IMPLEMENTACIÓN DE UN ALGORITMO DE MAPEO 2D MEDIANTE SENSOR RPLIDAR

Juan David Marín Quintero

Ingeniería Mecatrónica

Director: Juan Sebastián Botero Valencia

INSTITUTO TECNOLÓGICO METROPOLITANO

Agosto de 2016

RESUMEN

En el siguiente trabajo se hará la primera parte de un algoritmo de mapeo 2D por medio de la adquisición de profundidades de un sensor laser de la marca RoboPeak, el RPLidar para el movimiento de una plataforma móvil que funcionará como transportador del sensor. Este algoritmo se desarrollará por medio de la técnica de SLAM (Simultaneous Localization and Mapping). El fin de este algoritmo es el crear mapas de lugares sin explorar mediante la liberación de una plataforma móvil que se irá desplazando a medida que va tomando escaneos del área, para que, al finalizar su trayectoria, haya obtenido datos suficientes para crear un mapa del lugar. Este trabajo se enfocará en la conexión entre los diferentes elementos de hardware (tarjeta de adquisición de datos Odroid, plataforma móvil de la empresa IRobot, el IRobot Create 2 y el sensor laser de la empresa RoboPeak, el RPLidar) y el envío de datos entre estos, además de la obtención de los datos de entrada y salida para el algoritmo de mapeo, por medio de la obtención de los datos arrojados por el sensor laser RPLidar (calidad, ángulo y distancia), los datos arrojados por los encoder de la plataforma móvil y el movimiento de la plataforma. Se logró capturar los datos que se deseaban los cuales servirán para un trabajo futuro enfocado en el desarrollo del algoritmo de mapeo 2D.

Palabras clave: plataforma móvil, sensor laser, RPLidar, SLAM, Robopeak, IRobot, IRobot Create 2, ARM, Odroid.

RECONOCIMIENTOS

El resultado de este trabajo no hubiera sido posible sin la ayuda de algunas personas que influyeron en mi formación tanto profesional como personal. De esta forma quisiera agradecer a los laboratorios de Mecatrónica y de Sistemas de Control y Robótica y a sus respectivos laboratoristas Juan Esteban Zabala Daza y Mateo Rico Garcia por su colaboración y facilitación de los equipos necesarios para el normal desarrollo del proyecto. Además, un agradecimiento especial a mi asesor Juan Sebastián Botero Valencia por el acompañamiento y guía que me ha brindado durante el tiempo de realización de este proceso.

Le doy también gracias a aquellos docentes, que, durante mi estadía en este claustro me enseñaron no solo lo que debía saber de mi profesión, sino también aquellas enseñanzas para la vida que quedan impregnadas en el ser por siempre. Al profesor Wimar Alberto Moreno Silva por su amor a la enseñanza y la forma que nos entrega el conocimiento, ya que gracias a esto descubrí que cuando se habla del conocimiento no debe haber límites a la hora de brindarlo a los demás. También a la profesora Norma Patricia Guarnizo, por enseñarme indirectamente su forma de ser, ya que este expresa gran carácter y personalidad, una forma de ser atractiva digna de aprender.

Por último y más importante le doy gracias a mi familia por ser mi apoyo en los momentos más complicados de mi proceso de formación, momentos en los que los he necesitado y siempre han estado ahí, en especial de este último año y especialmente a mi padre (Q.E.P.D), el cual fue el que me brindaba la motivación necesaria para seguir adelante con mi formación a pesar de las dificultades que pudieran presentarse. Gracias infinitas a ti papa.

ACRÓNIMOS

En esta sección se listan los acrónimos, siglas, símbolos y abreviaturas propias, no halladas en diccionarios, que son utilizados en la escritura del reporte técnico. Deben incluirse como una lista sin viñetas, por ejemplo:

SLAM Simultaneous Localization and Mapping – Localización y Mapeo Simultaneos

LIDAR Light Detection and Ranging O Laser Imagining Detection and Ranging

DARPA Defense Advanced Research Projects Agency

VSLAM Virtual Simultaneous Localization and Mapping

SBC Single Board Computer

ARM Advanced RISC Machine

RISC Reduced Instruction Set Computer

USB Universal Serial Bus

HDMI High Definition Multimedia Interface

eMMC Multimedia Card

GND Tierra

BRC Baud Rate Change

LED Light Emitting Diode

VGA Video Graphics Array

DVI Digital Visual Interface

IDE Integrated Development Environment

ROS Robotics Open Source

SD Secure Digital Card

LAN Local Área Network

POO Programación Orientada a Objetos

TABLA DE CONTENIDO

1. INTRODUCCIÓN

La reconstrucción del entorno de forma autónoma ha sido uno de los objetivos más importantes de la robótica móvil debido a la libertad que lograría un robot con esta capacidad. De forma que pueda ser liberado en un entorno desconocido y por sí mismo recorrer y guardar los datos de su entorno, creando de esta forma un mapa.

En ese orden de ideas el objetivo del trabajo, es la implementación de un algoritmo de mapeo 2D por medio de la técnica de SLAM en una plataforma móvil, por medio de adquisición de profundidades mediante un sensor RPLidar, el cual es dirigido por el Sr. Juan Sebastián Botero Valencia y elaborado por cuatro estudiantes del pregrado en ingeniería mecatrónica. El autor de este trabajo es uno de ellos. De forma resumida la técnica de SLAM (Simultaneous Localization and Mapping) consiste en una técnica que permite mezclar datos de localización y de mapeo, con el fin de crear un mapeo de un área específica. En consecuencia, el trabajo está separado por dos fases. En una primera instancia el objetivo se trata de la captura de los diferentes sensores de la plataforma y el sensor laser encargados de obtener los datos de localización y mapeo respectivamente, los cuales son necesarios para servir como condiciones de entrada del algoritmo de mapeo 2D, con el fin de ser procesados para que produzcan una respuesta especifica de los diferentes actuadores de la plataforma y de esta forma lograr el movimiento de la misma que permitirá el mapeo dinámico del entorno. En segunda instancia se encuentra el desarrollo del algoritmo como tal.

En los siguientes se ampliará la información del trabajo de esta forma. En el capítulo 2, el marco teórico, se hablará de los conceptos teóricos en los que se basa todo el desarrollo del trabajo; en el capítulo 3, la metodología, se hablara de cómo se trabajaron los diferentes conceptos teóricos con el fin de desarrollar los objetivos planteados, en el capítulo 4, los resultados y discusión, se comentará los resultados obtenidos en la metodología y por último, en el capítulo 5, conclusiones, recomendaciones y trabajo futuro se hablará acerca de lo que se logró hacer e indicar varias recomendaciones acerca del trabajo, además de establecer las pautas para un trabajo futuro.

2. MARCO TEÓRICO

2.1. PEQUEÑA INTRODUCCIÓN A SLAM

La técnica de SLAM (Simultaneous Localization and Mapping) consiste en una técnica de reconstrucción espacial con el fin de mapear un lugar desconocido a la vez que calcula su localización, por esa razón se llama localización y mapeo simultaneo. Esta es una técnica compleja debido a las condiciones dinámicas del ambiente, por lo que los algoritmos enfocados en este tema son principalmente de índole probabilística. Dando como resultado, mapeos con limitaciones debido a la gran cantidad de capacidad computacional que tendría un mapeo completamente exacto. Aun así, esta técnica de SLAM ha sido implementada en autos autónomos, vehículos aéreos y acuáticos no tripulados, exploradores planetarios, últimamente en robots domésticos y hasta en elementos invasivos del cuerpo (Mountney, Stoyanov, Davison, & Yang, n.d.).

Uno de los métodos utilizados para realizar el mapeo mediante la técnica de SLAM es mediante el uso de mapas topológicos, los cuales son un método de representación ambiente que captura los objetos y la relación entre ellos, es decir no tiene distancias definidas entre un objeto u otro, pero conservando la misma escala para todos los objetos lo que le permite calcular distancias aproximadas. Existen enfoques topológicos del SLAM que intentan tener la misma consistencia que los mapas métricos (Cummins & Newman, 2008).

La técnica de SLAM parte de los datos del entorno entregados por los diferentes sensores involucrados tanto en los procesos de mapeo como en los procesos de localización de la plataforma móvil, lo que comúnmente se llama odometría. La técnica de SLAM utiliza una amplia gama de sensores, siempre dependiendo de la necesidad, pero lo que tienen todos en común es que, dependiendo su nivel de desarrollo, más específicamente en su exactitud, serán impulsadores de nuevos algoritmos (Magnabosco & Breckon, 2013).

Por parte de los procesos de mapeo, hay una gran variedad de sensores disponibles para la adquisición de estos datos, desde sensores táctiles hasta sensores ópticos, estos últimos siendo los más aptos para este tipo de trabajos debido a que son muy precisos para este tipo de tareas, y que vienen en diferentes tipos como los laser unidimensionales, los de barrido 2D, los LIDAR 3D de alta definición, o los sensores de sonar 2D y 3D además de las cámaras 2D, o varias de ellas para una toma 3D. En especial los enfocados en arquitectura laser (LIDAR) son especializados para los temas de SLAM. De hecho, el ganador del DARPA Grand Challenge de 2005, el robot Stanley, utilizó 5 sensores LIDAR los cuales trabajaron en conjunto para recrear un mapa 3D del entorno. Desde 2005 se ha incrementado la investigación orientada a técnicas de SLAM visuales (Virtual SLAM, VSLAM), utilizando cámaras tradicionales debido al incremento de estas en la actualidad llegando a los teléfonos móviles (Karlsson et al., 2005).

Por parte de los procesos de localización se hace uso de la odometría que es el estudio de la estimación de la localización relativa de los vehículos móviles durante la navegación. Este estudio se hace mediante la adquisición de los datos de las ruedas, los cuales son por general sensores encoders. Debido a las condiciones cambiantes del entorno se habla de estimación y no de determinación de coordenadas, en especial el efecto de deslizamiento de las ruedas el cual es un problema muy común.

En general la técnica de SLAM requiere una capacidad considerable de recursos computacionales debido a la cantidad de datos que se manejan, para las entradas, procesamiento y salidas de los algoritmos de mapeo y eso sin incluir el método elegido para dicho procesamiento (probabilístico o exacto). En caso de ser un método de procesamiento de los datos exacto, la cantidad de recursos computacionales se vuelve muy grande.

2.2. TARJETA DE ADQUISICIÓN DE DATOS ODROID

Para un algoritmo de mapeo 2D, no se requiere gran capacidad computacional, además de que esta capacidad debe ser portátil, por el hecho de que será montado en una plataforma móvil con funcionamiento autónomo. Por ello una tarjeta de adquisición de datos es ideal.

La Odroid es, según sus creadores, la primera tarjeta de adquisición de datos con una arquitectura de ordenador de placa reducida (Single Board Computer, SBC) con un procesador multiprocesos heterogéneo, es decir varios procesadores cada uno con características definidas y específicas. Un ordenador de placa reducida significa que todo lo necesario para ser en un ordenador se encuentra en la misma placa base, sin necesidad de recurrir a tarjetas externas, y que suele ser de tamaño reducido.

Figura 1. Tarjeta de adquisición Odroid XU3
("ODROID | Hardkernel", 2016)

El microprocesador multiprocesos de la Odroid está construido bajo la arquitectura ARM (Advanced RISC Machine, maquina RISC avanzada). RISC (Reduced Instruction Set Computer, ordenador con conjunto reducido de instrucciones). Esto significa que la Odroid funciona mediante un microprocesador que trabaja con un conjunto reducido de instrucciones y que, por lo tanto, permite potenciar la capacidad de rendimiento de la Odroid. “Con más de 50 millones de procesadores ARM producidos a partir de 2014, la arquitectura del conjunto de instrucciones ARM es el más utilizado en términos de cantidad producida” (Matas, 2013).

Entre lo más destacado de esta tarjeta de desarrollo se encuentra que está pensada para ser una tarjeta de adquisición de datos de gran potencia, al combinar gran capacidad de procesamiento, con otras importantes prestaciones como los puertos USB 3.0, puertos dedicados a la imagen de alta definición (Display Port y Micro HDMI) más otras prestaciones de los ordenadores de placa reducida. Es importante resaltar la capacidad de la Odroid de

operar una tarjeta eMMC, la cual es un homologo a los discos duros de estado sólido del cual es conocida su rapidez. Además, se le puede instalar un sistema operativo, que en cierta forma es una desventaja a comparación por ejemplo de la Raspberry que tiene un sistema operativo propio.

2.3. PLATAFORMA MÓVIL IROBOT CREATE 2

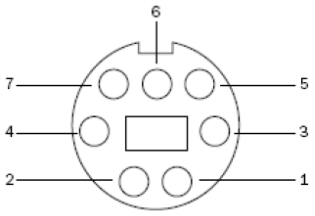
Para un algoritmo de mapeo es fundamental tener una plataforma móvil que sea capaz de ser controlada mediante una cantidad de sensores y actuadores que tenga a su disposición. En este caso una plataforma ideal para este tipo de mapeos, además de la relativa baja dificultad de este tipo de mapeos en comparación con los otros, debido a que es una plataforma con una buena disposición de sensores y actuadores a un relativo bajo precio.

La plataforma IRobot Create 2, de la empresa IRobot está orientada para ser utilizada a un nivel educativo.

Figura 2. Plataforma IRobot Create 2
("iRobot Create 2 brings DIY to Roomba robots", 2014)

Para algunos datos extra, revisar la guía adjunta de este elemento de hardware en el apartado de fundamento teórico.

Dado que la tarjeta Odroid estará conectada de forma serial tanto a la plataforma móvil como al sensor laser, dicha plataforma móvil debe tener un puerto serial por el cual serán enviados los datos. Este puerto está compuesto por pines de los cuales un par de ellos están dedicados a la emisión-transmisión (Rx-Tx). Los otros pines están dedicados a la velocidad serial y por último los pines dedicados al voltaje.



Pin	Name	Description
1	Vpwr	Roomba battery + (unregulated)
2	Vpwr	Roomba battery + (unregulated)
3	RXD	0 – 5V Serial input to Roomba
4	TXD	0 – 5V Serial output from Roomba
5	BRC	Baud Rate Change
6	GND	Roomba battery ground
7	GND	Roomba battery ground

Fig 3. Puerto Serial IRobot Create 2
("iRobot® Create OPEN INTERFACE", 2006)

De esta forma para acceder a los diferentes actuadores y sensores de la plataforma móvil de forma serial, dicha plataforma cuenta con una organización por paquetes de datos que están encabezados por un comando especial. De esta manera, para acceder a un actuador o sensor en especial se debe manipular mediante dicho comando especial. Como se dijo en la guía adjunta, para ser manipulada, la plataforma se le envía de forma serial cadenas de comandos por corchetes.

Entre los sensores más relevantes se encuentran los sensores encoders, los sensores de caída y un sensor digital de choque. En este caso se hablará de los sensores encoders que son los más importantes para el objetivo de este trabajo.

Left Encoder Counts **Packet ID: 43** **Data Bytes 2, unsigned**

The cumulative number of raw left encoder counts is returned as an unsigned 16-bit number, high byte first. This number will roll over to 0 after it passes 65535.

Range: 0 - 65535

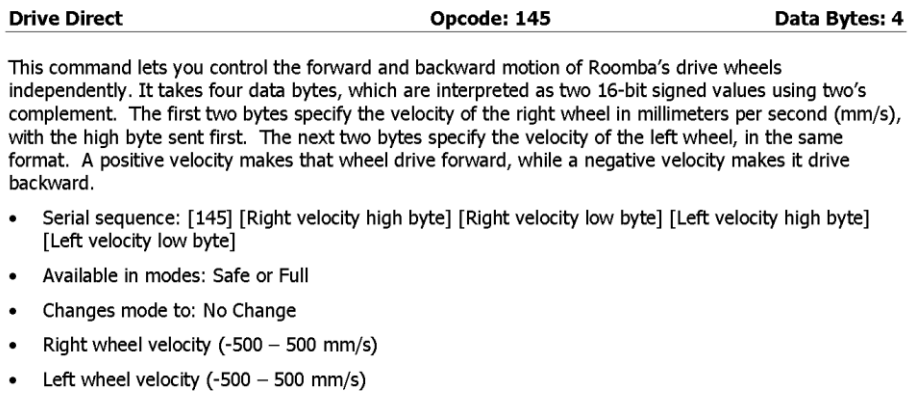
Right Encoder Counts **Packet ID: 44** **Data Bytes 2, unsigned**

The cumulative number of raw right encoder counts is returned as an unsigned 16-bit number, high byte first. This number will roll over to 0 after it passes 65535.

Range: 0 - 65535

Fig 4. Sensores Encoder Izquierdo y Derecho
("iRobot® Create OPEN INTERFACE", 2006)

De esta forma, se muestra el paquete de datos correspondiente al sensor deseado. Además, se muestran las características de dicho paquete de datos y como se relaciona con el fenómeno físico sensado. En este caso el paquete de datos de cualquiera de los dos sensores tiene un tamaño de 2 bytes sin signo, es decir de 0 a 65535. Esto también indica que los encoders tienen una resolución de 2 bytes, es decir de 16 bits, lo cual es una resolución por encima de la media para estos sensores.



Drive Direct **Opcode: 145** **Data Bytes: 4**

This command lets you control the forward and backward motion of Roomba's drive wheels independently. It takes four data bytes, which are interpreted as two 16-bit signed values using two's complement. The first two bytes specify the velocity of the right wheel in millimeters per second (mm/s), with the high byte sent first. The next two bytes specify the velocity of the left wheel, in the same format. A positive velocity makes that wheel drive forward, while a negative velocity makes it drive backward.

- Serial sequence: [145] [Right velocity high byte] [Right velocity low byte] [Left velocity high byte] [Left velocity low byte]
- Available in modes: Safe or Full
- Changes mode to: No Change
- Right wheel velocity (-500 – 500 mm/s)
- Left wheel velocity (-500 – 500 mm/s)

Fig 5. Comando utilizado en el código de movimiento de la plataforma ("iRobot® Create OPEN INTERFACE", 2006)

En el caso de los actuadores, se encuentran los motores y los indicadores LED. El movimiento de los motores está controlado mediante un paquete de datos de 4 bytes, tal y como se muestra en la imagen anterior. En él se especifica las velocidades de ambas ruedas.

2.4. SENSOR LASER RPLIDAR

Para un algoritmo de mapeo 2D es necesario que la plataforma móvil antes mencionada lleve consigo una especie de sensor que pueda detectar su entorno. Uno de los sensores más capacitados para este tipo de trabajos son los sensores ópticos de arquitectura laser o simplemente sensores láser.

Un sensor laser trabaja mediante el envío y recepción de un rayo láser en el cual se mide su tiempo de vuelo con relación a la velocidad de la luz. De esta forma se determina cuanta distancia recorre el láser a una velocidad constante. El sensor láser RPLidar de la marca RoboPeak funciona específicamente mediante la comunicación de un sistema Host (Administrador), por lo general un Pc y el núcleo del RPLidar. Esta comunicación es serial y funciona mediante el clásico (Tx-Rx). En el cual el sistema Host le manda ordenes, en este

caso llamadas Request Command (Comandos Requisito) y el nucleo del RPLidar le responde con los comandos correspondientes. (Response Command).

En la guía adjunta se explican los diferentes tipos de relación entre dichos comandos. En este caso se especifican los dos más importantes.

1. Petición Simple – Respuesta Simple

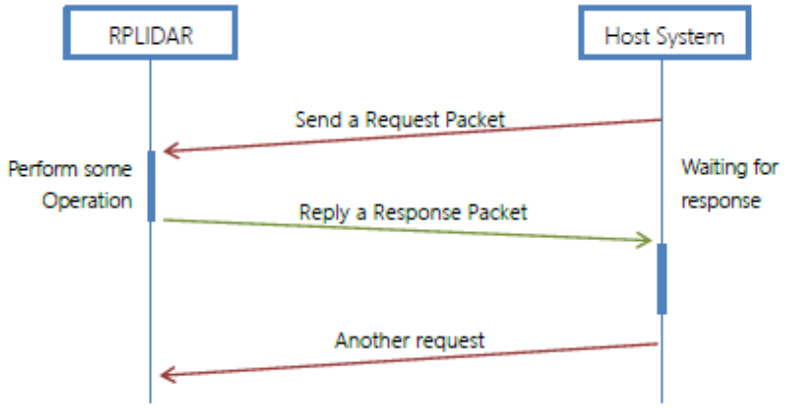


Fig 6. Petición simple – Respuesta simple.
("RPLIDAR Interface Protocol and Application Notes", 2016)

Como lo muestra la imagen, esta relación consiste en que se envía una cadena de respuesta por cada cadena de petición. Este tipo de comunicación sirve para el caso de datos que no constantemente. A diferencia de la próxima comunicación.

2. Petición Simple – Respuesta Múltiple

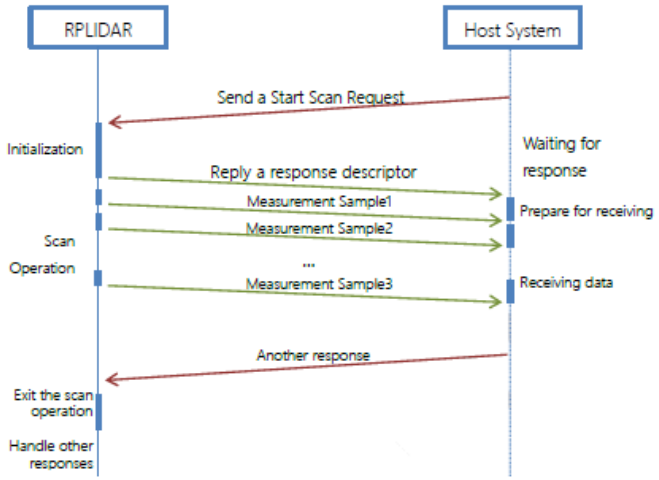


Fig 7. Petición simple – Respuesta múltiple (operación de escaneo).
("RPLIDAR Interface Protocol and Application Notes", 2016)

Como muestra la figura, esta comunicación entre comandos, a diferencia de la anterior consiste en múltiples respuestas por cada petición. Esto ocurre porque los datos que utilizan este tipo de comunicación cambian continuamente, por lo que se acude a un ciclo de comandos de respuesta. De ahí la respuesta múltiple.

Hay un tercer caso en el que no existe un comando de resupuesta, ya que este tipo de comandos de petición no lo requieren. Para el caso del RPLidar, los comandos de petición que no requieren respuesta son el STOP y el RESET.

Para aclarar como pueden ser útiles estas comunicaciones se toma como referencia la siguiente imagen.

Nombre de petición	Valor	Modo de respuesta	Longitud de respuesta	Operación
STOP	0x25	Sin respuesta	Sin respuesta	Se sale de cualquier operación
RESET	0x40	Sin respuesta	Sin respuesta	Reinicia el RPLidar
SCAN	0X20	Respuesta múltiple	5 bytes	Inicia el escaneo luego de verificar que el motor este girando
FORCE_SCAN	0X21	Respuesta múltiple	5 bytes	Inicia el escaneo sin verificar el estado del motor
GET_INFO	0X50	Respuesta simple	20 bytes	Obtiene la información del dispositivo como el serial
GET_HEALTH	0X52	Respuesta simple	3 bytes	Obtiene el estado del dispositivo

Fig 8. Paquetes de petición disponibles para el RPLidar.
("RPLIDAR Interface Protocol and Application Notes", 2016)

Esta imagen muestra los diferentes tipos de comandos de petición (Request Commands) que se pueden enviar desde el sistema Host. Por ejemplo, para el caso del comando RESET, se tiene el comando de petición y la respuesta. Este caso por ejemplo no necesita comandos de respuesta. Para el caso del GET_INFO se obtiene su respectivo comando de petición, su respuesta, y en este caso un comando de respuesta simple, ya que la información solicitada no cambiara de un momento a otro. Para el caso más importante, el SCAN, se tiene su propio comando de petición y su respuesta múltiple ya que con este comando se inicia un escaneo en el cual los valores que entren en dicho comando irán cambiando continuamente.

Como se ve en la imagen, el comando de respuesta del comando de petición SCAN tiene una longitud de respuesta de 5 bytes.

- **Byte 1:**

7	6	5	4	3	2	1	0
Calidad del láser reflejado						Ō	O

- **Byte 2:**

7	6	5	4	3	2	1	0
Ángulo [6:0]							C

- **Byte 3:**

7	6	5	4	3	2	1	0
Ángulo [14:7]							

- **Byte 4:**

7	6	5	4	3	2	1	0
Distancia [7:0]							

- **Byte 5:**

7	6	5	4	3	2	1	0
Distancia [15:8]							

Fig 9. Trama de datos correspondiente al comando de petición SCAN ("RPLIDAR Interface Protocol and Application Notes", 2016)

Como se ve en los recuadros anteriores se puede ver la disposición de dicho comando de respuesta del comando de petición SCAN. En dicho comando de respuesta se puede ver tanto la calidad del rayo láser expulsado en el escaneo respectivo, junto con el ángulo relativo entre la salida del rayo y el sistema coordenado de origen del RPLidar y por último la distancia reflejada por el rayo en el ángulo respectivo. La calidad del rayo es la relación entre la potencia de llegada del rayo comparada con su potencia de salida, con el fin de obtener cuanta potencia del rayo se perdió en el camino de emisión-recepción y, por tanto, saber la calidad de la muestra obtenida.

Para más información acerca del sensor laser, se pueden leer las guías de fabricante del sensor RPLidar.

3. METODOLOGÍA

En una primera instancia, el grupo de trabajo fue recibido por el Sr. Juan Sebastián Botero Valencia en el laboratorio de Automática y Robótica de Parque i. El cómo supervisor del trabajo dio indicaciones sobre cuales eran los objetivos a alcanzar, enseñó los diferentes elementos de hardware involucrados en el trabajo entre los cuales estaban una plataforma móvil de la marca IRobot, la IRobot Create 2; un sensor laser de la marca RoboPeak, el RPLidar; y una tarjeta de desarrollo Odroid cuya versión era la XU3. Además, mostró como trabajaba en conjunto la tarjeta de desarrollo Odroid y la plataforma móvil IRobot Create 2 con la finalidad de lograr el movimiento básico de la plataforma, (adelante, atrás, a los lados). Al girar la plataforma gira en su propio eje debido a que son dos ruedas apoyadas por una rueda loca de base. Nuestra primera tarea fue replicar este mismo experimento nosotros mismos.

Para esto el profesor hizo entrega de las guías de usuario hechas por el fabricante de los elementos de hardware que tenía disponibles, estos eran de la plataforma móvil y del sensor laser, además del código de programación en lenguaje Python encargado de los movimientos de la plataforma el cual fue brindado por el mismo fabricante.

Es importante señalar que parte de estos elementos de hardware también estaban disponibles en el laboratorio de Mecatrónica de la sede Robledo, los cuales eran la plataforma móvil IRobot y la tarjeta de adquisición de datos Odroid, de forma que la tarea encomendada podía ser realizada en la sede Robledo sin necesidad de desplazarse al centro de investigación Parque i.

Además, el supervisor facilitó dos adaptadores de puerto dedicado a la imagen digital, estos son el conversor Display Port a VGA y el conversor HDMI a DVI.

De esta forma, para lograr la tarea asignada por el supervisor del trabajo, se debía tener un sistema operativo basado en Linux que estuviera en la capacidad de procesar el lenguaje Python, además de un IDE (Integrated Development Environment) o un programa de

programación en lo posible especializado en lenguaje Python para poder correr el programa facilitado por el supervisor.

De esta manera se intentó con una Raspberry PI B+, el cual convenientemente tiene un sistema operativo integrado propio de Raspberry basado en Debian de Linux llamado Raspbian y un puerto dedicado a la imagen HDMI, por lo que se utilizó el adaptador de HDMI a DVI. Es conveniente aclarar que antes de hacer la instalación pertinente de software IDE o de librerías para el manejo del lenguaje Python, se debía realizar la instalación del ROS, (Robotics Open Source), el cual es un software libre que dispone de herramientas dedicadas a la técnica del SLAM, por lo que era necesario para el desarrollo del algoritmo. De esta manera la instalación del ROS se volvería en prioridad 1, por encima de cualquier instalación.

Fig 10. Raspberry Pi B+
("Raspberry Pi Modelo B+ Mini PC", 2016).

Al momento de realizar la instalación del ROS, se obtuvo un colapso de sistema, lo que comúnmente se llama como bloqueo, caracterizado por la detención de los procesos que se estaban ejecutando. Se concluyó entonces que la Raspberry no tenía la capacidad necesaria para la instalación del ROS, por lo que debía pensarse en otra tarjeta de desarrollo más potente, y como la Odroid estaba disponible se recurrió a ella.

La desventaja con la Odroid como se dijo antes es que no cuenta con un sistema operativo propio, sino que cuenta con un disco duro en el que debe instalarse el sistema. Para este disco duro hay dos opciones, la tarjeta eMMC o la tarjeta Micro SD. Como no había disponible ninguna de las dos, se optó por utilizar una tarjeta Micro SD de uno de los

integrantes del grupo de trabajo. Como es usual, la tarjeta Micro SD era estándar (clase 4). La instalación del sistema operativo en la tarjeta Micro SD es muy sencillo, solamente hay que seguir los siguientes pasos.

1. Descarga de imagen (.iso) del sistema operativo Linux deseado, en este caso Ubuntu 14.04.01.

Link de descarga de software para Odroid: http://com.odroid.com/sigong/nf_file_board/nfile_board_view.php?keyword=&tag=&bid=235 . Acá se encontrará todo tipo de versiones de Linux, se recomienda Ubuntu trusty 14.04 por compatibilidad con ROS. Para más detalles ver el link: http://www.ros.org/reps/rep-0003.html#id9.

2. Descarga del programa SDFormatter V4.0 y posterior instalación.
3. Ingresar la memoria MicroSD en el puerto respectivo y abrir el programa.
4. Seguir los pasos que indica el programa.
5. ¡Listo!

Como se dijo anteriormente, el sistema operativo que en el que se va a trabajar es el Ubuntu 14.04.01, el cual es un sistema operativo de distribución libre basado en Linux.

Una vez instalado el sistema operativo en la tarjeta Micro SD, se probó en la Odroid junto con uno de los dos puertos dedicados a la imagen que la tarjeta Odroid posee. Estos puertos son el Display Port y el Micro HDMI. En esta primera prueba se utilizó el conversor Display Port a VGA. El resultado fue que la pantalla no daba imagen. Después de investigar al respecto, se dedujo que no era posible que la pantalla diera imagen debido a la conversión realizada por el conversor utilizado de imagen en alta definición (Display Port) a imagen en formato estándar (VGA), el cual interfiere con la frecuencia adecuada del tránsito de datos del puerto dedicado a la imagen.

Como única salida se tuvo que recurrir al puerto Micro HDMI disponible en la Odroid. Por lo que fue necesario el Adaptador Micro HDMI a HDMI. Como no se contaba con un adaptador, este tuvo que ser adquirido el cual, por fortuna, es de bajo costo.

Como resultado se logró una imagen estable, por lo que la adaptación resultante se convirtió en la adaptación por defecto para la implementación del proyecto. Dicha adaptación logró conectar la Odroid a la pantalla de computador, de esta manera la adaptación resultante fue la siguiente.

Odroid- Adaptador Micro HDMI a HDMI- Adaptador HDMI a DVI- Cable DVI- Pantalla.

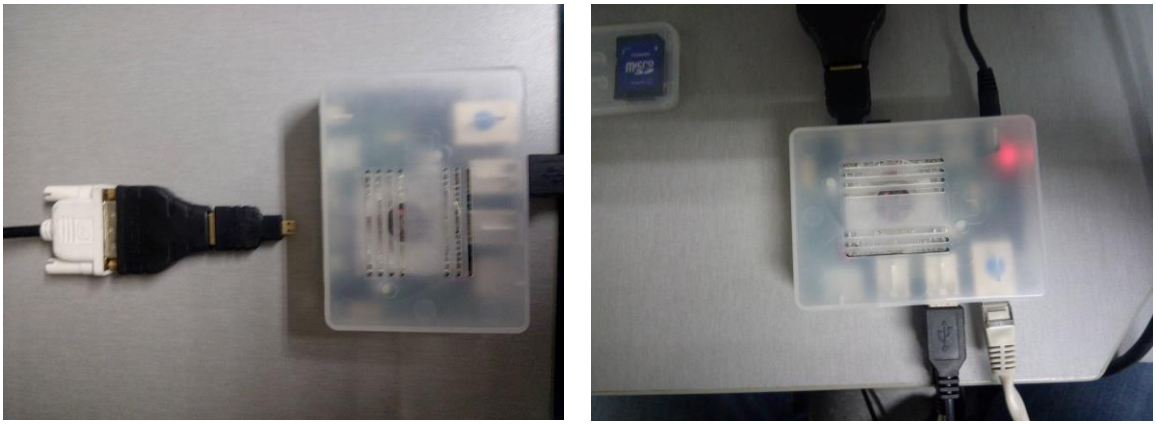


Fig 11. Disposición de adaptadores para conexión del puerto dedicado a la imagen. (Izquierda)
Montaje Completo Tarjeta de Adquisicíon Odroid. (Derecha)

En la imagen de la izquierda se muestra la adaptación resultante descrita anteriormente.

Por su parte la imagen de la derecha muestra el montaje completo de la tarjeta de adquisicón Odroid, incluyendo, el cable de poder, la adaptación de imagen, el disco duro (tarjeta Mcro SD, se encuentra al lado de la adaptación de imagen), el cable LAN y los periféricos necesarios. Por el momento solo mouse y teclado, en los otros dos puertos van la plataforma móvil y el sensor laser RPLidar.

Una vez se estabilizó la imagen y se inició el sistema operativo normalmente, dicho sistema operativo solicitó realizar las ultimas instalaciones de sistema, las cuales se realizaron. Una vez terminadas se procedió a instalar el ROS. El resultado fue que nuevamente el sistema se bloqueó, esta vez debido a la capacidad de la tarjeta Micro SD en la que se instaló el sistema operativo. Cabe recordar que la Micro SD es estándar (clase 4) es decir una velocidad muy baja en comparación con la velocidad optima de la Odroid. Cabe recordar también que el disco duro recomendado de la Odroid es la tarjeta eMMC la cual es un homólogo de los discos duros de estado sólido.

De esta manera había dos opciones, adquirir una tarjeta eMMC o una tarjeta Micro SD más potente (clase 10). Debido a la diferencia de precios se optó por comprar una tarjeta Micro SD clase 10 de 8 GB, la cual es importante aclarar que no es barata para un presupuesto limitado.

De nuevo se procedió a instalar el sistema operativo en la nueva tarjeta Micro SD e iniciar el procedimiento anteriormente mencionado de iniciar el sistema operativo en la Odroid, instalar las últimas actualizaciones de sistema requeridas y por ultimo instalar el ROS. En el proceso de instalación del ROS, no había señales de que se colapsara de nuevo el sistema, hasta que apareció un mensaje que decía que faltaba espacio en disco para terminar la instalación. Esto ocurrió en parte por que inicialmente se instalaron las actualizaciones de sistema requeridas. Aunque es muy posible que aun al no instalar estas actualizaciones aun no hubiera el espacio suficiente para instalar el ROS.

Inmediatamente se pensó en que el espacio en la tarjeta Micro SD era insuficiente por lo que sería necesario comprar una nueva tarjeta Micro SD con mayor espacio de almacenamiento. Gracias a nuestro supervisor que pudo informar a tiempo que la Odroid no utiliza el espacio completo disponible en la tarjeta Micro SD, sino que lo limita a un espacio libre determinado.

La solución era dirigirse a la opción del sistema de administración de particiones y una vez allí, redimensionar el espacio disponible hasta el espacio de almacenamiento total de la memoria Micro SD. Una vez hecho esto, se reinició la instalación del ROS, se sobrescribieron los archivos y se pudo completar la instalación.

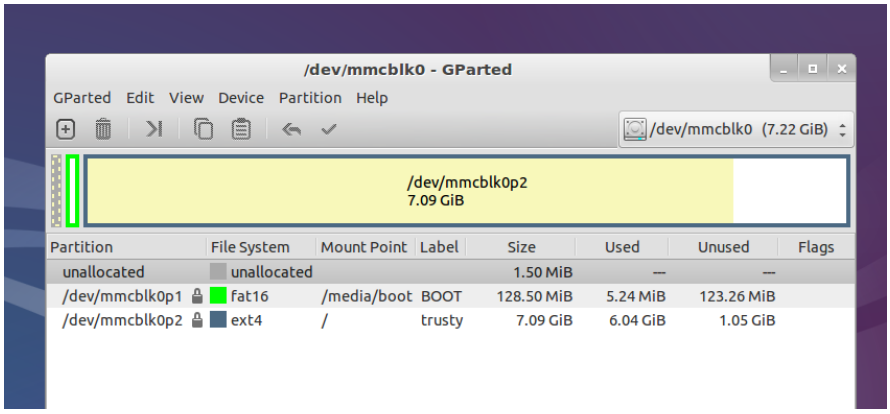


Fig 12. GParted. Aplicación para redimensionar el tamaño en disco.

En el inicio, en el apartado de herramientas, se encuentra la aplicación GParted. Al utilizar la Odroid por primera vez se verá que el espacio denominado extx, siendo x un numero cualquiera, permitirá redimensionar el espacio disponible en la tarjeta Micro SD hasta el límite total sobrante permitido de la tarjeta. En este caso se expandio hasta los 7.1 GB.

Una vez llegado a este punto, ya se tenía la libertad de iniciar con la tarea que había asignado el supervisor del trabajo. Para esto se instaló las librerías necesarias para trabajar en lenguaje Python y se descargó un IDE especializado en lenguaje Python llamado Spyder.

Lo siguiente fue iniciar con la lectura de la guía del fabricante de la plataforma móvil, específicamente en los temas básicos de movimiento (actuadores) y los diferentes modos de uso de la plataforma. Con respecto al movimiento de los actuadores, el código facilitado por el fabricante utiliza el código 145.

Entre las guías de laboratorio adjuntas se encuentra una enfocada en la plataforma móvil IRobot y de la implementación general de la misma para lograr la tarea propuesta al inicio de movimiento de dicha plataforma. En las imágenes 3 y 4 del punto 6 de dicha guía se muestran tanto los modos de funcionamiento de la plataforma y los movimientos de la plataforma, así como el comando de la plataforma encargado de manejar los actuadores (motores).

```
def callbackKey(event):
    k = event.keysym.upper()
    motionChange = False

    if event.type == '2':
        if k == 'P':        # Passive
            sendCommandASCII('128')
        elif k == 'S':      # Safe
            sendCommandASCII('131')
        elif k == 'F':      # Full
            sendCommandASCII('132')
        elif k == 'C':      # Clean
            sendCommandASCII('135')
        elif k == 'D':      # Dock
            sendCommandASCII('143')
        elif k == 'SPACE':  # Beep
            sendCommandASCII('140 3 1 64 16 141 3')
        elif k == 'R':      # Reset
            sendCommandASCII('7')
        elif k == 'UP':     # Forward
            callbackKey.up = False
            motionChange = True
        elif k == 'DOWN':   # Backward
            callbackKey.down = False
            motionChange = True
        elif k == 'LEFT':   # Counterclockwise
            callbackKey.left = False
            motionChange = True
        elif k == 'RIGHT':  # Clockwise
            callbackKey.right = False
            motionChange = True
```

Fig 13. Función de modos de funcionamiento y dirección de plataforma
Tomado de Código IRobot Create 2

```
if motionChange == True:
    velocity = 0
    velocity += VELOCITYCHANGE if callbackKey.up is True else 0
    velocity -= VELOCITYCHANGE if callbackKey.down is True else 0
    rotation = 0
    rotation += ROTATIONCHANGE if callbackKey.left is True else 0
    rotation -= ROTATIONCHANGE if callbackKey.right is True else 0

    # Compute wheel velocities
    vr = velocity + (rotation / 2)
    vl = velocity - (rotation / 2)

    # Send drive command
    cmd = struct.pack(">Bhh", 145, vr, vl)
    if cmd != callbackKey.lastDriveCommand:
        sendCommandRaw(cmd)
        callbackKey.lastDriveCommand = cmd
```

Fig 14. Función de Comando [145] para manejo de los motores.
Tomado de Tomado de Código IRobot Create 2

Una vez logrado el movimiento de la plataforma móvil, el siguiente objetivo fue lograr la adquisición del sensor laser de la marca RoboPeak, el RPLidar. Para este objetivo, así como con la plataforma móvil, lo primero fue estudiar los manuales de fabricante del sensor, en especial el manual de comunicación y protocolo, el cual fue fundamental para entender cómo funciona el sensor RPLidar.

Con el sensor laser, es importante anotar que, a diferencia de los otros elementos de hardware, este sensor laser RPLidar es sumamente limitado. Tanto así que la única unidad existente se encuentra en Parque i en el laboratorio de Automática y Robótica y que además es de uso exclusivo de este laboratorio. Por lo cual, para su manipulación, fue necesario ir hasta Parque i para realizar las pruebas pertinentes.

Una vez en Parque i, el asesor Juan Sebastián enseñó a los cuatro integrantes los elementos de hardware correspondientes al sensor laser RPLidar. Estos eran el sensor laser como tal, un cable USB y un conversor serial de TTL UART a USB, para poder conectar el sensor al computador. Al entregar estos elementos colocó una tarea al grupo y era obtener los datos del sensor. Para esto le dijo al grupo que se familiarizaran con el sensor y que por medio de un programa especializado en comunicación serial llamado XCTU, empezaran a adquirir los datos.

De esta manera se retomó la guía de fabricante del sensor RPLidar enfocada en protocolo de comunicación, y se revisó que comandos de petición enviar para empezar capturar datos. De esta manera se probaron estos comandos en el programa XCTU y se obtuvo una cadena de comandos que se pudo guardar en un documento tipo bloc de notas.

```
04-12-2016 16:00:06.800,-,SERIAL,"COM8 - 115200/8/N/1/N"

04-12-2016 16:00:10.510,0,SENT,A520
04-12-2016 16:00:10.530,1,RECV,A55A0500004081
04-12-2016 16:00:11.080,2,RECV,8D679D670AA60D9E270A96939EED
D2332167685242B138619251C130245
04-12-2016 16:00:11.130,3,RECV,2900002641264A162ADD268B152A
00002F5640000028D65000002276600
04-12-2016 16:00:11.190,4,RECV,0002BF6600000257670000002F167
67B9EFE099E079FC50996B39F9009A6
```

Fig 15. Fragmento de datos recopilados por software XCTU
Tomado de Toma de datos en bloc de notas

En la imagen anterior se puede observar el bloc de notas resultante, el cual corresponde al comando de petición SCAN. De esta forma se puede observar, el envío (SENT) del comando de petición, y el comando de respuesta múltiple (RECV), encabezado por el descriptor de respuesta (Descriptor Response), el cual indica que tipo de comando de petición va a ser retornado. De este modo se tomó una muestra de este block de notas, más específicamente del primer comando de respuesta y el cual fue separado en comandos para su análisis.

8D 67 9D 67 0A
A6 0D 9E 27 0A
96 93 9E ED 09
A6 33 9F B6 09
A6 CD 9F 83 09
B6 6B A0 53 09
8E 19 A1 2D 09
B2 AD A1 07 09
5A 45 A2 FC 08
26 FB A2 73 08

Fig 16. Fragmento de toma de datos separada.
Tomado de Toma de datos separada, archivo Word

Lo que se ve en la imagen es un fragmento de los datos separados de a 5 bytes (hexadecimal). Estos son los 5 bytes de respuesta del comando de respuesta resultante del comando de petición SCAN. Esta información se explicó en el marco teórico. Como se puede ver en la imagen cada byte está escrito en forma hexadecimal. de esta manera se logró separar los datos que nos interesaban para de esta forma realizar la respectiva conversión y así obtener lo que nos interesa (calidad, ángulo y distancia).

La tarea a partir de aquí fue buscar una manera de realizar todo lo mencionado anteriormente mediante código. De esta manera, navegando por la web, se encontraron varios códigos intentando hacer funcionar uno por uno. Después de varios intentos fallidos, se encontró un código que no presentaba problemas por librerías extrañas, o por incompatibilidades con el software ROS como los anteriores. Además, era ideal ya que era lo que estábamos buscando en todo sentido. Estaba programado en Python y tenía todas las características del sensor laser. Además, y como dato curioso está programado en la técnica de programación orientada a objetos al igual que el código de la plataforma móvil.

Es importante aclarar en este punto que tanto la técnica de programación orientada a objetos como la programación en lenguaje Python era nueva para los integrantes del trabajo, ya que estos temas se habían oído hablar, pero nunca se había consultado la teoría específica y menos llevarlo a la práctica como se está haciendo en este trabajo.

Continuando con el sensor RPLidar, el código encontrado fue puesto a prueba de forma inmediata siguiendo las instrucciones indicadas en forma de comentarios al inicio de dicho código.

```
'''>>> from rplidar import RPLidar
>>> lidar = RPLidar('/dev/ttyUSB0')
>>> info =lidar.get_info()
>>> print('\n'.join('%s: %s' % (k, str(v)) for k, v in info.items()))
firmware: (1, 15)
model: 0
hardware: 0
serialnumber: 64E699F3C7E59AF0A2E69DF8F13735
>>> lidar.get_health()
('Good', 0)
>>> process_scan = lambda scan: None
>>> for scan in lidar.iter_scans():
...  process_scan(scan)
KeyboardInterrupt
>>> lidar.stop()
>>> lidar.stop_motor()'''
'''
```

Fig 17. Instrucciones de manejo del código del RPLidar
Tomado de Código RPLidar.

Continuando con el sensor RPLidar, el código encontrado fue puesto a prueba de forma inmediata siguiendo las instrucciones indicadas en forma de comentarios al inicio de dicho código.

Lo que indican estas instrucciones es lo siguiente. Cabe recordar que la programación del manejo del sensor RPLidar está en bajo la técnica POO (Programación Orientada a objetos).

En la línea 1, se llama la clase (RPLidar) desde el archivo llamado (rplidar).

En la segunda línea se crea el objeto (lidar) y se le asigna el puerto de conexión USB por defecto (0).

En la tercera línea se llama a la función info del sensor RPLidar creada mediante funciones de la clase RPLidar. En la cuarta línea se imprimen los datos solicitados en la línea anterior. En la línea 9, se pregunta por el estatus del sensor RPLidar mediante la función get_health. En la línea 11, se inicializa la matriz que contendrá los valores de calidad, distancia y ángulo con respecto a cada ángulo.

En la línea 12, se asignan los valores anteriormente mencionados mediante la función iter_scans en la matriz antes mencionada.

```
def iter_scans(self, min_len=5, force=False):
    '''Iterate over scans. Note that consumer must be fast enough,
    otherwise data will be accumulated inside buffer and consumer will get
    data with increasing lag.

    Yields
    ------
    scan : list
        List of the measurments. Each measurment is tuple with following
        format: (quality, angle, distance). For values description please
        refer to `iter_measurments` method's documentation.
    '''
    scan = []
    for new_scan, quality, angle, distance in self.iter_measurments(force):
        if new_scan:
            if len(scan) > min_len:
                yield scan
            scan = []
        if quality > 0:
            scan.append((quality, angle, distance))
```

Fig 18. Fragmento Codigo RPLidar, función iter_scans
Tomado de Código RPLidar.

En la imagen anterior se muestra la función iter_scans, en el cual se muestra como se asignan los valores anteriormente mostrados, en las últimas dos líneas de las instrucciones.

De esta manera se puede ver como se asignan, los valores escaneados a la matriz scan.

Dichos valores escaneados ingresan a la matriz de scan en forma bytes en sistema hexadecimal y con un mínimo de 5 valores, representa un minimo de 5 bytes. Por los datos anteriores se puede concluir que la matriz (scan) formada en esta función es el comando de respuesta correspondiente al comando de petición SCAN.

Esta misma conclusión, nos lleva a otra conclusión. Las instrucciones dadas en el código si permiten que el sensor RPLidar cense, sin embargo, no nos permite visualizar los datos de

calidad, ángulo y distancia como queremos. Esto se debe a que como se dijo anteriormente en dicha trama de bytes en hexadecimal están mezclados los datos que deseamos, por lo que se hace necesario la separación de estos datos. Esta separación se realiza mediante la función _process_scan.

```
def _process_scan(raw):
    '''Procesa la entrada 'raw' (información bruta) y la convierte
    en: Calidad, ángulo y distancia
    '''
    new_scan = bool(_b2i(raw[0]) & 0b1)
    inversed_new_scan = bool((_b2i(raw[0]) >> 1) & 0b1)
    quality = _b2i(raw[0]) >> 2
    if new_scan == inversed_new_scan:
        raise RPLidarException('New scan flags mismatch')
    check_bit = _b2i(raw[1]) & 0b1
    if check_bit != 1:
        raise RPLidarException('Check bit not equal to 1')
    angle = ((_b2i(raw[1]) >> 1) + (_b2i(raw[2]) << 7)) / 64.
    distance = (_b2i(raw[3]) + (_b2i(raw[4]) << 8)) / 4.
    return new_scan, quality, angle, distance
```

Fig 19. Función _process_scan
Tomado de Código RPLidar

Como se dijo anteriormente, se necesitaba una función que capture la información de escaneo, y la separará en los datos que necesitamos, calidad, ángulo y distancia. Por lo tanto, se requiere una función que imprima estos valores.

```
lidar = RPLidar('/dev/ttyUSB0')
lidar.get_health()
lidar.stop()

l1 = lidar.iter_scans()  #quality, angle, distance

#for i in l1:            #Generador continuo
#    print(i)

for i in range(2):
    sc = l1.next()
    if len(sc)>200:
        print(sc)

lidar.stop()
lidar.disconnect()
```

Fig 20. Función Imprimir datos de escaneo
Tomado de Código RPLidar

La imagen anterior muestra cómo se crea una variable a partir de una función que tiene como resultado retornar los datos de calidad, angulo y distancia. Este paso es un poco confuso ya que desde dicha función entran en funcionamiento otras funciones, la función final es la encargada de retornar los valores ya mencionados. Para aclarar lo anterior se puede seguir este esquema de funciones.

Iter_scans - iter_ measurments - _process_scan.

De esta manera el resultado del código anterior es la trama de datos deseada.

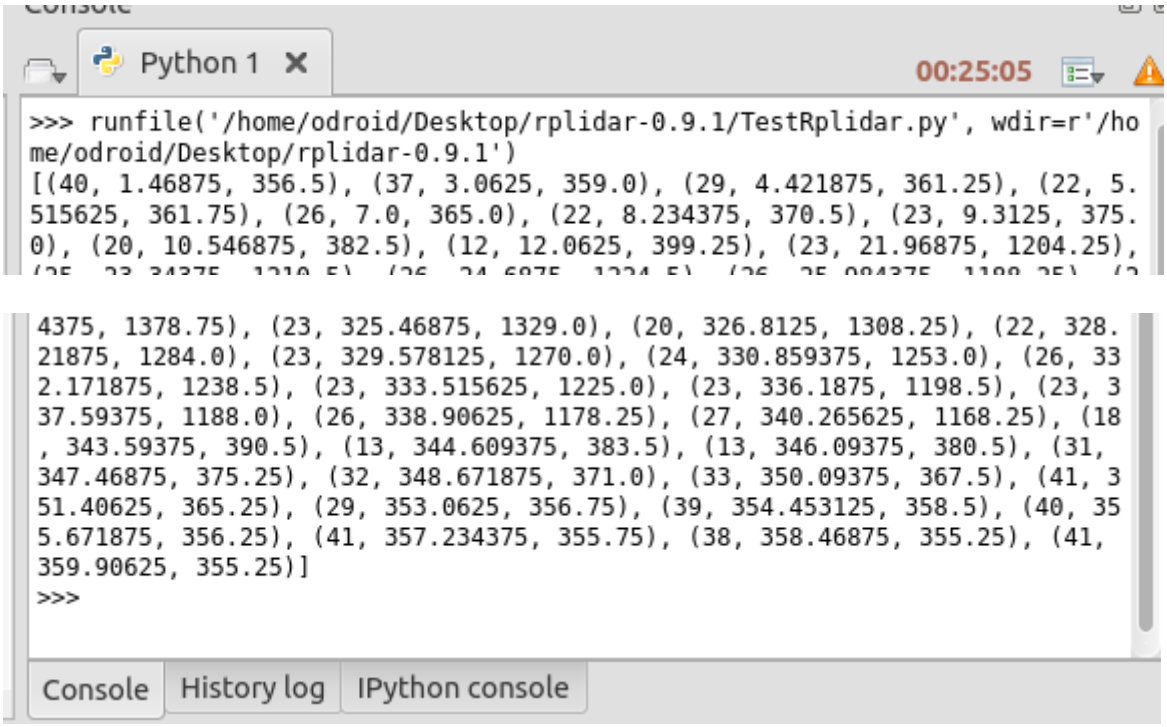


Fig 21. Trama de datos inicial (arriba) y final (abajo).
Tomado de Código RPLidar

Las imágenes anteriores muestran la trama de datos deseada, empezando por la primera imagen en el cual se muestra el vector correspondiente al ángulo (1.46875) y terminando por la segunda imagen en el cual se muestra el vector correspondiente al ángulo (359.90625). La trama de datos completa está en un documento de Word adjunto.

Una vez realizada la adquisición de datos del sensor laser RPLidar, la siguiente tarea fue realizar la adquisición de datos de los encoders. En forma de ensayo, se procedió entonces a realizar la adquisición de uno de los encoders. Para esta tarea, se acudió a la guía de fabricante de la plataforma móvil.

En dicha guía, se revisó la sección de comandos de entrada (Input Commands) y se encontró el comando serial para los datos de los sensores. Este comando es el 142. La guía de fabricante muestra que, para adquirir los datos, este valor debe ir acompañado por el número de paquete del sensor que deseamos. En el marco teórico se explicó que cada sensor está agrupado en un paquete especial.

Sensors **Opcode: 142** **Data Bytes: 1**

This command requests the OI to send a packet of sensor data bytes. There are 58 different sensor data packets. Each provides a value of a specific sensor or group of sensors.

For more information on sensor packets, refer to the next section, "*Roomba Open Interface Sensors Packets*".

- Serial sequence: [142] [Packet ID]
- Available in modes: Passive, Safe, or Full
- Changes mode to: No Change

Identifies which of the 58 sensor data packets should be sent back by the OI. A value of 6 indicates a packet with all of the sensor data. Values of 0 through 5 indicate specific subgroups of the sensor data.

Fig 22. Comando de Entrada dedicado a Sensores.
("iRobot® Create OPEN INTERFACE", 2006)

De esta manera, y aprovechando las funciones que nos brindaba el algoritmo facilitado por el fabricante, se pudo realizar una función que pudiera capturar dichos datos.

```
def getSensorInfo(self, sensorCode, data_bytes, signed = False):

    self.sendCommandASCII('142 ' + str(sensorCode))
    global connection

    if connection is not None:

        if signed is True and data_bytes == 1:
            return self.get8Signed()                    # Retorna un valor entre [-128,127]
        elif signed is True and data_bytes == 2:
            return self.get16Signed()                  # Retorna un valor entre [-32768, 32767]
        elif signed is False and data_bytes == 1:
            return self.get8Unsigned()                   # Retorna un valor entre [0, 255]
        elif signed is False and data_bytes == 2:
            return self.get16Unsigned()                  # Retorna un valor entre [0, 65535]

    else:
        tkMessageBox.showerror("Error", "You are not connected")
```

Fig 23. Función Creada para obtención de datos de sensores (GetSensorInfo)
Tomado de Tomado de Código RoombaLidar

Para el caso de los encoders la función resultante será la última debido a la forma en que se entregan los datos de dicho encoder. Para más información revisar marco teórico.

Al usarse la función se hará de la siguiente manera. En la imagen anterior se puede observar cómo se ingresa el comando 142 (Sensor Command), además de una variable llamada (sensorCode) la cual es la variable que corresponde al número de paquete del sensor, la variable (data_bytes) que indican el tamaño del paquete y por último la variable (signed) que indica si el paquete trae valores únicamente positivos, o si el paquete trae valores tanto positivos como negativos. Al mezclar estas variables es posible hacer uso de cuatro funciones que ya estaban diseñadas para las diferentes combinaciones de estas dos últimas variables. Dando como resultado cuatro diferentes posibilidades. (get8signed, get16signed, get8unsigned, get16unsigned).

Para el caso de los encoders la función resultante será la última debido a la forma en que se entregan los datos de dicho encoder. Para más información revisar marco teórico. Al usarse la función se hará de la siguiente manera.

```
elif k == 'M': # Petición de encoders
    print (self.getSensorInfo(43,2,False))
```

Fig 24. Petición para obtener datos de encoder.
Tomado de Tomado de Código RoombaLidar

En la imagen anterior se muestra la petición para obtener datos del paquete de datos dedicado al encoder izquierdo el cual está identificado con el número 43. Además, el paquete de datos es entero positivo de 2 bytes. Es importante aclarar que este encoder es un encoder absoluto multivuelta (multiturn encoder). Esto significa que el encoder tendrá un inicio absoluto (único) y que podrá medir varios giros del motor con este mismo origen como referencia, lo cual es muy útil en estos casos ya que se puede conocer no sólo la posición absoluta dentro de una vuelta, sino también el número de vueltas que ha dado el motor, para poder determinar la posición absoluta.

En la siguiente imagen se mostrará el resultado de la obtención del encoder.

```
145 255 56 255 56
145 0 0 0 0
145 0 200 0 200
142 43
1028
145 0 0 0 0
```

Fig 25. Resultado del encoder en el paquete 43.
Tomado de Código RoombaLidar

Luego de la obtención de los datos del encoder, la siguiente tarea fue realizar todo el proceso de forma automática. Para esta tarea se creó un nuevo botón en el menú principal llamado Auto, el cual llama a la función mostrada a continuación.

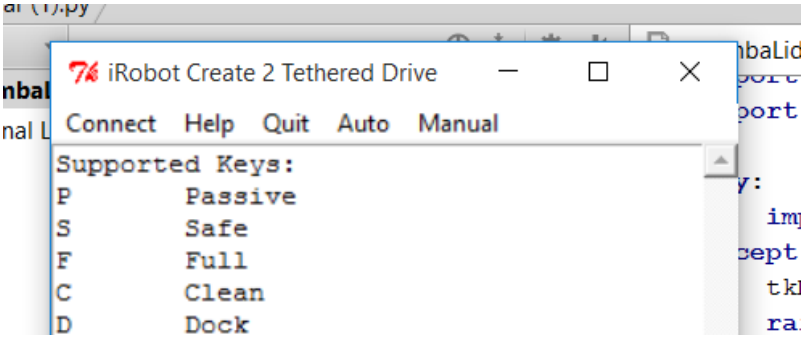


Fig 26. Menú Principal
Tomado de Tomado de Código RoombaLidar

```
def onAuto(self):
    print 'Changing to automatic mode'
    self.auto = True
    try:
        print 'Trying to connect to Lidar'
        ports = self.getSerialPorts()
        Lidar_port = tkSimpleDialog.askstring('COM','Enter COM port where Lidar is connected'
      # Lidar = RPLidar(Lidar_port)
        lidar = RPLidar('/dev/ttyUSB0')
        lidar.get_health()
        lidar.stop()
        l1 = lidar.iter_scans()  #quality, angle, distance
        #for i in l1:            #Generador continuo
        #    print(i)
        for i in range(2):
            sc = l1.next()
             self.creartxt()
            if len(sc)>200:
                print(sc)
                p = open ('archivo.txt','w')
                for elemento in sc:
                    p=open('datos.txt','a')
                    p.write('%s \n' % str(elemento))
                p.close()
                 self.grabartxt()
        lidar.stop()
        lidar.disconnect()

    except serial.SerialException:
        print 'Could not connect to Lidar'
        print 'Changing to manual mode'
        self.auto = False
        tkMessageBox.showerror('Error','Could not connect to: '+ Lidar_port)

    print (self.getSensorInfo(43,2,False))
```

Fig 27. Petición para obtener datos de encoder.
Tomado de Tomado de Código RoombaLidar

Por último se solicitó guardar los datos resultantes de la compilación, tanto del sensor laser RPLidar como del sensor encoder de la plataforma, para realizar una base de datos con varias tomas de estos datos. Para esta tarea se intentó crear un bloc de notas e imprimir los valores en dicho documento. Para esta tarea se crearon unas líneas en el ciclo for que se encuentra en la función que captura los datos del sensor laser y como se muestra en la siguiente imagen.

```
print(sc)
p = open ('archivo.txt','w')
for elemento in sc:
    p=open('datos.txt','a')
    p.write('%s \n' % str(elemento))
p.close()
 self.grabartxt()
```

Fig 28. Función para guardar datos en bloc de notas
Tomado de Tomado de Código RoombaLidar

4. RESULTADOS Y DISCUSIÓN

Se logró realizar el montaje de un sistema operativo estable por medio de una tarjeta Micro SD clase 10 en la tarjeta de adquisición de datos Odroid, además de la conexión entre esta y los diferentes elementos de hardware involucrados (plataforma móvil de la empresa IRobot, la IRobot Create 2 y el sensor laser de la empresa Robopeak, el RPLidar).

Fig 11. Disposición de adaptadores para conexión del puerto dedicado a la imagen. (Izquierda)
Montaje Completo Tarjeta de Adquisicíon Odroid. (Derecha)
Fuente: Autor

Fig 11. Montaje Completo (Odroid, RPLidar, Plataforma Móvil)
Fuente: Autor

La adaptación del puerto de imagen de la Odroid se hizo mediante la conexión entre el puerto Micro HDMI de la tarjeta Odroid y el puerto DVI de la pantalla de la siguiente manera.
Adaptador Micro HDMI a HDMI – Adaptador HDMI a DVI – Cable DVI – Pantalla.

La conexión serial entre los elementos es permitida gracias al archivo de comunicación serial de Python Pyserial, ya que este es el idioma de destino de los códigos de operación de los elementos de hardware involucrados.

Además, se logró instalar el software ROS en la tarjeta de adquisición de datos Odroid, por medio de la necesaria extensión de espacio disponible en la Micro SD brindada por la Odroid.

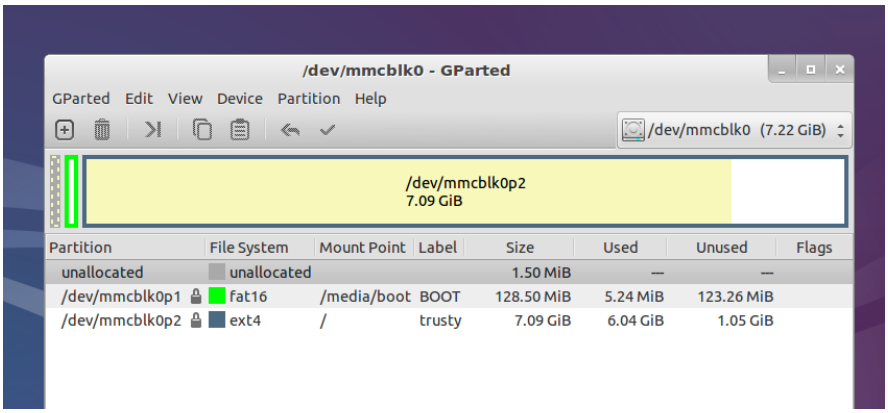
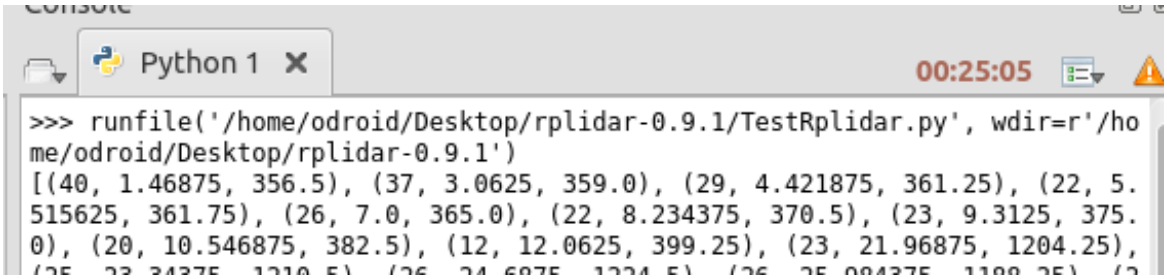


Fig 12. GParted. Aplicación para redimensionar el tamaño en disco.
Fuente: Autor

Se logró la adquisición de datos del sensor laser RPLidar, por medio de una matriz de (calidad, ángulo medido y distancia) por ángulo, es decir una matriz en la que por cada ángulo medido, se toma dicho ángulo, la calidad respectiva y la distancia medida en ese ángulo.



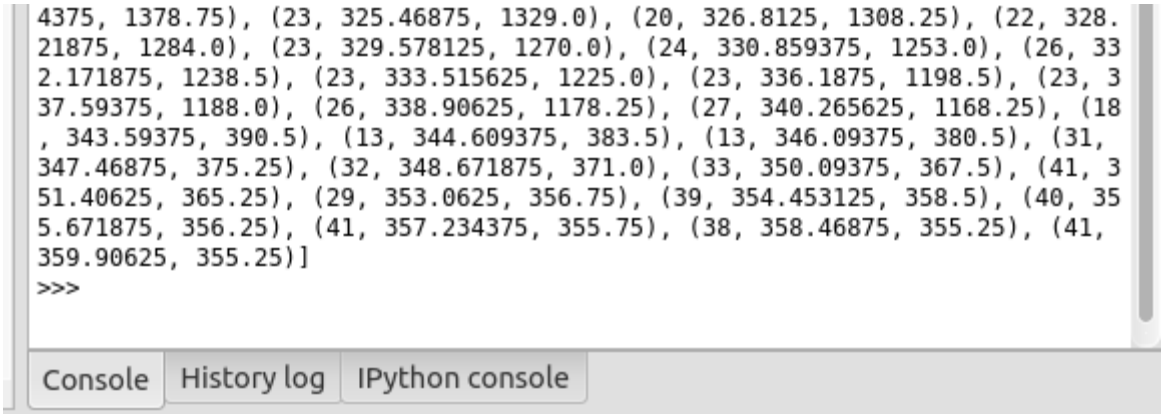


Fig 21. Trama de datos inicial (arriba) y final (abajo).
Tomado de Código RPLidar

Lo siguiente que se logró fue la adquisición de los datos del encoder por lo que se hicieron pruebas de su movimiento. Una de las pruebas fue la de averiguar que dato del encoder de (0 – 65535) corresponde a un giro de la rueda de la plataforma móvil. Al hacer cálculos, un giro de la rueda de la plataforma móvil equivale a un desplazamiento horizontal de 11.78 cm. El dato equivalente a un giro de rueda de la plataforma móvil es de 273.

Fig 21. Trama de datos inicial (arriba) y final (abajo).
Fuente: Autor

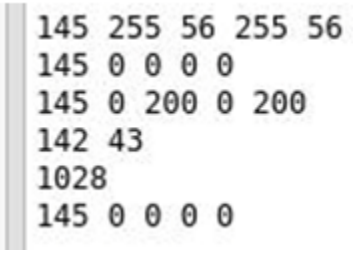


Fig 25. Resultado del encoder en el paquete 43.
Tomado de Código RoombaLidar

El resultado logrado es el dato del encoder (1028), el cual indica el movimiento de la plataforma a partir de un origen único o absoluto. Teniendo como referencia el dato arrojado por el encoder en una sola vuelta se puede determinar la distancia recorrida en este dato.

5. CONCLUSIONES, RECOMENDACIONES Y TRABAJO FUTURO

Se determina una obtención exitosa de los datos deseados para el algoritmo de mapeo 2D. Estos datos corresponden a los datos de entrada de dicho algoritmo. Estos datos corresponden al sensor laser RPLidar, referente a la calidad, el ángulo y la distancia medida en dicho ángulo. Estos datos permitirán realizar la parte de mapeo de la técnica de SLAM. Por parte de la plataforma móvil, el dato obtenido es el del encoder que permitirá la parte de localización de la técnica de SLAM.

De forma secundaría se determina que es necesario una tarjeta de datos mas potente que la Raspberry PI, para la realización del algoritmo de mapeo 2D, debido a la exigencia computacional que el software ROS, en el cual se encuentra entre otras cosas, la técnica SLAM necesaria para el desarrollo del algoritmo. Por esta razón se eligió la tarjeta de adquisición Odroid.

Se determina que no es posible una adaptación de imagen de formato de alta definición (Dsiplay Port) a formato de definición estándar (VGA), debido a que interfieren las frecuencias de funcionamiento de ambos tipos de imagen, por lo que una coincidencia sería imposible.

Se determina la adaptación de video final como la establecida por defecto de esta manera, adaptador Micro HDMI a HDMI, adaptador HDMI a DVI , cable DVI y pantalla.

Se recomienda que para replicar este trabajo, se estudie de forma previa lo correspondiente al lenguaje Python y su aplicación en programación Orientada a Objetos.

Se recomienda tener un manejo previo del sistema operativo Ubuntu, en especial lo correspondiente a las funciones y manejo de la consola.

Se recomienda leer cuidadosamente las guías de fabricante, ya que en ellos se presenta mucha información en poco espacio.

Para un trabajo futuro se espera que los datos obtenidos en este trabajo sean utilizados para la creación del algoritmo de mapeo 2D, el cual como se refirió en la introducción, por medio de la técnica de SLAM.

REFERENCIAS

Cummins, M., & Newman, P. (2008). FAB-MAP: Probabilistic Localization and Mapping in the Space of Appearance. *The International Journal of Robotics Research*, *27*, 647–665. http://doi.org/10.1177/0278364908090961.

Karlsson, N., Bernardo, E. Di, Ostrowski, J., Goncalves, L., Pirjanian, P., & Munich, M. E. (2005). The vSLAM Algorithm for Robust Localization and Mapping, (April), 24–29.

Matas, B. (2013). MCU Market on Migration Path to 32-bit and ARM-based Devices. *Research Bulletin*, 32–35. Retrieved from http://www.icinsights.com/data/articles/documents/541.pdf

Magnabosco, M., & Breckon, T. P. (2013). Cross-spectral visual simultaneous localization and mapping (SLAM) with sensor handover. *Robotics and Autonomous Systems*, *61*(2), 195–208. http://doi.org/10.1016/j.robot.2012.09.023.

Mountney, P., Stoyanov, D., Davison, A., & Yang, G. (n.d.). Simultaneous Stereoscope Localization and Soft-Tissue Mapping for Minimal Invasive Surgery.

ODROID | Hardkernel. (2016). *Hardkernel.com*. Retrieved 2 August 2016, from http://www.hardkernel.com/main/products/prdt_info.php?g_code=g140448267127

iRobot Create 2 brings DIY to Roomba robots. (2014). *SlashGear*. Retrieved 2 August 2016, from http://www.slashgear.com/irobot-create-2-brings-diy-to-roomba-robots-09358883/

iRobot® Create OPEN INTERFACE. (2006). *irobot.com*. Retrieved 2 August 2016, from http://www.irobot.com/filelibrary/pdfs/hrd/create/Create%20Open%20Interface_v2.pdf

RPLIDAR Interface Protocol and Application Notes. (2016). *slamtec.com*. Retrieved 2 August 2016, from http://www.slamtec.com/download/lidar/documents/en-us/rplidar_interface_protocol_en.pdf

Raspberry Pi Modelo B+ Mini PC. (2016). Pccomponentes.com. Retrieved 2 August 2016, from http://www.pccomponentes.com/raspberry_pi_modelo_b_.html

FIRMA ESTUDIANTES

FIRMA ASESOR: JUAN SE.

FECHA ENTREGA: _____

FIRMA COMITÉ TRABAJO DE GRADO DE LA FACULTAD _____

RECHAZADO___ ACEPTADO____ ACEPTADO CON MODIFICACIONES_____

ACTA NO._____

FECHA ENTREGA: _____

FIRMA CONSEJO DE FACULTAD_____

ACTA NO._____

FECHA ENTREGA: _____