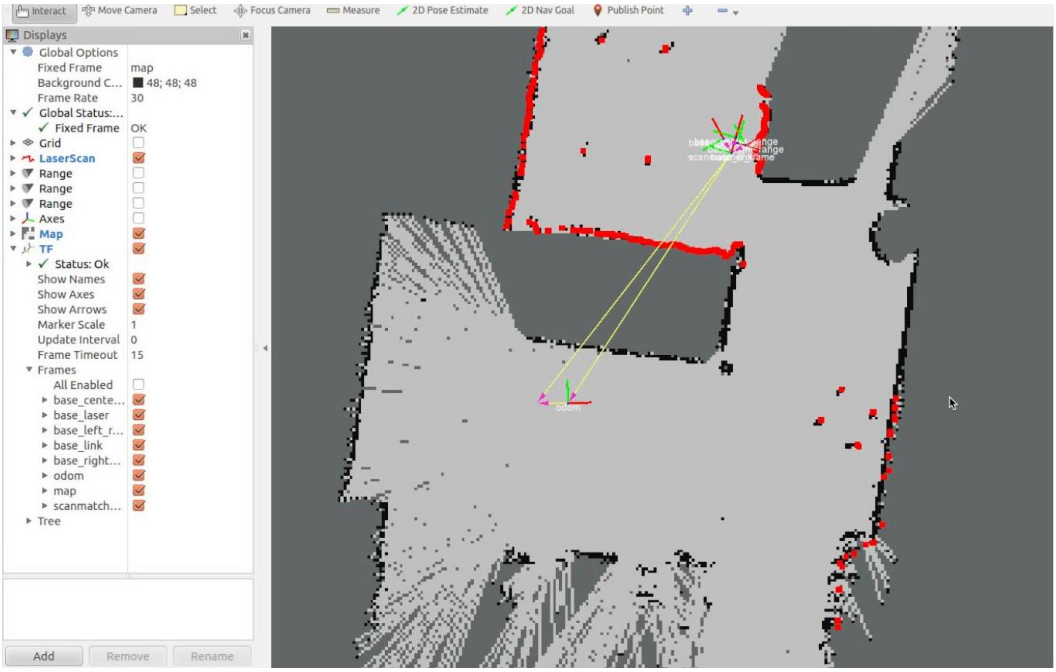


Figura 20-2: Mapa generado con Hector SLAM

Fuente: Team_Hector, 2017

2.5.3. Hardware para SLAM con soporte en ROS

En esta sección se describen el hardware compatible con ROS que permite realizar SLAM, específicamente a computadoras embebidas y a sensores de rango.

2.5.3.1. *Computadoras embebidas*

A continuación, se listan las algunas de las placas electrónicas que incorporan todo un computador en ellas y que son compatibles con ROS:

Tabla 1-2: Computadoras compatibles con ROS

Board	**Características técnicas**
Beagleboard 	Processor: AM335x 1GHz ARM® ▪ 512MB DDR3 RAM ▪ 4GB 8-bit eMMC on-board flash storage ▪ 3D graphics accelerator ▪ NEON floating-point accelerator ▪ 2x PRU 32-bit microcontrollers Connectivity ▪ USB client for power & communications ▪ USB host ▪ 802.11b/g/n and Bluetooth 4.1 plus BLE ▪ HDMI ▪ 2x 46 pin headers
Odroid C2 	* Amlogic ARM® Cortex®-A53(ARMv8) 1.5Ghz quad core CPUs * Mali™-450 GPU (3 Pixel-processors + 2 Vertex shader processors) * 2Gbyte DDR3 SDRAM * Gigabit Ethernet * HDMI 2.0 4K/60Hz display * H.265 4K/60FPS and H.264 4K/30FPS capable VPU * 40pin GPIOs + 7pin I2S * eMMC5.0 HS400 Flash Storage slot / UHS-1 SDR50 MicroSD Card slot * USB 2.0 Host x 4, USB OTG x 1 (power + data capable) * Infrared(IR) Receiver * Ubuntu 16.04 or Android 5.1 Lollipop based on Kernel 3.14LTS
Raspberry Pi 3 	• Quad Core 1.2GHz Broadcom BCM2837 64bit CPU • 1GB RAM • BCM43438 wireless LAN and Bluetooth Low Energy (BLE) on board • 40-pin extended GPIO • 4 USB 2 ports • 4 Pole stereo output and composite video port • Full size HDMI • CSI camera port for connecting a Raspberry Pi camera • DSI display port for connecting a Raspberry Pi touchscreen display • Micro SD port for loading your operating system and storing data • Upgraded switched Micro USB power source up to 2.5A

NVIDIA Jetson TK1 	> Tegra K1 SOC > NVIDIA Kepler GPU with 192 CUDA Cores > NVIDIA 4-Plus-1™ Quad-Core ARM® Cortex™-A15 CPU > 2 GB x16 Memory with 64-bit Width > 16 GB 4.51 eMMC Memory > 1 Half Mini-PCIE Slot > 1 Full-Size SD/MMC Connector > 1 Full-Size HDMI Port > 1 USB 2.0 Port, Micro AB > 1 USB 3.0 Port, A > 1 RS232 Serial Port

Fuente: ALDAS, Iván, 2017

Sin lugar a duda, hay una infinidad de placas en el mercado. En la tabla 3-1 aparecen las más accesibles económicamente. En realidad, cualquier placa que pueda correr la distribución de Linux denominada Ubuntu es buena candidata para alojar ROS, ya que es la versión más soportada por la comunidad.

2.5.3.2. *Sensores de rango utilizados para SLAM*

A continuación, se listan algunos sensores de rango utilizados para SLAM con soporte en ROS:

Tabla 2-1: Sensores de rango con soporte ROS

Sensor	**Características técnicas**
Hokuyo URG-04LX-UG01 Scanning Laser Rangefinder 	• Power source: 5VDC±5 (USB Bus power) • Light source: Semiconductor laser diode(785nm), Laser safety class 1 • Measuring area: 20 to 5600mm (white paper with 70mm×70mm), 240° • Accuracy: 60 to 1,000mm : ±30mm, 1,000 to 4,095mm : ±3 of measurement • Angular resolution: Step angle : Approx. 0.36°(360°/1,024 steps) • Scanning time: 100ms/scan • Noise: 25dB or less • Interface: USB2.0/1.1[Mini B](Full Speed) • Command System: SCIP Ver.2.0 • Ambient illuminance*1: Halogen/mercury lamp: 10,000Lux or less, Florescent: 6000Lux(Max) • Ambient temperature/humidity: -10 to +50 degrees C, 85% or less(Not condensing, not icing) • Vibration resistance: 10 to 55Hz, double amplitude 1.5mm each 2 hour in X, Y and Z directions • Impact resistance: 196m/s^2, Each 10 time in X, Y and Z directions • Weight: Approx. 160g

RPLIDAR A2 360° Laser Scanner 	• Model: RPLDIAR-A2 • Distance Range: 0.15 - 6 m • Angular Range: 0-360 degree • Distance Resolution: <0.5 (0.15~1.5 meters) and <1% of the distance (All distance range) • Angular Resolution: 0.9degree • Sample Duration: 0.25 millisecond • Sample Frequency 4000Hz • Scan Rate: 10Hz • Requires at least 1.5A@5V • Barrel ID: 0.65 mm • Barrel OD: 2.80 mm • Weight: 340g
LIDAR-Lite 3 Laser Rangefinder 	• Accuracy: +/- 2.5cm • Range: 0-40 meters • Power: 4.75-5 VDC; 6 V Max • Current consumption: 105ma, idle; 130ma, continuous • Rep rate: 1-500Hz • Interface: I2C or PWM • Operating temperature: -20 to 60° C • Laser wave length/Peak power: 905 nm/1.3 watts • Beam divergence: 4 m Radian X 2 m Radian • Optical aperture: 12.5 mm

Fuente**:** robotshop.com, 2017

CAPÍTULO III

3. DISEÑO E IMPLEMENTACIÓN

En este capítulo, se expondrá todo el proceso de diseño e implementación del proyecto, iniciando con una visión global de los subsistemas inmersos en él, y culminando con las configuraciones finales de control.

3.1. Esquema global del proyecto

En la figura 3-1 se muestra el esquema general del presente proyecto que sirve de base para el diseño de detalle.

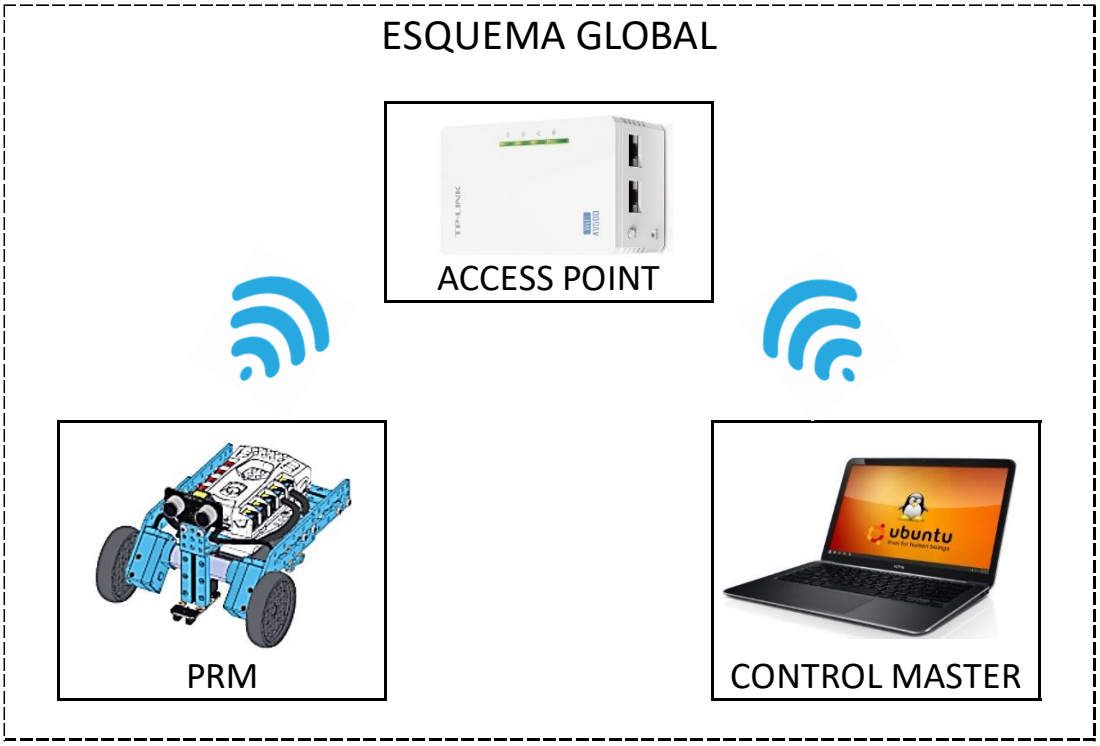


Figura 1-3: Esquema global del proyecto

Fuente: ALDAS, Iván, 2017

Como se puede apreciar en la figura anterior, el subsistema denominado PRM (Plataforma Robótica Móvil), es el objetivo primordial del proyecto y su diseño se basó en la simple premisa de hacerlo accesible a la economía de un estudiante de educación superior promedio, pero que incorpore las características de un robot autónomo o

semiautónomo con capacidad de realzar SLAM. Por lo que se especificó las siguientes consideraciones de diseño:

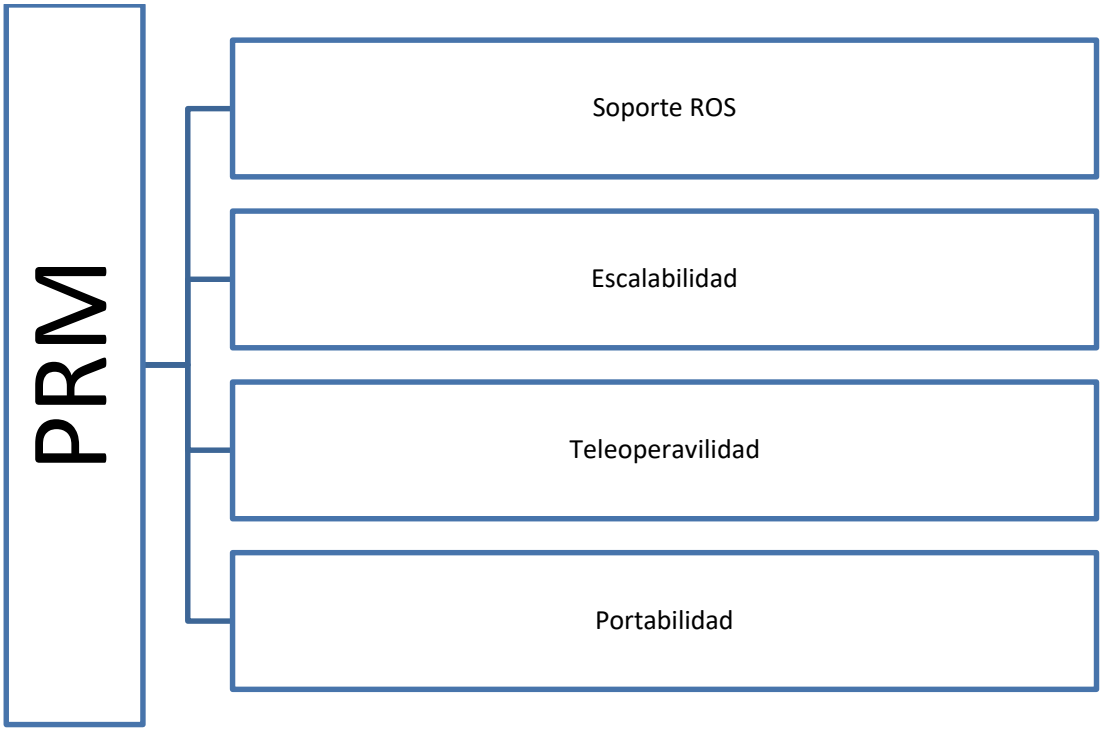


Figura 2-3: Consideraciones de diseño de la PRM

Fuente: ALDAS, Iván, 2017

En función de lo anterior, el diseño de la PRM se ha divido en las siguientes secciones:

a) Diseño estructural
b) Diseño del control hardware
c) Diseño del control software

3.2. Diseño estructural

El diseño preliminar de la estructura de la PRM contemplaba el modelado 3D de toda la estructura, basado en un robot de tracción diferencial. Para ello se haría uso ya sea de Autodesk Inventor o SolidWorks, ambos paquetes software con soporte de modelado paramétrico.

En poco tiempo esta idea fue rechazada debido al limitado acceso a una impresora 3D, esencialmente necesaria para materializar los modelados. En su lugar, se optó por adquirir una estructura comercial de base coste y adaptarla a los objetivos del presente trabajo. Esta estructura debía cumplir con las características de robustez, escalabilidad y

portabilidad originalmente establecidas. Es por ello que se adquirió el kit de robótica educacional denominado mBot Ranger, fabricado y distribuido por Makeblock. La figura 3-3 se visualiza el kit.

Figura 3-1: mBot Ranger de Makeblock

Fuente: makeblock.com, 2017

3.2.1. Tipos de estructura:

El kit del mBot Ranger, incluye partes mecánicas que permiten ensamblar tres tipos de estructura base de robots, siendo estas:

- Land Raider (Todo terreno)
- Dashing Raptor (Velocista)
- Nervous Bird (Equilibrista)

En la figura 3- 4 se ilustran los tres tipos.

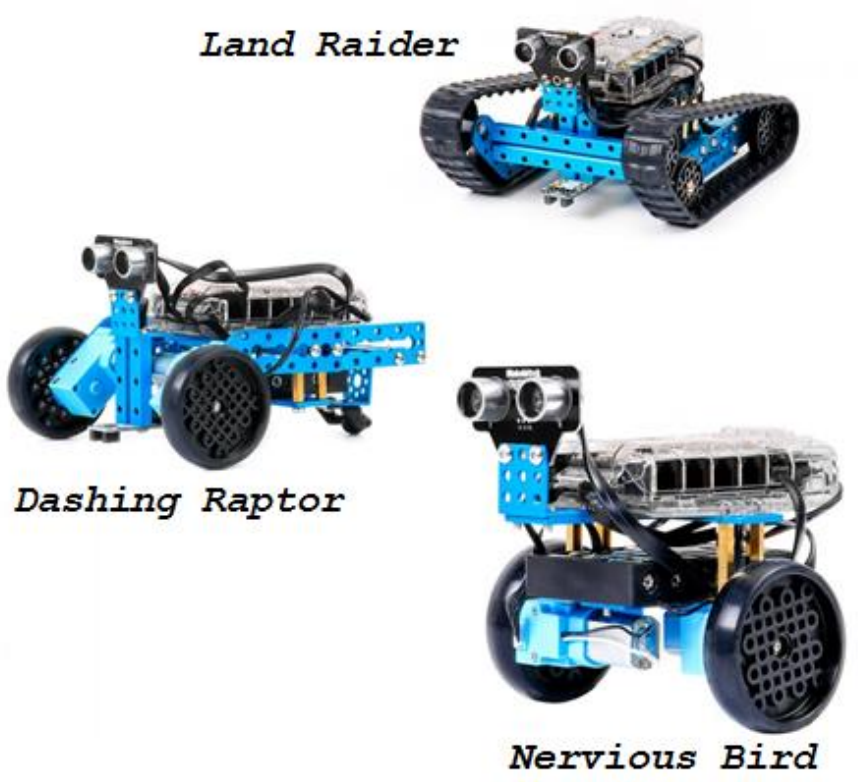


Figura 4-3: Motor para brazo dorado 1 y 2

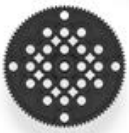
Fuente**:** makeblock.com, 2017

La estructura base que más ajusta a los propósitos del presente proyecto, sin lugar a dudas es la del Dashing Raptor ya que proporciona gran espacio libre para colocar las placas controladoras y sensores descritos en breve.

3.2.2. Ensamblaje de la estructura:

Antes de ilustrar el proceso de ensamblaje de la estructura, es necesario listar las partes de la misma. En la tabla 3-1 se encuentra la lista mencionada anteriormente.

Tabla 1-3: Lista de partes mecánicas

PARTE	FIGURA DE REFERENCIA	CANTIDAD
Soporte Motor		2
Soporte lateral		2
Soporte horizontal		2
Soporte tipo T		1
Soporte Holder Batería		1
Holder Batería		1
Soporte Rueda Universal		1
Rueda Universal		1
Ruedas 90T		2
Llanta 90T		2
Separador M4*30		4
Tornillo M4*8		18
Tornillo M4*14		4
Tuerca M4		6

Fuente: ALDAS, Iván, 2017

Con ayuda de la lista anterior, las figuras siguientes detallan el proceso de ensamblaje:

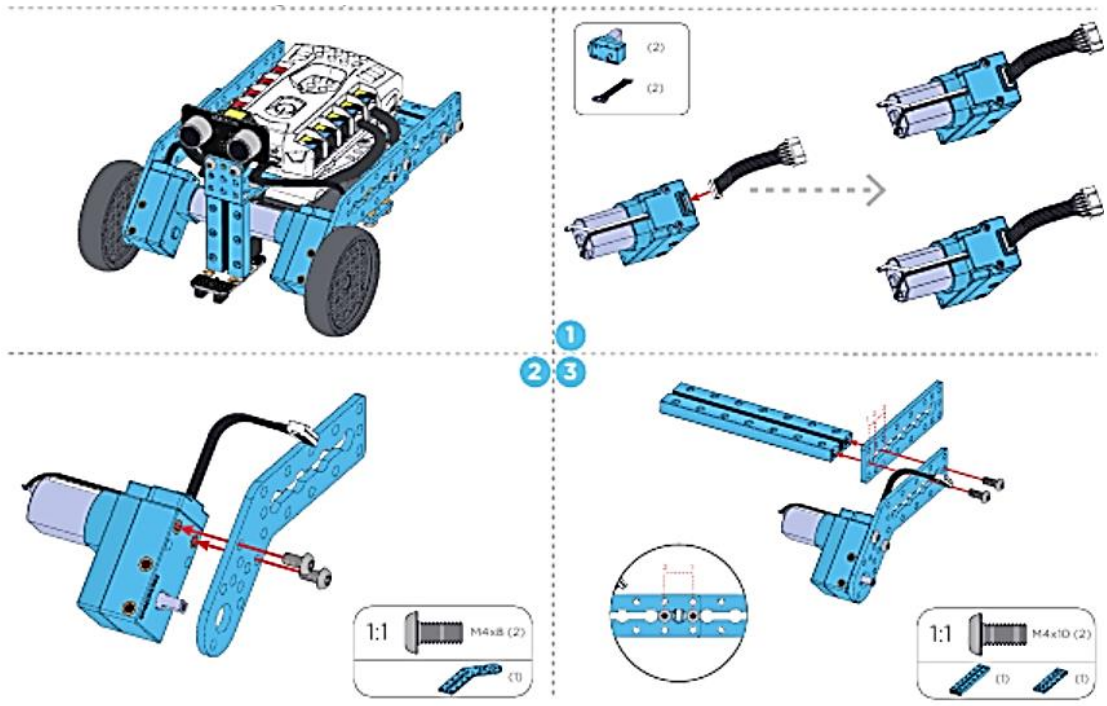


Figura 5-3: Ensamblaje - pasos 1-3

Fuente: makeblock.com, 2017

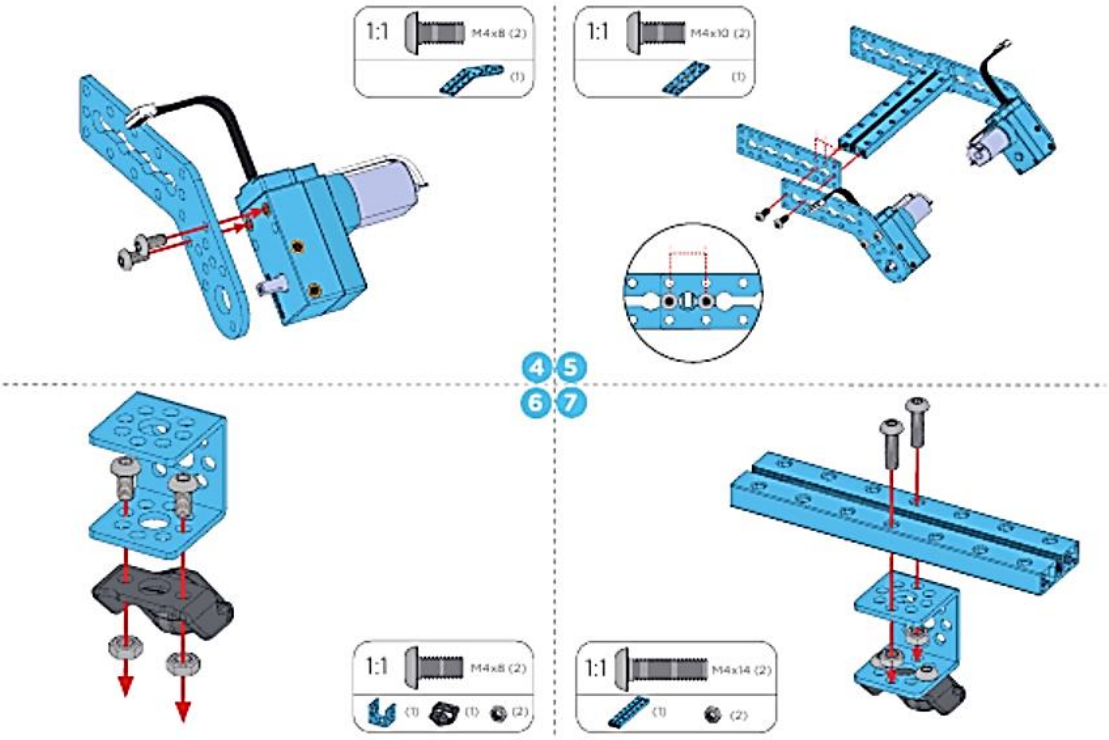


Figura 6-3: Ensamblaje - pasos 4-7

Fuente: makeblock.com, 2017

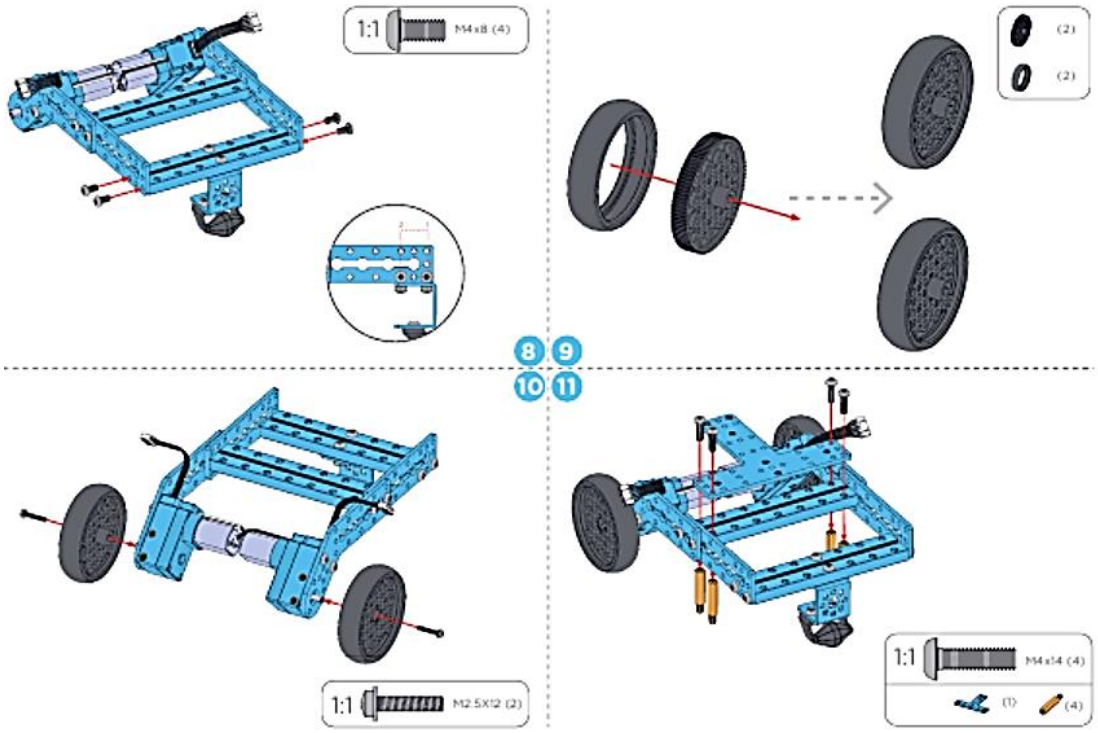


Figura 7-3: Ensamblaje - pasos 8-11

Fuente**:** makeblock.com, 2017

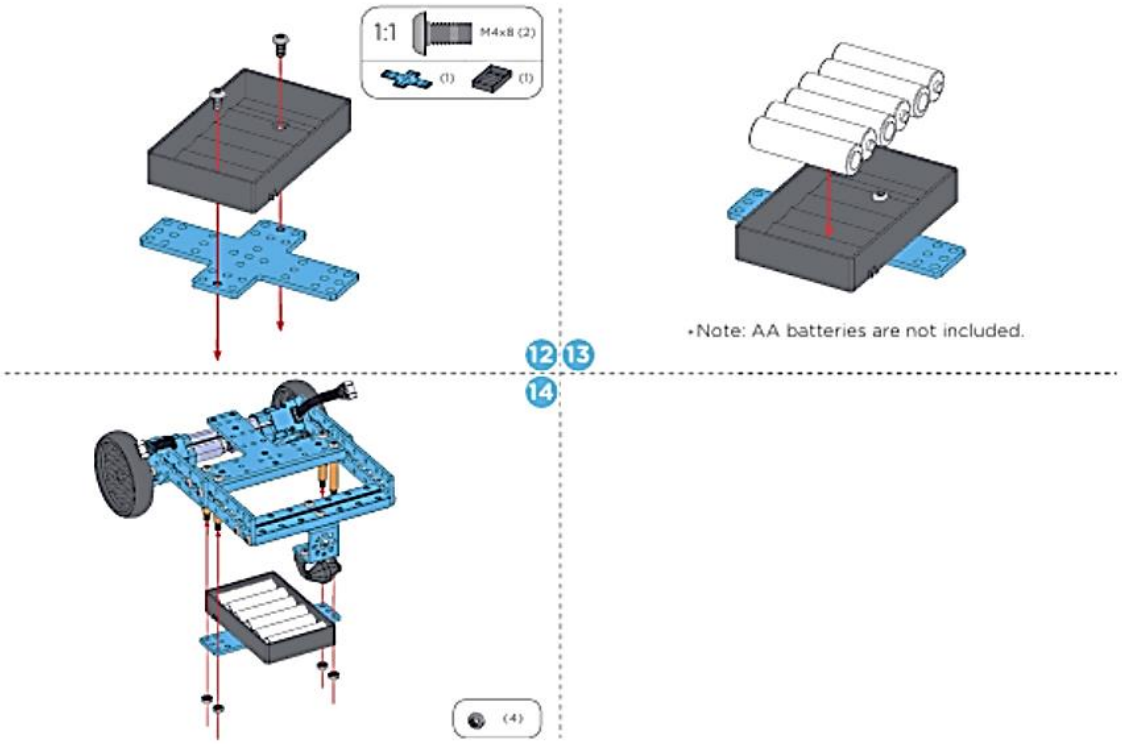


Figura 8-3: Ensamblaje - pasos 12-14

Fuente**:** makeblock.com, 2017

3.3. Diseño del control hardware

En esta sección, se describen los elementos utilizados para controlar la PRM, incluyendo sensores y actuadores, así como su integración.

3.3.1. Diagrama de bloques de la PRM

La PRM cuenta con los siguientes bloques funcionales:

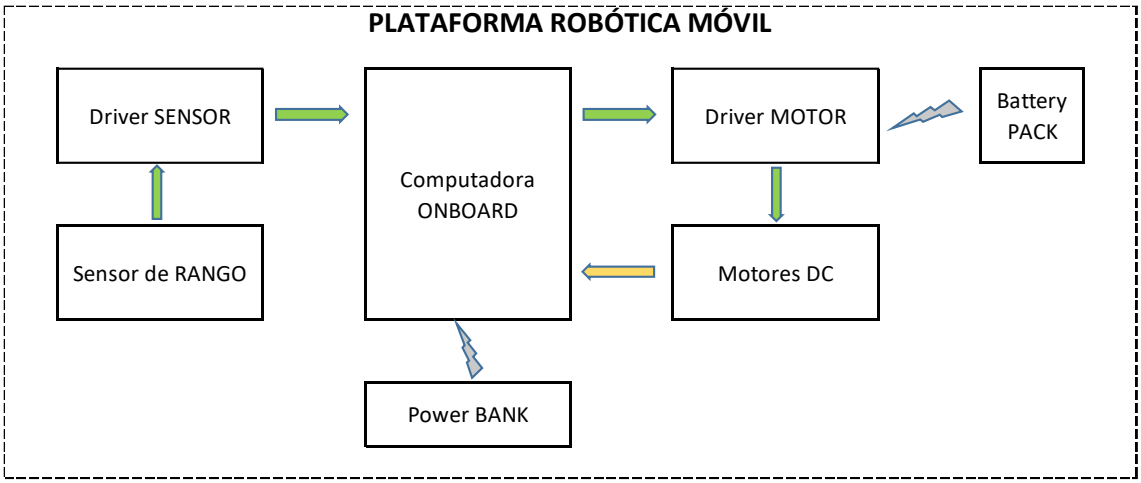


Figura 9-3: Bloques funcionales de la PRM

Fuente: ALDAS, Iván, 2017

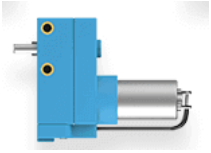
A continuación, se explica la función de cada uno de los bloques:

- Computador ONBOARD: Encargada de gestionar los movimientos de la PRM, recolectar los datos del sensor de rango y publicarlos en una red interna.
- Sensor de RANGO: Obtiene las distancia y posición de los objetos del entorno en 360°.
- Driver SENSOR: Permite la comunicación entre el sensor de rango y el computador onboard.
- Motores DC: Proporcionan la fuerza motriz a la PRM.
- Driver MOTORES: Gestiona la energía para los motores
- Power BANK: Proporciona la alimentación regulada de 5V para la computadora onboard.
- Battery PACK: Proporciona la alimentación aproximada de 9V para los motores.

3.3.2. Criterios de selección de componentes electrónicos

En la tabla 3-2 se listas todos los componentes con su criterio de selección correspondiente.

Tabla 2-3: Lista de componentes eléctricos/electrónicos

PARTE	FIGURA DE REFERENCIA	CANTIDAD	CRITERIO DE SELECCIÓN
Raspberry Pi 3		1	-Soporte ROS -Ubuntu Mate -Wifi integrado -Puertos USB disponibles -Alimentación de 5Vdc -Bajo precio
RaspiRobot V3f		1	-Compatible con RP 3 -Driver TB6612FNG de 2A. -Acceso directo a los GPIO
Rplidar A2		1	-Soporte ROS -Escaneo de 360° -Peso reducido -Precio moderado
Adaptador USB		1	-Compatible con Rplidar -Conexión USB
Motor M180		2	-Encoder óptico de cuadratura integrado con resolución de 360 pulsos. -1400 RPM -Alimentación de 7,4V. -Torque de 800 g.cm
Banco de Batería		1	-Capacidad de 10000 mA -Dos puertos de carga

Pilas AA		6	-Compatible con el Holder de pilas incluido en el mBot.

Fuente: ALDAS, Iván, 2017

En el anexo 1 se encuentran las hojas técnicas de los componentes principales.

3.3.3. Evolución de la PRM a la RasPi_mBot

Una vez que se ha seleccionado los componentes, el diseño conceptual de la PRM se ha materializado en la RasPi_mBot. En la figura siguiente se presenta la actualización de los bloques funcionales:

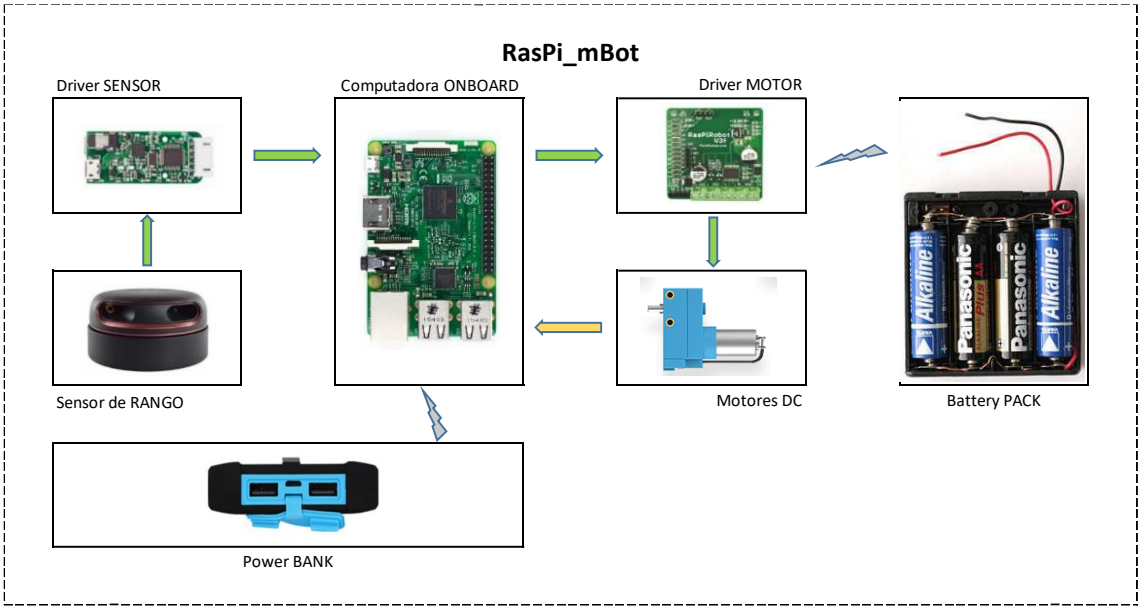


Figura 10-3: Bloques funcionales de la RasPi_mBot.

Fuente: ALDAS, Iván, 2017

3.4. Diseño del control software

Para que la RasPi_mBot pueda realizar SLAM, es necesario realizar varias tareas a nivel de software en los siguientes componentes:

- Computador Control MASTER
- Raspberry Pi 3

3.4.1. Setup del Computador Control MASTER

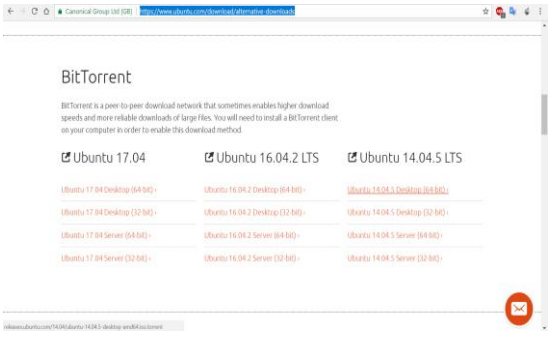
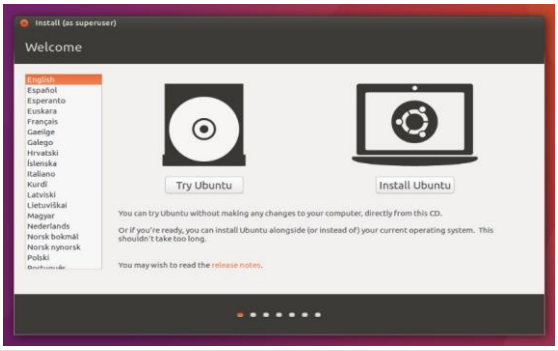
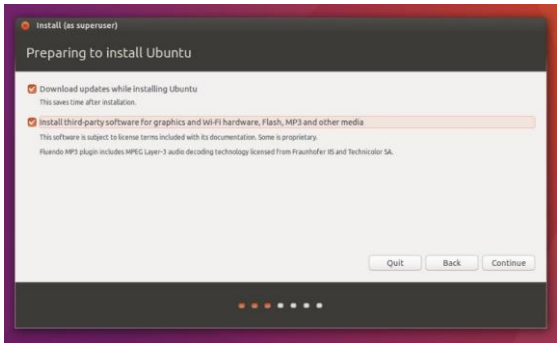
Una Laptop con procesador Intel CORE i7 con 8Gb de memoria RAM, hará las funciones del computador Control MASTER, la cual se encarga de enviar ordenes de funcionamiento a la RasPi_mBot, por lo que es necesario ejecutar las siguientes operaciones:

- Instalación Ubuntu 14.04
- Instalación ROS Indigo y paquetes SLAM
- Configuración IP estática

3.4.1.1. *Instalación de Ubuntu 14.04*

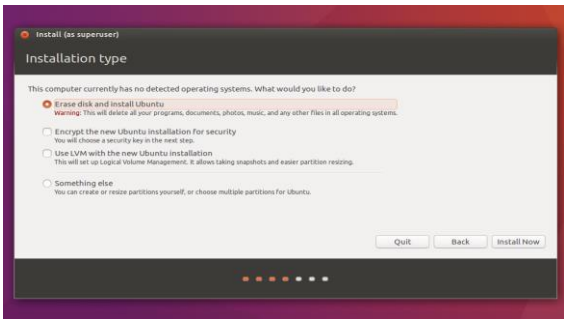
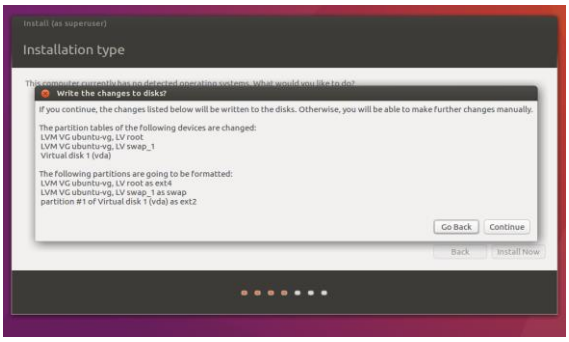
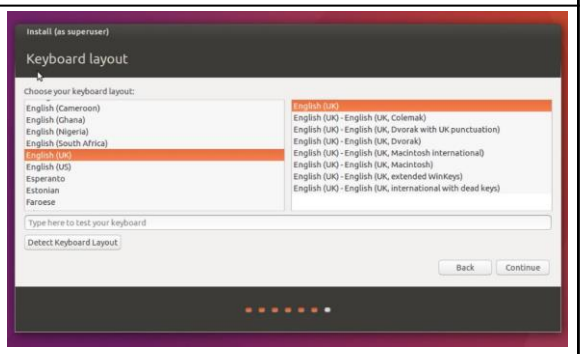
En las tablas siguientes se detalla el proceso de instalación.

Tabla 3-1: Instalación de Ubuntu 14.04

PASO 0	**REQUISITOS PRELIMINARES** -Conectar el computadora portátil a una fuente de alimentación. -Disponer de al menos 5 GB de espacio de almacenamiento libre. -Tener acceso a un DVD o una unidad flash USB que contenga la versión de Ubuntu 14.04	
PASO 1	**Descarga de imagen ISO** Descargar ingresando a los sigiuentes links: https://www.ubuntu.com/download/desktop https://www.ubuntu.com/download/alternative-downloads	
PASO 2	**Arranque desde una unidad flash USB** La mayoría de las computadoras se iniciarán automáticamente desde USB. Basta con insertar la unidad flash USB y encender el ordenador o reiniciarlo. Aparecera la ventana de bienvenida, solicitando que elija el idioma e instale o pruebe Ubuntu.	
PASO 3	**Preparado para instalar Ubuntu** Después de elegir instalar Ubuntu desde la ventana de bienvenida, se le preguntará sobre las actualizaciones y el software de terceros. Se aconsejamos activar la descarga de las actualizaciones y la instalación de software de terceros.	

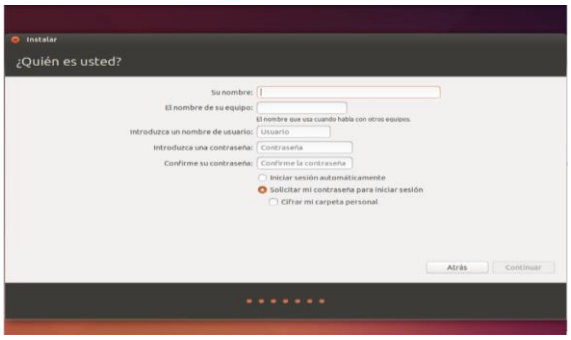
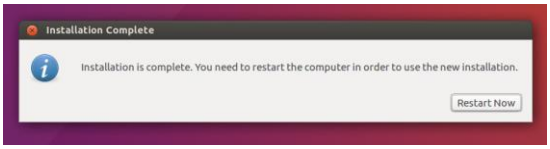
Fuente: ALDAS, Iván, 2017

Tabla 4-3: Instalación de Ubuntu 14.04 continuación I

PASO 4	**Asignar espacio en la unidad** Utilizar las casillas de verificación para elegir si desea instalar Ubuntu junto con otro sistema operativo, eliminar su sistema operativo existente y reemplazarlo con Ubuntu. Ubuntu necesita alrededor de 4,5 GB de espacio para una instalación mínima. Pero obviamente se necesitará espacio para instalar otros paquetes.	
PASO 5	**Comienza la instalación** Después de configurar el almacenamiento, haga clic en el botón "Instalar ahora". Aparecerá un pequeño panel con una visión general de las opciones de almacenamiento que ha elegido, con la posibilidad de volver si los detalles son incorrectos. Haga clic en Continuar para iniciar el proceso de instalación o en volver para modificar.	
PASO 6	**Seleccionar ubicación** Si está conectado a Internet, su ubicación se detectará automáticamente. Compruebe que su ubicación es correcta y haga clic en "Continuar".	
PASO 7	**Seleccionar Layout de teclado** Seleccionar el layout preferido y dar clic en "Continuar". Siempre se pueden cambiar los diseños de teclado y añadir esquemas adicionales desde el escritorio después de la instalación.	

Fuente: ALDAS, Iván, 2017

Tabla 5-3: Instalación de Ubuntu 14.04 continuación II

PASO 8	**Detalles de registro** Ingresar su nombre, el nombre del equipo y un nombre de usuario. Éstos se pueden cambiar fácilmente si usted prefiere. El nombre del equipo es la forma en que su computadora aparecerá en la red, mientras que el nombre de usuario será su nombre de usuario y de cuenta. A continuación, ingrese una contraseña segura. El instalador le informará si es demasiado débil.	
PASO 9	**Instalación en segundo plano** La instalación ahora se completará en segundo plano. La ventana de instalación le enseña un poco acerca de lo increíble que es Ubuntu. Dependiendo de la velocidad de la máquina y de la conexión de red, esto sólo debe tomar unos minutos.	
PASO 10	**Instalación completa** Después de que todo se ha instalado y configurado, una pequeña ventana aparecerá pidiendo que reinicie la máquina. Hacer clic en Reiniciar ahora y extraer el DVD o la unidad flash USB.	

Fuente: ALDAS, Iván, 2017

3.4.1.2. *Instalación de ROS Indigo y paquetes SLAM*

Los pasos a continuación permiten instalar ROS en Ubuntu 14.04 y configurar el ambiente de trabajo. La version de ROS para Ubuntu 14.04 es llamada Indigo. En las siguientes tablas se detalla el proceso de instalación.

Tabla 6-3: Instalación de ROS Indigo y paquetes SLAM

PASO 1	**Permitir todas las fuentes de ROS**
	Se configura la computadora para aceptar todo el software de packages.ros.org. ROS Indigo solo soporta paquetes Debian en las versiones de Ubuntu Saucy (13.10) y Trusty (14.04).
	En una ventana de terminal, ejecutar el siguiente comando: **sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'**
PASO 2	**Configuración del key**
	En una misma ventana de terminal, ejecutar el siguiente comando:
	sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 –recv- key 421C365BD9FF1F717815A3895523BAEEB01FA116
PASO 3	**Instalación general**
	Primero se debe asegurar que Ubuntu este actualizado, mediante: **sudo apt-get update**
	Luego que la actualización se complete, se puede instalar la versión completa de Indigo, la cual es la mas recomendada, mediante: **sudo apt-get install ros-indigo-desktop-full**
PASO 4	**Instalación de paquetes SLAM**
	Hay muchos paquetes que no vienen en la instalación full, para el presente proyecto son necesarios los siguientes paquetes que permiten implementar SLAM.
	En la ventana de terminal, ejecutar los siguientes comandos, uno a la vez: **sudo apt-get install ros-indigo-slam-gmapping** **sudo apt-get install ros-indigo-hector-slam**
PASO 5	**Inicialización de ROS**
	Ejecutar los comandos siguientes, uno a la vez:
	sudo rosdep init **rosdep update**
PASO 6	**Configuración del entorno de trabajo para Ubuntu**
	Es conveniente que las variables del entorno ROS sean automáticamente añadidas a la sesión bash cada vez que aperturamos una nueva ventana shell, eso se logra gracias a:
	echo "source /opt/ros/indigo/setup.bash" >> ~/.bashrc **source ~/.bashrc**
PASO 7	**Obtención de rosinstall**
	Una herramienta de comandos de línea frecuentemente utilizada en ROS es rosinstall, que se distribuye por separado. Esta herramienta permite descargar fácilmente paquetes ROS de diferentes fuentes con un solo comando.
	Para instalar esta herramienta en Ubuntu, ejecutar: **sudo apt-get install Python-rosinstall**
PASO 8	**Creación del entorno de trabajo ROS**
	Para crear y construir un entorno de trabajo (catkin workspace), ejecutar:
	mkdir -p ~/catkin_ws/src **cd ~/catkin_ws/** **catkin_make**

Fuente**:** wiki.ros.org, 2017

En el último paso de instalación, el comando catkin_make permite trabajar con catkin workspaces. Al ejecutar por primera vez en el entorno de trabajo, se crea un archivo denominado CMakeLists.txt en el directorio 'src'. Adicionalmente, en el directorio actual se crean los subdirectorios 'build' y 'devel'. Dentro de la carpeta 'devel' se crean archivos setup.*sh.

Para establecer la fuente del nuevo archivo setup.*sh, ejecutar:

source devel/setup.bash

Para asegurar que el espacio de trabajo y las variables del entorno ROS estén incluidos en el directorio actual, ejecutar:

echo $ROS_PACKAGE_PATH

Si todo anda bien, se debe obtener la siguiente respuesta al comando anterior:

/home/youruser/catkin_ws/src:/opt/ros/indigo/share

Al finaliza la instalación de ROS Indigo en el computador Control MASTER, se tendrá la siguiente estructura de directorios y archivos:

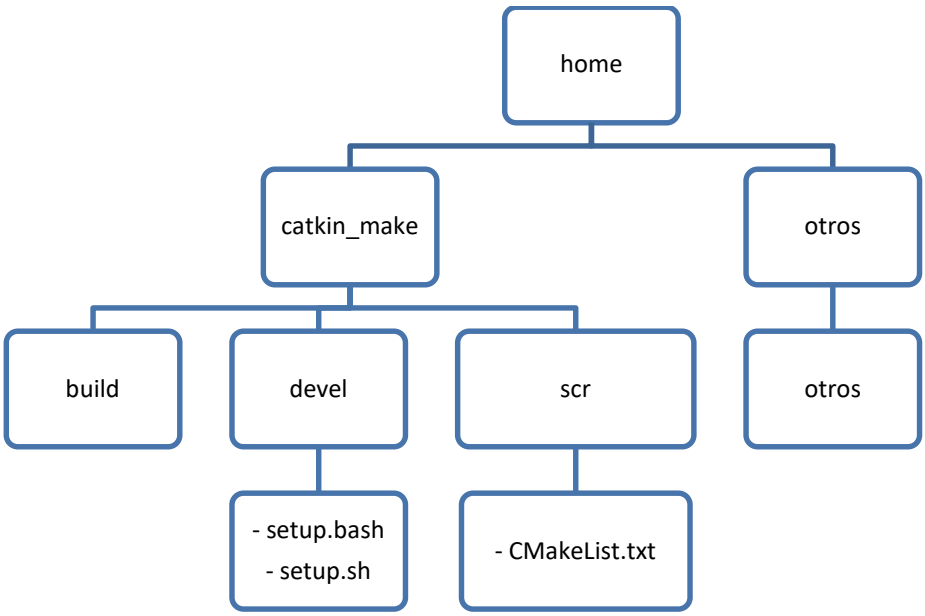


Figura 11-3: Estructura del entorno de trabajo ROS

Fuente: ALDAS, Iván, 2017

En la figura 3-12 en cambio se visualiza la estructura de archivos del directorio ROS en donde se encuentran los paquetes que fueron instalados anteriormente incluido los paquetes de SLAM, en los cuales se realiza especial énfasis.

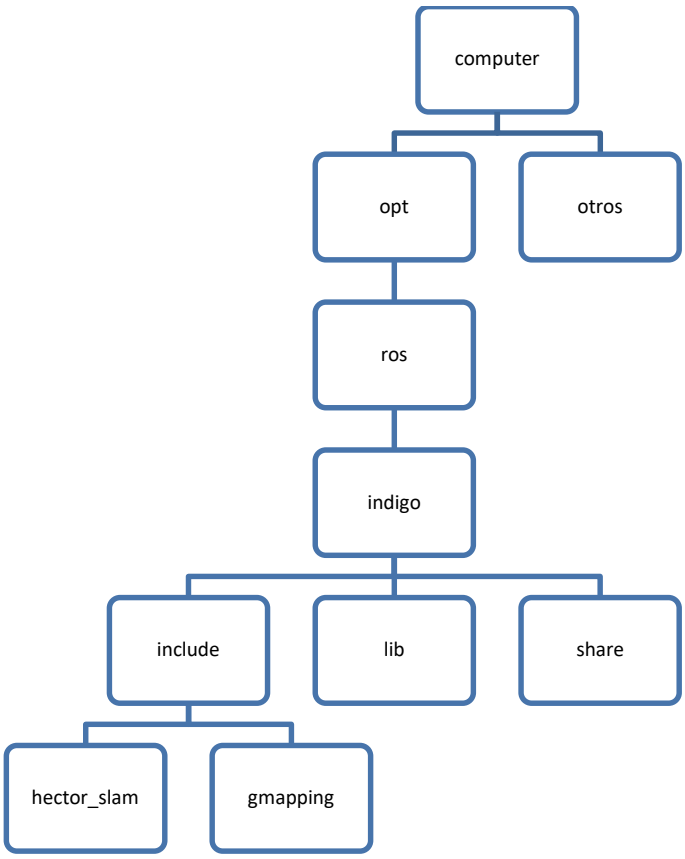


Figura 12-3: Estructura del directorio ROS

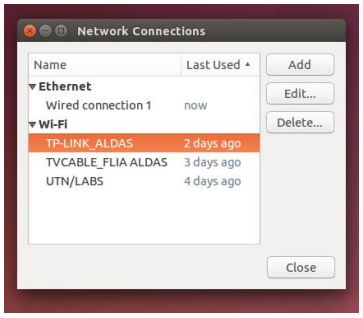
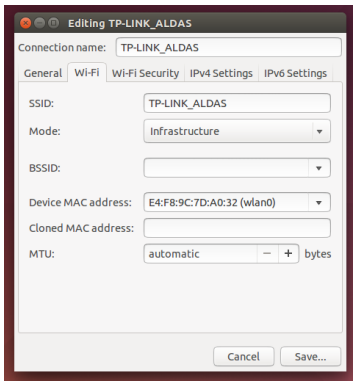
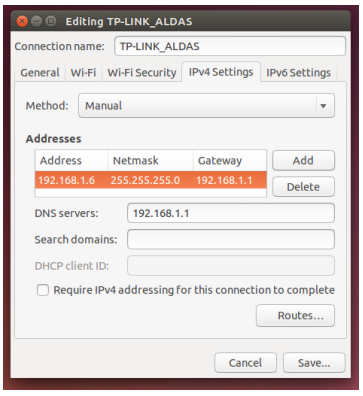
Fuente: ALDAS, Iván, 2017

3.4.1.3. *Configuración de IP estática*

En la figura 3-7, se muestra el proceso de configuración IP. La dirección IP debe ser estática, porque es necesario que siempre sea la misma, ya que como se vera en el próximo capítulo, todos los nodos ROS Hardware publican sus direcciones IP para ser ubicados en el entorno ROS.

En este caso la dirección de la computadora denominada Control MASTER elegida es: 192.168.1.6.

Tabla 7-3: Control MASTER con IP estática

<table>
<tr><td rowspan="2">PASO 1</td><td colspan="2">Editar conexiones</td></tr>
<tr><td>Dar click al icono de conexión WIFI y seleccionar 'Edit Connections'</td><td> 
</td></tr>
<tr><td rowspan="2">PASO 2</td><td colspan="2">Conexiones de red disponibles</td></tr>
<tr><td>Seleccionar la red para la red ROS y dar click en 'Edit...'</td><td> 
</td></tr>
<tr><td rowspan="2">PASO 3</td><td colspan="2">Información de la red TP-LINK_ALDAS</td></tr>
<tr><td>En la pestaña 'Wi-Fi' se presanta la información de la dirección MAC.</td><td> 
</td></tr>
<tr><td rowspan="2">PASO 4</td><td colspan="2">Editando la red TP-LINK_ALDAS</td></tr>
<tr><td>Ir a la pestaña 'IPv4 Settings' , luego seleccionar el metodo Manual y colocar la información IP deseada.</td><td> 
</td></tr>
</table>

Fuente: ALDAS, Iván, 2017

3.4.2. Setup de la Raspberry Pi

La Raspberry Pi 3 es la computadora onboard de la RasPi_mBot por lo que es necesario ejecutar las siguientes operaciones:

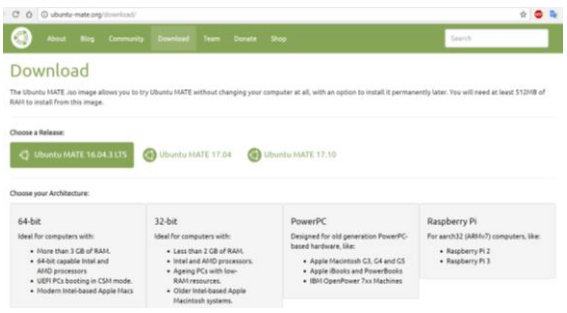
- Instalación de Ubuntu MATE
- Instalación de ROS Kinetic
- Creación y configuración del paquete rplidar_ros
- Instalación de librería rrb3
- Configuración IP estática

3.4.2.1. *Instalación de Ubuntu MATE*

En las siguientes tablas se detalla el proceso de instalación:

Tabla 8-3: Instalación Ubuntu MATE en RP3

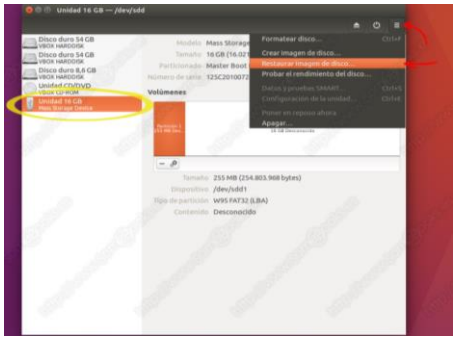
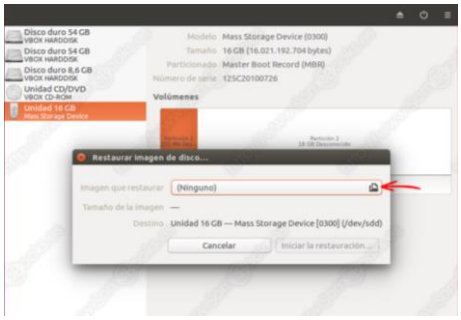
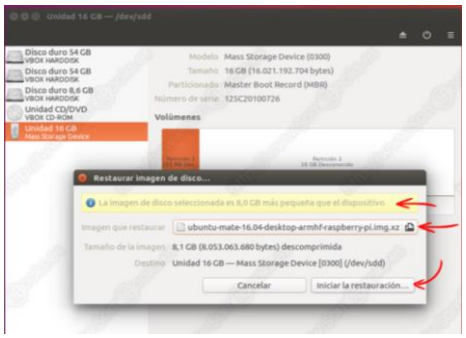
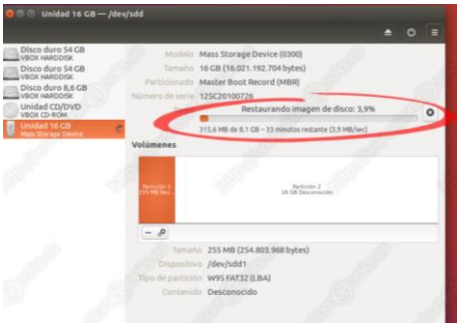
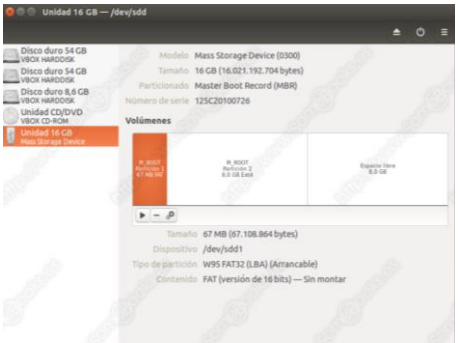
<table>
<tr><td colspan="2">Para instalar Ubuntu Mate 16.04 LTS en la Raspberry 3, se debe realizar las siguientes acciones generales:

- Descargar el archivo de imagen en el ordenador
- Volcar esa imagen a una tarjeta microSD.
- Instarlar Ubuntu Mate 16.04 LTS en la Raspberry Pi</td><td> 
</td></tr>
<tr><td rowspan="2">PASO 1</td><td>Descarga de imagen ISO</td><td rowspan="2"> 
</td></tr>
<tr><td>Descargar ingresando al sigiuente link oficial:

http://ubuntu-mate.org/download/</td></tr>
<tr><td rowspan="2">PASO 2</td><td>Volcar imagen a tarjeta microSD</td><td rowspan="2"> 
</td></tr>
<tr><td>Una vez descargado el archivo, habrá que volcarlo sobre la tarjeta microSD para poder trasladarlo a la Raspberry Pi. Algo que se puede conseguir usando la herramienta Discos que incorpora Ubuntu 16.04 LTS de forma predeterminada.
Para ejecutarlo, sólo tenemos que comenzar a escribir su nombre en el Dash y, cuando aparezca, hacer clic sobre él</td></tr>
</table>

Fuente: ALDAS, Iván, 2017

Tabla 9-3: Instalación Ubuntu MATE en RP3 (continuación)

PASO 3	**Restaurar imagen de disco** Una vez elegido el dispositivo correcto, ubicar el botón de menú que hay en la parte superior derecha de la ventana. Al hacer clic sobre él, aparecerá un menú con diferentes opciones. Seleccionar 'Reataurar imagen de disco' y continuar.	
PASO 4	**Buscar imagen** Hacer clic sobre el botón que permite buscar el archivo de imagen.	
PASO 5	**Iniciar la restauración** Una vez seleccionada la imagen, dar clic en 'Iniciar la restauración….'.	
PASO 5	**Restauración en proceso** Luego de aceptar el formateo de la microSD y la autenticación, el proceso de restauración avanzara.	
PASO 6	**Proceso terminado** Al final, se han creado dos nuevas particiones: La primera, de 67 MB, etiquetada como PI_BOOT, contendrá el arranque básico desde el que se iniciará la instalación del sistema durante el primer arranque. La segunda, de 8GB, etiquetada como PI_ROOT, será la que contenga al sistema operativo una vez completada su instalación y configuración.	

Fuente: ALDAS, Iván, 2017

Ya sólo queda insertar la tarjeta *microSD* en la *Raspberry Pi*, conectarla al monitor mediante un cable *HDMI*, un teclado y un ratón a los puertos *USB* y, finalmente, a la red eléctrica. En este momento se inicia la instalación del sistema operativo (SomeBooks.es, 2017). El proceso es similar al descrito previamente en el literal 3.4.1.1.

3.4.2.2. *Instalación de ROS Kinetic*

La tabla de abajo detalla el proceso de instalación:

Tabla 10-3: Instalación de ROS Kinetic

PASO 1	**Permitir todas las fuentes de ROS**
	Se configura la Raspberry Pi 3 para aceptar todo el software de packages.ros.org. ROS Kinetic solo soporta paquetes Debian en las versiones de Ubuntu Wily (Ubuntu 15.10) y Xenial (Ubuntu 16.04).
	En una ventana de terminal, ejecutar el siguiente comando: `sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'`
PASO 2	**Configuración del key**
	En una misma ventana de terminal, ejecutar el siguiente comando:
	`sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-key 421C365BD9FF1F717815A3895523BAEEB01FA116`
PASO 3	**Instalación general**
	Primero se debe asegurar que Ubuntu este actualizado, mediante: `sudo apt-get update`
	Luego que la actualización se complete, se puede instalar la versión completa de Indigo, la cual es la mas recomendada, mediante: `sudo apt-get install ros-kinetic-desktop`
PASO 4	**Inicialización de ROS**
	Ejecutar los comandos siguientes, uno a la vez:
	`sudo rosdep init` `rosdep update`
PASO 5	**Configuración del entorno de trabajo para Ubuntu**
	Es conveniente que las variables del entorno ROS sean automáticamente añadidas a la sesión bash cada vez que aperturamos una nueva ventana shell, eso se logra gracias a:
	`echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc` `source ~/.bashrc`
PASO 6	**Obtención de rosinstall**
	Una herramienta de comandos de línea frecuentemente utilizada en ROS es rosinstall, que se distribuye por separado. Esta herramienta permite descargar fácilmente paquetes ROS de diferentes fuentes con un solo comando.
	Para instalar esta herramienta en Ubuntu, ejecutar: `sudo apt-get install python-rosinstall python-rosinstall-generator python-wstool build-essential`

Fuente: wiki.ros.org, 2017

3.4.2.3. *Creación y configuración del paquete Rplidar_ros*

Debido a la selección del sensor LIDAR como instrumento de recolección de datos de posición de los objetos que rodean a la RasPi_mBot, es necesario la creación de un paquete compatible con ROS que permita activar y publicar los datos captados de este en el entorno ROS. Ventajosamente, la empresa SLAMTEC, fabricante del sensor Rplidar A2, distribuye gratuitamente dicho paquete. El paquete está disponible en: https://github.com/robopeak/rplidar_ros.git.

En la tabla 3-11 se presenta el proceso de creación:

Tabla 11-3: Creación del paquete rplidar_ros

PASO 1	**Clonar repositorio**	
	En una ventana de terminal, ejecutar los siguientes comandos, uno a la vez:	**cd catkin_ws/src/** **git clone https://github.com/robopeak/rplidar_ros.git**
PASO 2	**Construcción de paquete ROS**	
	Una vez concluida la clonación, ejecutar los siguientes comandos, uno a la vez, para crear los nodos 'rplidarNode' y 'rplidarNodeClient':	**cd catkin_ws/** **catkin_make** **source ~/catkin_ws/devel/setup.bash**

Fuente: ALDAS, Iván, 2017

Por otro lado, para asegurar la comunicación entre el sensor y el entorno ROS, se debe ejecutar los siguientes comandos:

Checar los permisos del puerto USB:

```
ls -l /dev |grep ttyUSB
```

Añadir los permisos de escritura del puerto USB:

```
sudo chmod 666 /dev/ttyUSB0
```

Instalar el remapeo del nombre del puerto USB:

```
sudo cp 'rospack find rplidar_ros'/scripts/rplidar.rules  /etc/udev/rules.d
```

Modificar el achivo rplidar.launch:

```
<param name="serial_port" type="string" value="/dev/rplidar"/>
```

3.4.2.4. *Instalación de Librería rrb3*

La librería rrb3 esencialmente permite controlar la placa RasPiRobot V3, que es un HAT (similar a los shields de Arduino) para Raspberry Pi 3, que hace la función del driver de los motores.

La placa RasPiRobot V3 es una tarjeta de expansión diseñada para convertir a la Raspberry Pi en una placa controladora de robots. Esta tarjeta viene completamente ensamblada e incluye una fuente de poder switched-mode que permite alimentar a la Raspberry Pi con una variedad de packs de baterías. En la figura 3.13 se muestra la placa mencionada anteriormente con sus características principales.

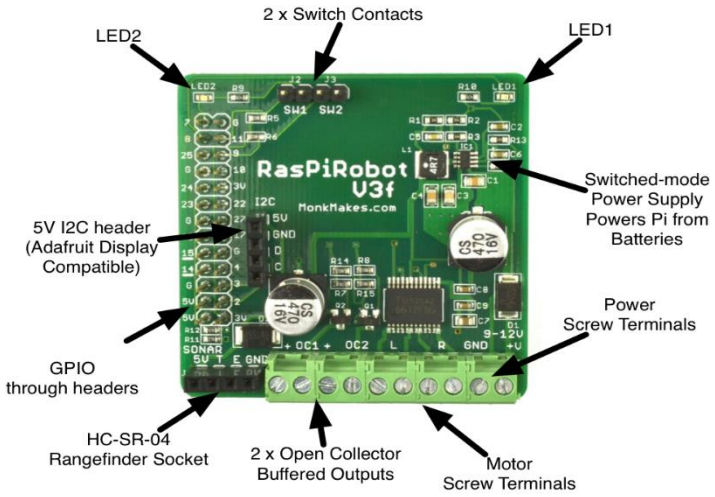


Figura 3-2: HAT RasPiRobot V3f

Fuente: monkmakes.com , 2017

La placa se ajusta en la parte superior del socket de los GPIO de la Raspberry Pi y permite el control bidireccional de dos motores con un chip controlador basado en puente H. También permite el control de la velocidad de los motores independientemente.

Las principales prestaciones de la versión 3 se listan a continuación:

- Compatible con Raspberry Pi modelos 3, 2, 1, Zero, A, A+, B y B+.
- Librería Python de código abierto con ejemplos.
- Totalmente ensamblada / no requiere de soldadura.
- Control Bi-direccional de dos motores DC.

- Control de potencia variable (PWM). Esto permite controlar la velocidad de los motores independiente y utilizar motores de menor potencia.
- Alimentación con 6 baterías tipo AA.
- Socket de conexión directa de sensor de rango infrarrojo compatible con HC-SR-04.
- Conector I2C de 5V compatible con displays Adafruit.
- Dos salidas de colector abierto con capacidad de 2A cada una.
- Dos salidas tipo LED.
- Dos entradas tipo Switch.
- Terminales tipo tornillo para los motores y la batería.
- Terminales tipo headers que permiten acceso a todos los pines GPIO (La RRB3 usa muchos de ellos).

La RRB3 se alimenta de un pack de baterías de entre 6 y 12V DC. Se recomienda utilizar al menos 6 pilas AA, sean estas recargables o regulares tipo heavy duty. Una batería LiPo de 7.2V también funcionará bien. No se necesita una fuente de alimentación separada para la Raspberry Pi. El RRB3 proporciona suficiente potencia. *Sin embargo, está perfectamente bien alimentar la Raspberry Pi a través de USB, incluso mientras las baterías de la RRB3 están conectadas.*

Una vez conocidas las ventajas del HAT, para instalar la Liberia rrb3 en la Raspberry Pi, se deben ejecutar los siguientes comandos en una ventana Terminal, uno a la vez:

```
cd ~
git clone https://github.com/simonmonk/raspirobotboard3.git
cd raspirobotboard3/python
sudo python setup.py install
```

La biblioteca implementa una clase llamada RRB3, la cual sólo está disponible para Python 2 y cualquier programa Python que escriba que use la biblioteca debe ejecutarse

como superusuario. Para importar la biblioteca y crear una instancia de la clase, se debe colocar en la parte superior del programa Python, lo siguiente:

```
from rrb3 import *
rr = RRB3(9, 6)
```

El primer parámetro '9' es el voltaje de la batería (6 pilas AA de 1,5V). El segundo parámetro '6' es la tensión del motor (6V para la mayoría de los motores de chasis de robots de bajo coste). Es importante establecer estos valores correctamente, ya que la biblioteca administrará el voltaje suministrado a los motores, para evitar que se quemen o corran demasiado rápido.

Para conocer todos los métodos disponibles de esta librería acceder a la página web del fabricante: https://www.monkmakes.com/rrb3/

También, se puede descargar el diagrama esquemático desde el siguiente link: https://github.com/simonmonk/raspirobotboard3/blob/master/hardware/RRBv3.pdf

Finalmente, es importante conocer los pines GPIO de la Raspberry Pi 3 utilizados por el RRB 3. En la tabla 3-7 se visualiza lo dicho.

Tabla 12-3: Relación de pines entre el RRB 3 y la Raspberry Pi 3

Pines del RRB 3	GPIO de la Raspberry Pi 3
RIGHT_PWM_PIN	14
RIGHT_1_PIN	10
RIGHT_2_PIN	25
LEFT_PWM_PIN	24
LEFT_1_PIN	17
LEFT_2_PIN	4
SW1_PIN	11
SW2_PIN	9
LED1_PIN	8
LED2_PIN	7
OC1_PIN	22
OC2_PIN	27
OC2_PIN_R1	21
OC2_PIN_R2	28
TRIGGER_PIN	18
ECHO_PIN	23

Fuente: monkmakes.com , 2017

3.4.2.5. *Configuración IP estática*

El proceso es similar al descrito en el punto 3.4.1.3, pero considerando la siguiente dirección IP: 192.168.1.8

3.5. Consumo de energía

En la siguiente tabla se determina el consumo de energía y la autonomía de la RasPi_mBot:

Tabla 13-3: Consumo de energía y la autonomía de la RasPi_mBot

ETAPA	ITEM	COMSUMO PROMEDIO (mA)	ALIMENTACIÓN (mAh)	AUTONOMÍA (h)
CONTROL	Raspberry Pi3	350	Power bank: 10000	12,2
	Sensor Rplidar	450		
	RaspiCAM	20		
	TOTAL:	820		
POTENCIA	Motores DC	1300	Battery pack: (2700 x 6)	6,2
	TOTAL:	2600	16200	

Fuente: ALDAS, Iván , 2017

CAPÍTULO IV

4. RESULTADOS Y DISCUSIÓN

A continuación, se representa el protocolo de pruebas a la que fue sometida la RasPi_mBot:

Tabla 1-4: Protocolo de pruebas

PRUEBA	OBJETIVO
Test Acceso remoto	Acceder vía SSH a la RasPi_mBot desde la computadora Control Master.
Test Control remoto de movimiento (Teleoperación)	Activar remotamente los motores de la RasPi_mBot.
Test Rplidar	Activar remotamente el sensor laser y visualizar datos en entorno gráfico.
Test Paquetes SLAM	Graficar mapas aplicando paquetes SLAM de ROS mediante bags.
Test SLAM - Rplidar	Obtener el mapa del entorno aplicando paquetes SLAM de ROS mediante Rplidar.

Fuente: ALDAS, Iván, 2017

4.1. Test Acceso remoto.

Para realizar esta prueba fue necesario habilitar el servicio de SSH en la Raspberry Pi. El servidor de Shell Seguro o SSH (Secure SHell) es un servicio muy similar al servicio telnet ya que permite que un usuario acceda de forma remota a un sistema Linux pero con la particularidad de que, al contrario que telnet, las comunicaciones entre el cliente y servidor viajan cifradas desde el primer momento de forma que si un usuario malintencionado intercepta los paquetes de datos entre el cliente y el servidor, será muy

difícil que pueda extraer la información ya que se utilizan sofisticados algoritmos de cifrado (ITE, 2017).

Para habilitar el servidor SSH, en una ventana de terminal se ejecuta el siguiente comando:

sudo service ssh enable

Y luego de reiniciar la Raspberry Pi, el servidor SSH se encuentra ya disponible.

4.1.1. Resultados

Como se aprecia en la siguiente secuencia de figuras, el test de acceso remoto mediante SHH fue un éxito.

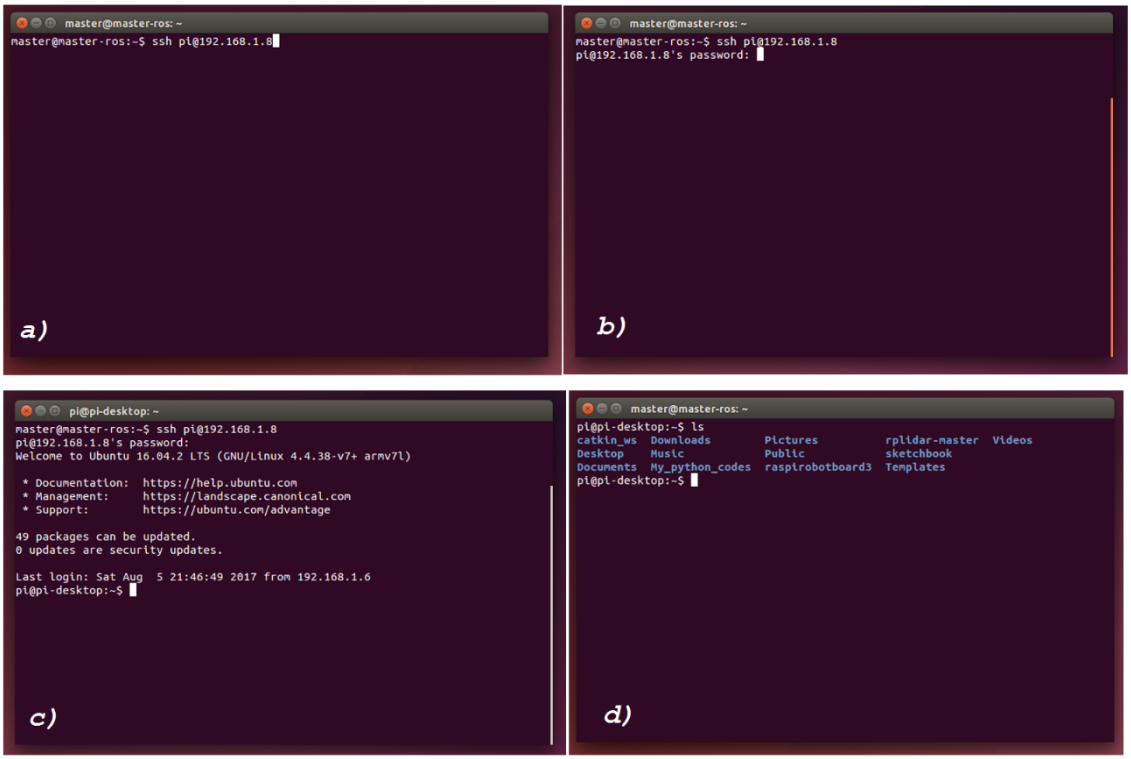


Figura 1-4: Test SSH

Fuente: ALDAS, Iván, 2017

En la secuencia a) se ingresa el comando ‘shh’ seguido del nombre de usuario, ‘@’ y la dirección IP correspondiente; en b) si los datos anteriores son correctos se solicita la clave; en c) se autoriza el ingreso y en d) ya se puede navegar por la RP3 en este punto se aplicó el comando ‘ls’ para listar el contenido de home como ejemplo.

4.2. Test Teleoperación

El objetivo de esta prueba fue utilizar una máquina para publicar un dato y emplear otra máquina para suscribirse a al mismo.

Se uso la laptop denominada ‘Computador MASTER’ para publicar los datos de las teclas (flechas) pulsadas del teclado y la RasPi_mBot en función de estos datos, ejecuto el movimiento correspondiente. Para el efecto, se realizaron los siguientes procedimientos:

- Creación de un paquete ROS
- Setup de la maquina Master
- Setup de la maquina Huésped

4.2.1. Creación del paquete ROS

En la figura 4-2 se aprecia la estructura de archivos creada a través del método catkin_create_pkg y los ejecutables Python específicamente creados para activar los motores del RasPi_mBot mediante botones de un mando inalámbrico de Xbox 360.
Para mayor información del método, ingresar en http://wiki.ros.org/ROS/Tutorials/CreatingPackage.

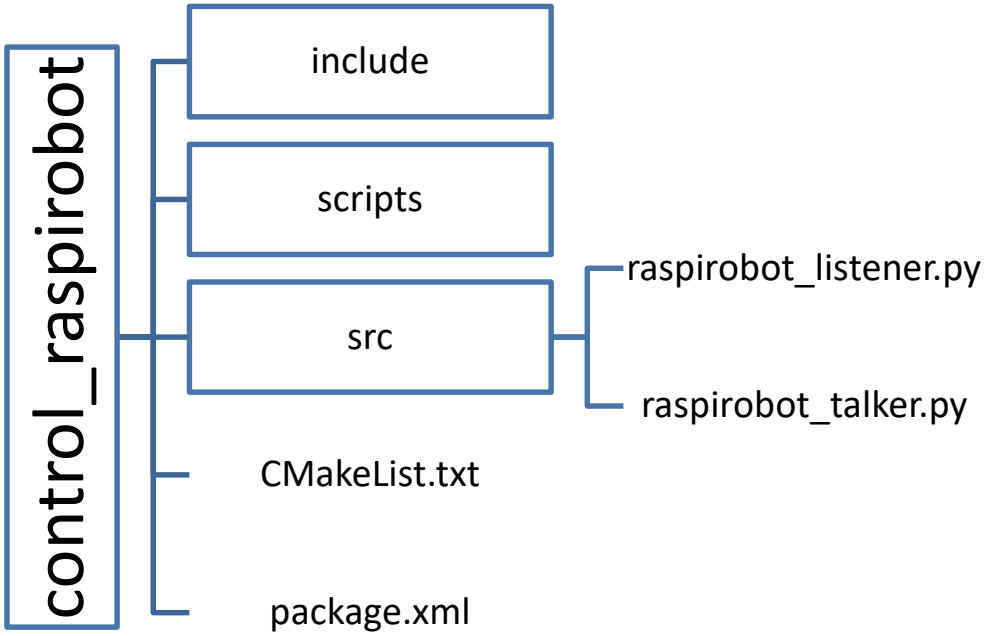


Figura 2-4: Paquete control_raspirobot

Fuente: ALDAS, Iván, 2017

La figura 4-3 presenta el código Python cargado en raspirobot_talker.py encargado de publicar el dato de los botones pulsados del mando. Un nodo ‘publisher’ y un nodo ‘subscriber’ en ROS tienen una estructura particular, para más detalles ingresar en:

http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28python%29.

Figura 3-4: Código raspirobot_talker.py

Fuente: ALDAS, Iván, 2017

Por otro lado, la figura 4-4 presenta el código Python cargado en raspirobot_listener.py encargado de suscribirse al dato publicado por el nodo anteriormente descrito.

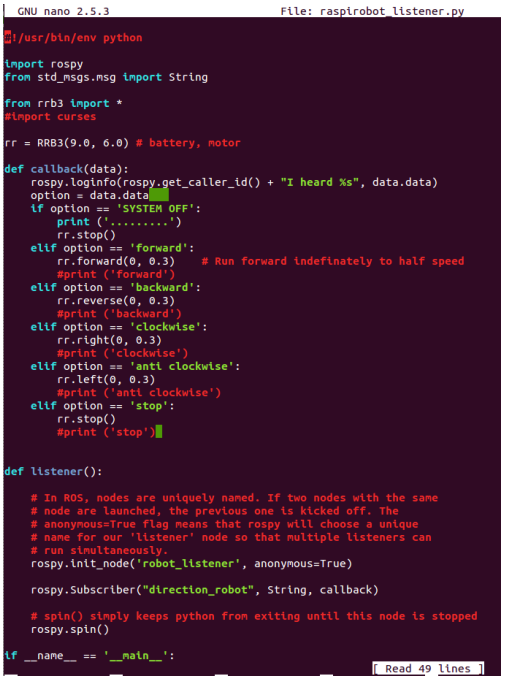


Figura 4-1: Código raspirobot_listener.py

Fuente: ALDAS, Iván, 2017

Los códigos también están más legibles en la sección de anexos del presente proyecto.

4.2.2. Setup de la maquina Master

La Computadora Master tiene la dirección IP: 192.168.1.6 y por defecto trasfiere información por el puerto 11311, por lo tanto, se debe editar el archivo /.bashrc mediante el comando siguiente:

gedit ~/.bashrc

y entonces se debe añadir las líneas de a continuación:

export ROS_MASTER_URI=http://192.168.1.6:11311

4.2.3. Setup de la maquina Huésped

La RasPi_mBot tiene la dirección IP: 192.168.1.8, siguiendo el procedimiento anterior, se debe incluir la siguiente información en su archivo /.bashrc:

export ROS_MASTER_URI=http://192.168.1.6:11311
export ROS_HOSTNAME=192.168.1.8

4.2.4. Resultados

Una vez configurado todo, ya se puede proceder con el test. Para ello se debe ejecutar los siguientes comandos en diferentes ventanas de terminal:

- Ventana 1 (Computador MASTER):

  roscore

- Ventana 2:

  shh pi@192.168.1.8

  rosrun contro_raspirobot raspirobot_listener.py

- Ventana 3:

  shh pi@192.168.1.8

  rosrun contro_raspirobot raspirobot_talker.py

Los resultados se reflejan en la tabla 4-2.

Tabla 2-4: Resultados Test Teleoperación

Evento / Tecla presionada	Numero de eventos	Acción esperada	Acción realizada	TEST
Flecha arriba	30	Movimiento hacia delante	Movimiento hacia delante	
Fecha abajo	30	Movimiento hacia atrás	Movimiento hacia atrás	
Fceha Izquierda	30	Giro izquierda	Giro izquierda	
Fecha Derecha	30	Giro derecha	Giro derecha	
Letra s	30	Parar los motores	Parar los motores	
Letra q	30	Detener el programa	Detener el programa	

Fuente: ALDAS, Iván, 2017

4.3. Test Rplidar

Para realizar la prueba del sensor LIDAR, es necesario conocer la estructura del paquete rplidar_ros distribuido por SLAMTEC.

4.3.1. Paquete RPLIDAR_ROS

En la figura 4-5 se aprecia la estructura de archivos del paquete rplidar_ros. Este paquete contiene archivos 'launcher' que permiten ejecutar varios nodos a la vez. Para mayor información del paquete, ingresar en http://wiki.ros.org/rplidar.

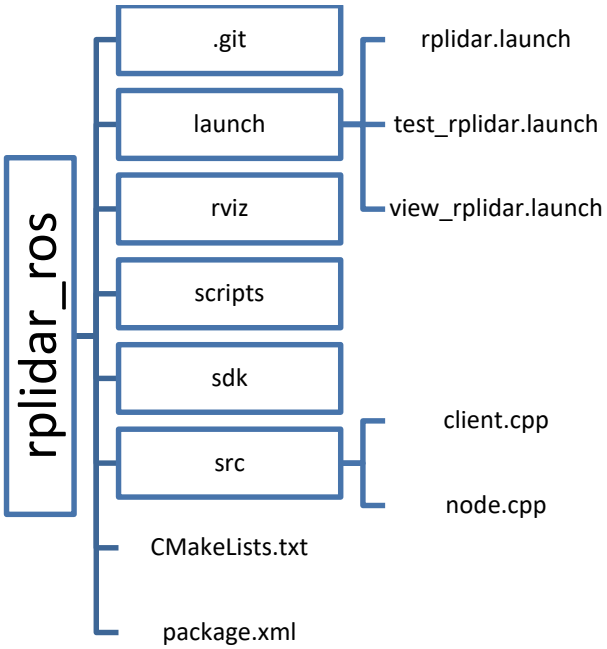


Figura 5-4: Paquete rplidar_ros

Fuente: ALDAS, Iván, 2017

En la figura 4-6, se visualiza el archivo ´rplidar.launch´, que como su extensión sugiere, lanza el ejecutable ´node.cpp´ con algunos parámetros iniciales, dentro de los cuales destacan: el puerto USB de conexión del sensor Lidar, la velocidad de comunicación y el nombre del frame que se utiliza para determinar la posición del sensor en relación a estructura de la RasPi_mBot.

```
<launch>
  <node name="rplidarNode"          pkg="rplidar_ros"  type="rplidarNode" output="screen">
  <param name="serial_port"         type="string" value="/dev/ttyUSB0"/>
  <param name="serial_baudrate"     type="int"    value="115200"/>
  <param name="frame_id"            type="string" value="laser"/>
  <param name="inverted"            type="bool"   value="false"/>
  <param name="angle_compensate"    type="bool"   value="true"/>
  </node>
</launch>
```

Figura 6-4: Archivo rplidar.launch

Fuente: ALDAS, Iván, 2017

4.3.2. Prerrequisitos

Para proceder con el test, antes se debe ejecutar los siguientes comandos en diferentes ventanas de terminal:

- Ventana 1 (Computador MASTER):

  roscore

- Ventana 2:

  shh pi@192.168.1.8

  roslaunch rplidar_ros rplidar.launch

- Ventana 3 (Computador MASTER):

  rosrun rviz rviz

Al ejecutar el comando de la ventana 3, se abre la herramienta grafica de ROS que permite visualizar los datos publicados por el nodo ´rplidarNode´. La aplicación RViz tiene el siguiente aspecto:

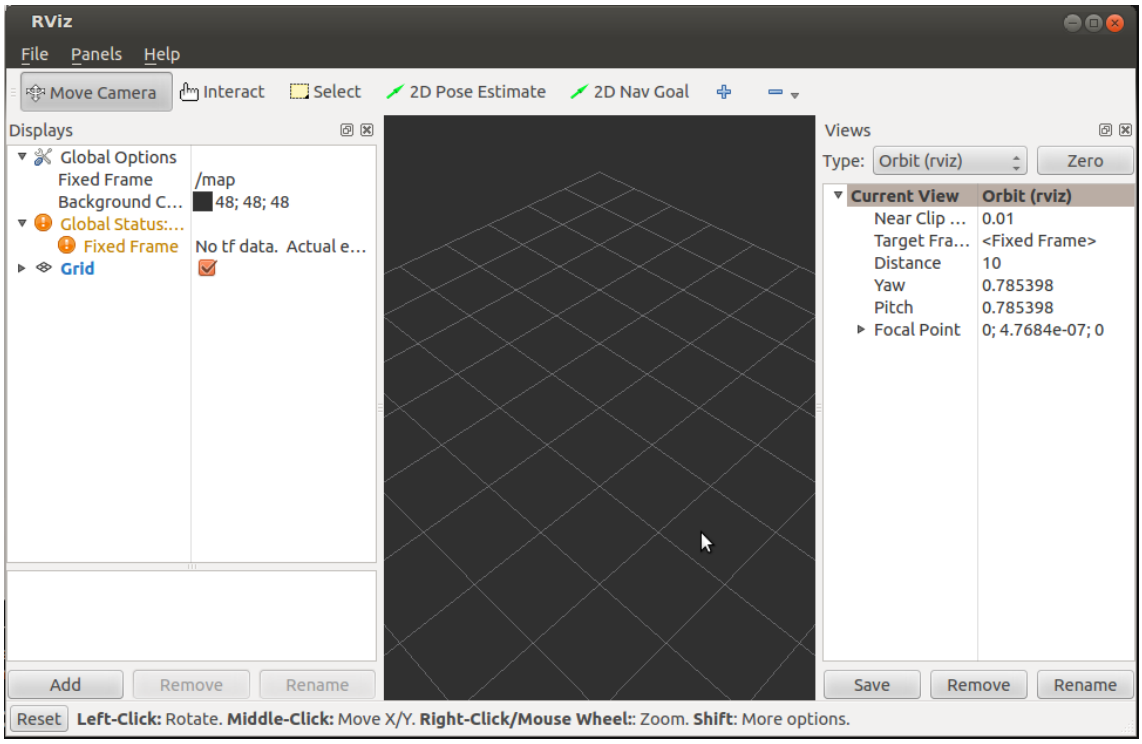


Figura 7-4: Aplicación RViz

Fuente: docs.ros.org, 2017

4.3.3. Resultados

Una vez ejecutados los comandos del punto anterior, se debe añadir el tópico publicado por ´rplidarNode´, denominado LaserScan. La figura 4-8 ilustra el proceso.

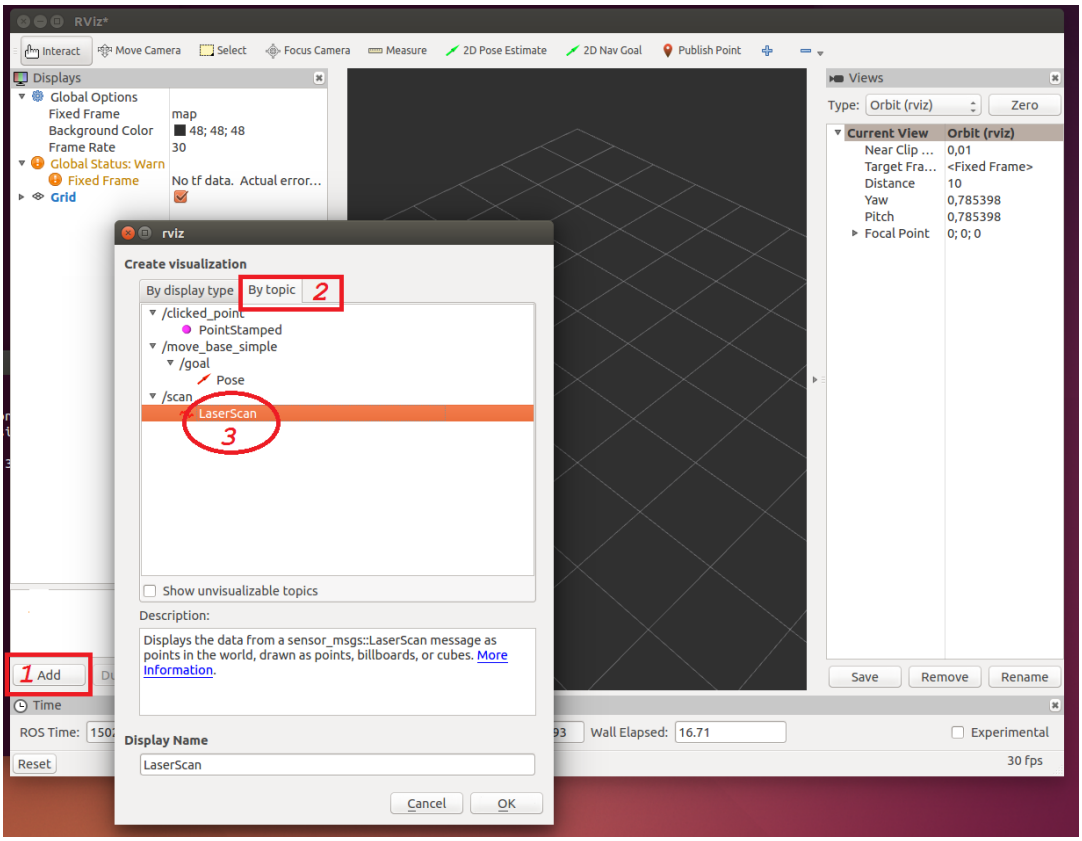
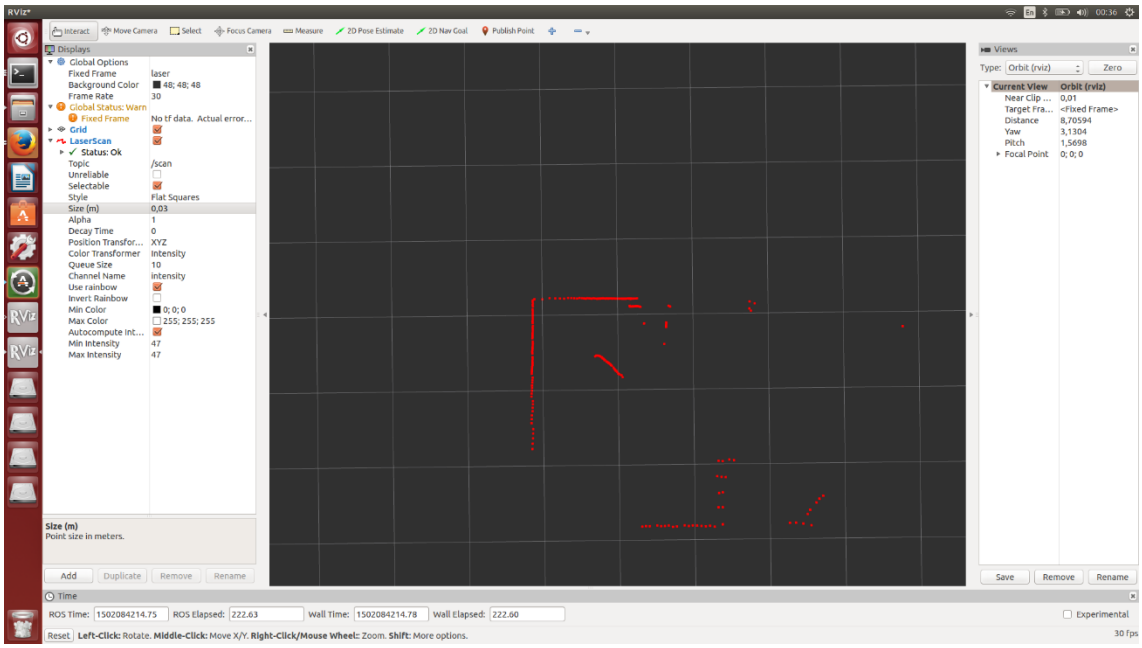


Figura 8-4: Adición del tópico LaserScan en RViz

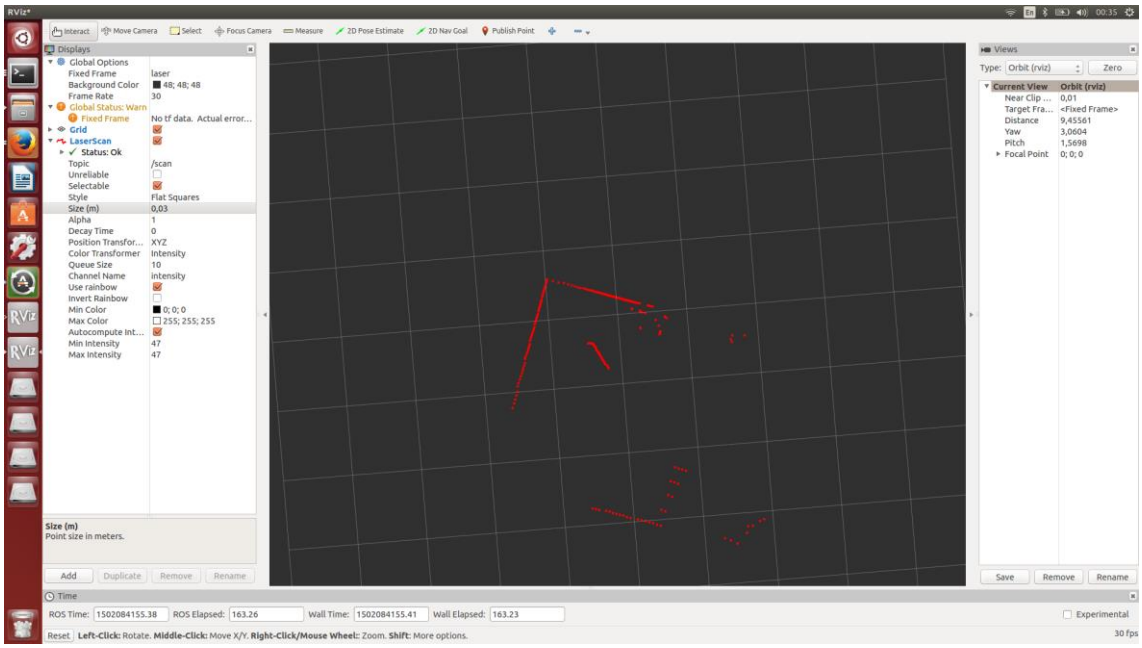
Fuente: ALDAS, Iván, 2017

En la siguiente secuencia de figuras, se evidencia el éxito del test.

a) Posición inicial



b) Giro de 45 grados



c) Giro de 90 grados

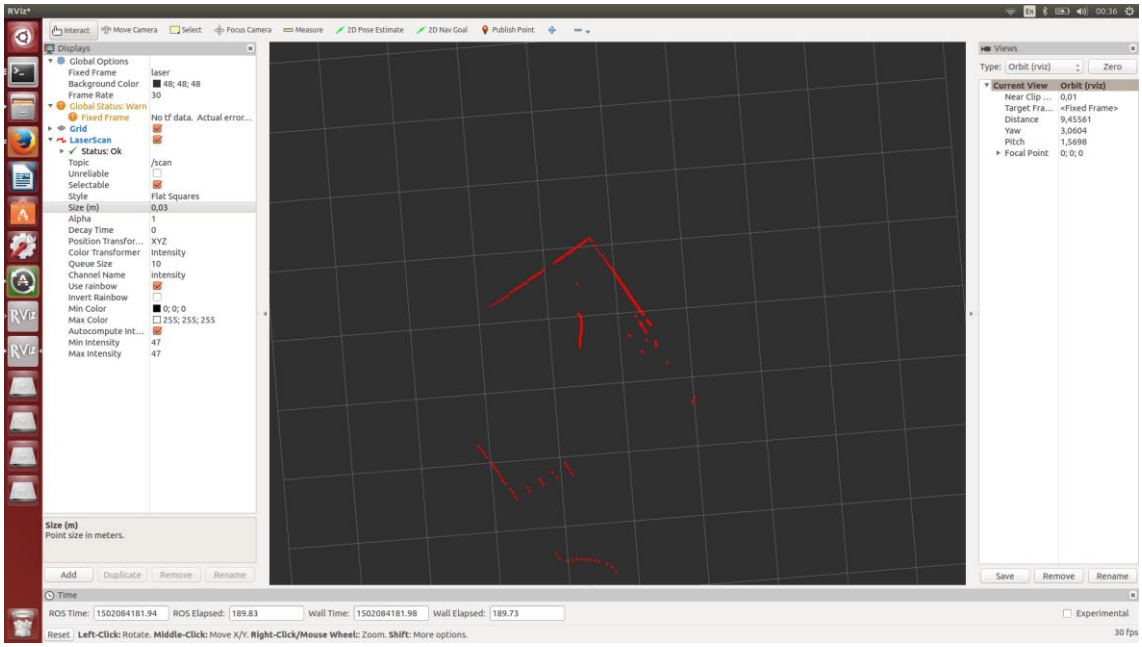


Figura 9-4: Rplidar en acción

Fuente: ALDAS, Iván, 2017

4.4. Test SLAM - Rplidar

Prácticamente esta es la prueba definitiva para la RasPi_mBot, ya que será una prueba de todo el conjunto. En la figura 4-10 se muestra el aspecto real del prototipo.

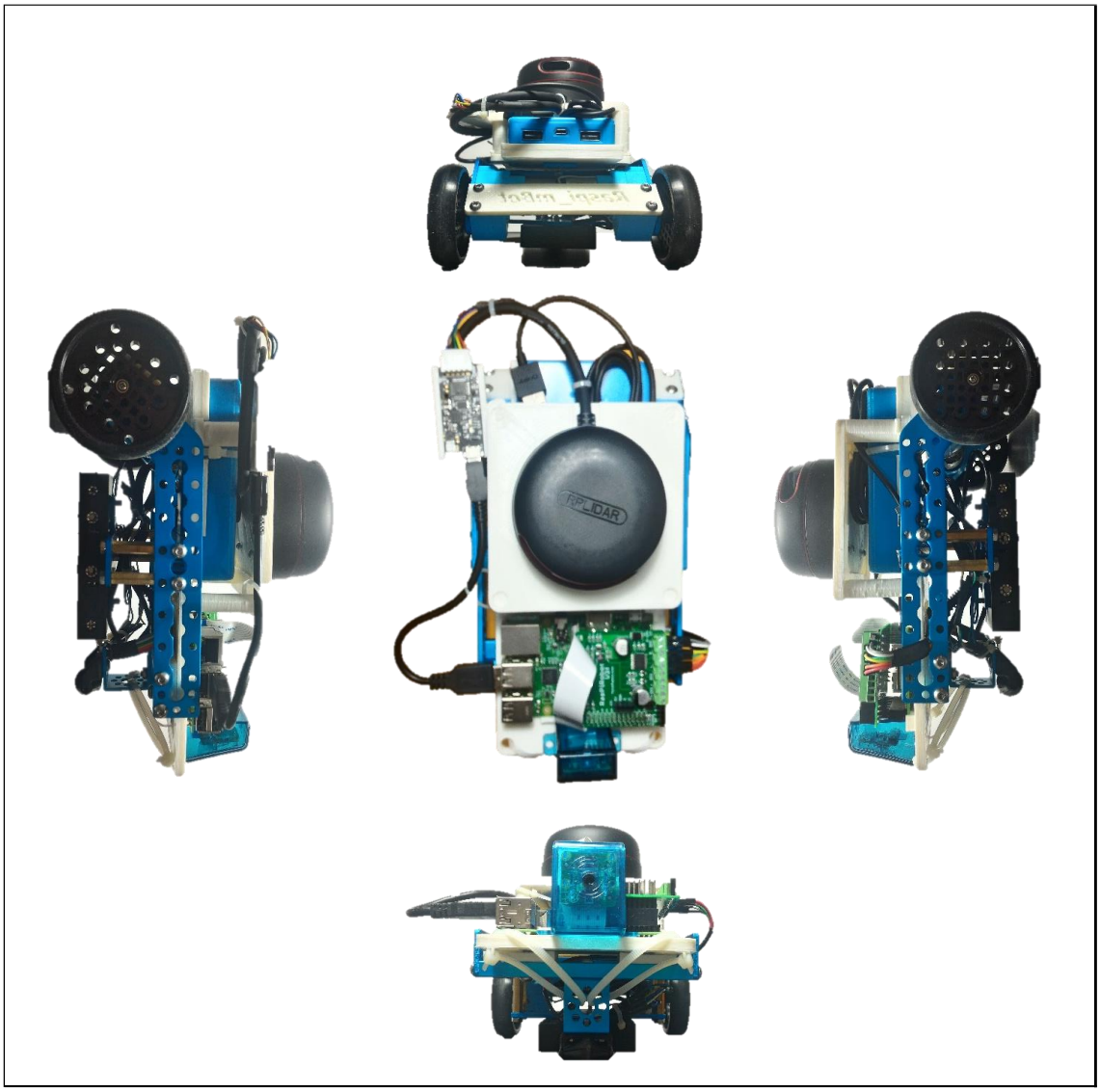


Figura 10-4: RasPi_mBot

Fuente: ALDAS, Iván, 2017

4.4.1. Paquete hector_slam

En la figura 4-11 se aprecia parte de la estructura de archivos del paquete hector_slam. Este paquete en realidad es un Metapaquete, ya que contiene varios paquetes que en conjunto permiten realizar SLAM. Para mayor información de los paquetes, ingresar en http://wiki.ros.org/hector_slam.

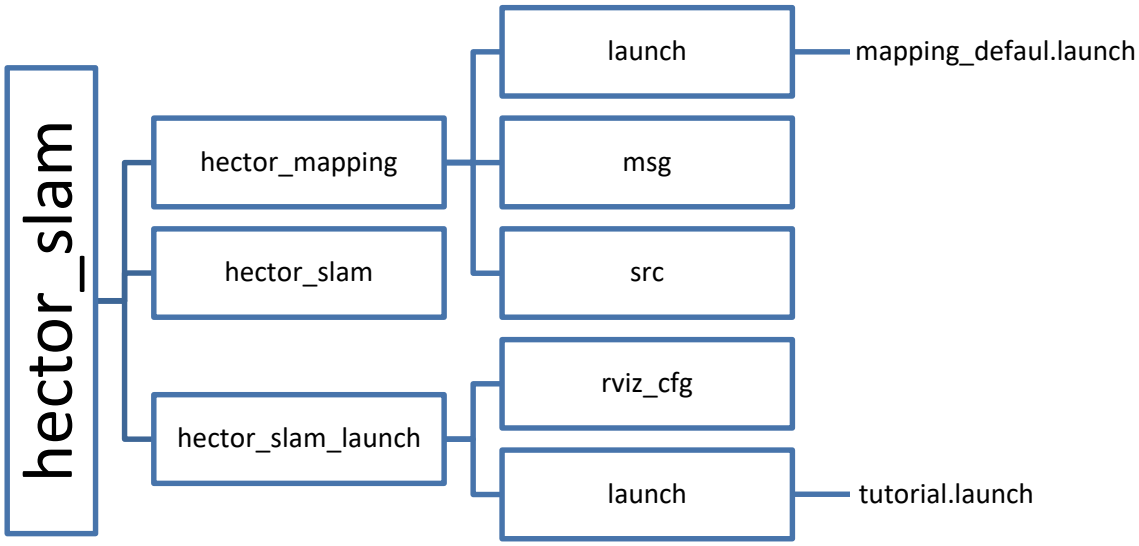


Figura 11-4: Paquete hector_slam

Fuente: ALDAS, Iván, 2017

En la figura anterior solo se muestra los archivos ´launch´ que se deben modificar para realizar SLAM utilizando el sensor Rplidar A2 y sin odometría extra.

En la figura 4-12, se visualiza el archivo ´tutorial.launch´, encargado de iniciar los varios nodos.

```
<?xml version="1.0"?>
<launch>
 <arg name="geotiff_map_file_path" default="$(find hector_geotiff)/maps"/>
 <param name="/use_sim_time" value="false"/>
 <node pkg="rviz" type="rviz" name="rviz"
  args="-d $(find hector_slam_launch)/rviz_cfg/mapping_demo.rviz"/>
 <include file="$(find hector_mapping)/launch/mapping_default.launch"/>
 <include file="$(find hector_geotiff)/launch/geotiff_mapper.launch">
  <arg name="trajectory_source_frame_name" value="scanmatcher_frame"/>
  <arg name="map_file_path" value="$(arg geotiff_map_file_path)"/>
 </include>
</launch>
```

Figura 12-4: Archivo tutorial.launch

Fuente: ALDAS, Iván, 2017

Dentro del código, el parámetro 'use_sim_time' seteado por defecto como 'true', indica al ros master que use el tiempo de simulación en lugar del tiempo real proporcionado por algún otro proceso. Sin embargo, como se utilizan los datos crudos del láser para construir el mapa, se debe establecer este parámetro como 'false'. Las otras líneas lanzan la interfaz gráfica de usuario 'rviz' con algunas configuraciones por defecto, también se lanza el

algoritmo predeterminado de mapeo 'hector'. El paquete 'hector_slam' genera la transformación entre el mapa y el base_frame, sin embargo, se necesitan transformaciones entre el world_frame y del map_frame; y entre el base_frame y el laser_frame. Un nodo que publica estas transformaciones estáticas, se describe en breve. En el presente proyecto el RasPi_mBot por el momento no está dotado de odometría, por lo tanto, se debe realizar modificaciones al archivo mapping_default.launch. Dichas modificaciones, se aprecian en la figura 13-4.

```
<?xml version="1.0"?>
<launch>
 <arg name="tf_map_scanmatch_transform_frame_name" default="scanmatcher_frame"/>
 <arg name="base_frame" default="base_footprint"/>
 <arg name="odom_frame" default="nav"/>
 <arg name="pub_map_odom_transform" default="true"/>
 <arg name="scan_subscriber_queue_size" default="5"/>
 <arg name="scan_topic" default="scan"/>
 <arg name="map_size" default="2048"/>

 <node pkg="hector_mapping" type="hector_mapping" name="hector_mapping" output="screen">

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

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

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

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

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

  <param name="scan_subscriber_queue_size" value="$(arg scan_subscriber_queue_size)"/>
  <param name="scan_topic" value="$(arg scan_topic)"/>

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

 <node pkg="tf" type="static_transform_publisher" name="map_nav_broadcaster" args="0 0 0 0 0 0 base_frame laser 100"/>
</launch>
```

Figura 13-4: Archivo mapping_defaul.launch

Fuente: ALDAS, Iván, 2017

4.4.2. Prerrequisitos

Para proceder con el test, antes se debe ejecutar los siguientes comandos en diferentes ventanas de terminal:

- Ventana 1 (Computador MASTER):

  roscore

- Ventana 2:

  shh pi@192.168.1.8

  roslaunch rplidar_ros rplidar.launch

- Ventana 3:

  shh pi@192.168.1.8

  rosrun control_raspirobot raspirobot_listener.py

- Ventana 4:

  shh pi@192.168.1.8

  rosrun control_raspirobot raspirobot_talker.py

- Ventana 5 (Computador MASTER):

  roslaunch hector_slam_launch tutorial.launch

4.4.3. Resultados

Finalmente, en la siguiente figura, se muestra el resultado del test.

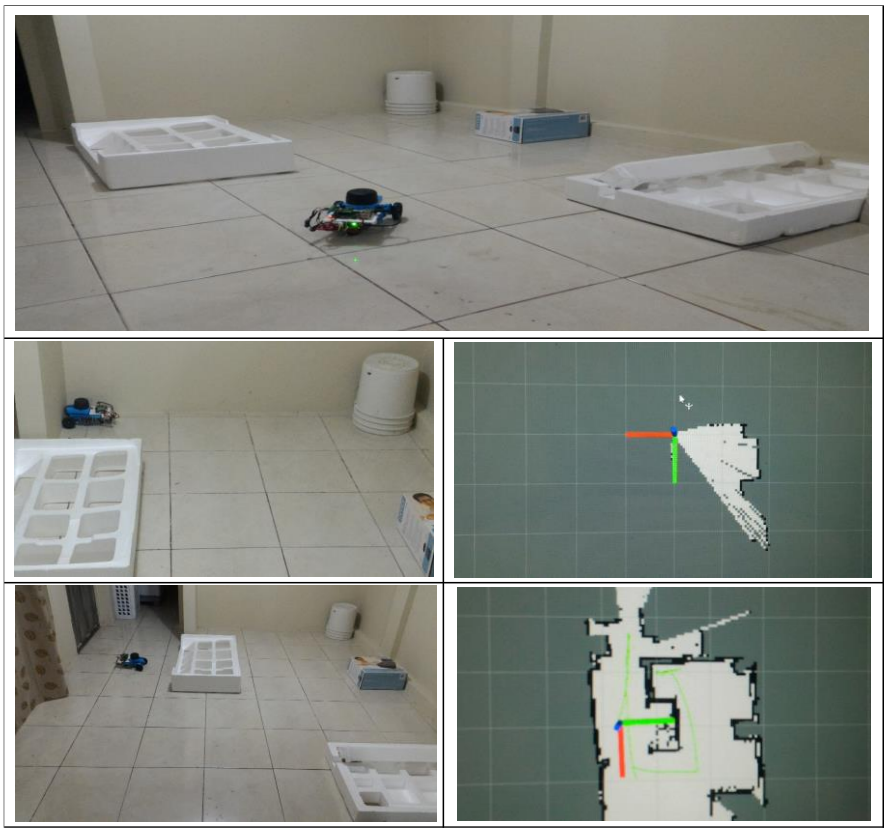


Figura14-4: RasPi_mBot realizando SLAM

Fuente: ALDAS, Iván, 2017

4.5. Comprobación de la hipótesis

La comprobación de la hipótesis, se realizo mediante la aplicación de una practica de laboratorio, cuyo guion se presenta a continuación:

GUION DE PRACTICA

LABORATORIO: ROBOTICA

FECHA: JULIO 2017

TEMA:
UTILIZACIÓN DE LA RASPI_MBOT PARA MAPEO Y LOCALIZACIÓN SIMULTÁNEA

INTRODUCCIÓN:
Esta practica cubrirá la instalación del paquete hector_slam y del paquete rp_lidar para generar un mapa del entorno a la RASPI_MBOT con los datos proporcionados por un sensor láser.

MATERIALES:
1.- Plataforma robótica móvil: RASPI_MBOT
2.- Computador MASTER con Ubuntu 14 y ROS Indigo
3.- Access point

PREREQUISITOS:
En el computador MASTER, ejecutar los siguientes comandos:

```
sudo apt−get install ros−indigo−hector−slam
roscd hector slam launch /launch/
sudo gedit tutorial.Launch
sudo apt-get install ros-indigo-joy
```

DESARROLLO:
Para proceder con el test, antes se debe ejecutar los siguientes comandos en diferentes ventanas de terminal:

- Ventana 1 (Computador MASTER):
  roscore
- Ventana 2:
  shh pi@192.168.1.8
  roslaunch rplidar_ros rplidar.launch
- Ventana 3:
  shh pi@192.168.1.8
  rosrun control_raspirobot raspirobot_listener.py
- Ventana 4:
  shh pi@192.168.1.8
  rosrun control_raspirobot raspirobot_talker.py
- Ventana 5 (Computador MASTER):
  roslaunch hector_slam_launch tutorial.launch

RESULTADOS:
Para guardar el mapa utilice el servidor de mapas:

```
rosrun map server map saver −f mymap
```

4.5.1. Aplicación de la guía de practica

En la siguiente figura se observan los resultados de la aplicación de la guía de practica:

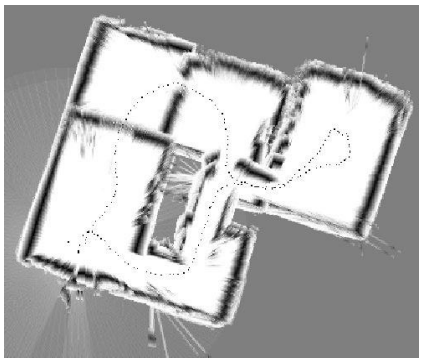
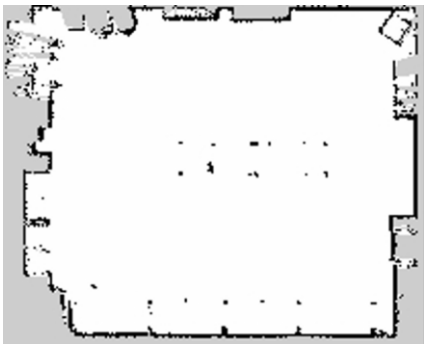
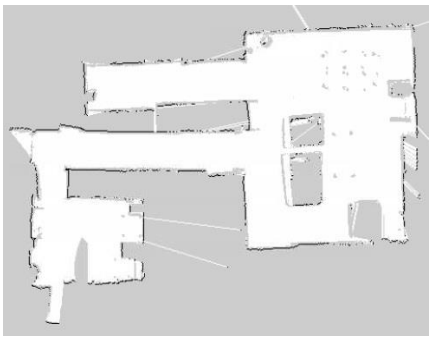
ESCENARIO 1 **UBICACIÓN:** UTN **FACULTAD:** FICA **OFICINAS:** CIME	
ESCENARIO 2 **UBICACIÓN:** UTN **FACULTAD:** FICA **LABORATORIO:** ROBOTICA	
ESCENARIO 3 **UBICACIÓN:** UTN **FACULTAD:** FICA **LABORATORIO:** SIMULACIÓN	
ESCENARIO 4 **UBICACIÓN:** UTN **FACULTAD:** FICA **LABORATORIO:** MECANIZADO	

Figura 15-4: RasPi_mBot realizando SLAM en diferentes escenarios

Fuente: ALDAS, Iván, 2017

CONCLUSIONES

Mediante la investigación bibliográfica se determina que los métodos que se emplean actualmente para la localización y mapeo instantáneo en robots autónomos móviles tienen dos enfoques, el bioinspirado y el probabilístico. De los cuales, los basados en técnicas de estimación de probabilidad son los más desarrollados hasta la fecha. Es importante recalcar, que gracias a la existencia de ROS muchas de estas técnicas se han logrado plasmar en algoritmos computaciones que se distribuyen en forma de paquetes con licencia open source. Entre los paquetes SLAM de ROS se destacan el 'gmapping' y 'hector_slam'. En el presente proyecto se utilizó el paquete 'hector_slam' ya que permite realizar SLAM sin odometría.

Para determinar los requerimientos del hardware y software de la plataforma robótica móvil, se establecieron algunos criterios, de los cuales la robustez, escalabilidad y precio fueron los más imponentes. Como estructura base se utilizó una plataforma comercial educativa por su capacidad de expansión y la inclusión de motores dc; como computadora a bordo se seleccionó la conocida Raspberry Pi en su versión 3 modelo B por su soporte ROS y sus capacidades de comunicación inalámbrica; y como elemento sensor principal un sensor Lidar por su soporte ROS y su capacidad de escaneo en 360°.

En el montaje de la plataforma robótica móvil, el principal desafío consistió en la disposición de componentes hardware, por lo que fue necesaria realizar adaptaciones de la plataforma comercial, principalmente añadiendo soportes impresos en 3D para el sensor Lidar y para la computadora onboard.

Finalmente, para evaluar el funcionamiento de la plataforma robótica móvil, se estableció un protocolo de pruebas, que consistió en test individuales de componentes y de un test integral, en el cual se ejecutó el paquete SLAM de ROS en diferentes ambientes controlados con buenos resultados.

RECOMENDACIONES

La investigación bibliográfica realizada en el presente trabajo, se enfocó solo en los métodos SLAM 2D mediante sensores de rango, por lo que se recomienda al lector indagar en métodos de SLAM 3D y el uso de sensores más complejos tipo Kinect.

En función de lo anterior, también es recomendable la adición de otros sensores que permitan mejorar la percepción del entorno del robot, así como de su propio comportamiento. La habilitación del Encoder óptico de cuadratura incluido en los motores ayudaría en este punto, al igual que un monitor de batería que alerte de niveles bajos de energía de las mismas y vendría bien una unidad de medida inercial para dotar a la plataforma de odometría total.

Si bien la estructura, está diseñada para operar en ambientes controlados como lo de los laboratorios de las universidades, todo el desarrollo del presente proyecto se puede aplicar en otros entornos. Por lo que se recomienda trabajar en una plataforma estilo todo terreno para extender el ámbito de aplicación.

Finalmente, se recomienda aplicar técnicas de navegación y planificación de trayectorias para dotar a la RasPi_mBot de autonomía completa.

BIBLIOGRAFÍA

ANDRADE, F., & LLOFRIU, M. (2013). *SLAM Estado del Arte.* Montevideo: Mina. Obtenido de https://www.fing.edu.uy/inco/grupos/pGrado/pgSLAM/documentos/eda.pdf

BBC, N. (22 de Diciembre de 2014). *Así será el primer auto no tripulado de Google*. Obtenido de http://www.bbc.com/mundo/ultimas_noticias/2014/12/141222_ultnot_google_auto_no_tripulado_prototipo_jg

CABRERA, A. P., & DELGADO, G. A. (2014). *DISEÑO Y CONSTRUCCIÓN DE UN ROBOT PARA MAPEO Y EXPLORACIÓN DE MINAS SUBTERRÁNEAS.* Cuenca: Universidad del Azuay.

CARPENTER, J., & FEARNHEAD, P. (1999). Improved Particle Filter for Nonlinear. *Radar, Sonar and Navigation, Vol. 146*.

FRESE, U. (2006). Closing a million-landmarks loop. *International Conference on Intelligent Robots and Systems* (págs. 5032-5039). Beijing: IEEE .

GENERATION_ROBOTS. (4 de Marzo de 2017). *Eddie Robot Platform*. Obtenido de https://www.generationrobots.com/en/401394-eddie-robot-platform-parallax.html

GOPALAKRISHNAN, B., & TIRUNELLAYI, S. (2014). *Design and development of an autonomous mobile smart vehicle: a mechatronics application.* West Virginia: Elsevier.

HERNÁNDEZ, J. J. (2015). *Diseño y construcción de una plataforma multipropósito y su aplicación con un robot explorador como material didáctico para la escuela de sistemas de la pucesa.* Ambato: PUCESA.

iROBOT CORPORATION, I. (20 de 12 de 2016). *Meet the Roomba family of vacuuming robots*. Obtenido de http://www.irobot.com/For-the-Home/Vacuuming/Roomba.aspx

ITE. (20 de Abril de 2017). *Servidor de SSH*. Obtenido de http://www.ite.educacion.es/formacion/materiales/85/cd/linux/m5/instalacin_de_servidor_de_ssh.html

LENTIN, J. (2015). *Mastering ROS for Robotics Programming.* Birmingham: Packt Publishing.

MARTINEZ, A., & FERNÁNDEZ, E. (2013). *Learning ROS for Robotics Programming.* Birmingham: Packt Publishing.

MILFORD, M., & WYETH, G. (2007). Spatial mapping and map exploitation: A bioinspired engineering perspective. *Spatial Information Theory, volume 4736 of Lecture Notes in Computer Science*, 203-221.

MILFORD, M., & WYETH, G. (2008). Mapping a suburb with a single camera using a biologically inspired slam system. *IEEE Transactions on Robotics VOL. 24 No. 5*, 1038-1053.

MOBILEROBOTS, O. A. (1 de Mayo de 2017). *Pioneer 3-AT*. Obtenido de http://www.mobilerobots.com/researchrobots/p3at.aspx

MONTEMERLO, M. (2003). *FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem with Unknown Data Association.* Pittsburgh: Carnegie Mellon University.

MURPHY, R. R. (2000). *Introduction to AI Robotics.* Cambridge: MIT Press.

OPEN SOURCE ROBOTICS FOUNDATION, I. (27 de Enero de 2017). *TurtleBot*. Obtenido de http://www.turtlebot.com/

OPENSLAM.ORG. (8 de Febrero de 2017). *GMapping*. Obtenido de http://openslam.org/gmapping.html

QUINGLEY, M., GERKEY, B., & SMART, W. (2016). *Programming Robots with ROS.* California: O'Reilly Media.

RAPADO, J. (2016). *Diseño e implementación de una interfaz grafica de usuario para mapeado de entornos y navegación en ROS.* Valencia: Universidad Politecnica de Valencia.

RBR. (10 de Febrero de 2017). *Karto SLAM Solution from SRI Adds Features and Open-Source Option in Version 2.0*. Obtenido de https://www.roboticsbusinessreview.com/uncategorized/karto-slam-solution-from-sri-adds-features-and-open-source-option-in-versio/

ROBOTNIK. (15 de Mayo de 2017). *Robot móvil SUMMIT XL*. Obtenido de http://www.robotnik.es/robots-moviles/summit-xl/

ROBOTSHOP. (7 de Abril de 2017). *Tetra DS-IV Advanced 2WD Robotic Platfrom*. Obtenido de http://www.robotshop.com/en/tetra-ds-iv-advanced-2wd-robotic-platfrom.html

ROS.org. (7 de Febrero de 2017). *vslam Package Summary*. Obtenido de http://wiki.ros.org/vslam

SANDERHAUF, N., & PROTZEL, P. (2010). Learning from nature: Biologically inspired robot navigation and slam. *Kanstliche Intelligenz, 24*, 215-221.

TEJADA, G., & BENAVIDES, F. (2014). *Estudio del estado del arte del SLAM e implementación de una plataforma fléxible.* Motevideo: Universidad de la Republica.

THRUN, S., BURGARD, W., & FOX, D. (2005). *Intelligent robotics and autonomous agents.* The MIT Press.

UTN. (15 de Junio de 2016). *Carrera de Mecatrónica - Perfil Profesional*. Obtenido de http://www.utn.edu.ec/fica/carreras/mecatronica/?page_id=29

VERDERO, J. (7 de Marzo de 2017). *Implementación del FKE para SLAM.* Obtenido de http://bibing.us.es/proyectos/abreproy/4535/fichero/

VILLAROEL, C., CALDERÓN, C., & CARRILLO, R. (2003). *Diseño en Implementación de un equipo de robots autónomos que toman decisiones en tiempo real: Fútbol Robótico -Componente Inteligente.* Guayaquil: ESPOL.

ANEXOS

Lista de anexos:

Anexo A: HOJAS DE DATOS TÉCNICOS

Anexo B: CÓDIGOS ROS

Anexo C: PRESUPUESTO