ALGORTIMOS PARA LA INTEGRACIÓN DE ESCÁNER LÁSER CON PLATAFORMA MÓVIL

STID RODRÍGUEZ BARRIENTOS

INGENIERÍA MECATRÓNICA

JUAN SEBASTIÁN BOTERO VALENCIA

INSTITUTO TECNOLÓGICO METROPOLITANO

AGOSTO 2016

RESUMEN

En este trabajo se realizan los algoritmos para la integración de tres dispositivos con el propósito de crear un prototipo robótico que reconozca su entorno, con el fin de que en trabajos futuros se pueda realizar una reconstrucción de mapas para la navegación, de modo que se utiliza un escáner láser para la adquisición de datos con el fin de reconocer el entorno, un dispositivo ARM para el procesamiento y la programación de los datos, y por último una plataforma móvil inteligente. Lo primero que se hace es instalar el Debian en el dispositivo controlador con los software y librerías necesarias para programar en Python y realizar la comunicación serial tanto del dispositivo como de la plataforma móvil, posteriormente se realiza un algoritmo para la adquisición de datos del escáner láser y otro para la manipulación de la plataforma móvil, para posteriormente integrar ambos algoritmos en un solo código con el fin de adquirir datos y dar órdenes a la plataforma con el mismo programa desde el dispositivo ARM. Así pues, se realizó dicho código obteniendo la adquisición de datos del escáner láser al mismo tiempo que manipulando la plataforma móvil condensados en un solo algoritmo. En conclusión, se puede observar que mediante la integración de varios dispositivos se puede realizar una programación versátil para la adquisición de datos de un entorno a mapear y el desarrollo de trayectorias en un entorno de prueba para su aplicación en un ambiente por ejemplo de trabajo industrial.

Palabras clave: Rplidar (escáner láser 2D), Python (lenguaje de programación), Odometría, Debían (sistema operativo).

RECONOCIMIENTOS

Al comenzar a redactar mis agradecimientos pensé en de todas aquellas personas que han contribuido en mi vida, tanto en mi formación personal como académica, sin embargo, nombrarlas a todas ellas me llevaría un sinfín de todo aquello que he vivido por estos días. No obstante, primero me gustaría expresar mis agradecimientos a mis padres por su formación y acompañamiento dentro de mis estudios, ya que gracias a ellos pude estar en esta linda institución, han dado todo el esfuerzo para que yo ahora esté culminando esta etapa de mi vida, les agradezco porque fueron ellos en últimas los que estuvieron en los días más difíciles de mi vida como estudiante.

Quiero agradecer a todos mis maestros debido a que fueron ellos los que me enseñaron a valorar los estudios y a superarme cada día, pero en especial a aquellos profesores que fueron diferentes a lo largo de mi desarrollo porque fueron ellos quienes me hicieron ver el mundo desde otro punto de vista, más crítico y mucho más profesional, pues sus discusiones y enseñanzas redundaron benéficamente tanto a nivel profesional como personal.

Doy mi gratificación al grupo de investigación de automática y robótica del ITM por su colaboración y por todos los insumos necesarios para hacer posible este trabajo, quiero expresar también mi más sincero agradecimiento al docente Juan Sebastián Botero por permitirme estar bajo su dirección, su apoyo y su capacidad para guiar mis ideas ha sido un aporte invaluable en este trabajo.

Finalmente le agradezco a todos los que me han ha apoyado durante toda mi carrera profesional, todos aquellos que estuvieron en mis momentos ya sean de alegría o tristeza, nunca me dejaron solo en el arduo deber de aprendizaje.

ACRÓNIMOS

DSP Digital signal processor

ARM Advanced risc machine

ROS Robotics open source

OI Open Interface

SLAM Simultaneous Localization and Mapping

TABLA DE CONTENIDO

1. INTRODUCCIÓN

En la actualidad, la robótica móvil está evolucionando para intentar solucionar problemáticas en algunos tipos de entorno volviendo más eficiente labores domésticas o industriales que conlleven tareas repetitivas en el entorno, algunas de estas tareas necesitan de un previo reconocimiento del ambiente físico para poder darle a un mecanismo autónomo la capacidad de realizar labores programadas. De ahí que, para que un mecanismo reconozca el ambiente, éste debe de poseer sensores que le indiquen tanto la posición en que se encuentra como las proximidades de objetos o paredes, pues dicen Ríos, Bueno, & Martínez que "el desarrollo sensorial es una de las partes más importantes dentro de la navegación de un robot móvil" (Ríos, Bueno, & Martínez, 2009, pág 1). Un caso específico en la industria moderna, es el de robots móviles utilizados para llevar cargas pesadas a lugares concretos navegando de forma autónoma, un prototipo actualmente conocido es el realizado por una empresa que hace robots móviles llamada Kiva Systems, algunos de los robots allí fabricados son hechos para navegar por el piso de bodegas guiándose por etiquetas magnéticas para buscar objetos que necesitan ser empacados y llevados a trabajadores, éstos reciben instrucciones inalámbricas de un ordenador central, quien hace que los robots no choquen unos con otros. (Ortigoza, Cruz, y Sánchez, 2015,pág 1), en este proyecto se desea indagar en la programación que conllevan este tipo de mecanismos autónomos teniendo presente una mayor autonomía esperando integrar un algoritmo que tenga un tipo de procesador propio y que además no siga un camino, más bien que reconozca su entorno para navegar libremente.

Así pues, se pretende implementar algoritmos en un dispositivo ARM, en este caso se usará una Odroid para la adquisición de datos de un escáner láser (Lidar) con el fin de reconstruir un entorno de prueba para enviar órdenes a una plataforma móvil, por lo cual se utilizará un robot móvil inteligente, el robot Irobot Create 2 para elegir un tipo de trayectorias

adecuada para la navegación del mecanismo. De esta manera, se busca primero realizar un algoritmo en Python capaz de realizar la adquisición de datos del escáner láser, segundo hacer un algoritmo para enviar comandos a través de comunicación serial para controlar la plataforma robótica, y finalmente integrar ambos algoritmos con el fin de incorporar todos los dispositivos mencionados para crear ese mecanismo de prueba que comprende el proyecto.

En este documento se pueden encontrar en la sesión 2 toda la teoría pertinente para el desarrollo de un dispositivo móvil autónomo, de esta manera en el 2.1 se introducirá en tipo de escáner láser que fue utilizado, allí se mostrarán los principios de funcionamiento de los que éste se vale para poder escanear y mandar los datos. Posteriormente en 2.2 se da una breve explicación y se muestran las características del dispositivo ARM que fue utilizado la Odroid, el cual se utilizó por su gran capacidad de procesamiento. En la sesión 2.3 se identifica el funcionamiento y la conexión de la plataforma móvil Irobot Create 2, un mecanismo que ya posee una estructura mecánica con un controlador para su movimiento, hecho de tres ruedas con dos encoder para las ruedas de manejo y en la sesión 2.4 se muestra el objetivo del proyecto, se identifica un esquema de como irá estructurado el mecanismo que se va a utilizar en la programación de los algoritmos. Luego en la sesión 3 se expone la metodología utilizada para el desarrollo del trabajo, pues en la sesión 3.1 se evidencia la arquitectura que comprende el prototipo, los programas y librerías necesarias, como los dispositivos que se usaron alrededor del desarrollo; en la 3.1.1 se detalla toda la instalación del ROS. En la sesión 3.2, se presentan la construcción de los códigos, primero en la 3.2.1 el algoritmo para la adquisición de datos del escáner láser, además de su conexión y segundo en la 3.2.2 los algoritmos de manejo de la plataforma móvil, además de sus principios de conexión. Finalmente, en la sesión 3.2.3 se realiza la integración de ambos códigos en uno sólo que permitirá todo el proceso del mecanismo final.

2. MARCO TEÓRICO

Para la creación de un robot móvil autónomo se requiere de una serie de componentes que hagan posible la navegación en entornos sin o con una pequeña intervención humana, pues debe ser indispensable que éste contenga sensores para reconocer el medio, asimismo de poseer un mecanismo capaz de recibir órdenes con el objetivo de moverse libremente para ir a un lugar determinado, no obstante para que ambos dispositivos hagan posible una navegación integrando sensores con el mecanismo actuador, es necesario utilizar un tipo de controlador que incorpore la adquisición de datos entregadas por el sensor, tanto como los datos para dar órdenes al mecanismo que finalmente es el que actúa.

Pero la integración de todos estos elementos conlleva a un propósito, el de proyectar el mecanismo mediante trayectorias a lugares específicos, la odometría se encarga de esto, pues es el estudio de la estimación de la posición de un robot móvil en el espacio. Ahora bien, dice Aníbal & González que para que un "robot autónomo pueda satisfactoriamente afrontar tareas como generar trayectorias, evitar obstáculos, monitorizar la ejecución, etc. Se requiere que éste sea capaz de determinar su localización (posición y orientación) con respecto a un sistema de referencia absoluto" (Aníbal & González, 1993, pág 3). Por esta razón se hace pertinente tener en el mecanismo actuador un tipo de sensor en las ruedas que mida el recorrido que ha tenido cada llanta desde el inicio del proceso para ubicarlo, pues a veces esa fricción de las ruedas es un factor para tener en cuenta dentro de la odometría, en donde se estudia la posición de un robot móvil en un entorno (Thum, Bichuniak & Börcsök, 2014, pág 2). Por otro lado, lo más importante para reconocer el entorno es realizar un mapeo del espacio, para este fin se puede utilizar un escáner láser de dos dimensiones, conocido también como láser radar o Lidar, es un tipo de sensor activo, el cual mide la distancia entre el sensor y la superficie tocada en un rango de 180 grados o 360 grados (Castejón, Blanco, López & Moreno, 2004, pág 2).

2.1 Escáner láser RPlidar

De modo que se utilizará un escáner láser (Lidar) para el reconocimiento del entorno con el fin de que más adelante se pueda realizar un mapeo del ambiente con los datos adquiridos, se utilizará un Rplidar, el cual es un escáner que posee un rango de 6 metros de alcance con una libertad de 360 grados, éste entrega datos de una nube de puntos 2D que puede ser utilizada en el mapeo de un entorno, su frecuencia de exploración es de 5,5 Hz, aunque puede ser configurada a 10 Hz.

Fig. 1. Rplidar, escáner láser fabricado por Robo Peak. ("Rplidar development kit user manual", 2016)

Su sistema de medición está basado en un principio de triangulación láser, en donde el Lidar emite una señal láser infrarroja modulada con el fin de que la señal sea reflejada por el objeto detectado. La señal que devuelve es muestreada por un sistema de adquisición de visión, posteriormente el DSP del Lidar comienza el procesamiento de los datos de muestreo y los valores de distancia y ángulo entre el objeto y éste mediante una interfaz de comunicación.

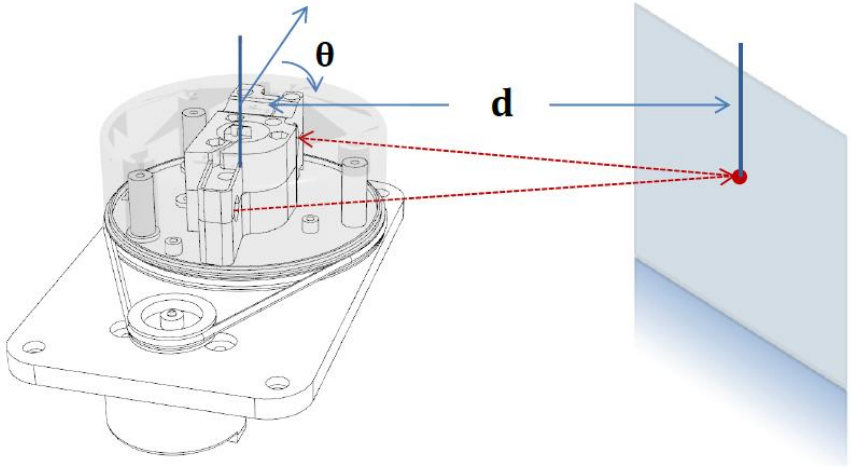

Fig. 2. Triangulación láser del Lidar. ("Rplidar introduction and datasheet", 2016).

Cuando el sistema Lidar trabaja, los datos de muestreo salen por medio de una interfaz de comunicación, manda datos continuamente, cada punto de muestreo contiene debajo información, los datos de envío o de respuesta están dados o se envían en paquetes de datos hexadecimales. La siguiente tabla muestra cada uno de los datos que son enviados como una respuesta del escáner:

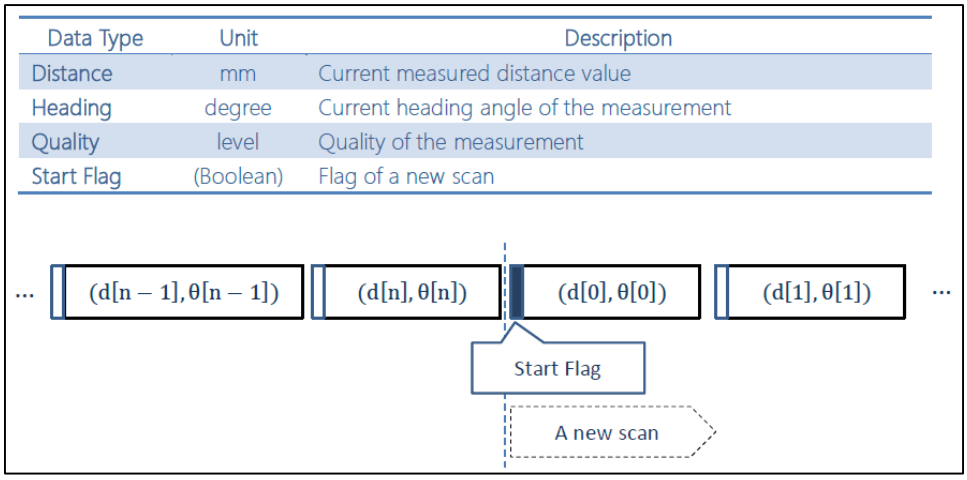

Fig. 3. Respuesta dada por el Lidar. ("Rplidar introduction and datasheet", 2016).

El Lidar tiene 4 estados, *ldle, escaneando, procesando petición* y *paro de protección.* El estado *ldle* es el estado por defecto con el cual iniciará automáticamente el LIdar, luego del encendido o reseteo de éste, el diodo láser y el sistema de medida están deshabilitados. Una vez el Lidar entra en estado *escaneando,* tanto el diodo láser como el sistema de medida son habilitados y de esta manera iniciará a medir distancia y a enviar datos continuamente, además el Lidar siempre comprobará el estado de rotación del motor. Asimismo, el Lidar entra en estado *procesando petición* una vez recibe el paquete de petición (más adelante se detalla la forma en que se envía la petición) desde el sistema Host, durante este estado el Lidar no realizará la operación de escaneo y tampoco enviará datos, sólo enviará datos de respuesta cuando la operación necesaria haya finalizado. Una vez que el Lidar detecte algo incorrecto con el Hardware, frenará su operación y posteriormente entrará en estado de *paro de protección.*

2.2 Dispositivo ODROID-XU3

Un dispositivo Odroid fue utilizado como controlador, ya que es una plataforma abierta de hardware, una nueva generación de dispositivos informáticos que contiene una variedad de placas de desarrollo de tamaño reducido y gran potencia, la placa puede ejecutar varias versiones de Linux, incluyendo la versión de Ubuntu 14.04 y el Android 4.4. Mediante la adopción de e-MMC 5.0 y USB 3.0. La Odroid cuenta con 2 Gbytes de memoria RAM con una velocidad de reloj de 933 MHz y cuatro núcleos con los cuales su velocidad de transferencia de datos es rápida, una característica que se requiere cada vez más para apoyar el poder de procesamiento de avanzada en dispositivos ARM que permite que los usuarios puedan disfrutar de una mejora en la computación, tales como arranque más rápido, navegación web entre otras. ("ODROID | Hardkernel", 2016).

Fig. 4. Dispositivo Odroid XU3. ("ODROID | Hardkernel", 2016)

2.3 Plataforma móvil Irobot Create 2

El iRobot Create 2 es una plataforma móvil completamente programable, con ésta el mecanismo tendrá una buena libertad para ir en trayectorias planeadas, además de una lectura muy precisa de los enconders que posee para verificar su recorrido. La plataforma pesa 8 Lb, mide 13 pulgadas de diámetro y 3,5 de altura, posee un puerto serial al lado superior por donde es posible conectar la plataforma con un sistema Host para el control del mecanismo, aparte de la programación que éste tiene por defecto. Posee dos ruedas con encoders en la parte trasera y una rueda loca en la parte delantera, además de una cantidad de sensores para choque, los cuales tienen como función activar los motores en sentido contrario cuando éste rose un obstáculo, por otra parte, está equipado con sensores en la parte delantera para evitar caídas cuando no se detecta suelo.

Fig. 5. Plataforma Irobot Create 2. ("iRobot Create2 Open Interface (OI) Specification based on the iRobot Roomba 600", 2016)

Para la comunicación serial se utiliza un conector que trae la plataforma llamado conector Mini-DIN, la Open interface (OI) es la interfaz que trae la plataforma para realizar dicha conexión, ésta proporciona una interfaz mediante la cual se pueden enviar un conjunto de comandos para el control de sus motores y sensores tanto como de sus sonidos, para llevar a cabo diversas acciones.

La Open Interface (OI) tiene cuatro modos de funcionamiento: *Apagado, pasivo, seguro,* y *completo*; por defecto, el OI está en *modo apagado*. El modo pasivo permite que los datos de los sensores que se solicitarán a través del puerto serial estén en funcionamiento, pero no hay comandos que puedan enviarse para cambiar los parámetros del actuador, como la velocidad de la rueda. Por otra parte, el *modo seguro* proporciona un control total del robot a través del puerto serial con unas pocas excepciones, en este modo los sensores de detección de caída y rueda hacia adelante están activos, lo que significa que el robot se detendrá en situaciones en donde esos sensores se activen, sin tener en cuenta el comando que se le haya enviado anteriormente, el robot no se cargará mientras esté en este modo. Si alguna de las excepciones mencionadas anteriormente se produce mientras esté en el modo seguro, el robot vuelve al *modo pasivo*. El último modo de funcionamiento es el *modo completo*, éste funciona de manera similar al *modo seguro*, pero con las excepciones citadas anteriormente.

Para utilizar la OI, debe ser conectado a un sistema Host quien mandará las órdenes al conector mini-DIN externo sobre la plataforma Create 2, este conector proporciona una comunicación bidireccional en serie en TTL (0 - 5 V). Además, el conector también proporciona una conexión directa no regulado a la batería de la plataforma Create 2, que se puede utilizar para alimentar las aplicaciones de OI, el conector mini-DIN se encuentra en la parte posterior derecha de la plataforma Create 2, debajo de un protector de plástico complemento de distancia. El puerto de comunicación se puede observar en la siguiente imagen:

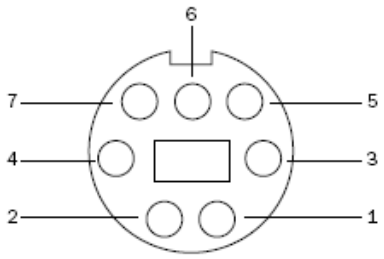

Fig. 6. Asignación de los pines de la entrada del Create 2. ("iRobot Create2 Open Interface (OI) Specification based on the iRobot Roomba 600", 2016).

Los pines 1 y 2 están conectados a la batería de la plataforma a través de un fusible reajustable de 200 mA, el pin 3 es el RXD encargado de recibir datos y el pin 4 es el TXD es el encargado de transmitir los datos; dado que los pasadores RXD y TXD utilizan 0 - 5V y el PC puertos serie utilizan diferentes voltajes (RS-232 niveles), es necesario cambiar los niveles de voltaje utilizando el cable USB del iRobot Create, el cual está hecho para esta conversación. Por defecto la plataforma Create se comunica a una velocidad de 115200 Baudios, por el pin 5 se puede modificar dicha velocidad a 19200 Budios y por último los pines 6 y 7 son conectadas a la tierra de la batería del Create.

2.4 Integración de los dispositivos

Al integrar los dispositivos anteriormente mencionados se identifica el rol que cada dispositivo va a cumplir para desarrollar la tarea que tiene como objetivo este trabajo, así pues, los dos dispositivos, tanto el Lidar como la plataforma se comunicarán con la Odroid con una comunicación bidireccional. El lidar es el encargado de la adquisición de datos para reconstruir el mapa de un entorno, éste recibe peticiones para una tarea y mandará los paquetes de datos a la Odroid como se muestra en el esquema. Posteriormente la plataforma Create recibirá los datos para su control desde la Odroid, además enviará la respuesta que va teniendo de sus enconders. El dispositivo más importante es el procesador, pues la Odroid posee un sistema operativo el cual procesará los datos dentro

de un software utilizando los algoritmos creados para llevar el control de ambos dispositivos.

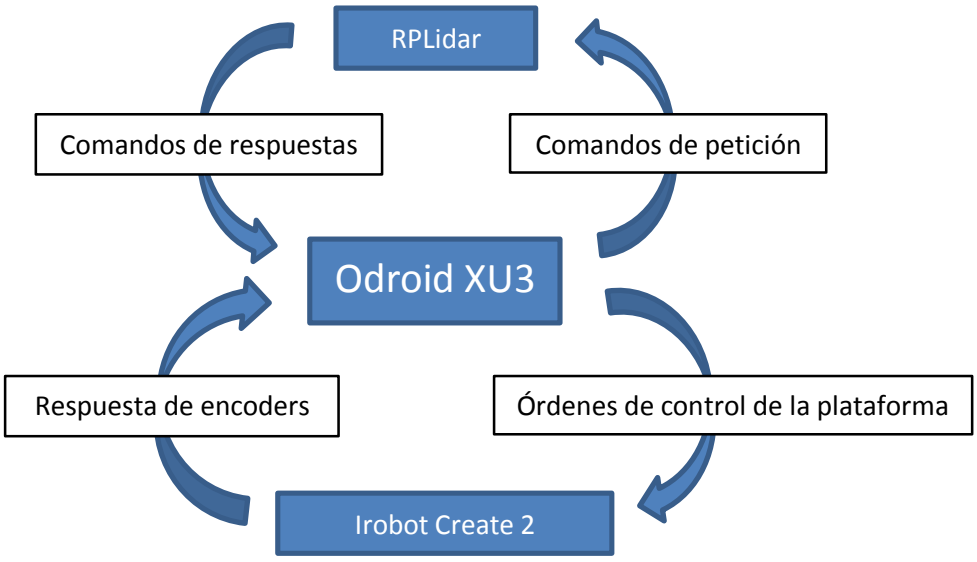

Fig. 7. Esquema del funcionamiento integrado del prototipo final.

3. METODOLOGÍA

3.1 Arquitectura del prototipo

El prototipo de prueba comprende tres dispositivos, la Odroid, la plataforma Irobot Create y el escáner láser Rplidar[1]. Cabe agregar que al comienzo en vez de una Odroid se intentó trabajar con un dispositivo Raspberry PI 2, pero no fue posible debido a que al momento de instalar un paquete de datos del ROS la Raspberry no mostraba un buen rendimiento por falta de capacidad de procesamiento.

Inicialmente se instaló un sistema operativo Linux llamado Ubuntu 14.04.01 para una arquitectura ARM (Apéndice A), para ello primero se formateó el dispositivo de almacenamiento, en este caso se utilizó una memoria micro SD clase 4, luego de formatearla se instaló mediante otro computador el sistema operativo anteriormente mencionado para posteriormente instalar dentro de éste, los software que se utilizarán para realizar los algoritmos, en este caso se programó en Python 2.7 utilizando Spyder. Para la comunicación serial con la plataforma y el Lidar se utilizó un paquete de librerías llamado Pyserial, fue necesario precisar el puerto en el que irá conectado cada dispositivo dentro del código realizado, la dirección de la raíz en el sistema operativo por defecto es "*/dev/ttyUSB0*", por lo cual para el Lidar la dirección será la anteriormente mencionada, mientras que para la plataforma Create será "*/dev/ttyUSB1*".

En cuanto a la conexión de la Odroid con una pantalla para poder visualizar todo, se utilizó una pantalla de computador, para esto es necesario utilizar un conversor de Display Port a VGA o un conversor HDMI a DVI, con la pantalla que se trabajó el conversor utilizado fue el conversor de HDMI a DVI junto con un pequeño conversor de HDMI a micro HDMI, el cual es la entrada original del dispositivo Odroid.

[1] Todos los implementos fueron proporcionados por el grupo de investigación de Automática, Electrónica y Ciencias Computacionales de parque I en el ITM.

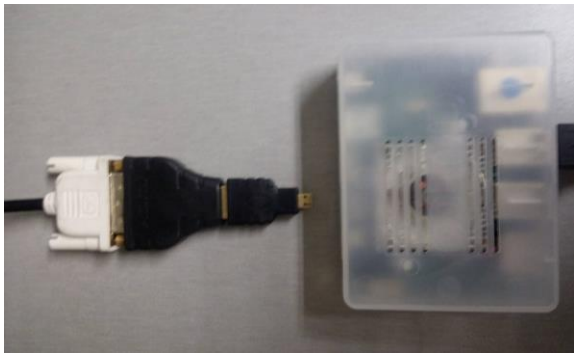

Fig. 7. Conversores para la conexión de la Odroid. (Fuente: Juan David Marín)

Además, para posteriores trabajos se instalaron los paquetes de librerías del ROS propias de Linux pertinentes para poder realizar mapeos con el Lidar mediante una técnica utilizada llamada Slam. (Apéndice B)

3.2 Algoritmos

Como se dijo anteriormente para la elaboración de estos algoritmos se utilizó Python 2.7, ya que es un lenguaje de alto nivel de propósito general, que tiene como objetivo ser muy fácil de leer y más fácil de usar que otros lenguajes de alto nivel como C ++ y Java. Es un lenguaje interpretado, es decir, que puede ejecutar instrucciones directamente, ya que son del tipo línea por línea. La comunicación serial es comúnmente utilizada para enviar datos de un lugar a otro ya sea por bus o por USB cómo se usa en esta aplicación. En la comunicación serial, como su nombre indica, los bits son enviados secuencialmente uno a la vez utilizando las librerías de Pyserial que posee Python.

3.2.1 Algoritmo de adquisición de datos

Para realizar la adquisición de datos se procedió a analizar la forma en cómo trabaja el Lidar, éste utiliza un protocolo de comunicación basado en paquetes para transmitir datos entre el sistema host y el núcleo del Lidar. Con el fin de recibir los resultados del escaneo y el control del Lidar, el sistema Host requiere comunicarse con el Lidar mediante la interfaz TTL UART proporcionada por el núcleo del sistema del Lidar.

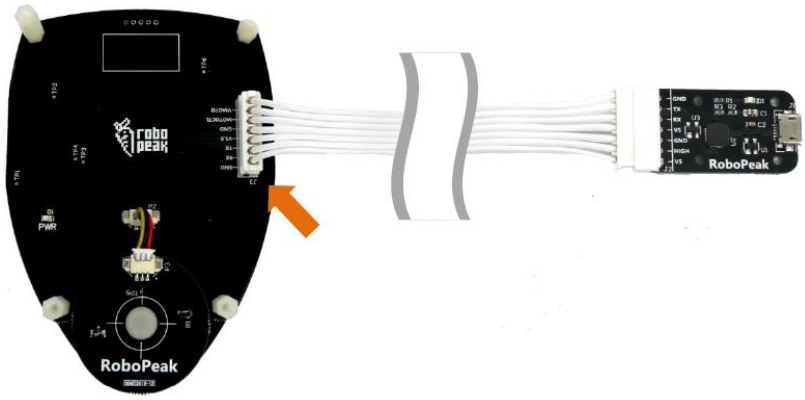

Fig. 8. Conexión del Lidar. ("Rplidar development kit user manual", 2016)

Se inició conectando el Lidar mediante un adaptador USB que el fabricante proporciona para que éste en últimas pueda enviar y recibir los datos al sistema Host, este adaptador alumbrará cuando el Lidar esté escaneando.

Ahora bien, para enviar los paquetes se debe tener en cuenta que al trasmitirlos en la interfaz deben ser enviados en un formato de números binarios, además de que éstos deben compartir un formato de paquetes uniforme. Se debe enviar un paquete de datos del sistema Host al Lidar, llamamos a este paquete *petición*, con la cual el Lidar recibirá dicha orden con la cual ejecutará una *petición* y la enviará al sistema Host. El lidar sólo iniciará si ha recibido una *petición* de parte del sistema Host, de esta manera el Lidar responderá, enviando uno o más paquetes de *respuestas,* esto se puede observar en la siguiente figura.

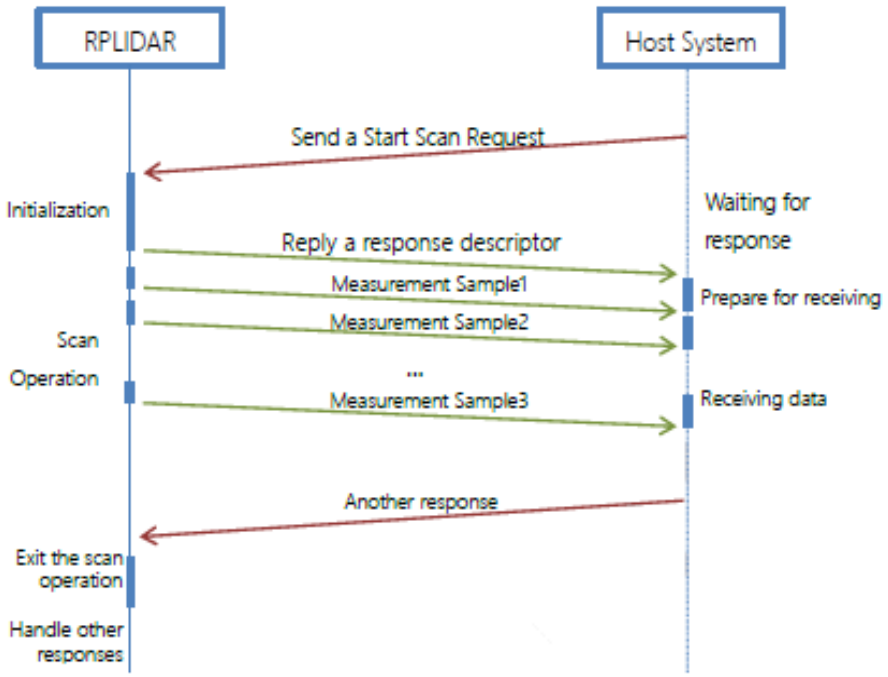

Fig. 9. Comunicación con el Lidar. ("Rplidar protocol interface and application notes", 2016).

Todos los paquetes comparten el siguiente formato y deben ser enviados en nomenclatura hexadecimal, tienen un orden de escritura a la hora de mandarlos para que el Lidar los interprete como una petición, en la siguiente figura se muestra como es la orden de los datos hexadecimales a enviar.

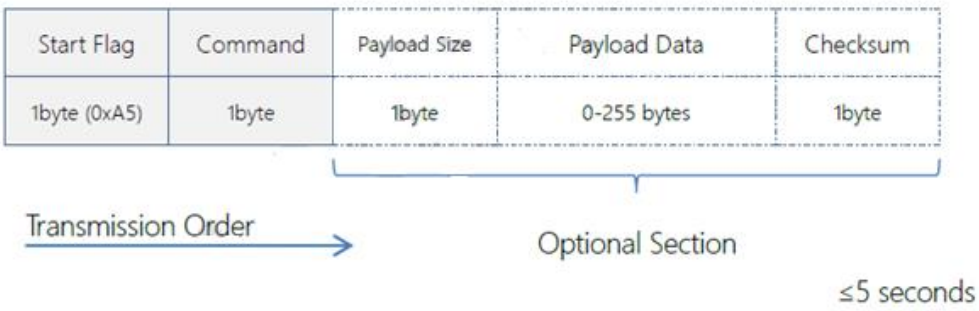

Fig. 10. Envío de paquetes al Lidar. ("Rplidar protocol interface and application notes", 2016).

En consecuencia, es necesario enviar un byte inicial, 0xA5 es un número hexadecimal fijo al inicio de cada envío de paquetes, pues el Lidar usa ese byte como la identificación de una nueva *petición.* Asimismo, seguido de la acción a ejecutar, el paquete que debe ser enviado

para que el Lidar empiece a escanear es el de **Start Scan**, para este se envía 0xA5 0x20, con el cual el Lidar empezará a escanear y a mandar paquetes de datos que contienen la distancia, el ángulo y la calidad láser de cada punto. Para poder salir del envío de esos paquetes de datos de respuesta, se envía un paquete de datos llamado **Stop**, para ello se envía 0xA5 0x25, inmediatamente el diodo láser y el sistema de medición serán deshabilitados y el lidar entraría en el estado Idle. Por otro lado, para reiniciar el núcleo del Lidar se envía **Reset** que corresponde al envío de 0xA5 0x40, un reseteo hará que el Lidar se revierta a un estado similar cuando recién se enciende. Para leer la información del Lidar se utiliza 0xA5 0x50, inmediatamente responderá con un paquete de datos de la información acerca de él, su número serial, su versión de firmware/hardware. Por otra parte, para verificar el estado del Lidar se envía el comando 0xA5 0x52, se devolverá un paquete en donde se muestra el estado del Lidar en tres posibles resultados *good, Warning* y *Error*.

El formato del descriptor de respuesta es representado así:

Bandera de inicio 1	***Bandera de inicio 2***	***Longitud de respuesta***	***Modo de envío***	***Tipo de información***
1 byte 0xA5	*1 byte 0x5A*	*30 bits*	*2 bits*	*1 byte*

Fig. 11. Descriptor de respuesta. (Fuente: Stid Rodríguez)

Así que, el descriptor de respuesta usa dos bytes de patrones de datos, por ejemplo, 0xA5 0x5A, para que el sistema Host identifique el inicio de respuestas del Lidar, los datos de respuesta en el campo de 30 bits son el tamaño de un solo paquete de respuesta de datos de entrada en bytes. Los dos siguientes campos describen la petición/respuesta de la sesión actual.

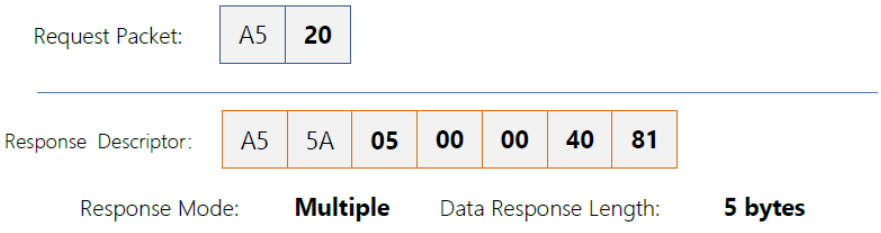

Fig. 12. Ejemplo de envío de paquetes y respuesta de datos. ("Rplidar protocol interface and application notes", 2016).

Cada resultado de medidas será enviado usando un paquete de respuestas individual, la respuesta relacionada del descriptor será enviada inmediatamente una vez que se envíe la petición y sea aceptada. Los formatos de los datos de respuesta son así:

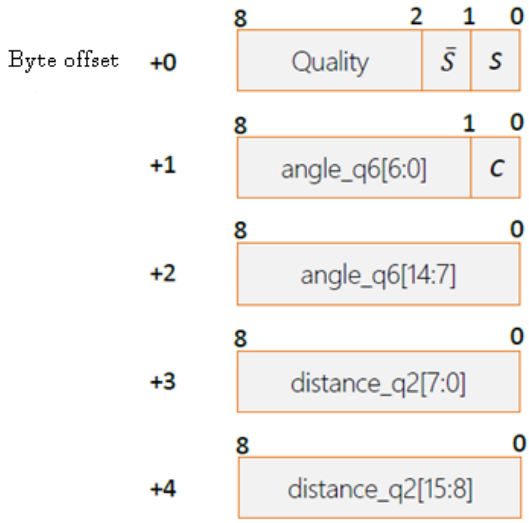

Fig. 13. Bytes entregados en el paquete de datos de resultado. ("Rplidar protocol interface and application notes", 2016).

El descriptor de respuesta entrega los datos de calidad del rayo láser, el ángulo y por último la distancia a la que llego el rayo.

De modo que el paso a seguir fue utilizar el puerto serial para comunicar el dispositivo Odroid con el Lidar, mediante la interfaz de comunicación USB UART con la ayuda del adaptador USB del Lidar. Para ello se instaló un paquete de librerías Pyserial las cuales son librerías realizadas para manipular el puerto serial, desarrolladas para el lenguaje de programación de Python 2.7, este último ya viene instalado en el sistema operativo. De esta manera sólo quedó programar el envío de peticiones anteriormente mencionado para recibir dentro de una variable los datos de respuesta del Lidar.

La primer función a ejecutar cuando se conecta el lidar es la función para enviar una petición para que el dispositivo pueda responder el Byte que contiene los datos.

```
def _send_cmd(self, cmd):
    '''Envía 'cmd' como comando al sensor'''
    req = SYNC_BYTE + cmd
    self._serial_port.write(req)
    self.logger.debug('Command sent: %s' % req)
```

Fig. 14. Función para el envío de peticiones.

Posteriormente a la hora de leer los datos que envía el Lidar se crean dos funciones, una para leer el descriptor,así mismo como otra interpretar la respuesta del byte de datos.

```
def _read_descriptor(self):
    '''Lee el paquete del descriptor'''
    descriptor = self._serial_port.read(DESCRIPTOR_LEN)
    self.logger.debug('Recieved descriptor: %s', descriptor)
    if len(descriptor) != DESCRIPTOR_LEN:
        raise RPLidarException('Descriptor length mismatch')
    elif not descriptor.startswith(SYNC_BYTE + SYNC_BYTE2):
        raise RPLidarException('Incorrect descriptor starting bytes')
    is_single = _b2i(descriptor[-2]) == 0
    return _b2i(descriptor[2]), is_single, _b2i(descriptor[-1])

def _read_response(self, dsize):
    '''Lee el paquete respesta con la longitud 'dzise' en bytes'''
    self.logger.debug('Trying to read response: %d bytes', dsize)
    data = self._serial_port.read(dsize)
    self.logger.debug('Recieved data: %s', data)
    if len(data) != dsize:
        raise RPLidarException('Wrong body size')
    return data
```

Fig. 15. Funciones para la lectura del descriptor y lectura de la respuesta.

Puesto que los datos irán incrementado uno por uno en cada iteracción, fue necesario crear en el algoritmo una función con un vector infinito en donde se vayan almacenando cada uno de los datos.

```
def iter_scans(self, min_len=5, force=False):
    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. 16. Función en donde construir el almacenamiento de las iteraciones de escaneo.

Ya que se almacenaron los datos, el paso a seguir fue construir apartir de éstos los datos a entregar, es decir, tanto la calidad de la medida, el ángulo y la distancia de cada punto medido según los datos almacenados.

```
def _process_scan(raw):
    '''Procesa la entrada 'raw' (informacion bruta) y la convierte
        en: Calidad, angulo 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. 17. Función para la conversión de los datos que entrega el Lidar.

Esta función anterior es importantísima porque es la que genera los datos finales que se necesitan para condensar una cantidad de datos necesarios para construir un entorno de mapeo. Primero el código interpreta el paquete de Bits que muestran la calidad del rayo, la calidad es la fuerza de pulso con que se refleja el rayo. Prosiguiendo, el segundo dato es el paquete de bits que contiene el ángulo el cual como punto de referencia muestra la dirección hacía donde el rayo reflecto un punto, para poder leer el dato que entrega el Lidar, se tiene que hacer una conversión, dividiendo el ángulo actual de la medida sobre 64 grados, Así se obtiene el dato real del ángulo que midió en un punto en unidades de 0 a 360 grados almacenando un número de puntos fijos. Por último, hay que interpretar el paquete de bits en donde se almacena la distancia, ésta dirá en donde el rayo se reflecto en un punto, para obtener el dato en milímetros se toma la distancia actual y se divide sobre 4 mm, de esta manera se adquirirá la medida en milímetros.

3.2.2 Algoritmo para el control de la plataforma

Esta plataforma está diseñada para realizar tareas de manera autónoma, pero también mediante la interfaz IO por comunicación serial puede ser controlada mediante comandos de órdenes, el fabricante ofrece un algoritmo hecho en Python 2.4 para mover la

plataforma de forma manual con un menú desde el computador con las flechas del teclado, de esta manera se utilizó dicha estructura del código para ser modificado y utilizarla a gusto con el fin de mover el dispositivo, además se incorporó el comando para mandar la petición de lectura de los encoders para la verificación de recorridos.

La OI proporciona al programador una multitud de comandos que se pueden enviar a la plataforma móvil en forma de cadenas de comandos por corchetes para ordenar un tipo de tarea según el tipo de comando enviado, por lo tanto, para controlar la plataforma móvil se enviará el tipo de comando a ejecutar, sea para mover los motores de las ruedas o para enviar la petición de lectura de los encoders.

Para los poner la plataforma Create en un modo de los 4 que posee, manualmente se puede teclear una letra según las propuestas en el siguiente código, para el modo *pasivo* se envía el comando 128, para el modo *seguro* se envía el comando 131, para el modo *completo* se manda el comando 132, el comando 135 se envía para que inicié como normalmente hace una labor de limpieza, los demás comandos realizan otros tipos de operaciones en el Create, pero no se ahondarán en este trabajo.

```
if event.type == '2': # KeyPress; need to figure out how to get constant
    if k == 'P':   # Passive
        self.sendCommandASCII('128')
    elif k == 'S': # Safe
        self.sendCommandASCII('131')
    elif k == 'F': # Full
        self.sendCommandASCII('132')
    elif k == 'C': # Clean
        self.sendCommandASCII('135')
    elif k == 'D': # Dock
        self.sendCommandASCII('143')
    elif k == 'SPACE': # Beep
        self.sendCommandASCII('140 3 1 64 16 141 3')
    elif k == 'R': # Reset
        self.sendCommandASCII('7')
```

Fig. 18. Menú para el envío de comandos para cada estado de la plataforma Create.

Para poder mover el Create con las flechas del teclado, se creó una condición para cada caso, para ir adelante se enviará un comando para que ambos motores giren en el mismo sentido generando movimiento hacia adelante. Por otra parte, para que gire en un sentido el comando enviado debe poner en un sentido un motor y el otro en el sentido contrario para gire teniendo presente para que lado va a voltear.

```
elif k == 'UP':
    self.callbackKeyUp = True
    motionChange = True
elif k == 'DOWN':
    self.callbackKeyDown = True
    motionChange = True
elif k == 'LEFT':
    self.callbackKeyLeft = True
    motionChange = True
elif k == 'RIGHT':
    self.callbackKeyRight = True
    motionChange = True
```

Fig. 19. Menú para el control manual de la plataforma desde el teclado del PC.

Para realizar el movimiento, la condición a ejecutar debió hacerse de una variedad de condiciones en donde se inicializan las variables *Velocity* y *Rotation* en cero y según el comando tecleado, cada variable indicará un signo positivo o negativo dependiendo si se pretende que el mecanismo vaya para atrás o para adelante o si va a girar para algún lado. posteriormente ambas variables son sumadas y se guarda en una variable llamada *Vr* en la cual se irá almacenando el valor que se le asignará al motor derecho para su movimiento, de igual manera se restará la variable *Rotation* con *Velocity* guardando el resultado en la variable Vl en la cual se irá almacenando el valor que se le asignará al motor izquierdo.

```
if motionChange == True:
    velocity = 0
    velocity += VELOCITYCHANGE if self.callbackKeyUp is True else 0
    velocity -= VELOCITYCHANGE if self.callbackKeyDown is True else 0
    rotation = 0
    rotation += ROTATIONCHANGE if self.callbackKeyLeft is True else 0
    rotation -= ROTATIONCHANGE if self.callbackKeyRight is True else 0

    # compute left and right wheel velocities
    vr = velocity + (rotation/2)
    vl = velocity - (rotation/2)

    # create drive command
    cmd = struct.pack(">Bhh", 145, vr, vl)
    if cmd != self.callbackKeyLastDriveCommand:
        self.sendCommandRaw(cmd)
        self.callbackKeyLastDriveCommand = cmd
```

Fig. 20. Condiciones para la rotación y movimiento de la plataforma.

Lo último que se le realizó a este algoritmo fue realizar la petición de los encoders de la plataforma, de igual manera que recibir los datos que éste entrega en otra función como se puede apreciar a continuación.

```
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. 21. Función para la recepción de los datos de los encoders entregada por la plataforma Create.

3.2.3 Integración de algoritmos

Finalmente, el algoritmo realizado para la adquisición de datos y el algoritmo para mover la plataforma se unieron en uno sólo integrando las librerías necesarias para utilizar cada una de las líneas de programación realizadas, de igual forma se pusieron las direcciones de los puertos USB diferentes para evitar confusiones en cada dispositivo.

Para ambos dispositivos fue fundamental utilizar la librería Pyserial, ya que es la única capaz de realizar la conexión USB con ambos dispositivos, las librerías del Create 2 son las necesarias con las que se crea un menú para manejar la plataforma manualmente, de igual forma con los mismos comandos se podrá manejar la plataforma, pero de una forma autónoma cuando se realice al mapeo. Las librerías del Lidar son para el envío y recepción de datos, pues se debe realizar la conversión de datos hexadecimales.

```
#Librerias para el create 2
from Tkinter import *
import tkMessageBox
import tkSimpleDialog
import struct
import sys, glob # for listing serial ports
try:
    import serial
except ImportError:
    tkMessageBox.showerror('Import error', 'Please install pyserial.')
    raise

#Librerias para el Lidar
import logging
import sys
import time
import codecs
```

Fig. 22. Librerías utilizadas en el desarrollo.

Luego, las dos funciones que se encargarán de enviar los comandos al Create. La primera función, sendCommandASCII, lleva en una cadena de comandos ASCII, lo divide y luego se llama a la segunda función, envía CommandRaw a través de la conexión serial. sendCommandRaw toma en forma de RAW un comando y lo envía al robot antes de

imprimir el programa en la ventana de Python, está imprimirá en pantalla si existe o no conexión.

```
def sendCommandASCII(self, command):
    cmd = ""
    for v in command.split():
        cmd += chr(int(v))

    self.sendCommandRaw(cmd)

# sendCommandRaw toma una strin interpretada como un arreglo de bytes
def sendCommandRaw(self, command):
    global connection

    try:
        if connection is not None:
            connection.write(command)
        else:
            tkMessageBox.showerror('Not connected!', 'Not connected to a robot!')
            print "Not connected."
    except serial.SerialException:
        print "Lost connection"
        tkMessageBox.showinfo('Uh-oh', "Lost connection to the robot!")
        connection = None

    print ' '.join([ str(ord(c)) for c in command ])
    self.text.insert(END, ' '.join([ str(ord(c)) for c in command ]))
    self.text.insert(END, '\n')
    self.text.see(END)
```

Fig. 23. Funciones que transforman los comandos para el envío de órdenes a la plataforma.

Para poder utilizar el escáner láser se debe conectar, con las dos anteriores funciones se conecta y se desconecta el Lidar con el sistema Host, antes de iniciar el motor y el escaneo se debe enviar una orden de conexión.

```
def connect(self):
    if self._serial_port is not None:
        self.disconnect()
    try:
        self._serial_port = serial.Serial(
            self.port, self.baudrate,
            parity=serial.PARITY_NONE, stopbits=serial.STOPBITS_ONE,
            timeout=self.timeout)
    except serial.SerialException as err:
        raise RPLidarException('Failed to connect to the sensor '
                               'due to: %s' % err)

def disconnect(self):
    if self._serial_port is None:
        return
    self._serial_port.close()
```

Fig. 24. Funciones de conexión y desconexión del Rplidar.

Para poder utilizar el escáner láser se debe conectar, con las dos anteriores funciones se conecta y se desconecta el Lidar con el sistema Host, antes de iniciar el motor y el escaneo se debe enviar una orden de conexión.

Se pensó en tener dentro del menú inicial dos opciones para el manejo de la plataforma y del Lidar, primero una opción para manejar de forma automática todo el prototipo mediante el teclado para controlar y probar todo, y segundo una opción automática para realizar el objetivo principal que es el de crear los algoritmos para que el prototipo de prueba sea autónomo, es decir que con esta opción el Lidar adquirirá los datos entregando los valores para que así la Odroid ordene los movimientos de la plataforma Create. Para esta opción se creó la siguiente función:

```
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. 25. Función para el manejo automático del prototipo.

Finalmente, el cuerpo el principal del código es el siguiente, en donde serán llamadas cada una de las funciones que se deben utilizar, además de programar el número de escaneos del Lidar.

```
if __name__ == "__main__":
    app = TetheredDriveApp()

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

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

    for i in range(2):
        sc = l1.next()
        if len(sc)>200:   #Condicion para tomar los datos
            print(sc)     #del lidar solo por un giro de 360 grados
```

Fig. 24. Función principal.

4 RESULTADOS Y DISCUSIÓN

Luego de realizar la adquisición de datos se obtuvieron satisfactoriamente los datos generados por el código realizado, el algoritmo logra interpretar los datos que entrega el Rplidar correctamente para poder ser entregados de una forma en que sea posible su interpretación numérica, tanto los datos de distancia, como ángulo y calidad del rayo, se esperaban almacenar en un bag, dentro de un bloc de notas o en Excel pero no fue posible, a continuación se pueden observar los datos como resultado de la compilación:

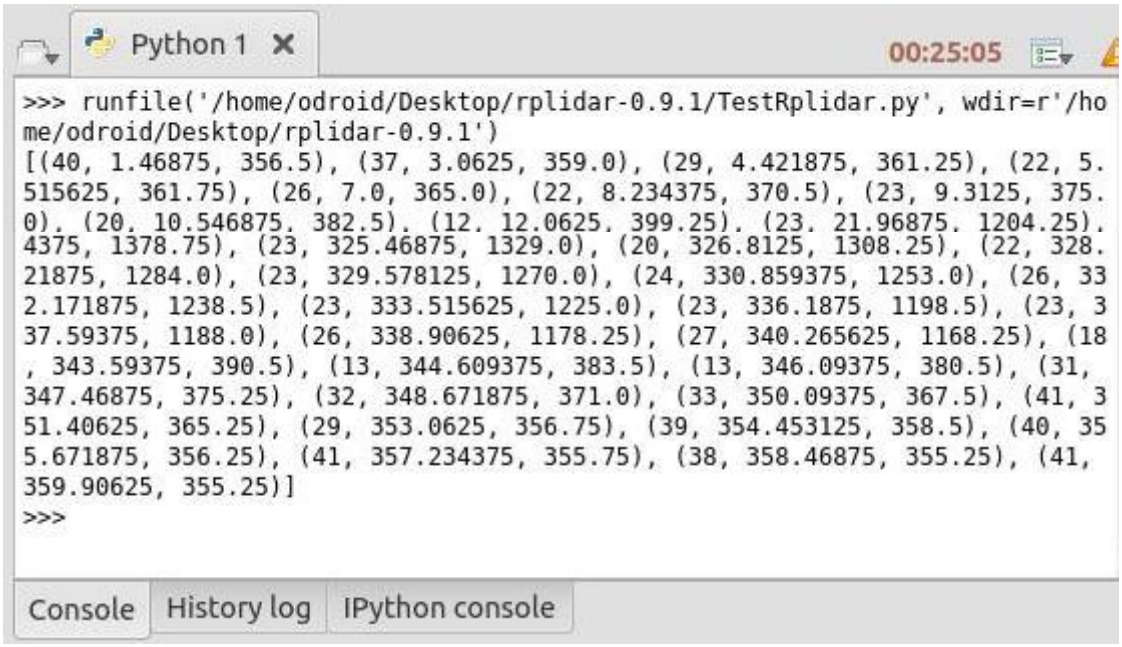

Fig. 25. Datos generados por el compilador Spyder. (Fuente: Stid Rodríguez)

Se realizaron pruebas de recorrido para verificar los datos que entregaban los encoders analizando el modo en que éstos se desempeñan, se pudo verificar que los encoders son absolutos multivueltas, éste además de la posición en una revolución, determina el número de revoluciones. Cada vez que la plataforma realiza un recorrido el encoder va acumulando una cifra comprendida entre 0 a 65535, quiere decir que son de 2 bytes los paquetes

entregados (16 bits de infrmación), cada vez que sobrepasa el valor máximo el encoder vuelve a iniciar desde cero, para tener un recorrido hecho se recomienda tener en cuenta lo anterior para realizar un conteo por cada 65535 de recorrido.

Fig. 26. Pruebas de medición del recorrido de la plataforma móvil. (Fuente: Juan David Marín)

Al imprimir el dato de lectura del encoder el compilador muestra el dato real de medida que tiene el encoder en el recorrido que ha realizado.

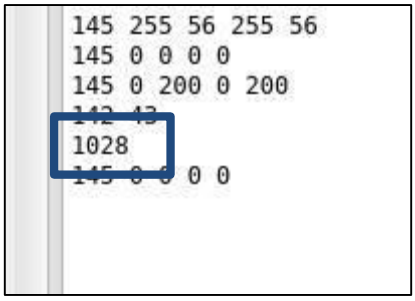

Fig. 27. Dato de medida de uno de los encoder. (Fuente: Stid Rodríguez)

Se realizó un menú con dos modos de funcionamiento, un automático y otro manual, pues con el modo manual se logra mediante teclas verificar el funcionamiento de la plataforma móvil, además de ver los datos que entrega el Lidar.

El modo automático es con el que finalmente el prototipo trabajará ya que se logró en este modo generar la integración conjunta de los dos dispositivos, el Lidar y el control de la plataforma, para posteriormente generar en trabajos futuros según el mapeo que se realice condiciones para las órdenes de movimiento del conjunto integrado.

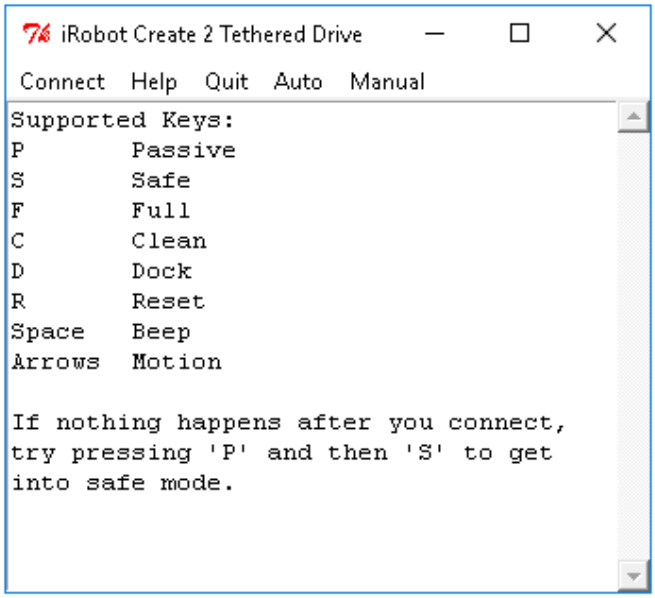

Fig. 28. Menú principal de la aplicación creada. (Fuente: Stid Rodríguez)

5 CONCLUSIONES, RECOMENDACIONES Y TRABAJO FUTURO

- Se elaboran los algoritmos pertinentes, como se había pretendido desde el inicio del trabajo, integrando ambos en un solo código robusto, para el debido procesamiento de los datos por medio del dispositivo Odroid. Hay que tener en cuenta que realizan las conexiones adecuadas con ambos dispositivos para la comunicación mediante el puerto serial.

- Se desarrolla el algoritmo de adquisición de datos, pues se comprende la forma en que trabaja el Rplidar, para posteriormente realizar las debidas peticiones que éste debe recibir para que envíe respuestas, de igual manera se interpretan aquellos paquetes de respuesta para almacenarlos, con el fin de que en futuros trabajos se realice un mapeo partiendo de estos datos.

- Ya se tenía un algoritmo para el manejo manual de la plataforma proporcionado por el fabricante, pero se modifica e incorpora a un algoritmo para el manejo de la plataforma de forma automática, además se agregan funciones como la función de los encoders para enviar la petición de lectura de éstos y el interpretador de los datos de respuesta para leer los datos que la plataforma envía.

- Luego para terminar se integran ambos algoritmos en uno solo, logrando la compilación de un solo código en Python para la adquisición de datos y el control de la plataforma.

- Finalmente se está entregando los datos en paquetes de lectura almacenados para ser utilizados en la reconstrucción del mapeo, Es decir, se realiza la adquisición de datos y el control de la plataforma el trabajo a seguir para futuros trabajos es el de simplificar

la adquisición de datos y guardarlos en un bag, para posteriormente construir un mapa a partir de éste guardar el mapa y después puedes realizar las trayectorias.

- Para la lectura de los encoders se deben programar condiciones para hacer un conteo cada vez que los encoders sobrepasen el valor máximo de sus medidas (65535), pues serán necesario para localizar el recorrido que cada uno realizó.

REFERENCIAS

- Aníbal, O. B. (2001). ROBOTICA Manipuladores y Robots Móviles. Barcelona: MARCOMBO.
- Stachniss, C. (2009). Robotic Mapping and Exploration. Freiburg, Germany: Albert-Ludwigs-University of Freiburg.
- Aníbal, O. B., & González, J. (1993). Estimación de la Posición de un Robot Móvil. Dpto. de Ingeniería de Sistemas y Automática. Málaga: Universidad de Málaga.
- Ortigoza, R. S., Cruz, M. A., & Sánchez, J. R. G. (2015). Robots móviles de ruedas : aplicaciones, (48), 34–36.
- Garc, F. J. A. (2011). Desarrollo e implantación de plataforma robótica móvil en entorno distribuido.
- Rios, L., H., Bueno M., & Martinez, L., h. (2009). Implementación de sensores extereoceptivos para una plataforma móvil utilizando microcontroladores, 41-45. Universidad tecnológica de Pereira.
- Lin, H., H., Tsai, C., C., & Chang, H., Y. (2007). Global Posture Estimation of a Tour-guide Robot using REID and Laser Scanning Measurements. Hsiuping Institute of Technology.
- Thum, R., Bichuniak, M., & Börcsök, J. (2014). Laser Scanner Based Position Detection for Safety Related Applications.
- Siadat, A., Djath, K., Dufaut M., & Husson, R. (1999). A laser based mobile robot navigation in structured environment. karlsruhe, Germany.
- Van Rossum, G. (2009). tutorial de python.
- Castejón, C., Blanco, D., López, B., Moreno, Luis. (2004). Desarrollo del sistema de percepción de una plataforma móvil para entornos exteriores.
- ODROID | Hardkernel. (2016). Hardkernel.com. Retrieved 2 August 2016, from http://www.hardkernel.com/main/products/prdt_info.php?g_code=G140448267127
- Rplidar introduction and datasheet. (2016). robotshop.com. Retrieved 2 August 2016, from https://www.robotshop.com/media/files/pdf/datasheet-rplidar.pdf

- Rplidar protocol interface 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
- Rplidar development kit user manual. (2016). robotshop.com. Retrieved 2 August 2016, from https://www.robotshop.com/media/files/pdf/user-manual-rplidar.pdf
- iRobot Create2 Open Interface (OI) Specification based on the iRobot Roomba 600. (2016). cdn-shop.adafruit.com. Retrieved 2 August 2016, from https://cdn-shop.adafruit.com/datasheets/create_2_Open_Interface_Spec.pdf
- *ODROID-XU3 Ubuntu 14.04.1 OS Image 20140814*. (2016). *Com.odroid.com*. Retrieved 8 August 2016, from http://com.odroid.com/sigong/nf_file_board/nfile_board_view.php?keyword=&tag=&bid=235
- *REP 3 -- Target Platforms (ROS.org)*. (2016). *Ros.org*. Retrieved 8 August 2016, from http://www.ros.org/reps/rep-0003.html#id9.

APÉNDICE A

Instalación del sistema operativo en la Odroid

Como disco duro del dispositivo Odroid se utilizó una tarjeta Micro SD clase 4, la instalación del sistema operativo en esta tarjeta es muy sencilla, solamente hay que seguir los siguientes pasos.

Lo primero que se debe hacer es descarga la imagen (.iso) del sistema operativo Linux deseado, en este caso se utilizó Ubuntu 14.04.01 ("ODROID-XU3 Ubuntu 14.04.1 OS Image 20140814", 2016). Encontrará todo tipo de versiones de Linux, pero es recomendable Ubuntu trusty 14.04 por su compatibilidad con ROS ("REP 3 -- Target Platforms (ROS.org)", 2016).

Para formatear la tarjeta Micro SD es necesario un software que realice dicha tarea, por tal razón se utilizó SDFormatter V4.0 un programa que formatea cualquier memoria, en este caso la clase 4 que se mencionó anteriormente.

Una vez formateada la tarjeta microSD hay que instalar la imagen (.iso). Para instalar la imagen se necesita de un software llamado Win32DiskImager, puntualmente es recomendable utilizar la versión de ODROID. Como lo que se pretende hacer es volcar la imagen de nuestro disco duro a la tarjeta externa, vamos a utilizar la operación “write” dentro del programa. Al terminar de cargar toda la barra, el proceso finalizó con éxito y ahora está listo el sistema operativo en la Odroid.

APÉNDICE B

Instalación del ROS Indigo en un dispositivo ARM

Para el correcto funcionamiento del Boost y algunas de las herramientas de ROS es necesario establecer la configuración regional del sistema, por lo cual lo primero que se hizo fue eso, con la siguiente línea:

sudo update-locale LANG = C = C IDIOMA LC_ALL = C = LC_MESSAGES POSIX

Posteriormente se configuró el dispositivo para que pudiera aceptar descargas de software desde la ruta packages.ros.org, debido a que hay recursos limitados, solo hay versiones activas para Trusty armhf (14.04). Para realizar esta configuración se ejecutó la siguiente línea:

sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu trusty main" > /etc/apt/sources.list.d/ros-latest.list'

El paso siguiente fue configurar las claves:

sudo apt-key adv --keyserver hkp: //ha.pool.sks-keyservers.net --recv-clave 0xB01FA116

Ahora bien, ya con todo listo para la instalación del ROS, se actualizó el Debian (sitema operativo), ya que siempre se debe actualizar antes de cualquier instalación:

sudo apt-get update

ROS posee una gran cantidad de librerías y herramientas, pero no todas funcionan correctamente en este tipo de dispositivos por lo que se pudó instalar una parte de estas herramientas, utilizando:

sudo apt-get install ros-indigo-ros-base

También es posible instalar cada uno de los paquetes individualmente, ya que el ROS tiene una multitud de librerías que se pueden en el sitio oficial de ROS. Para hacerlo se utiliza, package se intercambia por el nombre del paquete:

sudo apt-get install ros-indigo-PACKAGE

Antes de poder utilizar ROS es necesario instalar e inicializar ROSDEP, esta librería permite instalar fácilmente las dependencias del sistema de fuente que desea recopilar y es necesario para ejecutar algunos componentes de la base de ROS, para instalarlo se utilizó:

sudo apt-get install python-rosdep

init sudo rosdep

update rosdep

Posteriormente se realizó la configuración de entorno con la siguiente dirección, en ella se encuentra la ejecución del entorno.

echo "source /opt/ros/kinetic/setup.bash"

Finalmente se utilizó la herramienta ROSinstall, pues se utiliza con frecuencia en ROS y permite descargar varios complementos para los paquetes de ROS con un solo comando. Por ello se introdujo:

sudo apt-get install python-rosinstall

APÉNDICE C

En lo todo lo anterior se pudieron observar los algoritmos desarrollados y las conexiones utilizadas por puerto serial empleando la librería propia para ello de Python. Pero no se dio a conocer las conexiones físicas que conllevo el prototipo, a continuación, se pueden mostrar los dispositivos involucrados, pues fue así como se manipularon para lograr la integridad entre ellos para alcanzar el objetivo final.

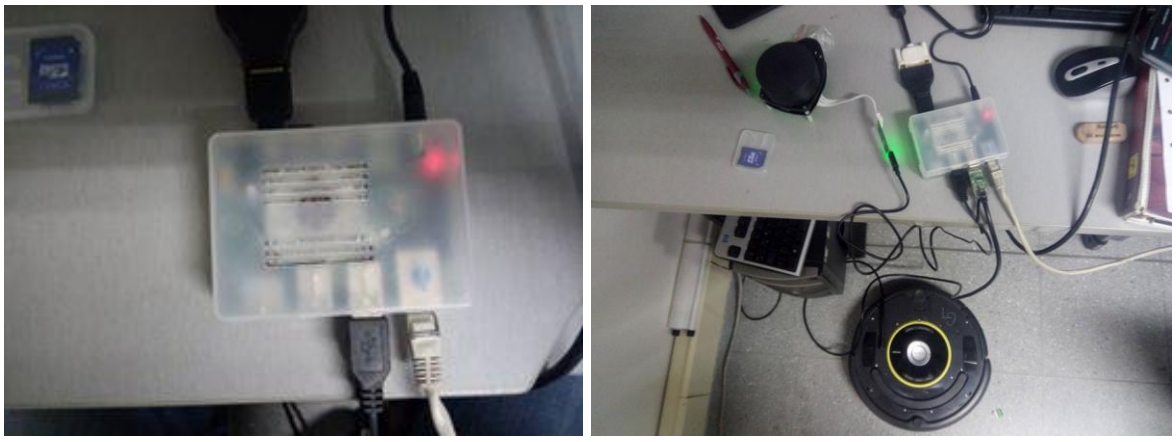

Fig. 27. Dispositivos involucrados en el trabajo. (Fuente: Juan David Marín)

FIRMA ESTUDIANTES _____

_____

_____

JUAN SE.

FIRMA ASESOR _____

FECHA ENTREGA: 9 DE AGOSTO

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: _____