ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Facultad de Ingeniería en Mecánica y Ciencias de la Producción

"Implementación de SLAM (Simultaneous Localization and Mapping) con un Robot Móvil"

TRABAJO FINAL DE GRADUACIÓN

Proyecto Integrador

Previo la obtención del Título de:

INGENIERO MECÁNICO

Presentado por:

Blas Francisco Hernández Castro

Jorge Esteban Morochz Sandoya

GUAYAQUIL - ECUADOR

Año: 2017

AGRADECIMIENTOS

Agradezco a la vida por todas las oportunidades que se me han brindado y las lecciones que me ha enseñado, a mi madre Roxana por todos sus sacrificios y esfuerzos, a mi novia Natalia por su tiempo, paciencia y el regalo que tiene en su vientre, a mi perrito, que en paz descansa, a mis hermanos, tías, familiares y amigos que estuvieron conmigo cuando más los necesité.

A los ingenieros Carlos Cifuentes y Jaime Vásquez por enseñarme el amor a la ciencia, y a todos aquellos profesores que dieron parte de su vida para formarme a lo largo de mi carrera e hicieron de la ingeniería mi más valiosa experiencia.

Blas Francisco Hernández Castro

AGRADECIMIENTOS

Agradecimiento eterno a Dios por todas las bendiciones que nos ha brindado, a nuestros padres, por su esfuerzo y apoyo incondicional a lo largo de toda nuestras vidas, familiares, amigos y profesores quienes con su ejemplo y sabios consejos nos guiaron y ayudaron durante toda nuestra formación profesional.

Jorge Esteban Morochz Sandoya

DECLARACIÓN EXPRESA

“La responsabilidad del contenido desarrollado en la presente propuesta de la materia integradora corresponde exclusivamente al equipo conformado por:

Blas Francisco Hernández Castro
Jorge Esteban Morochz Sandoya
Jorge Hurel Ezeta, Ph.D.

y el patrimonio intelectual del mismo a la Facultad de Ingeniería en Mecánica y Ciencias de la Producción (FIMCP) de la ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL”.

Blas Hernández Castro
Autor 1

Jorge Morochz Sandoya
Autor 2

Jorge Hurel Ezeta, Ph.D.
Tutor de Materia Integradora

RESUMEN

En tiempos modernos, las actividades de producción del ser humano se han vuelto monótonas, e incluso peligrosas, por lo que se ha tenido la necesidad de diseñar robots que reemplacen al hombre en determinadas labores. Específicamente, la robótica móvil ha ido evolucionando para dar paso a sistemas más autónomos que puedan cumplir sus tareas programadas. SLAM (*Simultaneous Location And Mapping*) es una técnica reciente que se ha venido estudiando e implementando para optimizar la percepción de la condición espacial que un robot tiene sobre sí mismo. A pesar de la poca vigencia del método SLAM, se han desarrollado soluciones satisfactorias, considerando la incertidumbre de los sensores de mapeo. El objetivo de este proyecto es implementar el algoritmo RTAB-MAP con un sensor Kinect 360 para realizar un SLAM del jardín de una casa mediante un robot Lego Mindstorms EV3 controlado por ROS (*Robot Operating System*), con la finalidad de validar al método al comparar el mapa obtenido por SLAM con uno real, debidamente acotado. Se utilizó el sistema operativo Ubuntu, una memoria micro SD con una versión de ROS para el EV3, H4R Yocto, y un adaptador de WiFi para la comunicación inalámbrica entre la PC y el robot. Se siguió el procedimiento de instalación y configuración descrito en ROS *tutorials*. Una vez mapeado el entorno, se calculó el error porcentual de SLAM comparado con la realidad, obteniendo un error promedio del 3.53% y una desviación del 2.87%, confirmando la factibilidad de aplicar SLAM en sitios desconocidos de difícil acceso o peligro inminente. También se pudo constatar la diferencia entre la nube de puntos generada en 3D y sus respectivas fotografías del jardín mapeado. Se recomienda continuar el proyecto para la implementación de la navegación autónoma del robot y evasión de obstáculos.

Palabras Claves: SLAM, Kinect, Robot móvil, ROS, Ubuntu.

ABSTRACT

In modern times, production activities of human being have become monotonous, and even dangerous, so it has been necessary to design robots that replace man in certain tasks. Specifically, mobile robotics has evolved to give way to more autonomous systems that can fulfill their assigned tasks. SLAM (Simultaneous Location And Mapping) is a recent technique that has been studied and implemented to optimize the perception of the spatial condition that the robot has of itself. Despite the lack of validity of the SLAM method, satisfactory solutions have been developed, although the uncertainty of the mapping sensors. The objective of this project is to implement the RTAB-MAP algorithm with a Kinect 360 sensor to perform a SLAM in the garden of a house by using a Lego Mindstorms EV3 robot controlled by ROS (Robot Operating System), in order to validate the method when comparing the map obtained by SLAM with a real one, duly dimensioned. It was used Ubuntu operating system, a micro SD memory with a version of ROS for the EV3, H4R Yocto, and a WiFi adapter for wireless communication between the PC and the robot. The installation and configuration procedure described in ROS tutorials was followed. Once the environment was mapped, the percentage error of SLAM was calculated compared with reality, obtaining an average error of 3.53% and a deviation of 2.87%, confirming the feasibility of applying SLAM in unknown places of difficult access or imminent danger. It was also possible to verify the difference between the point clouds generated in 3D and their respective photographs of the mapped garden. It is recommended to continue the project for the implementation of autonomous robot navigation and obstacle evasion.

Keywords: SLAM, Kinect, Robot móvil, ROS, Ubuntu.

ÍNDICE GENERAL

RESUMEN I

ABSTRACT II

ÍNDICE GENERAL III

ABREVIATURAS V

SIMBOLOGÍA VI

ÍNDICE DE FIGURAS VII

ÍNDICE DE TABLAS VIII

ÍNDICE DE PLANOS IX

CAPÍTULO 1 1

1. Introducción 1

1.1 Descripción del problema 1

1.2 Objetivos 2

1.2.1 Objetivo General 2

1.2.2 Objetivos Específicos 2

1.3 Marco teórico 3

1.3.1 Robótica Móvil 3

1.3.2 Dispositivo Kinect 5

1.3.3 Odometría. 7

CAPÍTULO 2 8

2. Metodología 8

2.1 Selección del Sistema Operativo. 8

2.2 Descripción del Sistema Operativo para Robótica (ROS) 10

2.2.1 Interfaz de Usuario en ROS. 12

2.3 Selección del Algoritmo para SLAM. 12

2.4 Descripción del algoritmo RTAB-Map 14

2.5 Diseño Conceptual 17

2.6 Diseño Detallado. 18
2.7 Consideraciones éticas y legales. 22
CAPÍTULO 3 23
3. Resultados 23
3.1 Análisis de Resultados. 23
3.2 Análisis de costos. 32
CAPÍTULO 4 34
4. Discusión y Conclusiones 34
4.1 Discusión. 34
4.2 Conclusiones. 39
4.3 Recomendaciones. 40
BIBLIOGRAFÍA
APÉNDICES
APÉNDICE A
Datos Técnicos del Equipo
APÉNDICE B
Códigos Utilizados
APÉNDICE C
Esquema de Grafos
APÉNDICE D
Plano Esquemático del Equipo

ABREVIATURAS

ESPOL	Escuela Superior Politécnica del Litoral.
FIMCP	Facultad de Ingeniería en Mecánica y Ciencias de la Producción.
SLAM	Simultaneous Localization and Mapping.
RGB	Red, Green, Blue.
RGB-D	Red, Green, Blue - Depth.
ROS	Robot Operating System.
LEGO	Leg Godt
KUKA	Keller und Knappich Augsburg.
IEEE	Institute of Electrical and Electronics Engineers.
MIT	Massachusetts Institute of Technology.
EV3	Evolution 3.
IDE	Integrated Drive Electronics.
GIRA	Grupo de Investigación en Robótica Autónoma.
ROS-PKG	Robot Operating System Package.
SO	Sistema Operativo.
MATLAB	Matrix Laboratory.
RTAB MAP	Real-Time Appearance-Based Mapping.
ICRA	International Conference on Robotics and Automation.
WM	Working Memory.
LTM	Long Term Memory.
STM	Short Term Memory.
SM	Sensorial Memory.
HMI	Human-Machine Interface.
2D	Dos Dimensiones.
3D	Tres Dimensiones.
AC	Alternating Current.
WiFi	Wireless Fidelity.
SD	Secure Digital.
FPS	Frames per Second.
CAD	Computer-Aided Design.
RAM	Random Access Memory.

SIMBOLOGÍA

mm	Milímetro.
cm	Centímetro.
m	Metro.
s	Segundo.
min	Minuto.
kHz	kiloHertz.
V	Voltio.
A	Amperio.
W	Watts.
HP	Caballo de Fuerza.
°	Grado Sexagesimal.
rad	Radianes.
rpm	Revoluciones por Minuto.
lx	Lux.
lm	Lumen.
fps	Cuadros por segundo.
GB	Gigabyte.

ÍNDICE DE FIGURAS

Figura 1.1. KUKA Titan, robot industrial. 4
Figura 1.2. Robot LEGO EV3. 5
Figura 1.3. Elementos de una Kinect. 6
Figura 1.4. Rango de visión del dispositivo Kinect. 6
Figura 2.1 Esquema de comunicación entre nodos en ROS. 11
Figura 2.2. Esquema de uso de la memoria fraccionada del robot. 16
Figura 2.3. Diseño Conceptual del Sistema que implementará el SLAM. 17
Figura 2.4. Esquema del producto final; especificaciones y limitaciones. 20
Figura 2.5. Diagrama de bloques de la metodología para SLAM. 21
Figura 3.1. Esquema del jardín usado en SLAM, con Autodesk AutoCAD. 23
Figura 3.2. Fotografía panorámica del lugar del ensayo para SLAM. 24
Figura 3.3. Fotografía del lado norte del jardín mapeado. 24
Figura 3.4. Fotografía del lado sur del jardín mapeado. 25
Figura 3.5. Mapa en 3D y nube de puntos respectivas para la Figura 3.2. 26
Figura 3.6. Nube de puntos equivalentes a la Figura 3.3. 26
Figura 3.7. Nube de puntos equivalentes a la Figura 3.4. 27
Figura 3.8. Vista superior del mapa en 3D en RTAB-Mapviz. 27
Figura 3.9. Detección de los lazos cerrados y odometría visual. 28
Figura 3.10. Mapeo en 3D de un patio. 29
Figura 3.11. Mapa en 2D del jardín con las líneas de la trayectoria del robot. 30
Figura 3.12. Proyección del mapa 3D en la rejilla de Rviz. 31
Figura C.1. Esquema de Grafos con todos los nodos utilizados en SLAM……52

ÍNDICE DE TABLAS

Tabla 2.1. Matriz de ponderación de los criterios para la selección del SO. 9
Tabla 2.2. Matriz de decisión para la selección del SO. 9
Tabla 2.3. Matriz de ponderación de criterios para la selección del algoritmo.....12
Tabla 2.4. Matriz de decisión para la selección del algoritmo.....13
Tabla 3.1. Comparación de cotas del mapa original y del mapeo por SLAM. 32
Tabla 3.2. Costos de los componentes asociados al proyecto. 33
Tabla A.1. Datos técnicos del equipo utilizado en la implementación de SLAM……46

ÍNDICE DE PLANOS

PLANO 1 Robot autónomo móvil para SLAM.

CAPÍTULO 1

1. INTRODUCCIÓN

1.1 Descripción del problema

Ante la necesidad de efectuar trabajos en zonas desconocidas, tareas repetitivas de movilización en entornos estáticos, actividades en lugares que presentan condiciones de baja seguridad para el ser humano o una gran dificultad para el ingreso de trabajadores, surgió la necesidad de desarrollar robots autónomos que realicen una exploración inicial en dichos lugares con el fin de obtener un mapa consistente del área en el que se llevarán a cabo dichas funciones. Debido a esto, con el paso de los años se fue desarrollando una alternativa que cubriría estas necesidades mediante el avance de la robótica móvil y la programación de algoritmos que, en la actualidad en su conjunto componen el método SLAM (Arroyo, 2013).

El problema SLAM (*Simultaneous Localization And Mapping*) se define como el desplazamiento de un robot móvil en un entorno desconocido, el cual a medida que va explorando, obtiene información del medio y va aprendiendo de este.

Utilizando un conjunto de códigos y técnicas usadas por robots y vehículos autónomos para analizar y planificar sus movimientos en tiempo real, se obtiene la capacidad de construir incrementalmente un mapa consistente del entorno desconocido en el que se encuentra, mientras que, en forma simultánea, este mapa es utilizado para determinar su propia localización y estimar la trayectoria óptima al desplazarse (Lugo, 2015).

Los principales problemas del método SLAM son consecuencia de las limitaciones físicas que presentan los diferentes tipos de sensores utilizados para obtener la información del entorno en donde se ubica el robot móvil, debido a esto, se utilizan modelos probabilísticos para reducir errores inherentes al proceso y realizar mejores estimaciones, desarrollando así la capacidad de aprender continuamente del entorno (Montemerlo, 2003).

En la última década, SLAM ha sido desarrollado con una variedad de sensores que permiten la solución a este problema dual, entre los que destacan los sensores RGB-D, los cuales brindan valiosa información visual que permite reconocer lugares ya visitados. A su vez, los sensores RGB-D tienen la capacidad de medir la profundidad de cada uno de los pixeles correspondientes a las imágenes obtenidas. Uno de los dispositivos más utilizados que contienen estos tipos de sensores, es la cámara Kinect de Microsoft.

A nivel mundial, gran parte de los trabajos desarrollados para explorar y explotar todas las bondades que ofrece el dispositivo Kinect, se han realizado empleando drivers y librerías para programar en C++ o en Python. Por lo cual, para resolver el problema que representa SLAM, se debe utilizar un sistema operativo para aplicaciones de robótica que posea un compilador eficaz de código fuente y a la vez nos ofrezca las herramientas necesarias para poder adquirir y controlar los datos de la cámara Kinect (Rapado, 2016).

1.2 Objetivos

1.2.1 Objetivo General

- Implementar la localización y mapeo simultáneos (SLAM) utilizando sensores de un dispositivo Kinect en un robot móvil.

1.2.2 Objetivos Específicos

- Elaborar un mapa consistente de un entorno desconocido.
- Controlar el desplazamiento de un LEGO Mindstorms EV3 para la implementación de SLAM.
- Calcular los errores existentes en las dimensiones del mapa generado por SLAM.

1.3 Marco teórico

A lo largo de la historia, el ser humano ha convivido con la idea de elaborar máquinas que tengan la capacidad de realizar tareas específicas sin ser controladas por personas, con el fin de reemplazar su trabajo y, en ocasiones, hacerlo de una forma más eficiente y segura. Como resultado del desarrollo de estas ideas, surgen los robots autónomos, que son máquinas con la capacidad de analizar información, tomar decisiones propias para realizar una tarea específica y conseguir un objetivo propuesto, el cual podría ser la exploración de entornos desconocidos o el desarrollo de actividades repetitivas que impliquen movilización dentro de entornos principalmente estáticos, esquivando todos los obstáculos presentes en dichos lugares.

SLAM tiene sus inicios en el año 1987 durante una jornada del IEEE, en donde se propuso una idea que supondría un avance de gran relevancia para la época. Se describió un sistema de navegación y mapeo basado en información obtenida con sonares y desarrollado en conjunto con un robot móvil para brindarle la capacidad de poder movilizarse en entornos desconocidos (Elfes, 1987).

Esencialmente, existen cuatro componentes primordiales para poder implementar SLAM: un robot móvil, sensores, códigos para SLAM basados en métodos estadísticos y un computador que integre, procese y controle toda la información obtenida, mediante el cual se llevarán a cabo las tareas de elaboración del mapa del entorno, lectura y filtrado de los datos recibidos desde los sensores, la decisión de la orden a ejecutar basados en el mapa y la posición del robot, y la comunicación con el robot mediante librerías (Mayans, 2012).

1.3.1 Robótica Móvil

Desde hace varias décadas, la robótica ha sufrido una marcada ampliación de las áreas y de las actividades en las que puede intervenir, clasificándose así, según su campo de aplicación, en robots industriales y robots de servicios.

Figura 1.1. KUKA Titan, robot industrial.

Fuente: https://www.kuka.com/es-es

Existen sectores en los cuales no es necesario mantener una alta productividad, en donde las actividades que se están desarrollando no son repetitivas y se carece de información a detalle del entorno de trabajo; debido a estas características no existe la posibilidad de sistematizar y clasificar los diferentes movimientos y acciones que se llevarán a cabo. Es entonces cuando las aplicaciones requieren de un robot de servicio, el cual debe contar con un mayor grado de inteligencia y autonomía mediante el uso de sensores y del software preciso que les permita tomar decisiones adecuadas rápidamente.

Para alcanzar la autonomía de los robots de servicio, es necesario emplear un robot móvil, el cual es una máquina con un diseño que le permite movilizarse en un determinado entorno. Estos, han sido foco de investigación durante las últimas décadas y hoy en la actualidad una gran parte de las instituciones universitarias promueven su uso para el desarrollo de aplicaciones en tareas domésticas, productivas, de rescate y hasta para la industria militar (López, 2016).

Bajo este contexto surgió un acuerdo entre LEGO y el MIT para lanzar una línea de juguetes para el desarrollo de aplicaciones en la robótica móvil mediante un modelo de sistema integrado con partes electromecánicas controladas por un entorno de desarrollo integrado (IDE) en un computador.

Este enfoque está alejado del clásico juguete comercializado por la compañía y busca experimentar con novedosos elementos que utilizan tecnología actual, tanto del área mecánica como de la electrónica. Hasta la actualidad, la compañía de juguetes ha desarrollado tres generaciones de estos productos, siendo el LEGO Mindstorms EV3 el último producto de esta línea.

Figura 1.2. Robot LEGO EV3.

Fuente: https://www.tcea.org/blog/ev3-software-update/.

1.3.2 Dispositivo Kinect

El dispositivo Kinect es un periférico de entrada desarrollado por la empresa Microsoft para la consola de videojuegos Xbox 360, cuyo objetivo era desarrollar un control para los videojuegos que permitiese interactuar al jugador con la consola por medio de la imagen y movimiento del usuario (Beltrán & Basañez, 2014).

Este dispositivo cuenta con un sensor infrarrojo que permite obtener los datos de profundidad y una cámara RGB que captura imágenes a color del entorno. El funcionamiento de estas cámaras se basa en el uso de sensores con la capacidad de convertir la señal obtenida en forma de fotones a una señal electrónica digital, la misma que se encarga de descomponer la luz capturada en tres componentes: rojo, verde y azul, de donde se obtiene su nombre (RGB) por las palabras en inglés *Red, Green & Blue* (Viñals, 2012).

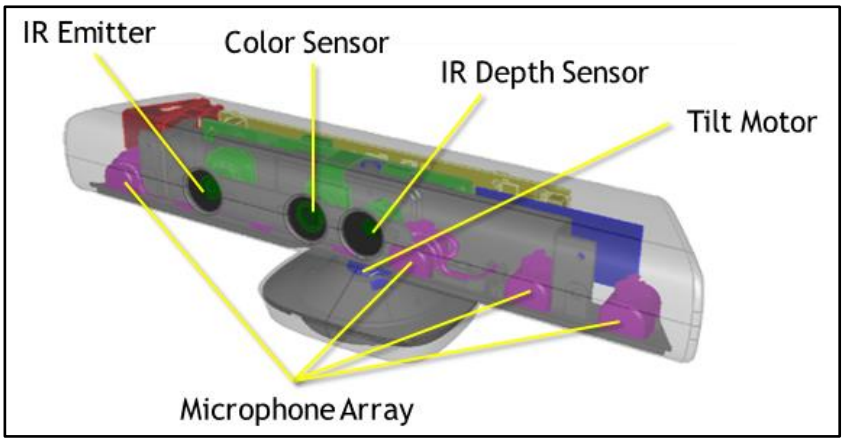


Figura 1.3. Elementos de una Kinect.

Fuente: https://www.msdn.microsoft.com/

El campo de visión en dirección horizontal del dispositivo Kinect es de aproximadamente 57°, mientras que el rango de distancia para la captación de profundidad es desde 80 centímetros (40 centímetros en la última versión del dispositivo) hasta aproximadamente 4 metros.

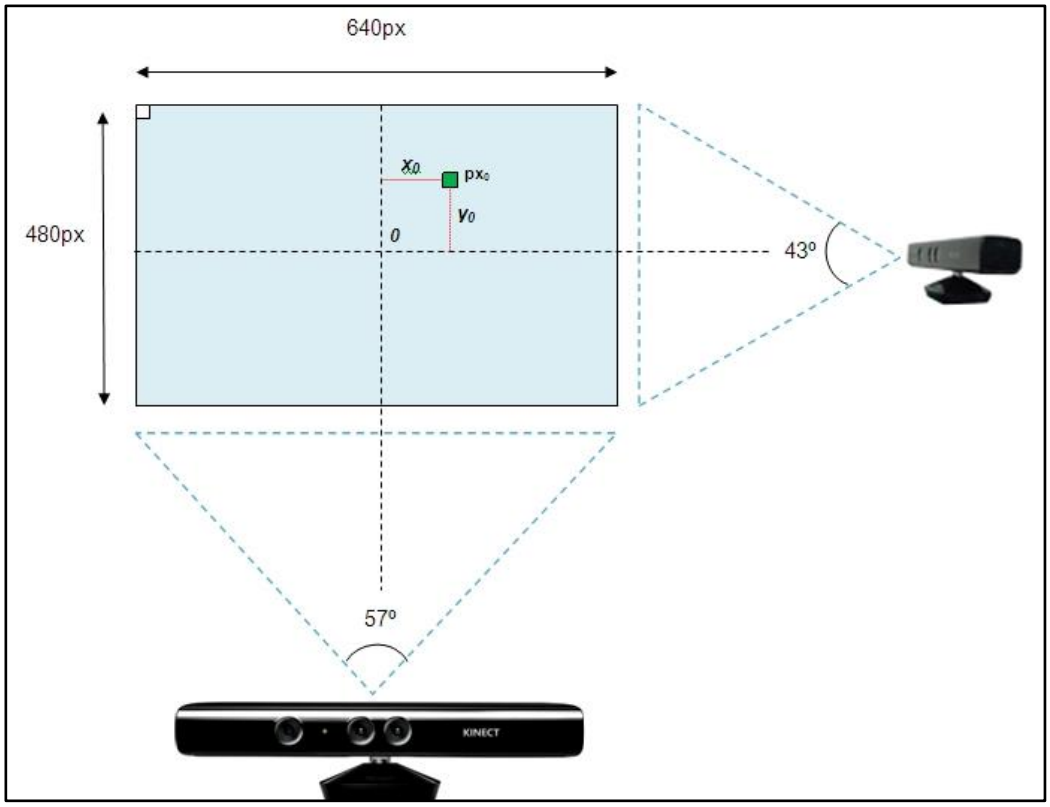


Figura 1.4. Rango de visión del dispositivo Kinect.

Fuente: http://tecnodacta.com.ar/gira/tag/kinect/.

A su vez, el campo de visión de la cámara en dirección vertical es de aproximadamente 43°. Cabe destacar que el dispositivo cuenta con un soporte motorizado que permite inclinar la Kinect ±27° adicionales en dirección vertical, lo cual permite cubrir cómodamente las dimensiones generales de cualquier persona independientemente de la posición de la cámara.

1.3.3 Odometría.

La odometría en la robótica móvil, se define como la estimación de la posición instantánea de un robot respecto al sistema de coordenadas. Esto se logra mediante la información obtenida por *encoders* (generadores de pulsos) de cada una de las ruedas del robot; los datos de las revoluciones de cada rueda son traducidos a un desplazamiento lineal, con el que se procede a calcular la localización instantánea del robot (Zabala *et al*., 2014).

Cabe recalcar que el cálculo incremental de desplazamientos en la odometría supone una acumulación de errores que afectarían el cálculo de la posición final. Estos errores pueden ser sistemáticos (al realizar la medición de los diámetros de las ruedas, la existencia de un desalineamiento e incertidumbre en la posición y orientación inicial del robot) o pueden presentarse en situaciones en las que las ruedas del robot patinen o experimenten una sobre-aceleración (Riisgaard & Blas, 2005).

CAPÍTULO 2

2. METODOLOGÍA

2.1 Selección del Sistema Operativo.

El primer paso para conseguir la implementación de SLAM supone la selección de un sistema operativo en donde se desarrollará la programación, control e integración de los diferentes dispositivos necesarios para el proyecto. Bajo este contexto, se han definido dos sistemas operativos (SO) que surgen como alternativas para cumplir con las funciones requeridas en SLAM, dichas alternativas son:

1) Desarrollar todo el proyecto en el sistema operativo Windows 10 con la ayuda de una herramienta de *software* matemático llamada Matlab.

2) Desarrollar todo el proyecto en el sistema operativo Ubuntu, haciendo uso a su vez, de un entorno de trabajo que cuenta con bibliotecas y algoritmos para aplicaciones en robótica llamado ROS.

De acuerdo con el análisis y comparación de las características de los diferentes sistemas operativos elaborado por (Rodríguez, 2011) y de las recomendaciones de (Mayans, 2012), que se basan en proyectos que utilizan SLAM en diversas aplicaciones, se procedió a definir los criterios de mayor relevancia para el proyecto y la importancia que estos tienen (Tabla 2.1). Con estos datos, se realizó una matriz de decisión (Tabla 2.2) con el objetivo de definir la mejor alternativa.

Para el desarrollo de la matriz de decisión, se estableció una escala del 1 al 10 para calificar de forma subjetiva a cada uno de los criterios, en donde el número 1 y el número 10 corresponden a la calificación más baja y a la calificación más alta, respectivamente. Luego, se ponderó cada una de estas calificaciones y se las sumó, obteniendo así un puntaje total tabulado en la

matriz de decisión, la misma que, muestra como mejor alternativa a la que posea el puntaje total más alto.

Tabla 2.1. Matriz de ponderación de los criterios para la selección del SO.

Criterio	Ponderación [%]
1) Conocimientos previos sobre el SO.	15
2) Facilidad de uso del Sistema Operativo para programación.	20
3) Desarrollo previo de proyectos SLAM en el SO.	30
4) Conocimiento, experiencia y recomendaciones acerca del SO. del profesor tutor y profesores involucrados.	35

Fuente: Elaboración propia.

Tabla 2.2. Matriz de decisión para la selección del SO.

Criterio por calificar según referencia de la Tabla 2.1	Sistema Operativo Windows 10		Sistema Operativo Ubuntu	
	Calificación	Calificación Ponderada	Calificación	Calificación Ponderada
1	8	1.20	1	0.15
2	7	1.40	10	2.00
3	7	2.10	10	3.00
4	7	2.45	9	3.15
Resultados	Total = 7.15		Total = 8.30	

Fuente: Elaboración propia.

Se observa que la mejor opción es el uso del sistema operativo Ubuntu, el cual, a pesar de no conocerlo previamente, se sabe que brinda muchas facilidades para la programación, tales como el acceso a la terminal (shell) que hace de interfaz entre el usuario y el propio sistema operativo. De igual forma, cuenta con una gran variedad de algoritmos y bibliotecas desarrollados por miles de personas en todo el mundo por ser un software libre. Otra ventaja es que Ubuntu cuenta con un entorno de trabajo para aplicaciones en el área

de la robótica (ROS), en donde se han llevado a cabo muchos trabajos relacionados a la implementación de SLAM, por lo que se constituye como una herramienta de gran valor.

Finalmente, cabe destacar que basados en la experiencia y conocimientos de las fuentes consultadas y recomendaciones de los profesores que estuvieron vinculados al proyecto, Ubuntu 16.04 (última versión estable y reconocida) es la mejor opción para el desarrollo e implementación de la técnica SLAM.

2.2 Descripción del Sistema Operativo para Robótica (ROS)

Desde el año 2008 se ha desarrollado un meta sistema operativo de código abierto para el desarrollo de *software* en el área de robótica, conocido como ROS (*Robot Operating System*), el cual está basado en una arquitectura de grafos donde todo el procesamiento de la información se lleva a cabo en unos nodos que reciben y envían mensajes desde los diferentes sensores, controladores y actuadores a través de tópicos.

El esquema o grafo de computación para la comunicación en ROS (Figura 2.1) está formado principalmente por las siguientes entidades que se encuentran relacionadas entre sí:

- Nodos: principales unidades funcionales de ROS. Cada nodo está encargado de realizar una parte específica de un trabajo, siendo responsable de la ejecución de un algoritmo. En la mayoría de los sistemas, solo se consigue ejecutar una tarea cuando hay la coexistencia de una amplia cantidad de nodos trabajando en conjunto.

- Tópicos: son los encargados de mostrar un punto de conexión en el sistema, con el fin de que los nodos sean capaces de intercambiar información a través de mensajes.

- Servicios: brindan al sistema un modelo de comunicación del tipo publicador/subscriptor que complementa el esquema de comunicación del tipo síncrona entre procesos.
- *Bags*: entidades que cuentan con un mecanismo para el almacenamiento de información, con el fin de brindar la capacidad de reproducirlos y utilizarlos en cualquier momento que se lo requiera.

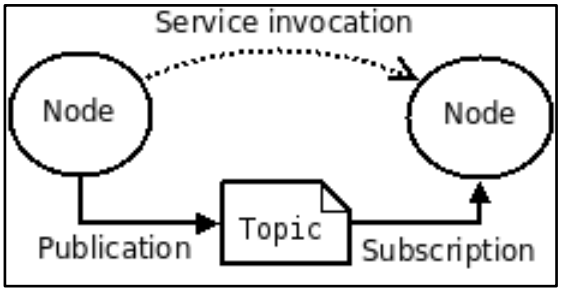


Figura 2.1 Esquema de comunicación entre nodos en ROS.

Fuente: ROS Basic Concepts, 2014.

ROS se compone de dos partes fundamentales: la primera es la del sistema operativo predeterminado y la segunda que se conoce como ROS-pkg, la cual es una agrupación de paquetes aportados por diferentes usuarios (estructurados en grupos conocidos como pilas) que permiten la implementación de diferentes funcionalidades que, por ejemplo, en su conjunto componen a SLAM, tales como percepción, localización, mapeo, evasión de obstáculos, entre otros (Quigley, 2010).

En ROS han sido desarrollados diferentes algoritmos para la implementación del SLAM, tales como: RTAB-Map, ORB-SLAM, RGBD-SLAM, Hector-SLAM, entre otros; los mismos que tienen la capacidad de poder generar una nube de puntos a partir de la información obtenida desde los sensores del dispositivo Kinect para elaborar un mapa del entorno observado, que posteriormente será utilizado para la localización y desplazamiento del robot móvil de forma autónoma.

2.2.1 Interfaz de Usuario en ROS.

Con el aumento de la complejidad de los sistemas que componen un robot y de las diferentes funciones que estos realizan, surgió la necesidad de emplear una interfaz entre el usuario y la máquina, denominada HMI (*Human-Machine Interface*), que permita mejorar la supervisión, adquisición y manipulación de los datos que componen el sistema.

Dentro de ROS existen diversos paquetes que contienen herramientas para la interacción entre el robot y el usuario (HMI), siendo una de las más importantes la interfaz gráfica RViz, puesto que permite la visualización 3D de mapas que son elaborados en tiempo real con información obtenida por sensores y representados mediante nubes de puntos. Por esta razón, RViz representa una herramienta clave en la implementación del SLAM

2.3 Selección del Algoritmo para SLAM.

La implementación de SLAM requiere del uso de un algoritmo que permita la extracción y procesamiento de la información obtenida desde el entorno por medio de los sensores RGB-D de la cámara Kinect 360. Debido a que en la actualidad existen varios algoritmos que cumplen con dichas funciones, surge la necesidad de definir cuál de ellos representa la mejor alternativa para el desarrollo del proyecto. Para esto, se determinaron ciertos criterios, mostrados en la Tabla 2.3 que fueron evaluados y ponderados en una matriz de decisión (Tabla 2.4), en donde finalmente la mejor alternativa es aquella que presenta el puntaje total más alto.

Tabla 2.3. Matriz de ponderación de criterios para la selección del algoritmo.

Criterio	Ponderación [%]
1) Rapidez del algoritmo.	15
2) Recursos computacionales necesarios.	20

3) Capacidad para relocalizarse ante movimientos bruscos de la cámara.	25
4) Fiabilidad del mapa generado.	40

Fuente: Elaboración propia.

Tabla 2.4. Matriz de decisión para la selección del algoritmo.

Criterio por calificar según referencia de la Tabla 2.1.		1	2	3	4	Resultado Ponderado
Héctor-SLAM	Calificación	9	9	6	7	7.45
	Calificación Ponderada	1.35	1.80	1.50	2.80	
RGBD-SLAM	Calificación	8	7	9	9	8.45
	Calificación Ponderada	1.20	1.40	2.25	3.60	
ORB-SLAM	Calificación	9	9	4	7	6.95
	Calificación Ponderada	1.35	1.80	1.00	2.80	
RTAB-Map	Calificación	8	10	9	9	9.05
	Calificación Ponderada	1.20	2.00	2.25	3.60	

Fuente: Elaboración propia.

Basados en los resultados mostrados en la matriz de decisión, se determinó que el algoritmo RTAB-Map es la mejor alternativa para el desarrollo del proyecto.

De acuerdo con experimentos realizados por López (2016) y Lugo (2015), el algoritmo RTAB-Map (*Real-Time Appearance-Based Mapping*) se ejecuta rápidamente sin presentar signos o señales de retraso, gracias a la técnica que este utiliza para el procesamiento de la información obtenida del entorno.

Otra de las características de este algoritmo es la demanda relativamente baja de recursos computacionales que necesita durante el desarrollo de SLAM, lo

cual se traduce en la capacidad de ejecutarse durante un largo período de tiempo y en grandes recorridos.

Finalmente, se conoce que los mapas generados por este algoritmo son muy aproximados a la realidad, lo cual es un factor clave para la localización del robot móvil y el desarrollo eficaz de SLAM en general.

2.4 Descripción del algoritmo RTAB-Map

Uno de los puntos claves en la localización y desplazamiento autónomo de un robot móvil es la capacidad de identificar si la información observada es nueva o si, por el contrario, la información observada corresponde a alguna localización que se haya visitado anteriormente, esto es conocido como detección de un bucle (lazo) cerrado (o en el inglés *Loop Closure Detection*).

Frecuentemente, la detección de un bucle (lazo) cerrado se consigue al realizar una comparación de la observación actual con cada una de las observaciones obtenidas previamente. Sin embargo, en caso de realizar recorridos muy prolongados, el tiempo de cómputo aumentará considerablemente al necesitar comparar demasiadas localizaciones, y en el caso en el que el tiempo de cómputo excediera al tiempo de adquisición de datos de las imágenes obtenidas del entorno, se presentarán retrasos y dificultades en el proceso de elaboración del mapa en tiempo real y en el peor de los casos ocurrirá un colapso del sistema.

Como solución a esta limitación surge el algoritmo RTAB-Map, el cual posee la capacidad de desarrollar el SLAM sin las limitaciones que representa el tiempo de cómputo en recorridos muy prolongados. Para esto, el algoritmo se elaboró bajo una estrategia que divide la memoria de ROS en una memoria de trabajo (WM), una memoria a largo plazo (LTM), una memoria a corto plazo (STM) y en una memoria sensorial (SM). El objetivo de este fraccionamiento es mantener en la WM de ROS a las localizaciones que han sido observadas últimamente y con mayor frecuencia, entretanto, las otras localizaciones se

guardarán en la memoria a largo plazo, reduciendo así la cantidad de cálculos y comparaciones entre las diferentes localizaciones observadas.

El flujo de la información correspondiente a cada imagen se muestra en el diagrama del modelo del algoritmo RTAB-Map (Figura 2.2), en donde se muestra que, inicialmente, las imágenes observadas del entorno son adquiridas por medio del módulo de percepción y enviadas hacia el módulo de memoria sensorial, en donde las imágenes son reducidas para extraerles las características apropiadas que ayuden en el hallazgo de un nuevo bucle cerrado.

El módulo de memoria sensorial se encarga de generar nuevas localizaciones que luego son enviadas a la memoria de corto plazo, en donde se realiza la actualización de las observaciones recientes por medio de un actualizador de pesos, este peso será el equivalente a la cantidad de ocasiones que ha sido visitada una localización. Si en este proceso se determina que la nueva actualización es igual a la última actualización registrada en la memoria de corto plazo, se las combina para generar una nueva actualización con un incremento en el valor de su peso. De esta forma, se registran las localizaciones que han sido visitadas con mayor frecuencia, de las cuales se conoce que representan una mayor tendencia para reproducir bucles cerrados. Finalmente, las localizaciones que posean un menor peso son enviadas a la memoria de largo plazo hasta finalizar el proceso.

De forma resumida, la función principal de la memoria a corto plazo es examinar las características existentes entre imágenes sucesivas con el fin de obtener similitudes entre ellas. Mientras que, la memoria de trabajo es la responsable de descubrir la existencia de bucles cerrados.

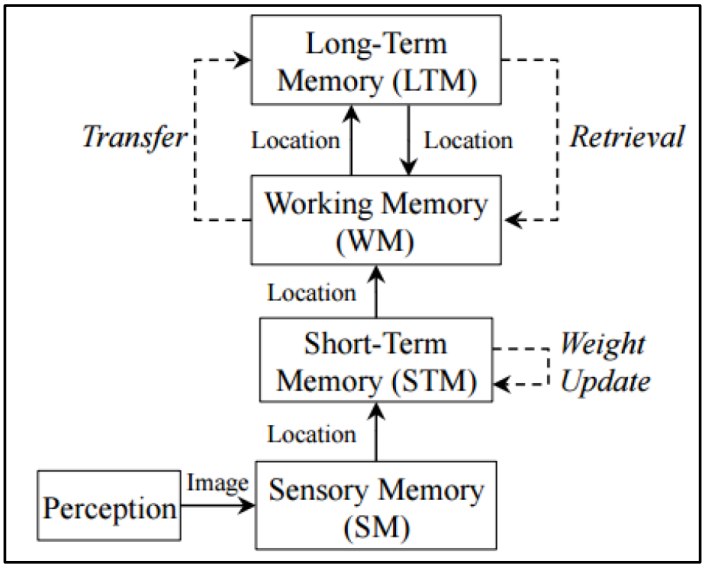


Figura 2.2. Esquema de uso de la memoria fraccionada del robot.

Fuente: Huang, 2017.

2.5 Diseño Conceptual.

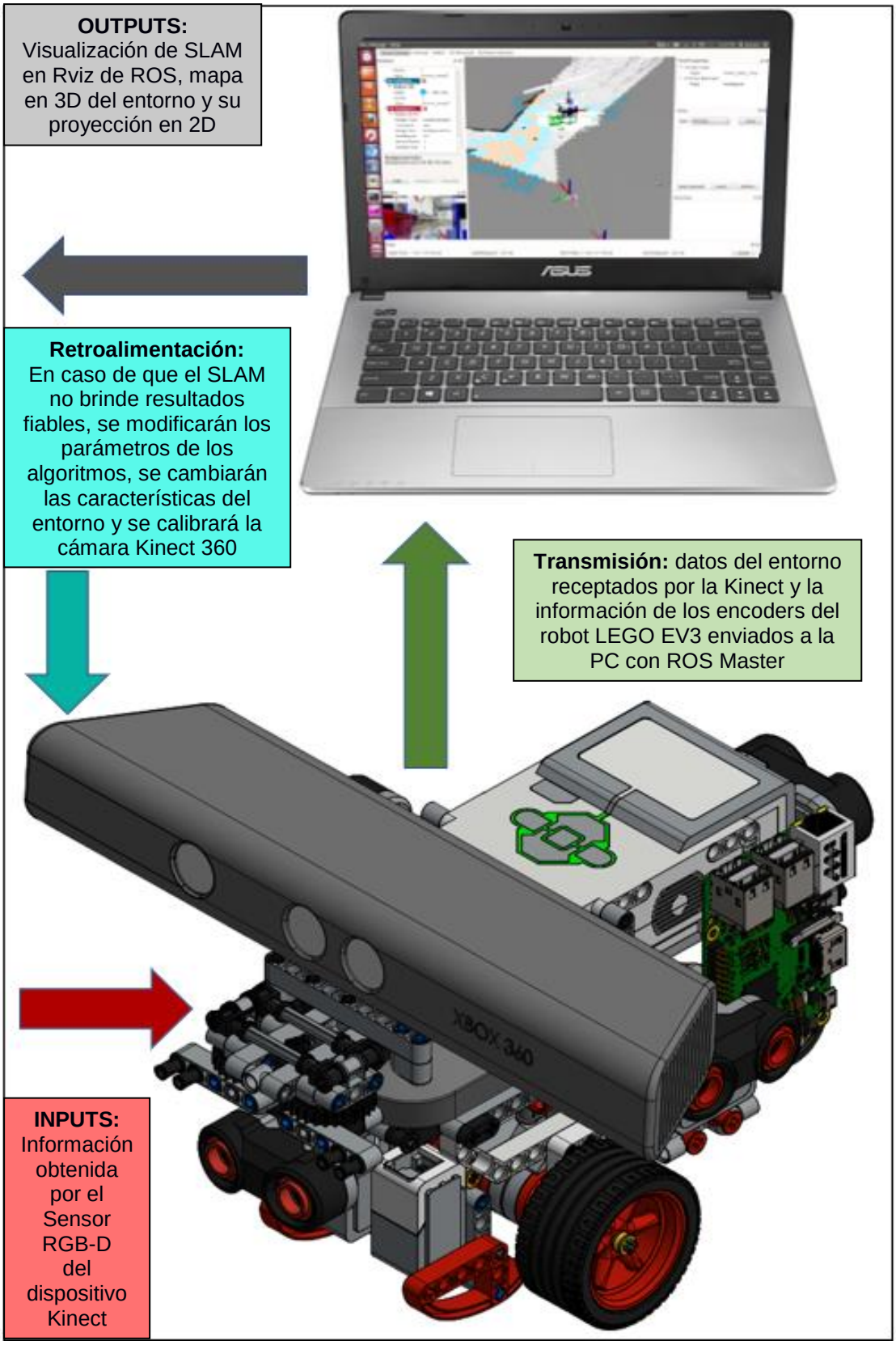


Figura 2.3. Diseño Conceptual del Sistema que implementará el SLAM.

Fuente: Elaboración propia.

La Figura 2.3 muestra los diferentes dispositivos necesarios para la implementación de SLAM y la interrelación existente entre ellos. Los dispositivos son el robot móvil LEGO Mindstorms EV3 cuya función es el desplazamiento del sistema a través del entorno en el que se desarrollará la técnica SLAM, la cámara Kinect con sus respectivos sensores RGB e infrarrojo, y una computadora portátil que se utilizará para el procesamiento de la información y la ejecución de los diferentes algoritmos en ROS. Cabe recalcar que la Figura 2.3 es esquemática y con fines ilustrativos. Existirán modificaciones de acuerdo con las exigencias del usuario.

2.6 Diseño Detallado.

Para la implementación de SLAM con el algoritmo RTAB-Map ejecutándose en ROS, usando una Kinect 360 y maniobrando con un Lego Mindstorms, descrito con anterioridad, fue necesario cumplir una serie de pasos que se detallan más adelante en los ítems correspondientes. La forma de cumplir estos pasos fue tipeando el código fuente en las líneas del terminal de Ubuntu desde las distintas páginas web que contienen las librerías y los paquetes que se descargaron. La Figura 2.4 muestra al robot definitivo utilizado para SLAM, con sus características y limitaciones como producto final (en la tabla A.1 del Apéndice A se tabulan las especificaciones técnicas del equipo), mientras que la Figura 2.5 resume de manera general el procedimiento realizado para cumplir los objetivos de este proyecto. Dichas líneas de código se encuentran en el Apéndice B de este documento.

Por último, con fines didácticos y debido a la dificultad de implementar el sistema de SLAM con el que se trabajó este proyecto integrador, se realizó un video tutorial explicando de manera ordenada y sistemática la instalación y ejecución de los programas con los que se podrían replicar los resultados obtenidos, así como el desarrollo de posteriores proyectos que usen SLAM. En la página oficial de Facebook del Club de Mecatrónica de ESPOL se subió este video, con el nombre de "Tutorial de Implementación de SLAM (*Simultaneous Localization and Mapping*) con un Robot Móvil”.

- Instalación del SO: Para empezar, es necesario realizar la instalación del sistema operativo Linux-Ubuntu 16.04, lo cual puede hacerse directamente desde la página web de Ubuntu. Se debe configurar para permitir las descargas desde cualquier servidor a nivel mundial, sin restricciones.

- ROS-Kinetic: Se instaló esta distribución de ROS debido a que es la versión más estable y respaldada para varios años por sus desarrolladores. Se instalaron los códigos de manejo del entorno, la creación del espacio de trabajo, compilación de los paquetes, visualización de nodos y de datos tridimensionales, entre otros (véase ROS Tutorials en la web para más información).

- *Drivers* de Open Kinect: El siguiente paso fue la instalación de las dependencias que permiten la conexión entre el dispositivo Kinect y ROS, específicamente el driver libfreenect (que ejecuta a freenect launch), para lograr la adquisición de la información desde los diferentes sensores del dispositivo. Luego de esto fue indispensable calibrar la cámara por medio de un tablero de ajedrez modificado de 7x9 cuadrados, con 108 [mm] de lado cada uno.

- Instalación de RTAB-Map: A continuación, se requiere la instalación del paquete de ROS RTAB-Map, que ejecuta el algoritmo de SLAM tipo online. Esto permitió visualizar los mapas generados en tiempo real en tres dimensiones y generar su proyección en dos dimensiones (mapeo del entorno), al mismo tiempo que calculaba la posición y trayectoria (odometría) de la cámara Kinect y, por ende, del robot. Es necesario ejecutar unos comandos en la terminal del sistema operativo cada vez que se desea realizar SLAM. Además, se tuvieron que realizar ciertas operaciones desde la interfaz gráfica de RTAB-Mapviz para optimizar los gráficos y realizar las capturas respectivas que se adjuntan como imágenes en los resultados.

- Instalación de ROS en el EV3: Para poder controlar los servomotores, por medio del teclado, del robot Lego Mindstorms EV3, fue necesario grabar un *firmware* en una memoria externa ejecutable dentro del *brick* del Lego. Este *firmware*, llamado H4R (Hacks for ROS) Yocto-Linux, permite visualizar los nodos de los motores para su control. La tele-operación (movimiento del robot usando el teclado) se realizó mediante un contenedor Docker, que es una pseudo máquina virtual, la misma que facilitó la implementación del algoritmo de tele-operación y arranque de los motores del EV3 sin usar el software predeterminado de Lego.

- Validación de RTAB-Map: Para probar el método de SLAM fue necesario realizar una comparativa entre todas las dimensiones de un mapa escaneado con la Kinect con aquellas cotas que representaron a las mediciones reales del entorno. Una comparación exhaustiva requerirá entonces tomar la mayor cantidad de mediciones posibles del medio y obtener cada respectiva dimensión en Rviz del mapeo, para realizar un posterior análisis estadístico y calcular el error promedio porcentual y la desviación estándar de SLAM.

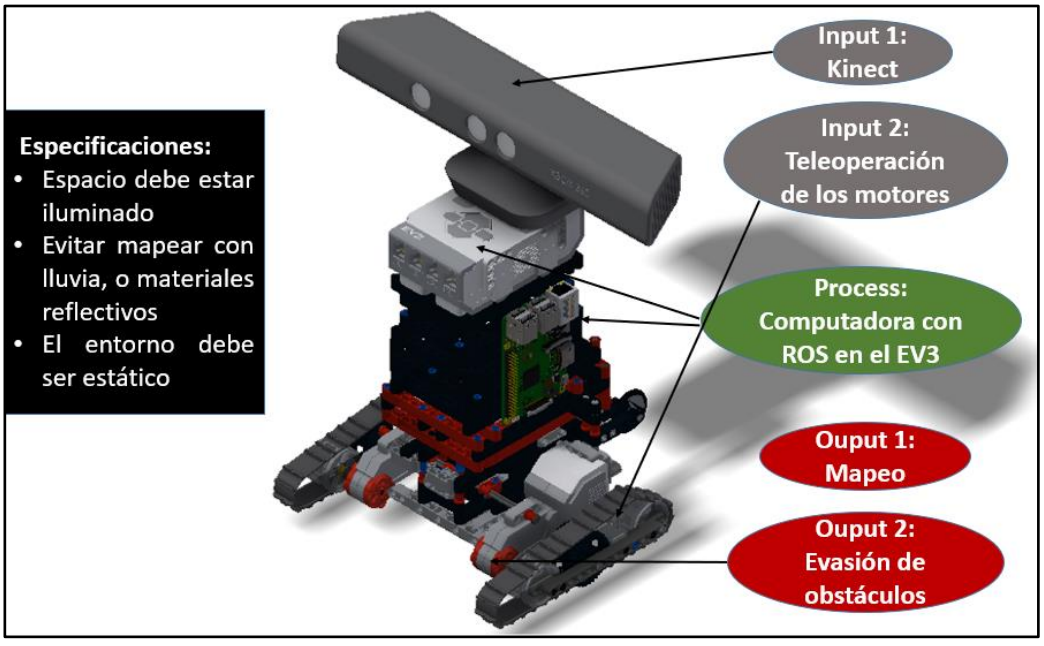


Figura 2.4. Esquema del producto final; especificaciones y limitaciones.

Fuente: Elaboración propia.

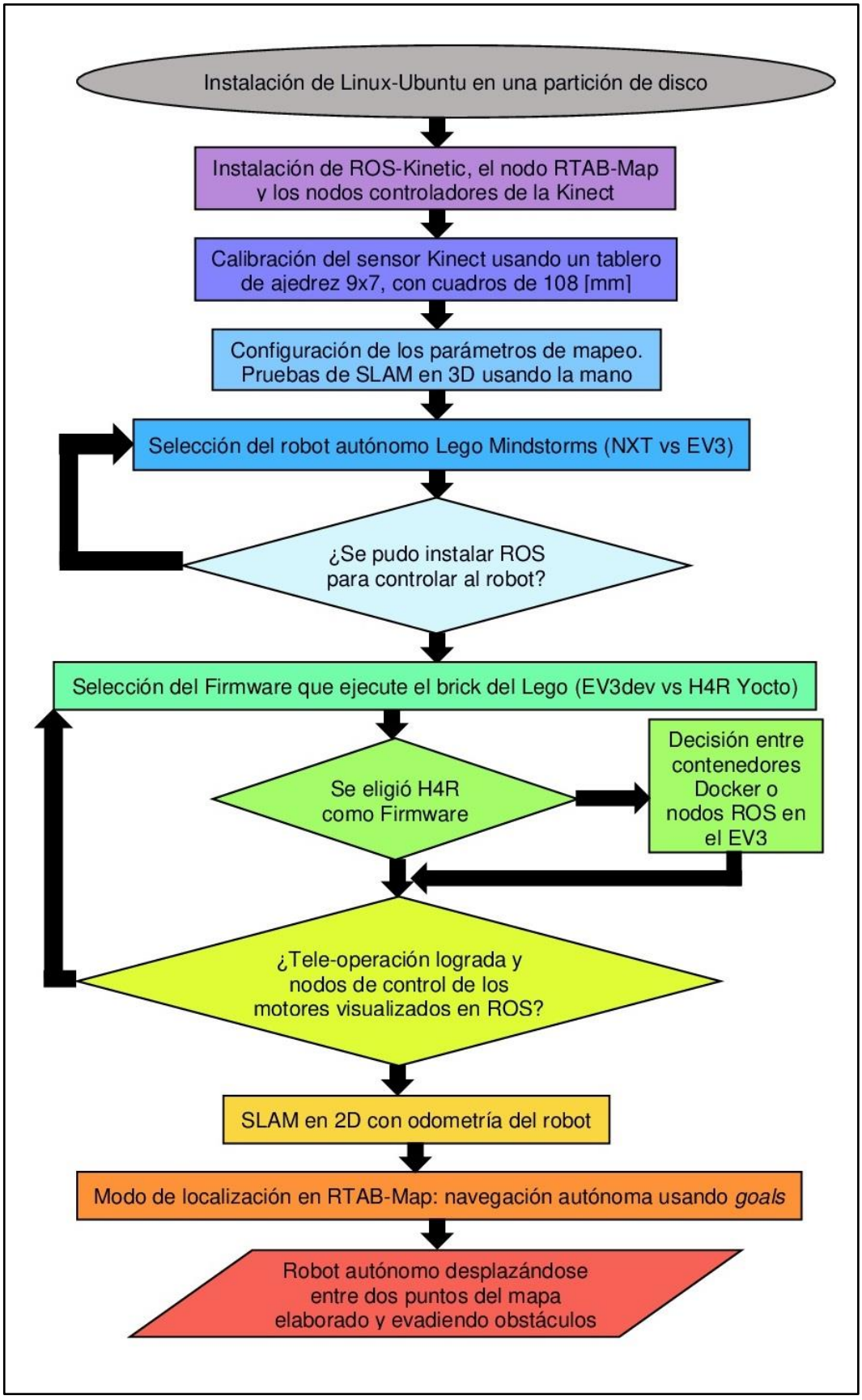


Figura 2.5. Diagrama de bloques de la metodología para SLAM.

Fuente: Elaboración propia.

2.7 Consideraciones éticas y legales.

El hecho de haber descargado e implementado *software*, *firmware*, librerías y algoritmos de *opensources*, excluye a los ejecutores de este proyecto de cualquier problema legal por derechos de autor; del mismo modo, el procedimiento propuesto se basó en tutoriales o páginas tipo wiki de ROS, Ubuntu, Github, y demás, por lo que cualquier persona en el mundo puede trabajar mediante esas páginas sin infringir la ley. No se modificaron los algoritmos, y aquellos códigos que se editaron, no fueron comercializados con fines de lucro. La finalidad de este proyecto (y de la continuación de este) no abarca más allá de las fronteras del campo académico o de investigación, con el último propósito de que la universidad pueda diseñar, construir e implementar sistemas robóticos móviles de seguridad o rescate que sirvan a la sociedad en general y mejoren la calidad de vida de quienes interactúen con estos sistemas.

CAPÍTULO 3

3. RESULTADOS

3.1 Análisis de Resultados.

Para demostrar la validez del algoritmo RTAB-Map se decidió mapear un entorno conocido, específicamente, el jardín de la casa del Sr. Blas Hernández. Se midieron las longitudes más representativas y se realizó un dibujo detallado en 2D, con el *software* AutoCAD, del jardín debidamente acotado (ver la Figura 3.1). En el mapa no se pudo dimensionar correctamente unas macetas con plantas en la esquina inferior izquierda de la Figura 3.1, pero en la Figura 3.2 se muestra una panorámica del jardín donde se realizó el ensayo. En las Figuras 3.3 y 3.4 se muestran fotos con otros ángulos del jardín, para tener una perspectiva más amplia del lugar donde se realizaron las pruebas de SLAM, y poder contrastar los resultados a continuación.

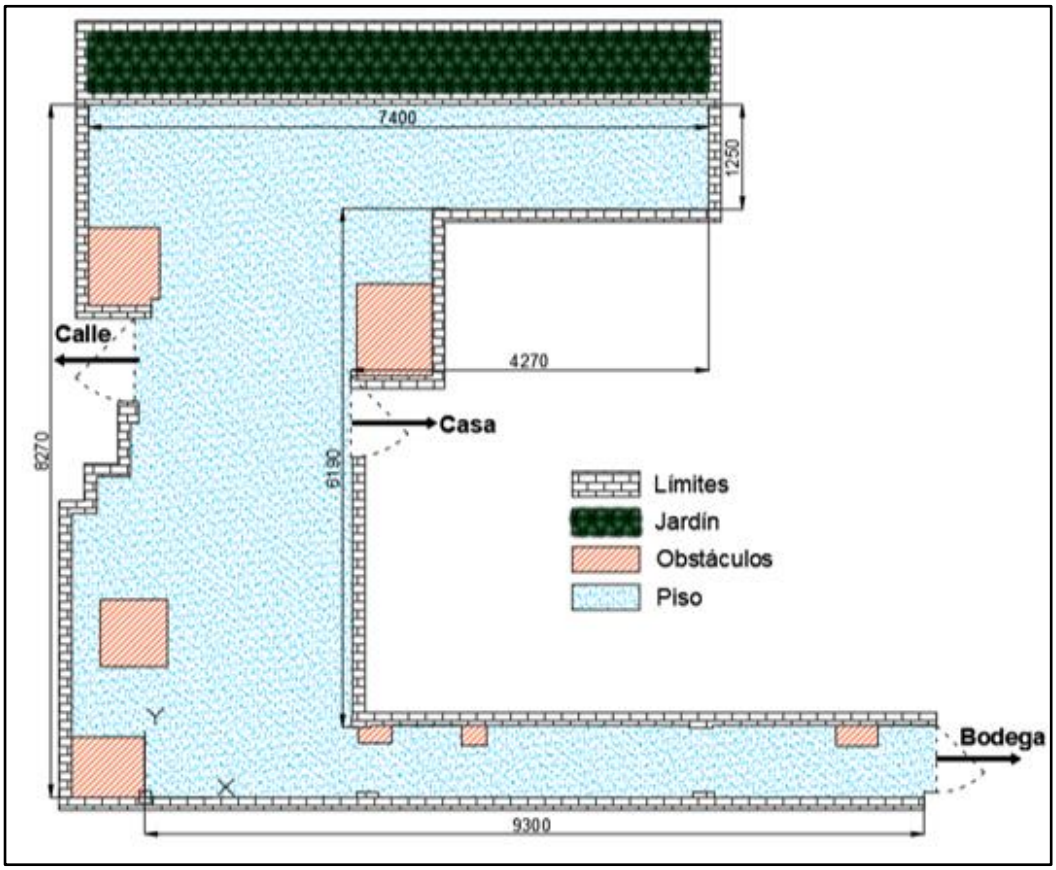


Figura 3.1. Esquema del jardín usado en SLAM, con Autodesk AutoCAD.

Fuente: Elaboración propia.

Figura 3.2. Fotografía panorámica del lugar del ensayo para SLAM.

Fuente: Elaboración propia.

Figura 3.3. Fotografía del lado norte del jardín mapeado.

Fuente: Elaboración propia.

Figura 3.4. Fotografía del lado sur del jardín mapeado.

Fuente: Elaboración propia.

Una vez terminado el mapeado del jardín mediante la tele-operación del robot, con el visor RTAB-Mapviz y Rviz, se obtuvieron las Figuras 3.5 hasta la 3.11, que son los resultados principales del método usado en SLAM. De la Figura 3.5 hasta la 3.7 se observan las nubes de puntos del jardín (discretización del mapa en 3D), con la perspectiva de las fotografías mostradas en las Figuras 3.2 hasta la 3.4, respectivamente. Estas imágenes representan una captura de pantalla tomada de RTAB-Mapviz, el cual en realidad muestra una nube de puntos tridimensional, que se puede manipular para observar sus regiones desde diferentes perspectivas e incluso exportar como un archivo .obj. Se puede observar que estas imágenes obtenidas no son continuas, sino más bien son imágenes discretas con límites identificables (conjunto finito de puntos).

En SLAM, estas nubes de puntos representan todos los potenciales obstáculos sensados por el robot, los cuales se almacenan en el algoritmo, para luego optimizarlos. De este modo aquellas nubes de puntos con

características similares pero desfasados una pequeña distancia, se agrupan, recreando un mapa en 3D con detalles menos difusos o ambiguos. Este mapa en 3D puede ser luego usado para el modo de *Localization* del robot en RTAB-Mapviz para su posterior navegación autónoma y evasión de obstáculos (cuyos algoritmos se encuentran dentro del paquete de RTAB-Map).

Figura 3.5. Mapa en 3D y nube de puntos respectivas para la Figura 3.2.

Fuente: Elaboración propia.

Figura 3.6. Nube de puntos equivalentes a la Figura 3.3.

Fuente: Elaboración propia.

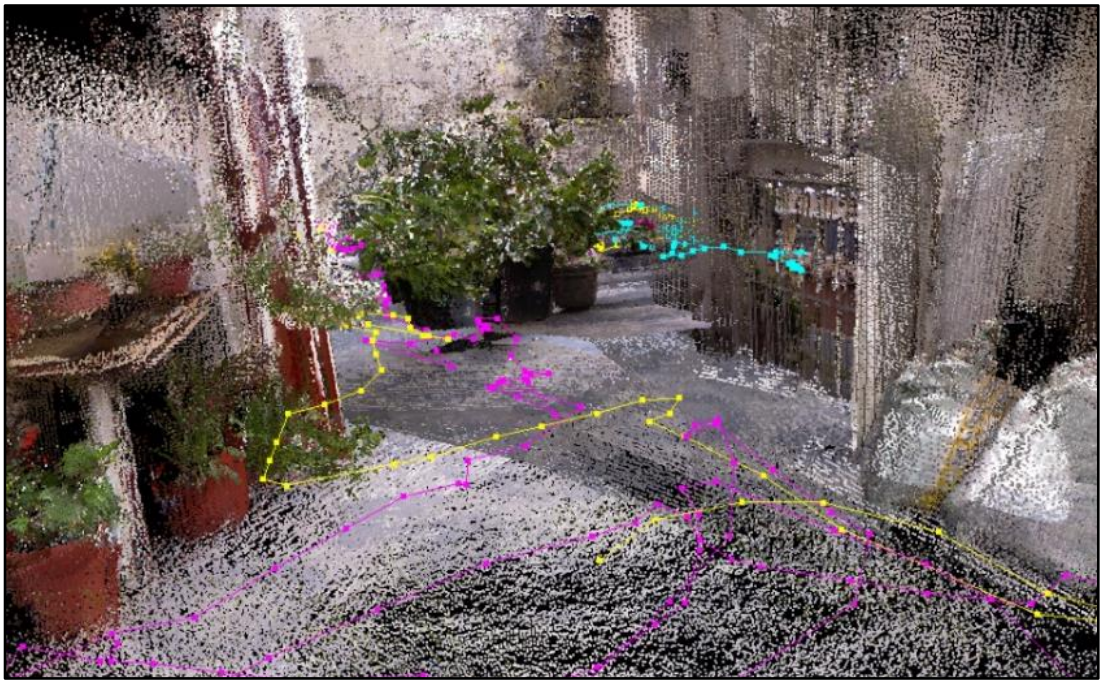


Figura 3.7. Nube de puntos equivalentes a la Figura 3.4.

Fuente: Elaboración propia.

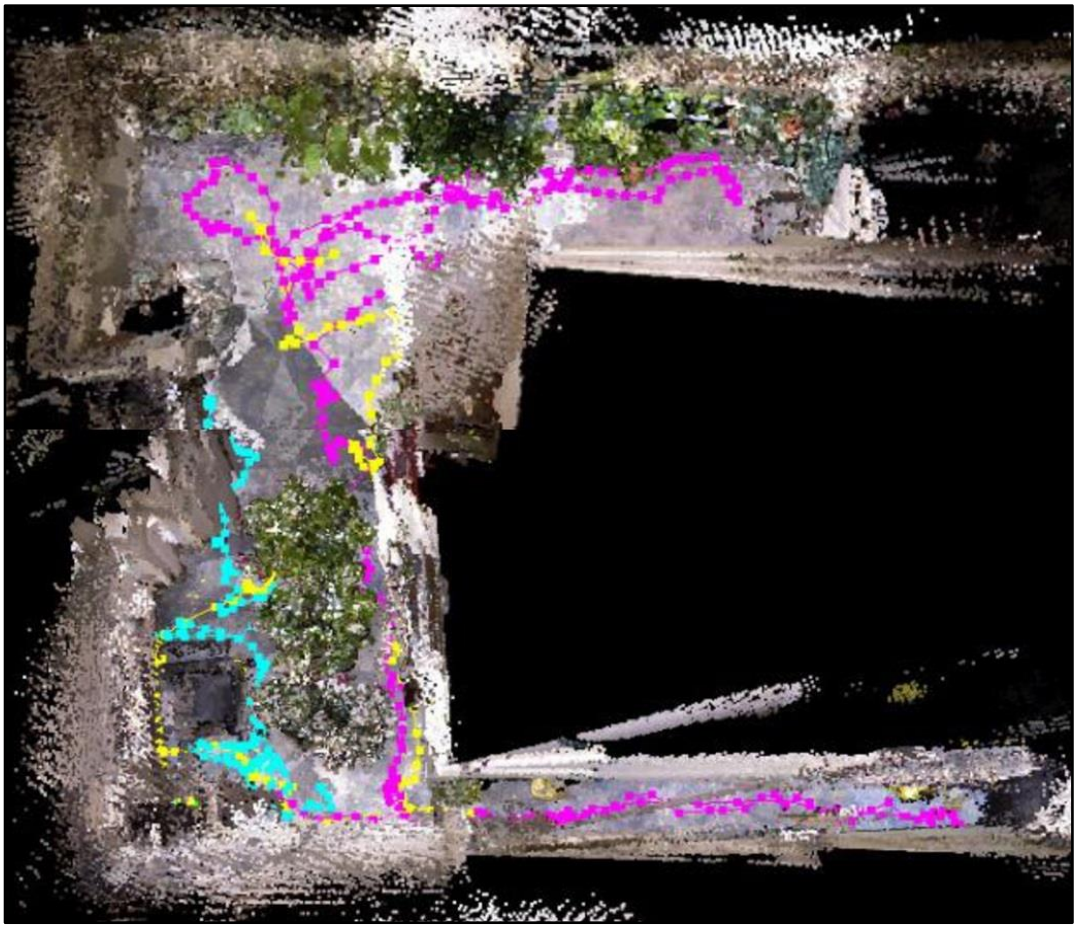


Figura 3.8. Vista superior del mapa en 3D en RTAB-Mapviz.

Fuente: Elaboración propia.

En la Figura 3.8 se muestra el mapa en 3D completo, desde una vista superior. En el mismo se aprecian regiones que no tienen relación al mapa dibujado en la Figura 3.1. Las nubes de puntos que están de más son errores al momento de mapear, que se originaron debido a la pérdida de la odometría visual del robot.

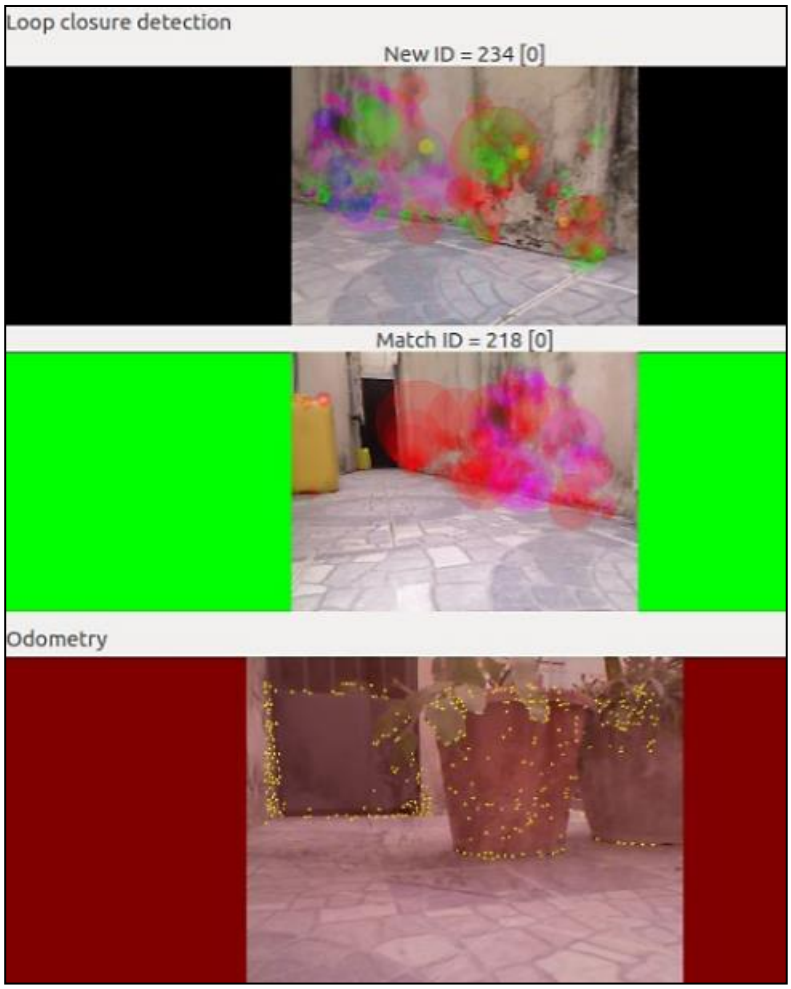


Figura 3.9. Detección de los lazos cerrados y odometría visual.

Fuente: Elaboración propia.

Para realizar el cálculo de la odometría, RTAB-Map realizó un *loop closure detection* (detención de cierre del bucle) que se basa en el siguiente principio: cada nube de puntos adquiría una característica sobresaliente del entorno (círculos de color verde en la Figura 3.9), con el fin de comparar esas nuevas características con las almacenadas previamente en la memoria. Cuando la Kinect volvía a sensar un entorno con las características guardadas, el algoritmo triangulaba la posición respecto a otras características y generaba su odometría visual (esto se muestra cuando el fondo negro de la pantalla de

en medio de la Figura 3.9 se ponía de color verde). En cambio, cuando el algoritmo no era capaz de reconocer las características en el orden respectivo, o cuando al superponerlas no cuadraban con un error mínimo, la cámara no era capaz de determinar su localización y mostraba el fondo de pantalla de color rojo, según la parte inferior de la Figura 3.9. Para evitar este error (que imposibilitaba la continuación del mapeo), fue necesario regresar al robot a alguna posición donde se había previamente calculado su odometría de la siguiendo la trayectoria de la misma forma en cómo había avanzado hasta el lugar en el que se perdió o, como segunda opción, habría que resetear su odometría (opción en RTAB-Mapviz), luego del cual el mapeo empezaba desde cero, borrándose el mapa, formándose otra nube de puntos por lo que el color de la trayectoria cambiaba (véase la Figura 3.11). Sin embargo, debido a que los *pointclouds* con sus características se almacenaban en la memoria de corto plazo de ROS, el mapa perdido se actualizaba cada vez que se detectaba un cierre de lazo (*Loop Closure Detection*). El *Loop Closure* es la razón por la cual el SLAM de RTAB-Map es considerado un SLAM *online* (instantáneo o en vivo), permitiendo una continua retroalimentación de la odometría visual de la Kinect.

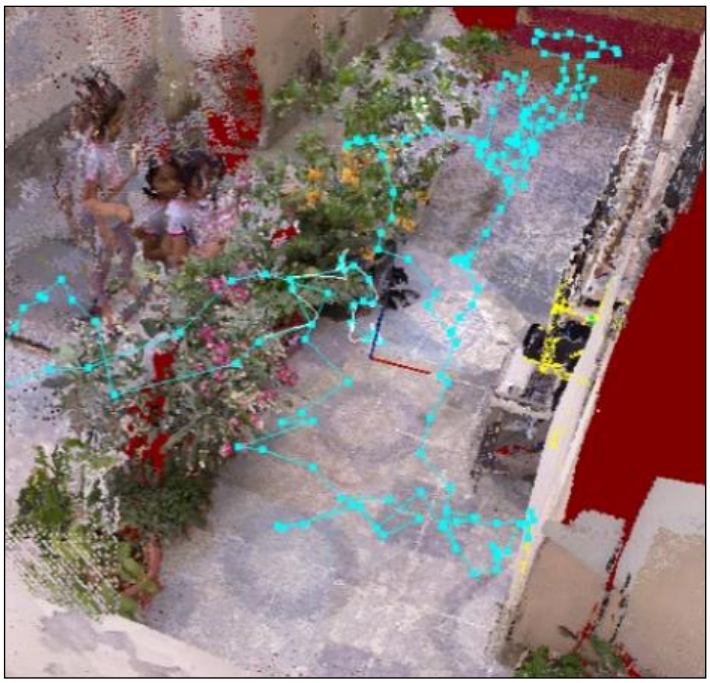


Figura 3.10. Mapeo en 3D de un patio.

Fuente: Elaboración propia.

Uno de los resultados obtenidos mediante el *mapping* del patio se muestra en la Figura 3.10, en donde cabe destacar la aparición de una niña en varias posiciones dentro de la misma imagen, como si hubiese algunas niñas cuando realmente sólo estuvo presente una, lo que permitió constatar que, para entornos dinámicos, el algoritmo de RTAB-MAP no actualiza la nube de puntos, sólo las superpone, por lo que no podría ser utilizado para vehículos autónomos en un lugar transitado.

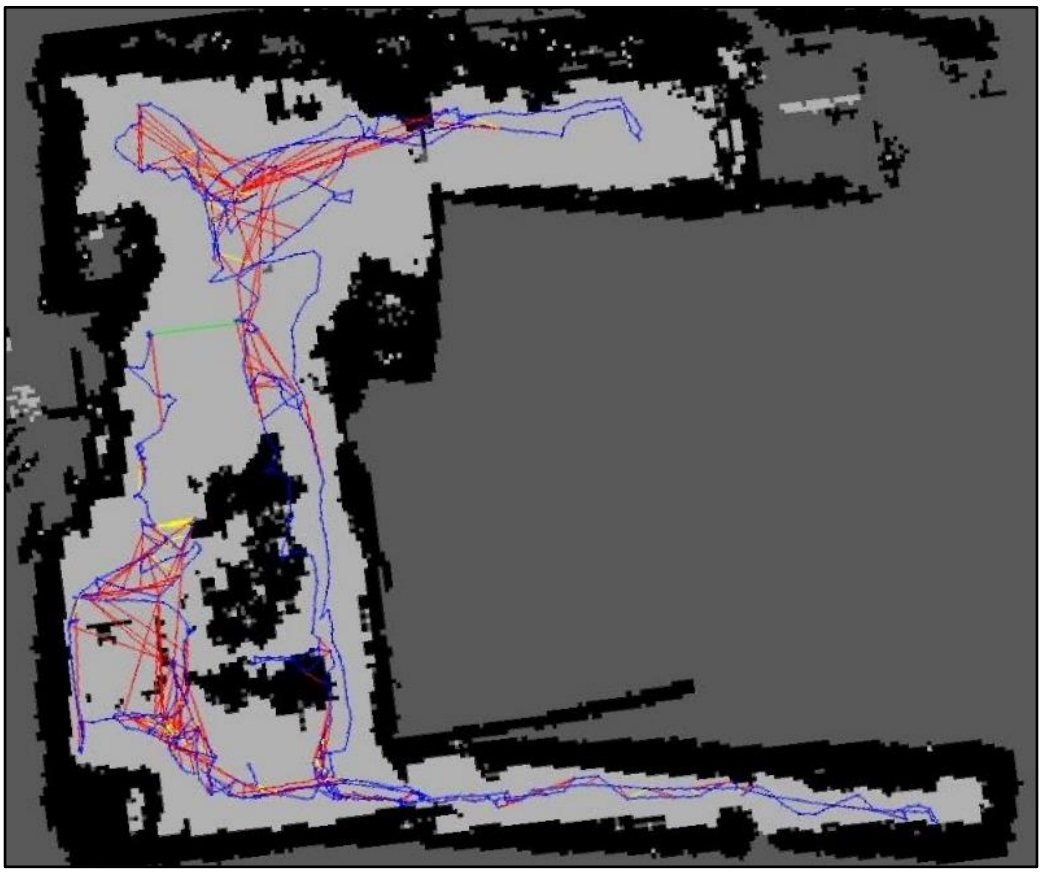


Figura 3.11. Mapa en 2D del jardín con las líneas de la trayectoria del robot.

Fuente: Elaboración propia.

En las Figuras 3.11 y 3.12 se observan los mapas en 2D proyectados por RTAB-Mapviz y Rviz respectivamente. La diferencia reside en los detalles mostrados. Mientras que el primero muestra todas las trayectorias seguidas por el robot mientras se realizó SLAM, en el segundo se aprecia una rejilla de dimensiones 1 [m^2]. En la figura 3.11 la trayectoria (localización del robot) se muestra de tres colores, amarillo, azul y rojo, indicando que se perdió la odometría visual dos veces en el proceso del mapeado. Además, se muestra

una línea verde, que es la recta que une a la posición inicial y final del robot durante el mapeo (el desplazamiento total). También se observa a la trayectoria de color azul superpuesta sobre el obstáculo negro; esto se justifica por el hecho de que, al haber macetas de plantas puestas como obstáculos, las ramas y las hojas se proyectaron sobre el suelo, resultando ser un pseudo obstáculo.

Figura 3.12. Proyección del mapa 3D en la rejilla de Rviz.

Fuente: Elaboración propia.

En ambas figuras se observan tres regiones con colores diferentes: el color gris claro indica el área del suelo donde el robot puede circular sin chocarse; el negro muestra a la nube de puntos en 3D proyectada sobre el suelo como obstáculos, los que finalmente dan el contorno al mapa si es un entorno cerrado, como el caso del jardín escaneado; el color de fondo, gris o plomo oscuro es la región que no ha podido ser sensada por la Kinect, por lo que no

existen *poinclouds* de la misma. De esta manera, se muestra un mapa en 2D bien definido, similar al de la Figura 3.1. En cambio, en la Figura 3.12 se muestra una línea plomo oscura, la cual fue generada para determinar su longitud, y conocer la distancia entre sus dos puntos extremos. Esta recta es el equivalente de la cota de 8270 [mm] del mapa de la Figura 3.1 y a su vez mide 8397 [mm] según el recuadro de la Figura 3.12. De manera similar, se obtuvieron seis mediciones del mapa en Rviz, las cuales se tabularon en la Tabla 3.1, para su comparación con sus respectivas cotas mostradas en la Figura 3.1. El error porcentual aproximado de la comparación de las mediciones entre SLAM y la realidad, tabulado en la tabla 3.1, se calculó mediante la Ec.1, mientras que su promedio y la desviación estándar, se calcularon mediante las funciones de Microsoft Excel PROMEDIO y DESVEST.P respectivamente.

$$Error(\%) = \frac{\left|Valor_{teórico} - Valor_{experimental}\right|}{Valor_{teórico}} * 100\% \qquad \textbf{(Ec.1)}$$

Tabla 3.1. Comparación de cotas del mapa original y del mapeo por SLAM.

Mediciones [m]		Error porcentual [%]
SLAM	Real	
7.23	7.40	2.30
1.13	1.25	9.60
4.44	4.27	3.98
6.12	6.19	1.13
9.05	9.30	2.69
8.39	8.27	1.45
Promedio [%]		3.53
Desviación [%]		2.87

Fuente: Elaboración propia.

3.2 Análisis de costos.

En la Tabla 3.2 se muestra los componentes para la elaboración de un robot completamente autónomo que use la técnica SLAM descrita. Debido a que los programas que se instalaron, así como el sistema operativo, fueron gratuitos,

no se incluyen en el análisis. Los costos de los equipos que muestran un asterisco (*) fueron obtenidos de la página oficial de Amazon, e incluyen los gastos de traslado. Nótese que con letra roja se tienen componentes que no son necesarios para un robot que opere en un ambiente controlado (como el Laboratorio de Mecatrónica en FIMCP-ESPOL o el jardín que se mapeó como ensayo), sin embargo, facilitan su control, la visualización de datos y compilación de algoritmos y mejoran las capacidades y limitaciones del robot. Cabe recalcar que la Facultad facilitó el uso de una Kinect 360 y el Lego Mindstorms EV3. Además, se debe asumir que se dispone de una computadora de generación I3 con 4 [GB] de RAM, como mínimo, para correr ROS Master ejecutando el algoritmo de RTAB-Map, y se disponga un modem *wireless* para la comunicación (el recomendado es el Edimax EW-7811Un).

Tabla 3.2. Costos de los componentes asociados al proyecto.

Nombre	Cantidad	Costo unitario [$]	Costo parcial [$]
Lego Mindstorms EV3*	1	350	350
Baterías recargables 1.5 [V] AA	17	5	85
Cargador de baterías	1	30	30
Cámara Kinect 360*	1	150	150
Wifi Dongle*	1	20	20
Micro SD 32 [GB] Clase 10	1	25	25
Fusible 2 [A]	2	10	20
Raspberry Pi 3*	1	110	110
Componentes electrónicos varios	1	20	20
Linterna recargable 3000 [lm]	1	20	20
Monitor LCD	1	60	60
Joystick PC	1	10	10
		Costo total [$]	810 (900)

Fuente: Elaboración propia.

CAPÍTULO 4

4. DISCUSIÓN Y CONCLUSIONES

4.1 Discusión.

Debido al gran potencial que tiene ROS en la robótica a nivel internacional, tanto en el sector educativo-investigativo como en el empresarial-industrial, el dominar este meta sistema operativo se ha convertido en una necesidad para muchas personas, específicamente aquellas que se desarrollan en carreras modernas como la Ingeniería en Mecatrónica, en donde se encuentran muchas aplicaciones de robótica.

La gran fortaleza de este trabajo radica en la cimentación de una base académica que promueva la apertura hacia nuevos campos, usos y conceptos que fueron explorados y documentados durante la implementación del método SLAM. Esto permitirá trabajar en nuevas aplicaciones que empleen algoritmos, programas y dispositivos ya analizados, desarrollados y registrados en este documento, debido a que la razón por la cual se vuelve una necesidad implementar un SLAM es su potencial uso en una gama de aplicaciones tales como actividades de búsqueda en entornos de desastres, exploración de lugares desconocidos o peligrosos, vigilancia de zonas específicas y movilización de personas discapacitadas u objetos a través de un entorno, entre otras tantas aplicaciones posibles.

Las grandes debilidades que se presentaron durante el desarrollo de este trabajo se basaron principalmente en la necesidad de aprender a utilizar ROS y entender todo lo relacionado a su entorno de trabajo. En la literatura hay suficiente información acerca de los programas, algoritmos y dispositivos empleados en SLAM, pero, a diferencia de lo que se esperaba, implementar dichos algoritmos y entablar una conexión entre los diferentes dispositivos empleados y los nodos de control de ROS, fue una tarea más complicada de lo que se había previsto al inicio del proyecto.

Otra problemática se basó en el hecho de que, a pesar de que se realizó una investigación exhaustiva antes de definir la metodología del proyecto, al empezar cada paso se cometían muchos errores o, simplemente, no se llegaba a una respuesta adecuada. Debido a la inexistencia de un tutorial que explique detalladamente como realizar cada etapa de la compilación de los algoritmos, para demostrar que un paso no estaba en lo correcto, hubo que intentarlo algunas semanas sin resultado exitoso alguno.

Específicamente, primero se utilizó un Lego Mindstorms NXT 2.0 (versión predecesora del EV3) como robot móvil, puesto que los algoritmos de control de este ya habían sido implementados en ROS con anterioridad en otros trabajos. El error consistió en que, debido al auge y actualidad de los Legos EV3, la programación compilada en los NXT ya estaba caducada para las nuevas distribuciones de ROS y de Ubuntu, y sólo se ha estado realizando investigación con los EV3. No se pudo constatar esto sino hasta el momento en que la instalación de los programas de control del Lego NXT no compilaban y emitían mensajes de error en su instalación.

Cuando se decidió utilizar el Lego EV3, hubo que crackearle el sistema operativo (es decir, bootear con otro S.O.) por medio de una memoria externa MicroSD con un *firmware* llamado EV3dev, el mismo que permitía controlar al robot por medio de instrucciones escritas en Python desde la computadora. Además, existía un video-tutorial que explicaba (sin detalles) como instalar ROS en el EV3dev. Al igual que en el caso anterior, se utilizó bastante del tiempo del proyecto para llegar a la conclusión de que no se iba a poder controlar al robot con el EV3dev debido a errores en la compilación de ROS en el *brick* del EV3.

Finalmente, se tuvo una tercera opción que fue optar por otro *firmware* llamado H4R (Hacks for ROS) Yocto-Linux. Fue la tercera opción debido a que no se detallaba casi nada en el modo de escribir los archivos de booteo en la microSD. Se tardó en entender el procedimiento de quemar esos archivos en la microSD, pero al final se pudo lograr. Además, se demostró que los nodos de ROS pueden correr en el EV3, de modo que pueda existir

una comunicación entre el computador (con ROS Master) y el *brick* (con ROS Slave). Sin embargo, debido a que la primera parte del proyecto de SLAM consiste en el mapeo de una región desconocida, se vuelve necesario tele-operar al EV3. Esto se pudo lograr por medio de la implementación de Docker *Containers*, que son un conjunto de paquetes de software que emulan una especie de máquina virtual que no demanda altos recursos y contiene todo lo necesario para ejecutarse en el terminal de Ubuntu.

La ventaja de usar contenedores Docker consistió en la facilidad de implementar códigos complejos y de evitar que los mismos varíen entre un programador y otro. Esto se verificó al configurarse los contenedores de Docker para correr el algoritmo de tele-operación dentro del *brick* del EV3. En cambio, la desventaja respecto a usarlo en conjunto con ROS radica en que Docker trabaja en un entorno cerrado, tal si fuese una computadora externa, lo que complica la visualización de sus procesos intermedios internos, sólo mostrando los *outputs* del paquete que se esté ejecutando en Docker. Esto imposibilita la detección de los nodos de ROS del EV3 que se corren en el contenedor, para su posterior enlace con los nodos de SLAM de RTAB-Map en ROS Master en la PC. En conclusión, al utilizar Docker, a pesar de que se pudo realizar la tele-operación del EV3 para el mapeo del jardín durante el ensayo, no se pudo utilizar el cálculo de la odometría de los motores del EV3 para la localización del robot al momento de mapear. De manera que la odometría fue calculada visualmente mediante RTAB-Map. Si no existe una comunicación efectiva entre ROS Master en la PC y Ros Slave del Lego EV3, entonces no se podrá realizar la segunda parte de SLAM, que es la navegación autónoma y la evasión de obstáculos cuando se le asigne un *goal* (meta o trayectoria a trazar) al robot.

Existe una última opción para alcanzar la segunda etapa del proyecto SLAM. Consiste en ejecutar el proceso que se realiza dentro de los contenedores Docker manualmente, que es corriendo los archivos *launch* de los motores a través de una comunicación SSH con el EV3, por medio de un paquete de ROS Control que se descarga en la misma página de H4R Yocto. El dilema de este método es la falta de un tutorial explícito donde se indiquen los pasos

correctos a seguir y el orden en que se deben entablar. Se trabajó cerca de un mes en este método, pero no se logró un resultado favorable, únicamente el de compilar el paquete ROS Control del EV3. Por esta razón, este Proyecto de Localización y Mapeo Simultáneo debió ser dividido en dos partes; la parte de navegación autónoma y validación del algoritmo de evasión de obstáculos deberá ser continuado como otro Proyecto de Materia Integradora.

Debido a la imposibilidad de visualizar los nodos de control del robot en ROS Master, no se pudieron configurar los tópicos de los nodos para que publiquen y reciban mensajes. Necesariamente, el nodo de control de la Kinect debe publicar los *pointclouds* en un tópico para que el nodo de odometría de los servomotores los reciba. A su vez, el nodo de odometría debe publicar su posición para que una transformada de coordenadas cartesianas actualice correctamente la ubicación de la Kinect durante el mapeo. Este es el principio de la evasión de obstáculos, puesto que el robot sensa las zonas de posible impacto e indica a los motores a que distancia se encuentra de ellas. Además, usando la odometría de los motores, se evita el uso de una odometría visual de la cámara, y se evitaría el problema de la pantalla roja al perder la Kinect su ubicación. Es necesario, una vez controlados los motores con ROS Master, en la siguiente etapa del proyecto, cumplir las condiciones previas para una correcta navegación espacial del robot en su entorno, las cuales son: configurar matrices de rotación de las transformadas de coordenadas del sensor hacia el robot, asegurar la correcta comunicación del sensor y ROS, y calcular correctamente la odometría de los servomotores (sería necesario calibrar la odometría mediante prueba y error).

Otra variable que se debe considerar es la independencia de la conexión por cable con la PC principal. Una alternativa para la solución es utilizar una tarjeta Raspberry Pi 3 como computadora de lectura de los datos sensados por la Kinect 360 y su posterior envío a través de un tópico en ROS hacia la PC. Usar a la Raspberry es considerada la mejor opción para la autonomía total del robot, puesto que sirve únicamente como puente entre la Kinect y la computadora principal con ROS Máster. Su baja memoria RAM (1 [GB] de RAM) para el procesamiento de datos usando RTAB-Map se recompensa con

su pequeño tamaño y peso. Sin embargo, el problema principal consistió en la dificultad de una conexión SSH entre la Raspberry Pi 3 (sobre la cual se instaló Ubuntu Mate 16.04) y la PC con Ubuntu Desktop. Este no es un problema de fondo, puesto que ya existen trabajos y tutoriales sobre el uso de ROS con una tarjeta Raspberry. De ser necesario, se utilizaría otro S.O. en caso de que Ubuntu Mate no sea el adecuado para la conexión SSH de la Raspberry Pi 3 con la PC.

En el análisis de costos se consideraron todos los elementos para dotar de una autonomía completa al robot, específicamente la autonomía energética a base de pilas recargables AA. Los tres componentes, Kinect 360, Lego Mindstorms EV3 y Raspberry Pi 3 usan 12, 9 y 5 voltios respectivamente (8, 6 y 3 baterías AA). Hay que tener cierto cuidado con la instalación electrónica de las baterías para la Kinect y la Raspberry para evitar cortocircuitos y daños de equipos costosos. El cable de alimentación de la Kinect debe ser cortado para usar las 8 baterías directamente conectadas con un fusible de resguardo. Del mismo modo, al cable de alimentación de la Raspberry se le debe conectar un adaptador para usar las baterías en serie y otro fusible de resguardo. Los componentes opcionales de la Tabla 3.2 no se consideran puesto que son independientes de las conexiones electrónicas mencionadas. Lo recomendable es asesorarse de una persona con conocimientos técnicos de circuitos eléctricos.

La importancia de haber utilizado un Lego Mindstorms EV3 como robot móvil y una Kinect 360 como sensor para SLAM es que no existen trabajos previos de los mismos, sea en el idioma español o inglés. Esto convierte al proyecto realizado como pionero en su aplicación. Aunque se utilizó la versión Track3r del Lego Mindstorms EV3 (cuyos modelos están patentados), se modificó en gran medida su diseño inicial para evitar el desbalance del robot al navegar, como consecuencia del desfase de su centro de gravedad hacia la parte superior del robot. El centro de gravedad del robot se ubica a 25 [cm] sobre el suelo, debido a que, en la etapa del mapeo, la cámara Kinect no podía registrar los *pointclouds* ubicados en el suelo. Esto fue consecuencia de instalar la Kinect encima del *brick* del EV3 con la versión Track3r del robot.

Como la Kinect necesita por lo menos un espacio de 40 [cm] como mínima distancia para sensar los *pointclouds*, se vio la necesidad de aumentar la altura del robot usando otros elementos de Lego (véase el Apéndice D), por lo que su centro de gravedad cambió en altura. Se podría aterrizar el centro de gravedad hasta cerca de los 15 [cm] si se desmontara la carcasa de la Kinect para dejar únicamente ensamblados los sensores RGB-D en el *brick* del EV3. El hecho de trabajar con una versión modificada del robot Lego permitiría el uso de este sin caer en la consideración de los derechos de autor al momento de publicar la investigación desarrollada al utilizarlo.

Que el Laboratorio de Mecatrónica de FIMCP disponga de un robot móvil autónomo que esté en capacidad de realizar un SLAM para ejecutar cualquier otra tarea asignada, sería un gran avance en la investigación académica de ESPOL. Este robot podría ser utilizado en prácticas de laboratorio, en ferias de exposición, en concursos de robots móviles, entre otros. Además, serviría de precedente para el diseño de robots con mayores capacidades de carga y menos limitaciones de movimiento o de adquisición de datos en sus sensores. Inspiraría futuros proyectos en la carrera de Ingeniería en Mecatrónica, específicamente aquellos de diseño de robots móviles con otras funciones o aplicaciones de SLAM (robots de rescate, de reemplazo o de seguridad), para beneficio de la ciudadanía en general. Utilizando ROS, se contribuiría a la comunidad científica con la experiencia ganada en estos proyectos, respaldando a la labor del *freeware* como medio de investigación y desarrollo a nivel mundial.

4.2 Conclusiones.

- Se realizó exitosamente la conexión entre el dispositivo Kinect 360 y la computadora portátil con Ubuntu por medio del entorno de trabajo ROS.
- Se controló el desplazamiento del robot móvil por medio de algoritmos de tele-operación compilados en ROS y Docker *Containers*.
- Se comprendió el potencial de ROS en aplicaciones robóticas a nivel académico-investigativo, así como la facilidad de la ejecución de sus paquetes una vez implementado los códigos de sus algoritmos.

- Se generó un mapa en 3D de un entorno conocido haciendo uso de los sensores RGB-D de una cámara Kinect 360 y del paquete RTAB-Map.
- Se realizó la proyección de un mapa generado en 3D a un plano paralelo a la superficie del suelo y se observaron los detalles, tales como los obstáculos, el camino libre, las paredes, entre otros.
- Se resolvió el problema de SLAM, obteniendo un mapa en 2D de un entorno escaneado mediante el dispositivo Kinect 360, ensamblado en un Lego Mindstorms EV3 y controlado por ROS.
- Se determinó la trayectoria recorrida por el robot durante el proceso del mapeo, ubicando la odometría visual de la Kinect dentro del espacio.
- Se pudieron contrastar las fotografías del jardín de la casa respecto a las imágenes de *cloudpoints* generadas en RTAB-Mapviz, percatándose de sus similitudes.
- Se confirmó la validez de RTAB-Map como algoritmo de SLAM *online*, obteniendo un error promedio del 3.53% y una desviación del 2.87% comparados con su mapa real respectivo.

4.3 Recomendaciones.

- Con el fin de aumentar la independencia del robot móvil en el momento de la implementación del SLAM, se recomienda realizar la conexión de todos los dispositivos con la computadora portátil de forma inalámbrica, debido a que la longitud de los cables de conexión es un limitante en la distancia que puede recorrer el robot móvil.
- Se requiere mantener la cámara Kinect a una determinada altura mínima (30 [cm]), lo cual permite mapear el suelo de forma eficiente sin generar puntos ciegos.
- Para el desarrollo de futuras aplicaciones, se requiere el uso de sensores que permitan obtener datos con menos incertidumbre de los que se presentan con los sensores actuales (ejemplo, los escáneres láser superan en resolución y costos a los sensores RGB-D).
- Antes de realizar un mapeo con SLAM, se recomienda realizar una calibración de los sensores RGB-D, por medio de un tablero de ajedrez modificado de 9x7 cuadrados con 108 [mm] de lados.

- Se recomienda utilizar los sensores ultrasónicos del Lego Mindstorms EV3 y crear un programa para que se ejecute en ROS, con el objetivo de realizar una comparación entre los datos obtenidos por los *cloudpoints* y las distancias sensadas por el sensor ultrasónico, en un plano definido paralelo al suelo.
- Es imprescindible tener conocimientos previos, o haber tomado un curso, de robótica aplicando ROS. Se necesita un nivel intermedio-avanzado de dominio con este sistema para completar la segunda parte de este proyecto, con el fin de completar las condiciones para una navegación correcta y establecer una comunicación efectiva entre los nodos de ROS Master (controlando a RTAB-Map) y ROS Slave (controlando al Lego EV3) para ordenar al robot el movimiento sin necesidad de tele-operarlo, de modo que se pueda ejecutar una navegación autónoma entre dos puntos, evadiendo obstáculos.

BIBLIOGRAFÍA

Arroyo Sánchez, G. (2013). *Localización y Mapeo Simultáneos para robot de búsqueda en entornos de desastre* (Tesis inédita de maestría). Universidad Nacional Autónoma, Ciudad de México, MX.

Beltrán Guerrero, D. & Basañez Villaluenga, L., (2014). *Microsoft Kinect* (General Technical Report IOC). Cataluña: Instituto de Organización y Control de Sistemas Industriales.

Bouchier, P. & Henriksen, J. (Directores). (2015). ROS Tutorial part 1 [archivo de video]. Disponible en https://www.youtube.com/watch?v=nKIOb6lyrlI

Chadrockey (2017). Ros-perception/depthimage_to_laserscan. GitHub. Recuperado de https://github.com/ros-perception/depthimage_to_laserscan

Creative Commons. (2014). Ros On Ev3. EV3DEV.ORG. Recuperado de http://www.ev3dev.org/projects/2014/11/17/ROS-on-EV3/

Creative Commons. (2017). Getting Started with ev3dev. EV3DEV.ORG. Recuperado de http://www.ev3dev.org/docs/getting-started/

Creative Commons. (2017). ROS Robot with EV3 and Docker. Autodesk, Inc. Recuperado de http://www.instructables.com/id/ROS-Robot-With-Lego-EV3-and-Docker/

Doxygen. (2016). H4R_EV3 ROS Nodes. Aelen, Alemania. Hacks for ROS. Recuperado de http://hacks4ros.github.io/h4r_ev3_ctrl/

Elfes, A. (1987). Sonar-Based Real-World Mapping and Navigation. *IEEE Journal of Robotics and Automation*, vol. 3(3), pg. 249-265.

Hernández, B. (Director). (2018) Tutorial de Implementación de SLAM *(Simultaneous Localization and Mapping)* con un Robot Móvil [archivo de video]. Disponible en https://www.facebook.com/pg/Club-Mecatr%C3%B3nica-ESPOL 58407001803082/ videos/?ref=page_internal

Introlab. (2014-2017). RTAB-Map. Real-Time Appearance-Based Mapping. Sherbrooke, Canadá. Introlab, Github. Recuperado de http://introlab.github.io/rtabmap/

Jaimez, M., Kerl, C., Gonzalez-Jimenez, J. & Cremers, D. (2017). Fast Odometry and Scene Flow from RGB-D Cameras based on Geometric Clustering. *IEEE Int. Conf. on Robotics and Automation (ICRA),* doi: 10.1109/ICRA.2017.7989459

Labbé, M. & Michaud, F. (2013) Appearance-Based Loop Closure Detection for Online Large-Scale and Long-Term Operation. *IEEE Transactions on Robotics*, vol. 29 (3), pg. 734-745. doi: 10.1109/TRO.2013.2242375

López. Torres, P. (2016). *Análisis de algoritmos para localización y mapeado simultáneo de objetos* (Tesis inédita de maestría). Escuela Técnica Superior de Ingeniería, Sevilla, ES.

Lugo Sánchez, O. (2015). *Localización y mapeo simultáneo en interiores mediante sensor RGBD* (Tesis inédita de maestría). Instituto Politécnico Nacional, Ciudad de México, MX.

Mayans Roca, V. (2012). *Desarrollo multiplataforma de aplicaciones de control y comunicación para robots móviles* (Tesis inédita de maestría). Universitat Politécnica, Valencia, ES.

Montemerlo, M. (2003). *FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem with Unknown Data Association.* (Tesis inédita de doctorado). Carnegie Mellon University, Pensilvania, US.

Mur-Artal, R. & Montiel, J. (2015). ORB-SLAM: A Versatile and Accurate Monocular SLAM System. *IEEE Transactions on Robotics*, vol. 31(5), pg. 1147-1163. doi: 10.1109/TRO.2015.2463671

Open Source Robotics Foundation (2015). Navigating the ROS Filesystem. ROS Wiki. Recuperado de http://wiki.ros.org/ROS/Tutorials/NavigatingTheFilesystem

Open Source Robotics Foundation. (2016). Navigation/Tutorials. ROS Wiki. Recuperado de http://wiki.ros.org/navigation/Tutorials

Open Source Robotics Foundation (2017). Installing and Configuring Your ROS Environment. ROS Wiki. Recuperado de http://wiki.ros.org/ROS/Tutorials/Installingand ConfiguringROSEnvironment

Open Source Robotics Foundation. (2017). Navigation. ROS Wiki. Recuperado de http://wiki.ros.org/navigation

Open Source Robotics Foundation. (2017). RGB-D Hand-Held Mapping. ROS Wiki. Recuperado de http://wiki.ros.org/rtabmap_ros/Tutorials/HandHeldMapping

Open Source Robotics Foundation. (2017). Rtabmap-ros. ROS Wiki. Recuperado de http://wiki.ros.org/rtabmap_ros

Open Source Robotics Foundation. (2017). Ubuntu install of ROS Kinetic. ROS Wiki. Recuperado de http://wiki.ros.org/kinetic/Installation/Ubuntu

Open Source Robotics Foundation. (2018). ROS Tutorials. ROS Wiki. Recuperado de http://wiki.ros.org/ROS/Tutorials

OpenKinect Project (2016). Getting Started. MediaWiki. Recuperado de https://openkinect.org/wiki/Getting_Started

OpenKinect Project (2017). Libfreenect. GitHub. Recuperado de https://github.com/OpenKinect/libfreenect/issues

Quigley, M., Gerkey, B., Conley, K. Faust, J., Tully, F., Leibs, J., Berger. E., Wheeler, R. & Ng, A. (2010). ROS: an open-source Robot Operating System. Recuperado de http://www.willowgarage.com/sites/default/files/icraoss09-ROS.pdf

Radiation Laboratories. (2016). H4R_EV3 ROS Nodes Documentation. GitHub Pages. Recuperado de http://hacks4ros.github.io/h4r_ev3_ctrl/

Radiation Laboratories. (2017). ROS Node for managing EV3 with ROS Control. GitHub Pages. Recuperado de https://github.com/Hacks4ROS/h4r_ev3_ctrl

Rapado García, J. (2016). *Diseño e implementación de una interfaz gráfica de usuario para mapeado de entornos y navegación en ROS* (Tesis inédita de maestría). Universitat Politécnica, Valencia, ES.

Riisgaard, S. Blas, M. (2005). *Slam for dummies a tutorial approach to simultaneous localization and mapping.* Recuperado de https://ocw.mit.edu/courses/aeronautics-and-astronautics/16-412j-cognitive-robotics-spring-2005/projects/1aslam_blas_repo.pdf

Rodríguez, A., Rodríguez, J., Luna, K., Rodríguez, P. & Rivera, L. (2011). Proyecto de Sistemas Operativos: Windows vs Linux. Recuperado de https://yaxtux.files.wordpress.com/2011/03/windows-vs-linux1.pdf

The Construct. (Director). (2017). RTAB-Map in ROS 101: ROS Tutorial [archivo de video]. Disponible en https://www.youtube.com/watch?v=gJz-MWn7jhE&t=428s

Viñals Pons, J. (2012). *Localización y generación de mapas del entorno (SLAM) de un robot por medio de una Kinect* (Tesis inédita de maestría). Universitat Politécnica, Valencia, ES.

Xbox Soporte Técnico. (2014). Condiciones de iluminación de la sala para Kinect. Microsoft. Recuperado de https://support.xbox.com/es-EC/xbox-360/kinect/lighting

Zabala, G., Teragni, M., Morán, R., Blanco, S. & Pérez, G. (2014). Localización y Mapeo Simultáneo (SLAM) utilizando un sensor de profundidad por infrarrojo. Recuperado de http://imgbiblio.vaneduc.edu.ar/fulltext/files/TC119406.pdf

APÉNDICES

APÉNDICE A

Datos Técnicos del Equipo

Tabla A. 1. Datos técnicos del equipo utilizado en la implementación de SLAM.

Sensores del dispositivo Kinect.	• Lente con sensor de color y profundidad. • Array de micrófonos para voz. • Tilt motorizado para ajuste del sensor.
Campo de visión del dispositivo Kinect.	• Campo de visión horizontal: 57 grados. • Campo de visión vertical: 43 grados. • Rango de tilt físico: ± 27 grados. • Rango del sensor de profundidad: 0.4m - 3.5m.
Data streams.	• 320x240 profundidad de 16-bit a 30 FPS. • 640x480 color de 32-bit a 30FPS. • 16-bit audio @16 kHz.
Dimensiones del equipo.	• Altura Total: 308 [mm]. • Longitud Total: 220 [mm]. • Ancho Total: 205 [mm].
Intensidad de iluminación de trabajo recomendada.	• Rango de 500 a 700 lux.
Velocidad del robot recomendada en el SLAM.	• Máxima velocidad lineal recomendable de acuerdo con nuestras pruebas: 21 cm/s • Máxima rotación angular del robot sobre el plano de referencia (suelo) de acuerdo con nuestras pruebas: 20 grados por segundo.
Tiempo aproximado de duración de las baterías.	• De 3 a 4 horas en operación continua del robot.
Limitaciones para operar.	• Implementación únicamente en entornos estáticos. • Desplazamiento del robot solo en superficies planas. • Ineficiente en zonas con objetos reflectivos. • Evitar el uso del equipo en zonas lluviosas.
Requerimientos de Software	• Procesador de 64 bits. • Doble núcleo físico de 3,1 GHz. • Controlador USB 2.0 (o superior). • 4 GB de RAM. • Tarjeta gráfica que admita DirectX 11.

Fuente: Microsoft, 2014.

APÉNDICE B

Códigos Utilizados

A continuación, se muestran los códigos ejecutados en el terminal de Ubuntu, para la instalación correcta de ROS Kinetic, así como sus complementos y el algoritmo de RTAB-Map. Cabe recalcar que es imprescindible que el usuario que requiera continuar este proyecto tenga conocimientos de Ubuntu, debido a que muchas veces existen errores en la compilación de los códigos. Sin embargo, en el wiki de Ubuntu y de ROS ofrecen soluciones para toda índole consultada.

Instalación de ROS Kinetic.

```
$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
$ sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-key 421C365BD9FF1F717815A3895523BAEEB01FA116
$ sudo apt-get update
$ sudo apt-get install ros-kinetic-desktop-full
$ sudo rosdep init
$ rosdep update
$ echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc
$ source ~/.bashrc
$ sudo apt-get install python-rosinstall python-rosinstall-generator python-wstool build-essential
```

Configurando el ambiente de ROS.

```
$ printenv | grep ROS
$ source /opt/ros/kinetic/setup.bash
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws/
$ catkin_make
$ source devel/setup.bash
$ echo $ROS_PACKAGE_PATH
```

Navegación en el sistema de archivos de ROS.

```
$ sudo apt-get install ros-kinetic-ros-tutorials
$ rospack find roscpp
$ roscd roscpp
```

$ pwd

$ echo $ROS_PACKAGE_PATH

Instalación de Open Kinect y Libfreenect.

$ git clone https://github.com/OpenKinect/libfreenect

$ cd libfreenect

$ mkdir build

$ cd build

$ cmake -L ..

$ make

$ sudo apt-get install git cmake build-essential libusb-1.0-0-dev

$ sudo apt-get install libfreenect

$ sudo apt-get install git-core cmake libglut3-dev pkg-config build-essential libxmu-dev libxi-dev libusb-1.0-0-dev

$ sudo ldconfig /usr/local/lib64/

$ sudo freenect-glview

$ sudo apt install cmake-curses-gui

$ ccmake ..

$ cd libfreenect/build

$ ccmake ..

$ make

Instalación de RTAB-Map.

$ sudo apt-get install ros-kinetic-slam-gmapping

$ sudo apt-get install ros-kinetic-freenect-launch

$ sudo apt-get install ros-kinetic-rtabmap-ros

$ sudo apt-get install ros-kinetic-depthimage-to-laserscan

$ rosrun gmapping slam_gmapping scan:=base_scan

$ rosrun rtabmap_ros rgbd_odometry --params

$ rosrun rtabmap_ros rtabmap --params | grep Optimize

$ nautilus /home/blass/.ros

Ejecución de los códigos de SLAM para mapeo.

$ roslaunch freenect_launch freenect.launch depth_registration:=true

$ roslaunch rtabmap_ros rgbd_mapping.launch rtabmap_args:="--delete_db_on_start" rviz:=true rtabmapviz:=false

$ roslaunch rtabmap_ros rgbd_mapping.launch rtabmap_args:="--delete_db_on_start"

Si no se puede establecer conexión entre la Kinect y ROS, probar:

$ lsusb | grep Xbox

$ echo -1 | sudo tee -a /sys/module/usbcore/parameters/autosuspend

Para quemar H4R Yocto Linux en micro SD:

$ sudo tar -xzvf boot.tar.gz -C /media/blass/Root

$ sudo tar -xjvf h4r_ev3_alpha_root_fs.tar.bz2 -C /media/blass/Root

Para editar los *hostnames* dentro del Ev3:

$ vi /etc/hosts

APÉNDICE C

Esquema de Grafos

A continuación, se muestran los nodos y los tópicos que se ejecutan en ROS Kinetic mientras se realizó SLAM con RTAB-Map. En la parte superior izquierda de la figura C.1 se muestra los sistemas de coordenadas para los puntos RED, GREEN, BLUE y DEPTH sensados por la Kinect, que por medio de una transformada con matrices de rotación permiten el cálculo de la odometría visual de la Kinect (cuyo nodo se muestra en la parte central). Los nodos de los sensores de la cámara se encuentran en la parte inferior de la figura C.1. Por último, todos los nodos convergen al nodo RTAB-Mapviz (parte superior derecha) que es el encargado de la presentación de SLAM en Rviz.

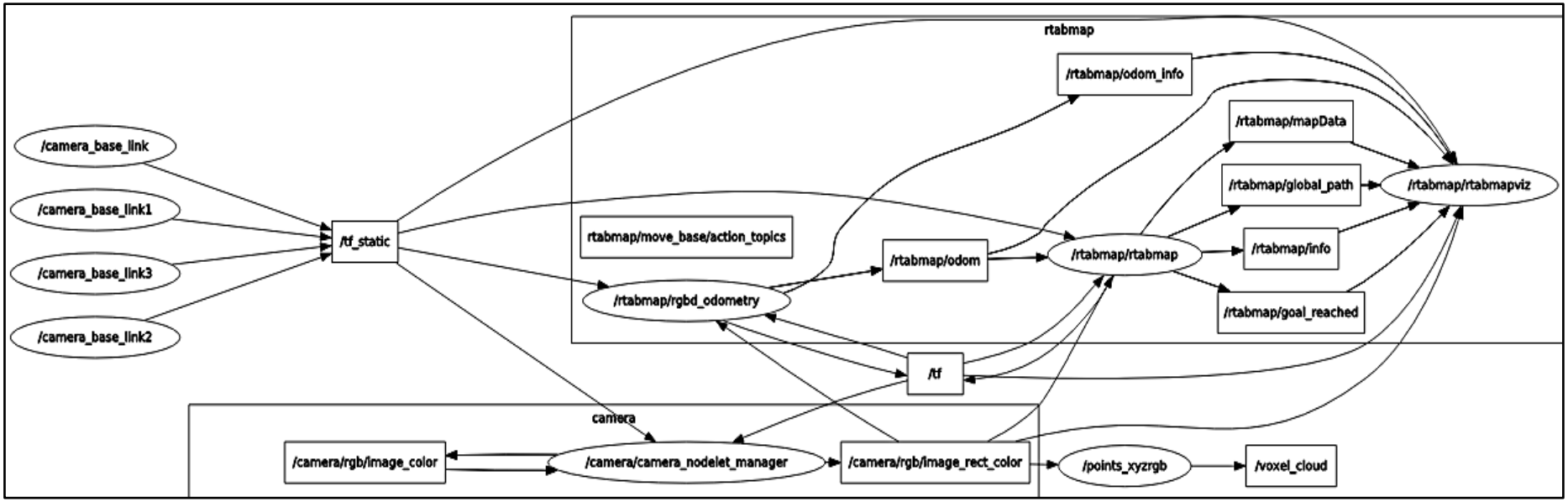


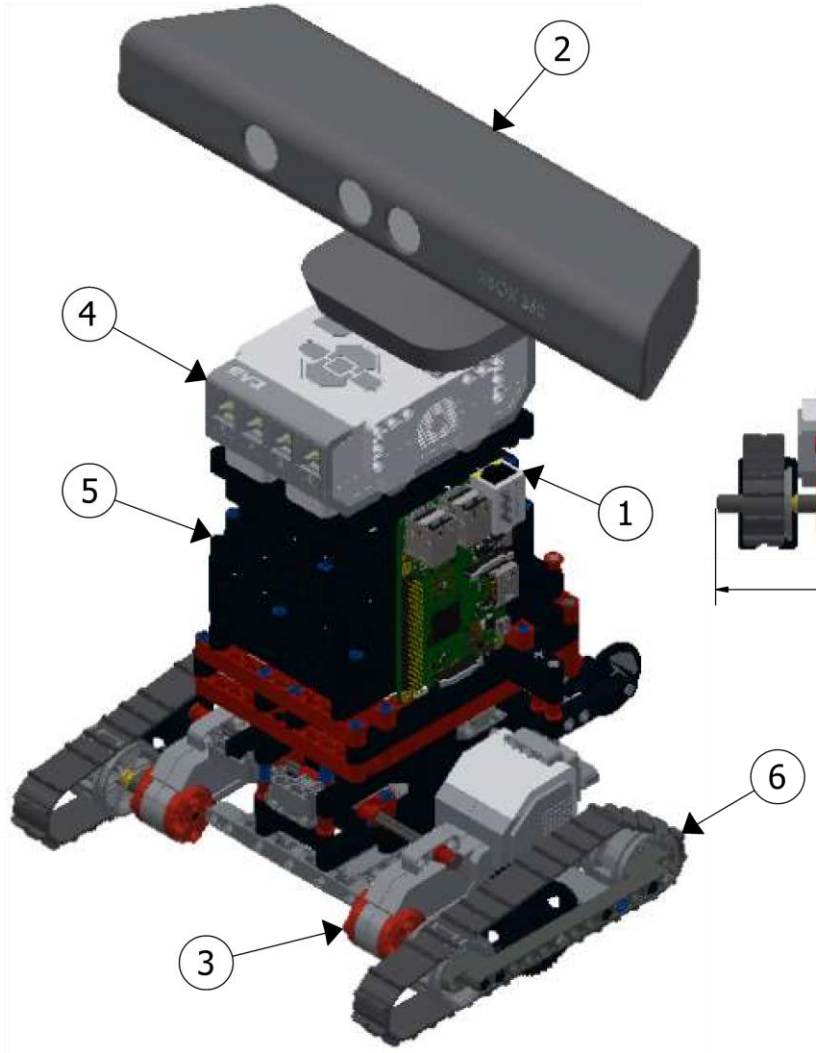
Figura C. 1.Esquema de Grafos con todos los nodos utilizados en la implementación del SLAM.

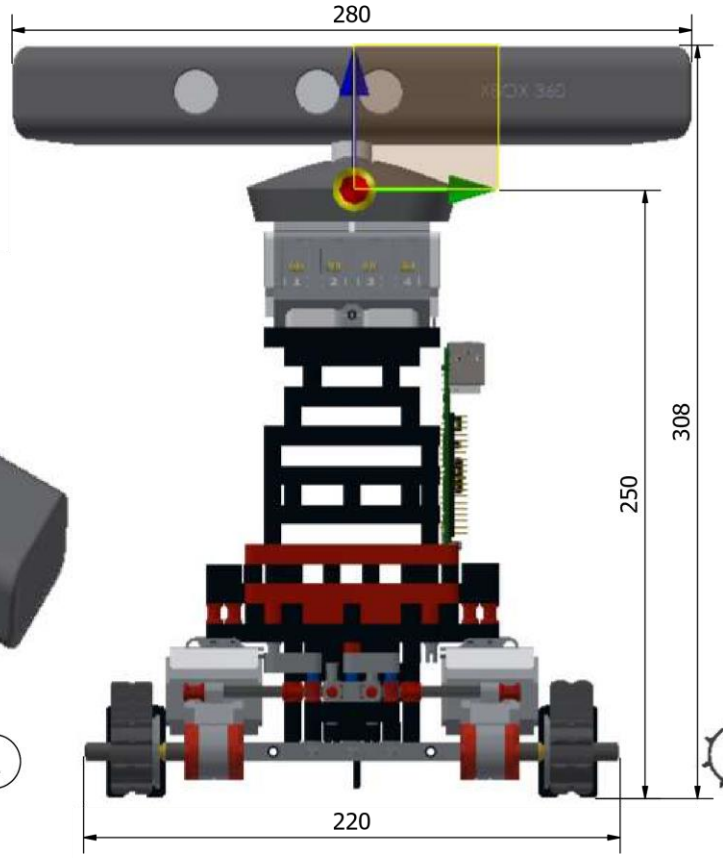
Fuente: Elaboración propia.

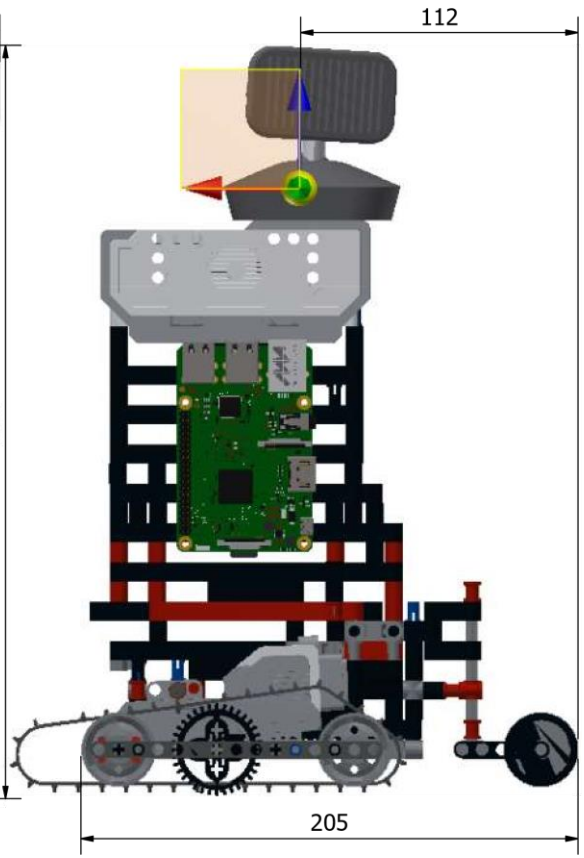
APÉNDICE D

Plano Esquemático del Equipo

PARTS LIST		
ITEM	QTY	PART NUMBER
1	1	Raspberry Pi 3
2	1	Kinect 360
3	2	Servomotor Lego EV3
4	1	Brick Lego EV3
5	1	Estructura Lego
6	2	Ruedas tipo caterpillar







Diseño de	Revisado por	Aprobado por	Fecha	Escala
Bláss Hernández	Ing. Jorge Hurel	Ing. Jorge Marcial	25/02/2018	1:2

ESPOL-FIMCP

Robot autónomo móvil para SLAM

Proyecto Integrador

Dimensiones	Plano
[mm]	N°1