UNIVERSIDAD SAN FRANCISCO DE QUITO USFQ

Colegio de Ciencias e Ingeniería

Diseño e implementación de un vehículo autónomo con navegación basada en un sensor tipo LIDAR

José Efrén Barbosa Costales
Christian Sebastián Hernández Mosquera

Ingeniería Electrónica

Trabajo de fin de carrera presentado como requisito
para la obtención del título de
Ingeniero Electrónico

Quito, 15 de diciembre de 2020

UNIVERSIDAD SAN FRANCISCO DE QUITO USFQ

Colegio de Ciencias e Ingeniería

HOJA DE CALIFICACIÓN
DE TRABAJO DE FIN DE CARRERA

Diseño e implementación de un vehículo autónomo con navegación basada en un sensor tipo LIDAR

José Efrén Barbosa Costales
Christian Sebastián Hernández Mosquera

René Játiva Espinoza, Ph.D.

Quito, 15 de diciembre de 2020

© DERECHOS DE AUTOR

Nombres y apellidos: José Efrén Barbosa Costales

Código: 00130608

Cédula de identidad: 0503384471

Lugar y fecha: Quito, 15 de diciembre de 2020

Nombres y apellidos: Christian Sebastian Hernández Mosquera

Código: 00132301

Cédula de identidad: 1720408499

Lugar y fecha: Quito, 15 de diciembre de 2020

ACLARACIÓN PARA PUBLICACIÓN

Nota: El presente trabajo, en su totalidad o cualquiera de sus partes, no debe ser considerado como una publicación, incluso a pesar de estar disponible sin restricciones a través de un repositorio institucional. Esta declaración se alinea con las prácticas y recomendaciones presentadas por el Committee on Publication Ethics COPE descritas por Barbour et al. (2017) Discussion document on best practice for issues around theses publishing, disponible en http://bit.ly/COPETheses.

UNPUBLISHED DOCUMENT

Note: The following capstone project is available through Universidad San Francisco de Quito USFQ institutional repository. Nonetheless, this project – in whole or in part – should not be considered a publication. This statement follows the recommendations presented by the Committee on Publication Ethics COPE described by Barbour et al. (2017) Discussion document on best practice for issues around theses publishing available on http://bit.ly/COPETheses.

RESUMEN

Se ha implementado un autómata terrestre cuyo sistema de navegación se basa en la reconstrucción en dos dimensiones de su entorno empleando múltiples sensores de alta precisión como: un sensor tipo Lidar, un módulo GPS, una brújula electrónica y varios sensores de proximidad. Todos los sensores y el sistema del vehículo fueron implementados en la plataforma Raspberry pi 3 B utilizando ROS. Los diferentes tipos de sensores y la interfaz de usuario permiten que el prototipo tenga una gran versatilidad para diferentes aplicaciones dentro del mundo de la ingeniería como el mapeo de zonas y reconstrucción interior a bajo costo.

Palabras clave: Vehículo autónomo, Navegación Inteligente, Sensor de alta precisión, ROS, Caracterización de sensores.

ABSTRACT

A terrestrial automaton has been implemented whose navigation system is based on the reconstruction in two dimensions of its environment using multiple high precision sensors such as: a Lidar-type sensor, a GPS module, an electronic compass and several proximity sensors. All sensors and the vehicle system were implemented on the Raspberry pi 3 B platform using ROS. The different types of sensors and the user interface allow the prototype to have great versatility for different applications within the engineering world such as zone mapping and interior reconstruction at low cost.

Key words: Autonomous vehicle, Smart Navigation, High precision sensor, ROS, Sensor characterization

TABLA DE CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

INTRODUCCIÓN

De manera general, un vehículo autónomo consiste en un dispositivo que utiliza distintos tipos de sensores para recibir información de su entorno, analizar dicha información para tomar decisiones de tipo secuencial y de control para realizar una tarea específica (Li, Díaz, Morantes, & Dorati, 2018). La navegación de manera autónoma presenta un amplio número de aplicaciones que van mucho más allá de autos que se conducen solos. Esta tecnología puede ser aplicada en distintas áreas como servicios de mensajería, e incluso en muchas áreas más técnicas como la exploración y el acceso a lugares de riesgo, minería, etc (Luettel, Himmelsbach, & Wuensche, 2012).

El desarrollo tecnológico en los últimos años ha favorecido el adelanto de este tipo de vehículos a tal punto que actualmente tienen un alto impacto en cuatro áreas:

La primera de innovación más reciente, se refiere a su uso comercial ya sea en autos autónomos con ejemplos de renombre como vehículos Tesla, Audi, entre otros, o por otro lado en dispositivos de uso cotidiano como aspiradoras inteligentes. La segunda, el área industrial siendo parte fundamental en la denominada Industria 4.0, cuyo objetivo es generar un proceso integrado o interconectado mediante IoT (Gonzales, Alves, Viana, Carvalho, & Basilio, 2018). La tercera, en áreas de investigación, siendo esta la más extensa. Como por ejemplo, la USFQ posee un prototipo de submarino dedicado a la investigación de vida marina en la región insular o Galápagos. La cuarta, se refiere a prototipos de búsqueda y rescate en zonas de difícil acceso tanto en entornos acuáticos (Leegstra, y otros, 2019) como aéreo (León & Barrientos, 2017).

De acuerdo a "*The Society of Automotive Engineers"* (SAE) existen seis niveles de autonomía (Izquierdo, Curiel, Bustamante, & Ramirez , 2019):

1. Sin automatización
2. Conducción asistida
3. Automatización parcial
4. Automatización condicional
5. Alta automatización
6. Completa automatización

Este proyecto está enfocado en el diseño e implementación de un prototipo autónomo cuyo sistema de navegación se base en sensores de alta precisión como un sensor tipo *Laser Imaging Detection and Ranging* (LIDAR), un módulo *Global Positioning System (GPS), un* módulo *Inertial Measurement Unit* (IMU) que incluye una brújula, giroscopio, acelerómetro y un barómetro, sensores de proximidad de ultrasonido, todo con el fin de generar una reconstrucción bi-dimensional del entorno para mapear la posición relativa del vehículo y evitar obstáculos en tiempo real mientras realiza una tarea asignada. Por lo tanto, el prototipo diseñado cumple con los requerimientos de un nivel de autonomía 5 (Alta automatización) pues requiere supervisión humana por motivos de seguridad.

DISEÑO DEL PROTOTIPO

Lista de materiales.

A continuación, se enumeran todas las partes del prototipo, incluidos los sensores, sistema de propulsión, sistemas de potencia.

- RPLidar A2M8

- Raspberry pi 3b
- Chasis
- 4 motores DC
- 4x L298N Driver
- Módulo Ublox Neo 6m GPS
- Berryiumu V2 brújula - giroscopio
- 2 baterías Li-Po
- Cargador de baterías
- 4 sensores ultrasónicos
- Módulo WiFi
- Mando Bluetooth (PS4 Dualshock)

Descripción del prototipo.

El prototipo usa de base un RaspBerry Pi 3B encargado del procesamiento de datos y de comunicación con los periféricos. Este ordenador monoplaca (SBC - Single Board Computer) recibe toda la información provista por los sensores, la procesa y es capaz de: determinar objetos cercanos, reconstruir un modelo 2d de su entorno y calcular rutas de movilización. Cuenta con dos tipos de sensores: los de proximidad que permiten al vehículo evitar colisiones e interactuar con su entorno que son los sensores ultrasónicos y LIDAR y los de localización como sensor IMU y modulo GPS que permiten establecer su posición relativa en tiempo real. El diagrama de bloques del prototipo se muestra en la figura 1.

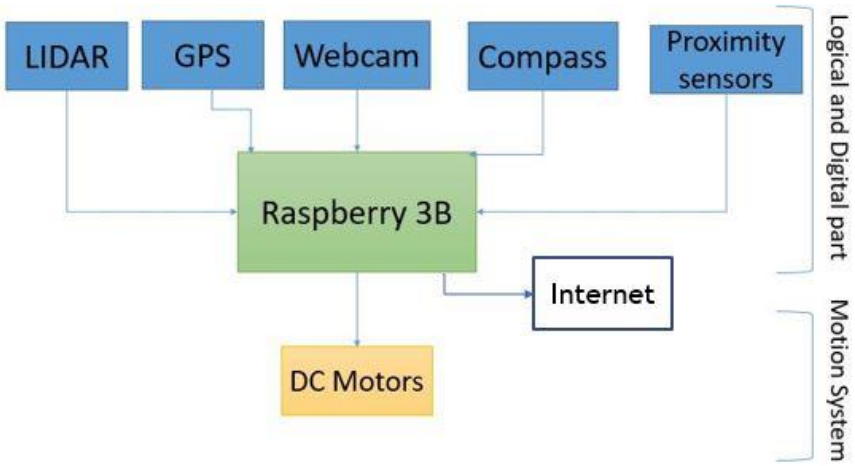


Figura 1 Diagrama de bloques del prototipo

Actualmente cuenta con dos modos de operación: modo manual en el cual se controla el movimiento del vehículo por medio de un mando bluetooth (PS4 Dualshock), en este modo se recolecta y almacena información sobre los alrededores por medio de LIDAR y Modo automático en el cual se utiliza los datos recolectados en el modo manual para imponer y ejecutar una ruta de movilización sobre el entorno mapeado. La transición entre los dos modos se produce de manera manual por comando en terminal o pulsando el botón 'triangulo' del mando, de esta manera podemos garantizar el control del vehículo incluso en el caso de falla en los sensores. Todo el sistema es alimentado por 2 baterías Li-Po de 3 celdas, cada batería con un voltaje nominal de 11.1v. La primera batería se utiliza para energizar el sistema de movilización, es decir, los motores y sus controladores; la otra se utiliza para brindar energía al Raspberry pi y todos sus periféricos. Se requiere un conversor DC-DC (Buck converter) para regular el voltaje suministrado a 5.15v.

CARACTERIZACIÓN DE SENSORES

En esta sección se describen y caracterizan los sensores que implementa el prototipo.

RpLidar

Sensor basado en láser, que gira a una frecuencia de 10 Hz y logra un escaneo de 360 grados y detección de rango del entorno circundante (SLAMTEC, 2016). La tabla 1 muestra las más relevantes características del SLAMTEC RPLIDAR A2M8 empleado.

Parámetro	Unidad	Valor Mínimo	Valor Máximo
Rango	Metros [m]	0.15	12-15
Frecuencia de medición	Hercio [Hz]	2000	8000
Frecuencia de escaneo	Hercio [Hz]	5	15
Ángulo de escaneo	Grados [°]	0	360

Tabla 1 Caracteristicas del RPLIDAR SLMATEC A2

WebCam

Una cámara web es un dispositivo periférico conectado por USB. Eso permite video en Full HD (1080p) y audio de alta calidad, sin embargo, la configuración utilizada opera la

cámara a resolución estándar (SD) para optimizar los recursos de procesamiento (Genius, 2012).

Módulo GPS

El sensor U-blox Neo 6M (U-blox, 2012) , se utiliza para el módulo GPS. Este sensor fue elegido por su pequeño tamaño, bajo costo y su versatilidad en proyectos. La Tabla 2 resume las más importantes características de este sensor.

Sensor	Tipo	Alimentación	Interface
Neo 6M	GPS	[2.7-3.6] V	UART-USB-SPI

Tabla 2 Caracteristicas del módulo GPS NEO 6M

El tiempo de espera entre encender el dispositivo y conectarlo con un satélite es de aproximadamente 27 segundos. Sin embargo, para un funcionamiento correcto minimizando el margen de error, el sensor debe estar conectado al menos a 9 satélites (U-blox, 2012), un proceso que dependiendo de las condiciones externas puede llevar entre 1 y 5 minutos.

Como se puede ver en la Tabla 2, existen diferentes tipos de interfaz. El prototipo implementa la interfaz USB versión 2.0 FS (velocidad máxima, 12 Mbit / s) desde un adaptador UART a USB. Esta modificación tiene dos ventajas: primero, mejora la precisión de los datos recibidos utilizando el algoritmo PPP desarrollado por U-blox; y segundo, esta interfaz permite usar la Interfaz UART de Raspberry exclusivamente para comunicación Bluetooth.

Como prácticamente todos los GPS disponibles en el mercado, el sensor NEO 6M utiliza el protocolo de información NMEA con una velocidad en baudios de 9600 (U-blox,

2012). En cuanto a la fiabilidad del sensor, se basa en chips GPS calificados AECQ100, por lo que pasa la ISO 16750 "Vehículos de carretera estándar-Condiciones ambientales y pruebas de equipos eléctricos y electrónicos" (U-blox, 2012),

Módulo IMU

BerryIMU, es un dispositivo IMU (Unidad de medida inercial) compuesto por un acelerómetro, giroscopio y magnetómetro (Ozzmaker, 2019), que operando en conjunto proporcionan información sobre la velocidad, orientación y fuerzas gravitacionales que experimenta el dispositivo. Además, el modelo BerryIMU v2 también incluye un sensor tipo barométrico BMP180, que evalúa la presión atmosférica del ambiente para estimar la altitud con respecto al nivel del mar. El circuito integrado LSM9DS0 es responsable de que la IMU tenga las funcionalidades de acelerómetro, giroscopio y magnetómetro. Según su hoja de datos, los rangos que cada uno de estos se muestran en la Tabla 3.

Acelerómetro	$\pm 2 ; \pm 4 ; \pm 6; \pm 8; \ \pm 16 \ [g]$
Magnetómetro	$\pm 2 ; \pm 4 ; \pm 6; \pm 8; \ \pm 12 \ [gauss]$
Giroscopio	$\pm 245 ; \pm 500 ; \pm 2000 \ [dps]$

Tabla 3 Rango de mediciones del sensor BerryIMU V2

Sensores de proximidad

Los sensores de proximidad utilizados son el Ultrasonido HC−SR04 (ELECFreak) . Este tipo de sensores emiten una señal de ultrasonido en una frecuencia de 40 KHz. Cuando la

señal rebota en un objeto vuelve, el tiempo se mide desde que la señal fue emitida hasta que vuelva al sensor para calcular la distancia usando la ecuación 1.

$$Distancia\ medida = \frac{\Delta time * (340m/s)}{2} \quad (1)$$

Con:

- *Distancia medida, es* la distancia calculada experimentalmente medida en metros.
- $\Delta time$, es el tiempo expresado en segundos que le toma a la onda sonora el regresar, a partir de su emisión.
- $(340m/s)$, rapidez del sonido expresada en metros sobre segundos.

ESQUEMA DE CONEXIÓN

A continuación, en la figura 2 se muestra el diagrama de conexiones del prototipo.

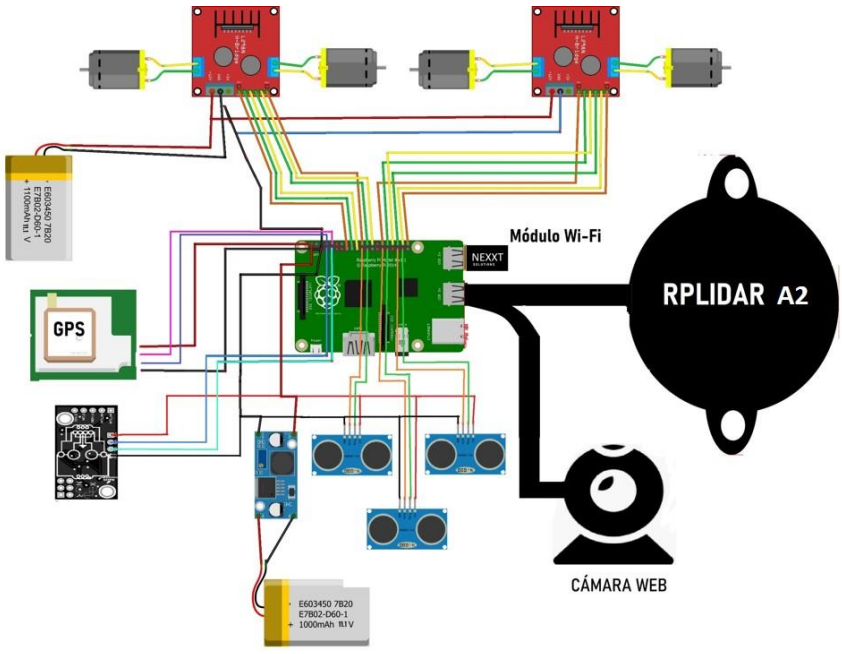


Figura 2. Diagrama de conexiones

PUESTA EN MARCHA

Esta sección explica la calibración y la puesta en marcha de cada sensor, ya que cada uno fue configurado para funcionar según parámetros útiles para todo el proyecto, sobre todo teniendo en cuenta la limitación de recursos del Raspberry empleado.

RpLidar

El sensor Lidar fue configurado para funcionar con una velocidad de transmisión de 11.520 Baudios. El dispositivo está conectado a un convertidor de comunicación UART a USB, por lo que es necesario declarar los derechos de lectura y escritura del puerto que ocupa. Esta configuración es la más adecuada para el proyecto como tal sobre todo teniendo en cuenta las limitaciones del microcontrolador Raspberry, cuyo puerto UART se utiliza ya para la comunicación con el mando Bluetooth. Se prefirió esta configuración que garantiza la detección de objetos a partir de una distancia de 15 centímetros y dentro de un rango de hasta 12 metros. El uso de la velocidad de transmisión más alta inhibe el rango de medición conforme a las características contenidas en el Data Sheet del dispositivo.

WebCam

La cámara Web fue configurada de modo tal que utiliza de la manera más eficiente los recursos del microprocesador Raspberry, es decir, la resolución de la imagen fue establecida en una definición estándar (SD por sus siglas en ingles), esto es 720 x480 pixeles a una tasa de 15 fps (cuadros por segundo por sus siglas en inglés). Los demás parámetros de la cámara como: enfoque, balance de blancos, exposición y saturación se establecieron en modo automático.

Módulo GPS

Para la configuración y calibración del módulo GPS se utilizó el programa *U-center* (distribuido por U-Blox, fabricante del GPS). La velocidad de transmisión fue configurada a una tasa de 9600 Baudios, con la cual se obtiene buenos resultados y se optimiza los recursos de procesamiento.

Por otro lado, en cuanto a la calibración del módulo GPS usando el programa U-center se revisó el grado de precisión del sensor. El error promedio en lugares cerrados fue de 5 metros, mientras que para lugares abiertos éste se redujo a 1,5 metros. La telemetría del GPS se verifica con ayuda del programa, es decir, se obtiene información como el número de satélites conectados, así como sus nombres, posiciones relativas, la calidad de transmisión de datos como se observa en la figura 3. Toda esta información se basa en los protocolos de comunicación NMEA (National Marine Electronics Asociation) 0183.

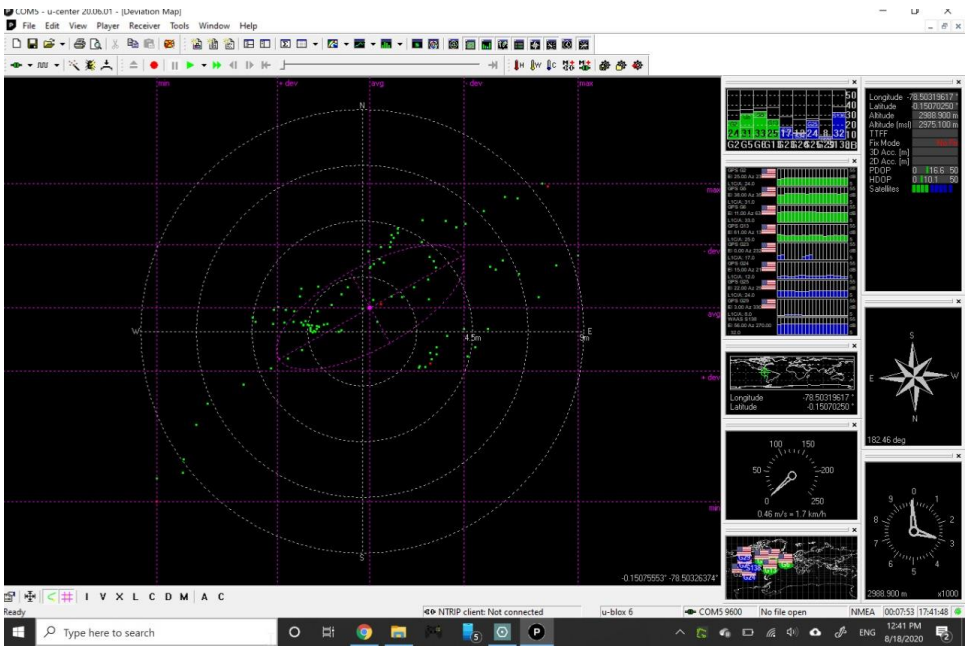


Figura 3 Calibracion de GPS por U-Blox 1

Módulo IMU

La configuración del módulo IMU fue realizada a partir de una interface I2C, de modo que se controla el acelerómetro y giroscopio para obtener información, que el prototipo usa para la navegación. El acelerómetro genera un *array* de 6 bits donde se almacenan los valores de información sin procesar o en bruto, por lo que se utiliza la función *readBlock( )* para expresar estos valores en complemento a 2, y posteriormente usando la función *Atan2( )* se obtienen finalmente el azimuth y elevación de utilidad para la navegación. Sobre el giroscopio se utiliza una metodología muy similar. En este caso usando la función *readACC ( )* se leen los datos provenientes del sensor y se transforman en ángulos de movimiento.

Sensores de proximidad

Los sensores de ultrasonido son configurados para emitir pulsos cada 10 µS, esto les permite calcular la distancia a un objeto cercano casi en tiempo real. El prototipo está equipado con tres sensores de este tipo: uno al frente y uno a cada lado. Esta configuración permite identificar obstáculos de manera óptima en cortas distancias, sobre todo entre 5 y 15 centímetros. A partir de este rango se utiliza la información proveniente del Lidar. Es decir, los sensores de proximidad son utilizados como complemento del Lidar, esto debido a los rangos de funcionamiento de cada tipo de sensor. El rango de funcionamiento del Lidar es de entre [0.15- 12] metros, mientras que el rango de funcionamiento de los sensores de ultrasonido es de [0.02 - 4] metros, de forma que empleando ambos tipos de sensores obtenemos un rango entre [0.02- 12] metros.

SISTEMA OPERATIVO

Este proyecto fue desarrollado sobre el sistema operativo Linux en su distribución Ubuntu Mate 16.04 LTS (Xenial Versus) sobre un raspberry Pi 3B donde instalamos ROS (Robot Operating System) en su versión Kinetic que es la única compatible con la versión de Linux que utilizamos. Kinetic no es la última versión de ROS, sin embargo, es la que tiene todos los paquetes de software totalmente actualizados y con soporte. Además, cuenta con una comunidad que desarrolla software sobre esta distribución de manera activa por lo que los foros de resolución de problemas son extremadamente útiles.

ROS

Robot Operating System (ROS) es una colección de librerías y paquetes de software pensado para facilitar el desarrollo de robots. Es considerado un Meta-Sistema operativo ya que ofrece características propias de un sistema operativo como: Abstracción de Hardware, control de dispositivos de bajo nivel, transmisión de mensajes a través de procesos y administración de paquetes; sin embargo, debe funcionar sobre un sistema operativo y no por si solo. ROS es compatible con la mayoría de las distribuciones de Linux, Mac OS y de manera experimental y limitada con Windows.

El propósito general de este software es el desarrollo de un sistema completo para robots que sea robusto y que sea adaptable a cualquier tipo de prototipo con cualquier tipo de hardware. Para esto se implementa una visión modular del sistema. Es posible dividir el funcionamiento general de un robot en varios comportamientos específicos y luego enlazarlos para que operen en forma conjunta. Este tipo de aproximación se conoce como robótica colaborativa y nos permite, por ejemplo, construir un equipo de trabajo que se dedique solo al desarrollo del control de motores, otro solo a la reconstrucción 2D del entorno a partir de LIDAR, otro equipo dedicado exclusivamente a la detección y reconocimiento de objetos

usando cámara. Al final, el resultado de cada equipo será mucho más eficaz pues dedicaron todo su tiempo a perfeccionar o crear software que resuelva un problema en específico al que solo faltaría enlazar cada producto de cada equipo para que se intercomuniquen.

Cada división tendrá como producto uno o varios nodos que son scripts programados para transmitir los valores de salida a otros nodos. Existen de dos tipos: Publicadores que emiten la información salida del script a cualquier otro nodo y Suscriptores que reciben la información de los nodos publicadores y la usan como valores de entrada para su propio código. La transmisión de datos se obtiene por medio de mensajes y a través de tópicos (canal de comunicación). Los nodos, en general, no tienen registro de su comunicación, es decir, no saben con qué nodo se comunican y se puede enlazar un nodo publicador a varios nodos suscriptores. La versatilidad de ROS permite generar cada nodo en diferente lenguaje de programación (un nodo escrito en Python y otro en C++ por ejemplo) y el resultado sería el mismo. Todos los nodos son comandados por un nodo maestro (ROS MASTER) que es siempre el primero en ejecutarse y gestiona la comunicación entre nodos, el diagrama de funcionamiento de ROS a partir de los distintos nodos se muestra a continuación en la figura 4.

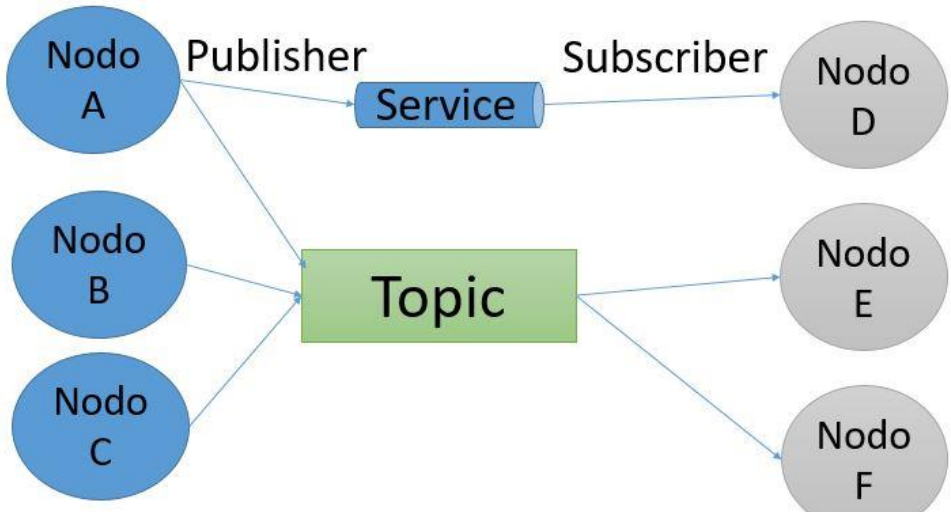


figura 4. Diagrama de operación de ROS.

En este proyecto se dividió el sistema completo del vehículo en secciones de control de movimiento, comunicación, SLAM (simultaneous localization and mapping) y rastreo GPS. Cada una de las secciones tiene su representación de nodos de manera gráfica utilizado el paquete RQT de ROS que permite la visualización de un diagrama de conexión entre nodos (el diagrama se presenta en la siguiente sección para mejor explicación).

ROS también permite el procesamiento distribuido. Implica la ejecución de nodos en un computador externo de manera que libere a la computadora principal de alguna carga de procesamiento. De esta manera se pueden ejecutar procesos que requieran de un hardware más avanzado y solo transmitir los valores de salida del nodo a través de mensajes y tópicos hacia un nodo suscriptor, sin embargo, la velocidad de transferencia de datos va a afectar el rendimiento del robot por lo que no es eficiente implementar acciones que requieran respuesta rápida con este método. Para este proyecto no se utilizó procesamiento distribuido debido a que el retraso de la salida de los nodos externos limitaba la operación óptima del vehículo.

APLICACIÓN DE PROCESOS EN ROS

Usando la metodología de ROS, el prototipo diseñado implementa 9 nodos principales, a continuación, en la figura 5 se muestra el esquema generado por ROS donde se detalla cada nodo, sus distintas dependencias y el diagrama de flujo.

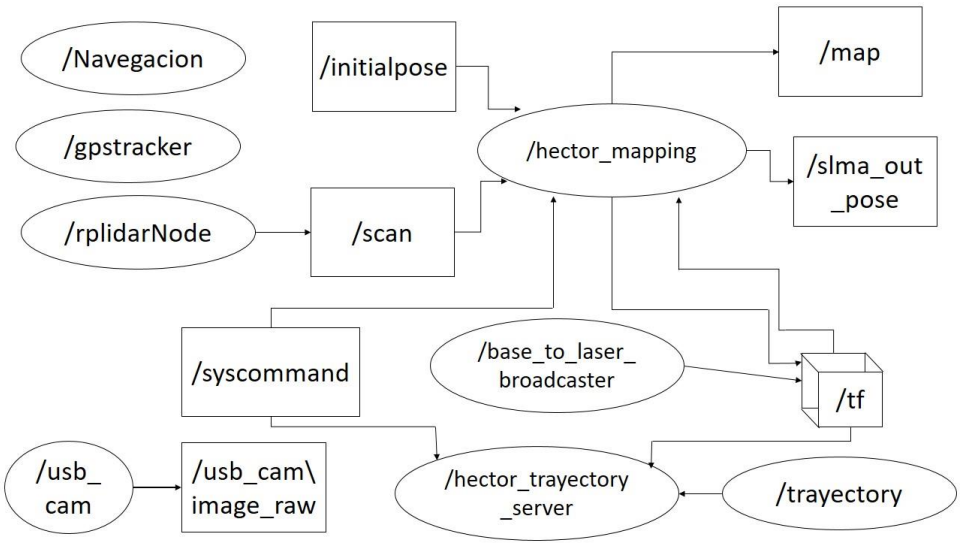


Figura 5 Diagrama de nodos obtenidos en RQT de ROS.

Para entender el funcionamiento de todo el diseño en conjunto primero es necesario describir o detallar el comportamiento de todas las partes que lo componen, por lo que a continuación especifica los nodos mostrados en la figura 5.

Nodo *USB CAM*

Este nodo es el encargado del control y transferencia de información proveniente de la cámara web. Dicha información es empaquetada usando el sistema YUYV, el mismo que consiste en un sistema de procesamiento de imágenes de color usando un ancho de banda reducido para los componentes de iluminancia (Nobuhara, Pedrycz, & Firota, 2005)**.** Este nodo se conecta de manera interna con la herramienta *rviz* de ROS, la cual hace el papel de interfaz con el usuario, pues es la encargada de mostrar la información de la cámara, a una tasa de 15 fps de modo tal que el cerebro humano pueda percibir movimiento fluido en la transmisión de video. La tasa de refresco (fps) es un parámetro que puede ser modificado, sin embargo, una tasa más alta implica un requerimiento de mayor poder computacional, siendo éste un recurso sumamente limitado en nuestro hardware.

Nodo RpLidar

Este nodo implementa el control y transferencia de información por parte del sensor Lidar, para lo cual implementa las librerías *slamtec* propias del fabricante del sensor. La información obtenida del sensor, distancias y ángulos, son enviadas en forma de tópico al nodo *hector_mapping* que se encarga de interpretar dicha información usando un algoritmo especifico (desarrollado por Slamtec) para generar Mapeo y Localización Simultanea (*SLAM* por sus siglas en ingles). La información de mapeo es enviada al nodo *map,* que implementa comunicación directa con la herramienta de ROS *rviz,* que como se explicó previamente realiza la función de interfaz con el usuario y despliega la sección del mapa construida a partir de las mediciones del LIDAR. De igual forma la información del mapeo se envía al nodo *slam_out_pose* como parte del algoritmo dedicado a realizar el SLAM, es decir el posicionamiento relativo del vehículo dentro de la sección del mapa.

Trayectoria a partir del Sensor Lidar

El sistema de trayectoria implementa distintos nodos, en especial los nodos *hector_mapping* y *hector_trayectory_server*. El primero, como se explicó previamente, obtiene información del sensor Lidar y realiza una reconstrucción de entorno o SLAM, dicha reconstrucción es compartida al nodo *hector_trayectory_server,* el cual es el encargado de fijar la trayectoria recorrida por el prototipo. El algoritmo que utiliza (desarrollado en Norwegian University of Science and Technology en Alemania) se basa en analizar el SLAM en dos momentos continuos en un intervalo de tiempo y a partir de las diferencias calcula el movimiento efectuado. Esta es la forma en que se realiza la trayectoria referencial a partir del

sensor Lidar (Stranden, 2019). Una segunda forma en que se fija una trayectoria geo referencial es a partir del uso del GPS.

Nodo *GPS_tracker*

Este nodo es el encargado del control y transferencia de información proveniente del módulo GPS, la información recibida usa el protocolo NMEA 0183 (U-blox, 2012)**.** Se utiliza las librerías de U-blox (fabricante del módulo GPS) para así obtener información de longitud, latitud, altura y velocidad de desplazamiento. Estos datos se cargan mediante una conexión de internet a un servidor de la plataforma en tiempo real **Pubnub,** disponible en su versión gratuita con recursos restringidos. De esta forma, utilizando un programa de página Web desarrollado en HTML (el mismo que está descrito en la sección de interfaz de usuario) es posible visualizar la trayectoria en tiempo real del prototipo, para lo cual se utiliza el API de Google Maps.

Nodo de Navegación

Este es el nodo encargado del sistema de movimiento del prototipo. Implementa dos tipos de navegación. La primera en forma automática se encarga de recibir la información de los distintos sensores para así controlar la trayectoria del vehículo. La segunda en forma manual se implementa a partir de comunicación Bluetooth con un control remoto externo, en este caso un mando de PS4.

En resumen, este nodo se encarga del movimiento de los cuatro motores para generar el desplazamiento a partir de la información de los sensores (modo automático) o a partir de las órdenes del usuario (modo manual).

Nodo Compass

Este nodo a pesar de no estar diagramado en la figura 5, es el encargado del control y transferencia de información del BerryIMU V2, para lo cual se utiliza las librerías propias del fabricante. La información proveniente del sensor se procesa y convierte en información útil como el ángulo en relación al norte magnético, aceleración o posición respecto al eje horizontal, de modo que es posible obtener una telemetría más precisa del prototipo en tiempo real.

INTERFAZ DE USUARIO

La interfaz de usuario puede ser dividida en dos etapas principales: la primera que se relaciona con el sistema de seguimiento a partir del módulo GPS, y la segunda enfocada con la información de navegación, para lo cual se usa la herramienta de ROS, *rviz*.

La interfaz asociada al módulo GPS, como se mencionó en la sección previa (GPS_Tracker), utiliza un servidor en Pubnub en conjunto a una página web diseñada exclusivamente para el seguimiento. A continuación, en la figura 6 se muestra una captura de la aplicación web

Figura 6. Aplicación web usada para rastreo GPS.

Se observa en esta imagen que la interfaz de usuario es sumamente amigable con el usuario. De forma general, utiliza un solo botón para iniciar o detener el seguimiento del prototipo. Al implementar la Api de Google Maps su uso es bastante simple; existen dos tipos de vistas, en forma de mapa o una vista satelital, y presenta botones que permiten al usuario realizar *zoom in* o *zoom out.* A medida que el vehículo se mueva la trayectoria se verá reflejada en el mapa permitiendo observar el recorrido completo.

La segunda etapa usa la herramienta de ROS, *rviz*, para mostrar los resultados del SLAM y de la cámara. La figura 7 muestra una captura de la aplicación.

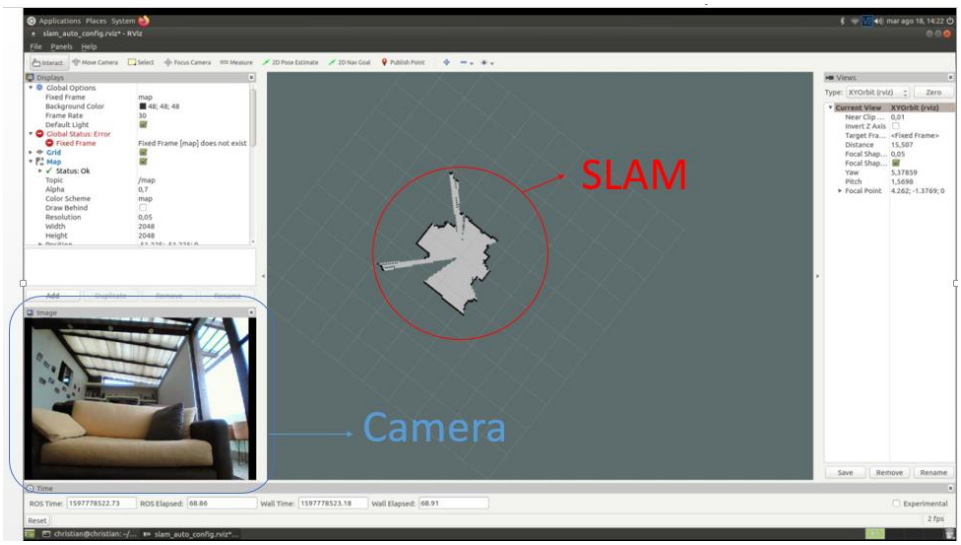


Figura 7. Interfaz en RVIZ.

La gran ventaja de usar *rviz* es que es posible configurarlo de la manera que resulte más conveniente para el usuario; por ejemplo, en la figura 7 se configuró de modo tal que el SLAM es la visualización más importante por lo que tiene un tamaño principal comparado al resultado de la cámara web. Por lo tanto, dependiendo de la aplicación que se realice se puede cambiar el tamaño, posición e información desplegada en la interfaz de usuario.

FUNCIONAMIENTO

Previamente se explicó el funcionamiento de cada sensor y cada nodo de manera individual, a continuación, en esta sección se detalla el funcionamiento en conjunto de todas las partes que componen el prototipo. Iniciando con el módulo GPS, este sensor se comunica con el microprocesador usando un convertidor UART a USB, dado las características del GPS existe un tiempo de espera hasta establecer comunicación con el número mínimo de satélites para garantizar su correcto funcionamiento. Una vez que la comunicación es establecida el nodo *gps_tracker* es ejecutado, recolecta la información de longitud y latitud para enviarla al servidor de Pubnub de modo que sea posible iniciar el proceso de

seguimiento o *tracking* a partir de la aplicación Web utilizando la API de Google Maps. En la figura 8 se muestra el proceso de transmisión de los datos de longitud y latitud entre el prototipo y el servidor.

Figura 8. Monitoreo de transmisión de datos en PUBNUB.

Por otro lado, los nodos *usb_cam_node , rpLidar:node* y los otros nodos correspondientes a la georeferencia son ejecutados en forma simultánea, lo que permite efectuar el algoritmo encargado del SLAM y de la trayectoria del prototipo La principal limitacíón del metodo es que presenta una alta sensibilidad en relación a la posición relativa por lo que es necesario realizar movimientos fluidos. A continuación en la figura 9 se muestra una captura del funcionamiento del vehiculo.

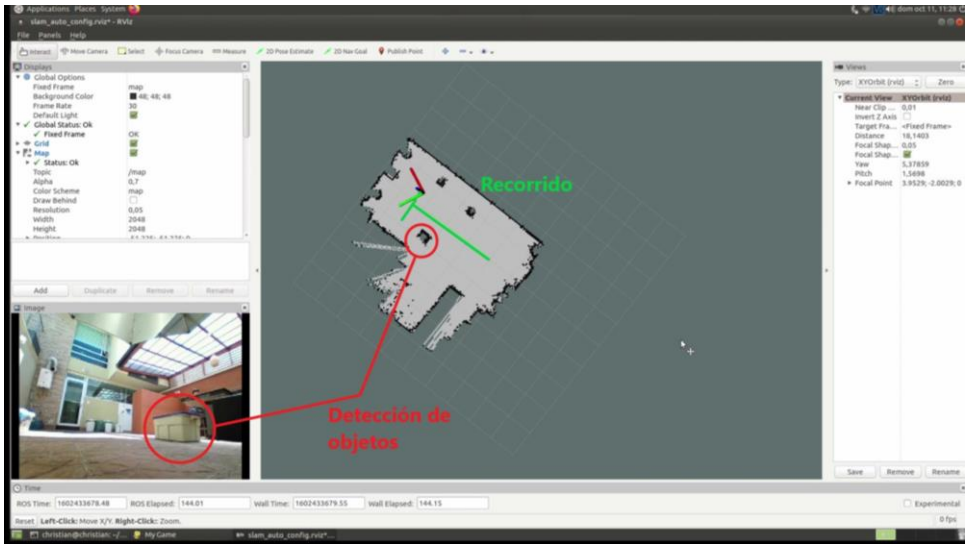


Figura 9. Ejemplo de SLAM/Recorrido.

En la figura mencionada, se observa el mapeo completo de un entorno cerrado junto con la perspectiva a nivel del suelo obtenida por la cámara. Los círculos de color rojo representan al mismo objeto, una caja, que fue situada en una posición aleatoria del entorno para demostrar que el sensor LIDAR detecta objetos cercanos y que el SLAM los sitúa en dentro del entorno mapeado a medida que realiza el recorrido por el cuarto. La línea verde es el recorrido desde el reposo del prototipo y sirve para determinar su posición relativa.

Prototipo físico Final

En esta sección se detalla la parte física de la versión final del prototipo, en las figuras 9, 10 , 11 se muestran las vistas frontal, posterior y lateral respectivamente.

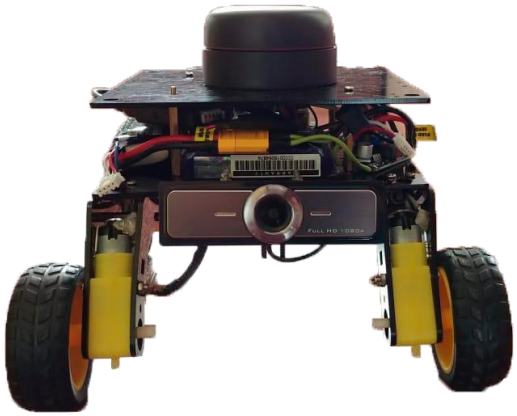


Figura 10. Vista frontal del prototipo.

Figura 11. Vista posterior del prototipo

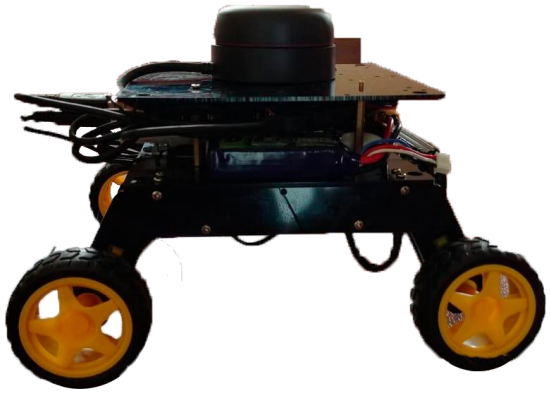


Figura 12. Vista lateral del prototipo

El prototipo fue ensamblado sobre un chasis genérico de acrílico, implementa 4 motores DC de 5V con sus respectivas cajas reductoras con un piñón plástico y un eje biaxial, el peso aproximado de cada motor es de 35 gramos. El driver controlador de cada motor implementa un control en Modulación de Ancho de Pulso (PWM) para generar un máximo de 250 revoluciones por minuto (TIMOTOR). Sin embargo, a partir de distintas pruebas realizadas y teniendo en cuenta que el movimiento del vehículo debe lo más fluido posible, se utiliza solo el 30 porciento de la capacidad máxima, es decir, 75 revoluciones por minuto.

ESCALABILIDAD DEL PROYECTO

Gracias a la versatilidad del software, es posible realizar una gran cantidad de mejoras, por lo que este proyecto es altamente escalable. Haciendo uso de la ejecución distribuida de nodos en ROS es posible obtener un mayor poder computacional al correr en simultáneos nodos fuera del microprocesador Raspberry. En otras palabras, se puede correr subrutinas del programa en sistemas externos como computadoras o servidores. Suponiendo las debidas mejoras en el poder computacional, el algoritmo de navegación puede implementar Machine Learning para mejorar su precisión, trabajando en conjunto de un

sistema de visión artificial a partir del reconocimiento de imágenes de la cámara, de forma que hacer posible la identificación y el reconocimiento por ejemplo de los bordes de las calles, señales de tráfico, si el problema de interés se refiere a la movilidad en carretera. En forma análoga, dentro del sector industrial sería posible reconocer señaléticas específicas como parte de un proceso, brindando al prototipo un mayor nivel de autonomía. Estos desarrollos futuros podrían ayudar, aspirando a llegar al nivel 6 o completa autonomía.

De igual forma, con un mejor presupuesto es posible mejorar las características mecánicas del vehículo, es decir, utilizar por ejemplo como base un carro de batería para niños para lograr un mayor tiempo de funcionamiento, mayor alcance, mayor capacidad de carga siendo posible aumentar la cantidad de equipos, sensores, entre otros, mejorando asi las prestaciones del prototipo.

CONCLUSIONES

Se diseñó e implementó un prototipo de vehículo terrestre, equipado con diferentes tipos de sensores, que le permiten moverse evitando de forma segura colisiones en la configuración automática. Desde la interfaz de usuario es posible visualizar en tiempo real la trayectoria del vehículo utilizando la API de Google Maps. Esta aplicación permite tener un error medio de 2,5 m. El sistema de navegación, por su parte, permite observar en tiempo real el mapeo, reconstrucción del lugar (SLAM) y una visualización del sitio (usando la cámara). Las dos baterías que usa el prototipo le permiten tener una autonomía promedio de 24 minutos lo que le permite cubrir áreas relativamente grandes.

El uso del microprocesador Raspberry con el sistema operativo basado en Linux con una distribución de Ubuntu permitió emplear la versión Kinetic de ROS, con tres ventajas principales. La primera y más importante se refiere a su compatibilidad con todos los tipos de sensores empleados y sus respectivas librerías. Segunda, la capacidad de implementar todas las subrutinas del prototipo en distintos nodos con una gran versatilidad. Tercera, al tratarse de software libre, existe una gran cantidad de información, tutoriales y referencias de proyectos similares que fueron de gran utilidad a lo largo del desarrollo de este proyecto.

El prototipo desarrollado es similar a otros robots existentes en el mercado, como por ejemplo TURTLEBOT3 WAFFLEPI; sin embargo, estos robots comerciales tienen precios más altos (alrededor de USD 1300) y no poseen la versatilidad que el modelo que se ha propuesto. Nuestro prototipo puede configurarse fácilmente para escenarios múltiples y complejos, a un costo de aproximadamente USD 500. Por lo tanto, es una herramienta muy útil y con gran potencial dentro del mundo ingenieril. Por ejemplo, se puede utilizar como asistente de arquitectura ya que permite reconstruir un lugar de forma fácil, rápida y de forma prácticamente automática. De la misma forma se puede utilizar como vehículo de exploración en lugares de difícil acceso, donde el posicionamiento GPS es de gran importancia; sin embargo, para aplicaciones en exteriores se requiere un nuevo diseño mecánico que relacione debidamente el tamaño, la configuración de las ruedas y el tipo de motores con la aplicación que se le va a dar. Vehículos autónomos tienen aplicaciones potenciales en agricultura, en industria y hospitalidad en la cadena de suministros, etc.

REFERENCIAS BIBLIOGRÁFICAS

ELECFreak. (s.f.). Ultrasonic Ranging Module HC - SR04. *Development Kit User Manual*.

Genius. (2012). Eye 312 Web Cam. *User Manual*.

Gonzales, A., Alves, M., Viana, G., Carvalho, L., & Basilio, J. (2018). Supervisory Control-Based Navigation Architecture: A New Framework for Autonomous Robots in Industry 4.0 Environments. *IEEE Transactions on Industrial Informatics, vol. 14, no. 4*, 1732-1743.

Izquierdo, J., Curiel, L., Bustamante, R., & Ramirez , R. (2019). Perspective of Autonomous Driving in Mexico. *2018 International Conference on Mechatronics, Electronics and Automotive Engineering (ICMEAE).* Cuernavaca: IEEE.

Leegstra, R., De Paula, M., Carlucho, I., Solari, F., Rozenfeld, A., Acosta, G., . . . Villar, S. (2019). MACÁBOT: Prototipo de Vehículo Autónomo de Superficie (ASV). *Revista Tecnología y Ciencia No: 36*, 142-154.

León, J., & Barrientos, A. (2017). Estudio de los patrones de marcha para un robot hexápodo en tareas de búsqueda y rescate. *Actas de las XXXVIII Jornadas de Automática.* (págs. 701-709). Gijón: Universidad de Oviedo.

Li, Y., Díaz, M., Morantes, S., & Dorati, Y. (2018). Vehículos autónomos: Innovación en la logística urbana. *Revista de Iniciación Científica*, 34-39.

Luettel, T., Himmelsbach, M., & Wuensche, H.-J. (2012). *"Autonomous Ground Vehicles—Concepts and a Path to the Future.* Proceedings of the IEEE, vol. 100, no. Special Centennial Issue, pp. 1831-1839.

Nobuhara, H., Pedrycz, W., & Firota, K. (2005). Relational image compression: Optimizations through the design of fuzzy coders and YUV color space. En *Soft Computing* (págs. 471 - 479). DBLP.

Ozzmaker. (2019). BerryIMU Quick Start Guide. *BerryIMU Quick Start Guide User Manual*.

SLAMTEC. (2016). RPLIDARA2. *Development Kit User Manual*.

Stranden, J. E. (Junio de 2019). Autonomous driving of a small-scale electric truck model with dynamic (Master tesis). Darmstadt , Alemania: Norwegian University of Science and Technology.

TIMOTOR. (s.f.). TGP01D-A13 Gear motor. Development user manual.

U-blox. (2012). Neo 6M. *User Manual*.

ANEXO A: ARTÍCULO PUBLICADO EN ICMEAE 2020

Design and Implementation of an autonomous vehicle with LIDAR-based navigation

José Barbosa
Universidad San Francisco de Quito
Colegio de Ciencias e Ingenierías
Quito, Ecuador
jebarbosa@ieee.org

Christian Hernandez
Universidad San Francisco de Quito
Colegio de Ciencias e Ingenierías
Quito, Ecuador
cshernandez@ieee.org

Daniel Paredes
Universidad San Francisco de Quito
Colegio de Ciencias e Ingenierías
Quito, Ecuador
dsparedes@estud.usfq.edu.ec

René Játiva E.
Universidad San Francisco de Quito
Colegio de Ciencias e Ingenierías
Quito, Ecuador
rjativa@usfq.edu.ec

***Abstract*—**
A terrestrial automat has been implemented whose navigation system is based on the reconstruction in two dimensions of its surroundings from multiple sensors such as: a Lidar type sensor, a GPS module, an electronic compass and various proximity sensors. Further, the characterization of the system and all these sensors were carried out in the Raspberry pi 3 B platform.The different types of sensors and the user interface allow the prototype to have great versatility for different specific applications within the engineering world such as zone mapping and interior reconstruction at low cost

***Index Terms*—Autonomous vehicle, Navigation, LIDAR, ROS, Characterization of sensors.**

I. Introduction

Autonomous navigation is quite promising as it presents a large number of applications that go far beyond a car that is driven by itself. This technology can be used in multiple areas, such as in the parcel service, mail, and even in much more technical areas such as exploration and access to remote and risky places, mining, etc [1].

On the other hand, this technology is also used to create search and rescue prototypes in air [2] and water [3] environments. More than ninety percent of traffic accidents are attributed to human error [4], so the implementation of automated vehicles on a daily basis will mean a significant improvement in safety issues in land transportation.

The current development of fully electric vehicle prototypes for personal use and for the transport of merchandise already implements mapping technology based on 2D and 3D lidar sensors in addition to an array of cameras around the vehicle that allow an interpretation of the surroundings in real time, based on machine learning [5]. It is necessary to create or improve algorithms that allow the use of this technology efficiently, with fewer resources.

This article focuses on the design and implementation of an autonomous prototype whose navigation system is based on a Laser Imaging Detection and Ranging (LIDAR) sensor to generate an environment bi-dimensional reconstruction, in order to map vehicle position and thus avoid obstacles in real time. Further, the prototype has a GPS module, a compass electronic and various proximity sensors. All of these devices work together in the Raspberry Pi 3 B platform, allowing the creation of multiple engineering oriented applications.

Due to a proper work of these sensors is fundamental to the navigation of the prototype, they were characterized in terms of accuracy and precision to integrate them in the best way and minimize the navigation errors.

According to the The Society of Automotive Engineers (SAE) there are six levels of vehicular autonomy [6].

1) No automation
2) Driver assistance
3) Partial automation
4) Conditional automation
5) High automation
6) Full automation

Our prototype is designed to meet the requirements of the level of autonomy 5 (high automation) because it requires human action only when defining its objective.

II. Prototype design

A. List of Materials

Listed below are all parts of the prototype, including sensors, propulsion system, power systems, and hardware.

- - RPLidar A2 sensor
- - Raspberry pi 3b
- - Chassis
- - 4 DC engine
- - 4x L298N Driver
- - Ublox Neo 6m GPS module
- - Berryiumu V2 Compass
- - 2 lipo battery
- - Battery charger
- -Infrared proximity sensors
- - WiFi Module

B. Prototype Description

The prototype control system is based on a Raspberry Pi 3 B microprocessor used both as brain and communication node. This microprocessor receives all the signals coming from the sensors, processes them and incorporates them into the navigation system to establish the route. The output information is also sent to the vehicle engines for commissioning, as we can see in Fig. 1 The prototype has two types of sensors: those of proximity that allow the vehicle to interact with its environment and avoid collisions; and those of geolocation that enable to know its position in real time and to establish the route from the navigation system. The current prototype has two modes of operation: Manual operation, where a user controls the movements of the vehicle in real time through bluetooth communication; and the Automatic mode where navigation is based exclusively on information collected from all sensors. For security reasons it is also possible to change from automatic to manual mode in a matter of seconds. Two power sources were implemented, each one corresponding to a 11.4V battery. A battery is used exclusively to power the vehicle's movement system, that is, the 4 DC motors with their respective controllers. The second one is responsible for supplying energy to all the logical and digital part of the prototype, that is, to the microprocessor and the different sensors. Further, to expand the versatility in terms of control and data visualization, a WiFi module was added so the vehicle has an Internet connection from which it can transmit images, results and telemetry in real time.

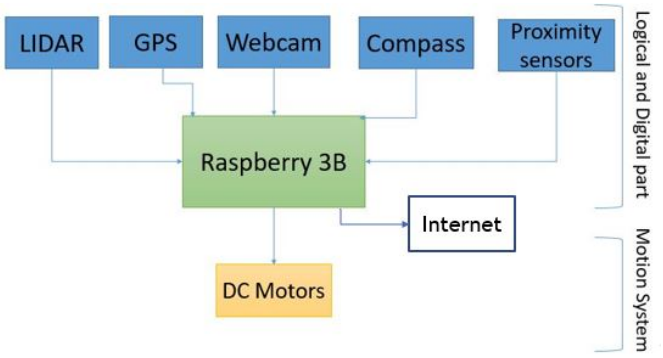


Figure 1: Blocks diagram.

III. Sensors

This section describes the main sensors, their operation and their main characteristics.

A. RpLidar

Laser-based sensor, that rotates at a frequency of 10 Hz and achieves 360-degree scanning and range detection of the surrounding environment [7]. Table I shows the most relevant characteristics of the SLAMTEC RPLIDAR A2M8 employed.

Table I: Characteristics of SLAMTEC RPLIDAR A2M8

Parameter	Unit	Min	Max
Range	m	0.15 m	12 - 15 m
Measurement frequency	Hz	2000	8000
Scan frequency	Hz	5	15
Scan Angle	grades	0-360	–

B. WebCam

A webcam is a peripheral device connected by USB. It allows video in full HD (1080p) and high quality audio, however, the configuration used operates the camera at standard resolution (SD) to optimize processing resources (other parameters were altered and will be explained in the following sections) [8].

C. GPS Module

The U-blox Neo 6M [9] sensor, is used for the GPS module. This sensor was chosen due to its small size, low cost and its versatility in projects. Table II summarizes the most important characteristics of this sensor.

Table II: Characteristics U-blox Neo 6m sensor

Sensor	Type	Supply	Interface
Neo 6M	GPS	[2.7-3.6]V	UART, USB, SPI

The waiting time between turning on the device and connecting it with a satellite is approximately 27 seconds. However, for correct operation minimizing the margin of error, the sensor must be connected to at least 9 satellites [9], a process that depending on the external conditions can take between 1-5 minutes.

As can be seen in Table II, there are different types of interface. The prototype implements USB interface version 2.0 FS (full speed, 12 Mbit/s) from a UART to USB adapter. This modification has two advantages: first, it improves the accuracy of the received data using the PPP algorithm developed by U-blox; and second, this interface allows to use the Raspberry UART interface exclusively for Bluetooth communication.

Like practically all available GPS, the NEO 6M sensor uses the NMEA information protocol with a Baud Rate of 9600 [9]. Regarding the reliability of the sensor, it is based on AEC-Q100 qualified GPS chips, which is why it passes the ISO 16750 standard "Road vehicles-Environmental conditions and testing for electrical and electronic equipment" [9],

D. Compass-gyroscope-accelerometer

BerryIMU, is an IMU (Inertial Measurement Unit) device composed of an accelerometer [10], gyroscope and magnetometer, which together provide information about the speed, orientation and gravitational forces that the same device passes through under a given environment. In addition, the BerryIMU v2 model also includes the BMP180 barometric sensor, which evaluates the atmospheric pressure of the environment to estimate the altitude with respect to sea level. The integrated LSM9DS0 is responsible for the IMU to have the accelerometer, gyroscope and magnetometer functionalities since this integrated is a packaged system that has these three

measurement devices. According to its datasheet, the ranges that each of these have are the following:

Accelerometer: ± 2; ± 4; ± 6; ± 8; ± 16 [g]

Magnetometer: ± 2; ± 4; ± 6; ± 8; ± 12 [gauss]

Gyroscope: ± 245; ± 500; ± 2000 [dps]

E. Proximity sensors

The proximity sensors used are the Ultrasound $HC-SR04$ [11]. This type of sensors emit an ultrasound signal at a frequency of 40 KHz. When the signal bounces off an object it returns, the time is measured from when the signal was emitted until it returns to the sensor in order to calculate the distance using equation 1.

$$Test_{Distance} = \frac{\Delta time \times (340m/S)}{2} \quad (1)$$

- $Test_{Distance}$ Calculated experimental distance expressed in meters
- $\Delta time$ Time it takes for the sound wave to return to the sensor expressed in seconds
- $340m/s$ Speed of sound expressed in metres over seconds

The main characteristics of HC-SR04 Sensor are shown in Table III

Table III: Characteristics HC-SR04 sensor

Voltage (DC)	Current	Frecuency	Range
5 V	15 mA	40 KHz	[2cm-4m]

IV. Connection scheme

The connection diagram of the designed prototype is exhibited in Fig.2:

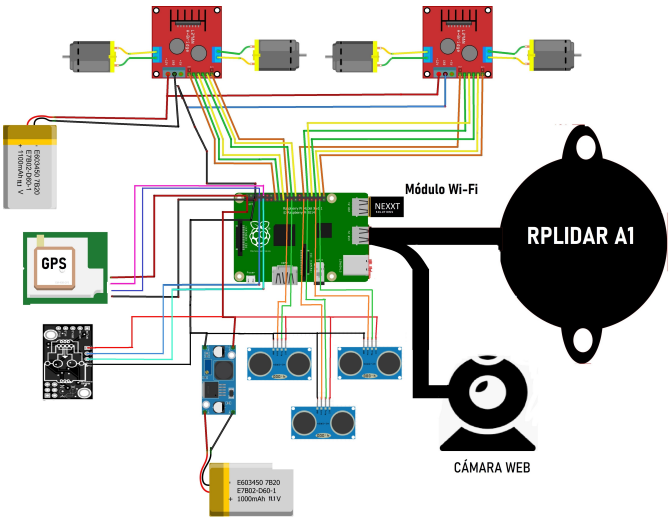


Figure 2: Connection diagram.

V. Start up

This section explains the calibration and commissioning process for each sensor, since each one was configured to work according to useful parameters for the entire project.

A. RpLidar

The lidar sensor was configured to work with a Baud rate of 11520 and with the parameters specified in table 1, the device is connected to a UART to USB converter, so it is necessary to declare reading and writing rights for the port it occupies. This type of sensor is very accurate at medium distances and is unable to detect obstacles less than 15 cm.

B. WebCam

The configuration of the web camera is designed so that it uses the resources of the raspberry pi efficiently, the resolution of the image was fixed to standard definition (SD), that is, 720x480 pixels and 15 fps (frames per second). The other parameters such as: focus, white balance, exposure and saturation were set to automatic.

C. GPS Module

The configurations made in the GPS module were made using the *U-center* program (distributed by U-blox) [9], in which the data transmission speed was set at a baud rate of 9600. This parameter was chosen to optimize use limited Raspberry resources.

Similarly, using the U-center program, the accuracy of the sensor was checked. The average error in closed environments is 15 meters, while in open places it is reduced to 5 meters. The telemetry of the GPS is also appreciated showing data such as the number of satellites connected and the quality of data transmission.

Below is a screenshot of the U-center program in Fig. 3 with all the previously mentioned settings.

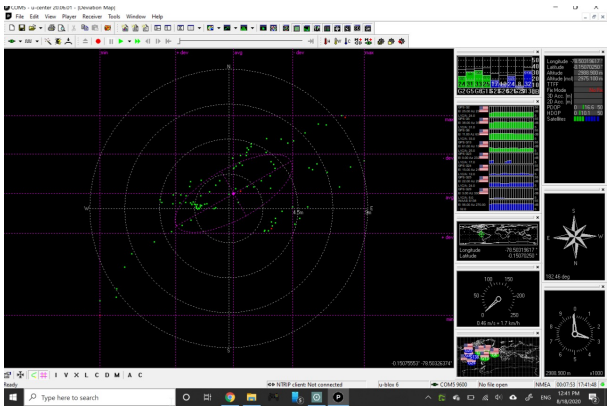


Figure 3: GPS calibration process in U-center

D. Compass

The compass configuration was done through an I2C interface, which will control the activation of the gyroscope and the accelerometer so that they can provide us with the data we require from the prototype navigation. In the accelerometer, a 6-byte array is generated where the raw values are stored and the readBlock () function is in charge of expressing these values in 2's complement, so that, using the Atan2 function, we

can convert these raw values into angles. With the gyroscope, we use a function similar to readACC () to read the raw values, so that we can convert them into angles by having the sensitivity level and the number of dps as data. [10]

E. Proximity sensors

the ultrasonic sensors are programmed to emit pulses every 10us, this allows us to calculate the distance to a nearby object almost in real time. Bear in mind that these types of sensors are designed to work optimally in short distances, that is, from 5 to 15 cm, after this range the data received by the lidar sensor is used.

VI. Operating system

The project is based on the Raspberry 3b microprocessor, therefore it uses the Linux operating system with an Ubuntu distribution. ROS (Robot Operating System) is used to control all sensors and navigation systems, specifically the 16.04 (LTS) Xenial version is used.

A. ROS

In general terms ROS is a collection of frameworks for the development of robot software. It enables hardware abstraction, low-level device control, implementation of commonly used functionality, message passing between processes, and package maintenance. This project uses Kinetic ROS Version following guidelines in [12].

ROS presents great versatility because its architecture is based on graphs, as it is shown in fig. 4. That is, the processing is based on ”nodes” (understood as an executable subroutine) and how these are related to each other. For example, there is an exclusive node for receiving and transmitting data from the GPS module, this being a publisher node, it communicates with another node (subscriber) that is in charge of processing and using said information. The nodes share information through a topic. A node can be subscribed to one or more publisher nodes.

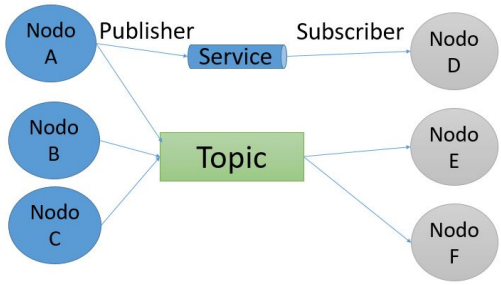


Figure 4: ROS operating diagram

Given ROS’s versatility in terms of handling nodes, it is possible to program each subroutine in a different programming language. Python was used for this project.

Another advantage of using ROS is the availability and great variety of libraries on its home page based on the philosophy of solidarity and collaboration so characteristic of free software projects.

VII. Application of processes in ROS

As it is shown in Fig. 5 the prototype implements 9 different nodes, each of them and their different dependencies are detailed below.

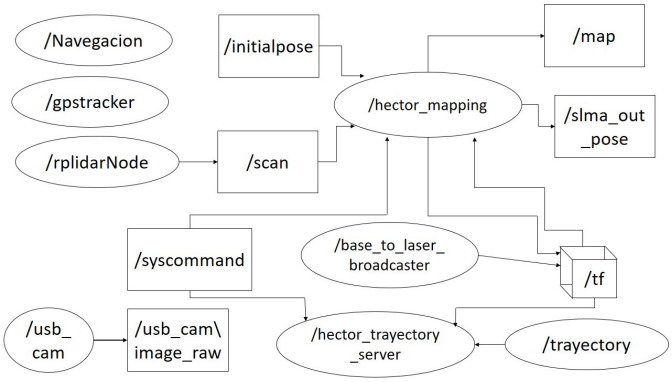


Figure 5: Node schema in ROS

A. USB Cam node

This node implements the control and transfer of information from the web camera. The information packaging system is YUYV, consisting of a signal system to process color image generating a reduced bandwidth for luminance components [13] To be able to observe the images obtained from the camera in real time using the ROS rvis tool, a data transmission is performed at a speed of 15 fps.

B. RpLidar Node

This node implements the control and transfer of information from the LIDAR sensor, for which the manufacturer’s own libraries *slamtec* are executed. The information obtained from the sensor is sent in the form of topic to the node hector_mapping which interprets said data using a specific algorithm to generate the Simultaneous localization and mapping (SLAM), finally the real-time mapping is shown in the rviz.

C. GPS Node

This node implements the control and transfer of information from the GPS module, the information is received using the NMEA 0183 protocol [9].

Using the U-blox (the GPS manufacturer) own libraries, the information received is transformed into the format of longitude, latitude , height and travel speed. This information is sent as a topic to the gps_tracker node to later be uploaded to the **Pubnub** server. In this way, using a web page program in HTML it is possible to follow the trajectory in real time. To visualize the displacement route, a Google Maps *API* is used

D. Navigation Node

This node is in charge of the prototype's movement system. It is in charge of managing the information from the sensors and in this way controlling the trajectory of the vehicle. Additionally, in this node, manual control is implemented from a Bluetooth communication with an external remote control, in this case a PS4 control.
In summary, this node is in charge of directing the 4 motors to generate the displacement based on the information from the sensors (autonomous mode) or the user's orders (manual mode)

E. Compass Node

This node implements the control and transfer of information from the BerryIMU V2 Compass sensor, for which the libraries provided by the sensor manufacturer (BerryIMU) are used. The data is processed and converted into useful information for navigation such as the angle in relation to the magnetic north, the acceleration or position with respect to the horizontal. This information is useful for the SLAM process and is also displayed in real time in the user interface.

VIII. Functionality

Previously, the operation of each sensor and each node was mentioned. This section explains how all the parts that make up the prototype work together.
Starting with the GPS module, this sensor connects to the Raspberry using a UART to USB converter. There is a waiting time until the module connects with the minimum number of satellites to guarantee its correct operation (9 satellites [9]). Once communication link has been established, the gps_tracker node is executed,and it collects the longitude and latitude information. Then, this information is sent to the Pubnub [14] server to start the transmission to the web page,as it is shown in Fig. 6, designed for this purpose, generating a georeference using the Google Maps API.

```
christian@christian:~$ cd Desktop/
christian@christian:~/Desktop$ ls
gps.py  pruebamotor.py  prueba.py
christian@christian:~/Desktop$ python gps.py
Latitude=-0.150828833333and Longitude=-78.5034138333
Latitude=-0.1508255and Longitude=-78.5034116667
Latitude=-0.150828166667and Longitude=-78.503413
Latitude=-0.150826833333and Longitude=-78.5034116667
Latitude=-0.150824666667and Longitude=-78.5034108333
Latitude=-0.1508255and Longitude=-78.5034115
```

Figure 6: Publisher node to Pubnub

On the other hand, the nodes usb_cam_node, rpLidar_node and the nodes corresponding to geolocation and georeference are executed simultaneously. The mapping process uses the georeference information to implement the SLAM algorithm. That is, the first image obtained from the Lidar is taken as a reference and as the prototype moves, the next image is compared with the first one to create the environment based on the differences. The main limitation of this method is that it has a high sensitivity in relation to relative position, especially in terms of sudden movements. Figure. 7 and 8 below contain photographs of the final prototype. This configuration is suitable for indoor aplication.

Figure 7: Real prototype,front view

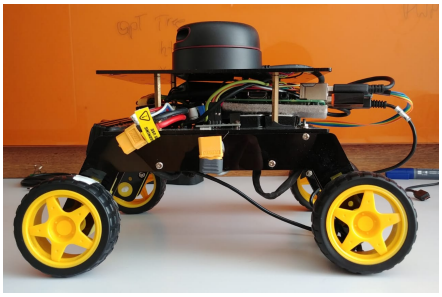


Figure 8: Real prototype, side view

Regarding the physical part of the vehicle, the implemented engines have a 1:48 gearbox with plastic pinion and biaxial shaft. The approximate weight of each engine is 35 grams, and this also carries a PWM generator that allows each engine of the prototype to reach 250 revolutions per minute [15]. However, during the navigation tests of the vehicle, we only used 30 percent of the maximum capacity, giving us about 75 revolutions per minute at 1KHz as the operating frequency.

IX. User Interface

The interface with the user can be divided into two main stages, the first one is the interface of the tracking system from the GPS module, which, as mentioned in the previous section, uses the Pubnub server together with a web page, the same shown in Fig.9 .

Figure 9: Website used for tracking

In the second stage, the navigation data is shown, for which the *rviz* tool is used. This tool takes the outputs of the Lidar, the SLAM algorithm and the camera to generate a fairly easy-to-use interface, the same one shown in Fig.10.

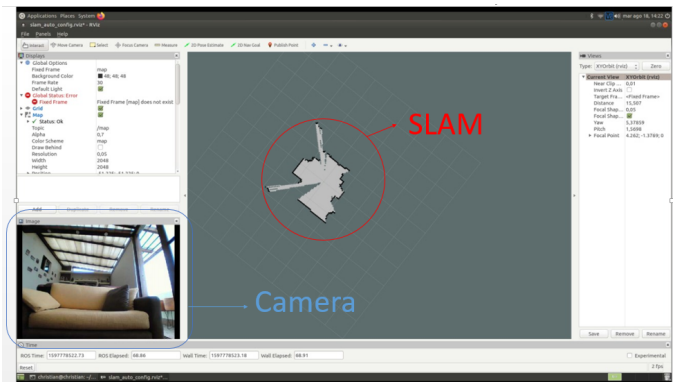


Figure 10: Rviz interface

X. Project Scalability

Thanks to the versatility of the software, it is possible to make a large number of improvements, there so this project is highly scalable. The main improvement or future work is based on the use of Machine Learning. It is possible to incorporate image recognition from the camera so that the prototype can identify and recognize the edges of the streets, traffic signs in order to achieve a higher level of autonomy. Machine Learning can be implemented in terms of the Mapping algorithm so that the precision of the results is even greater.

XI. Conclusion

A land vehicle was designed and implemented,the prototype implements different types of sensors, which allow it to move safely avoiding collisions in the automatic configuration. From the user interface it is possible to visualize in time real time the trajectory made by the vehicle in real time using Google Maps API, this Application allows to have an average error of 2.5 m.

The navigation system allows us to observe in real time the mapping, reconstruction of the place (SLAM) and a real visualization of the place (using the camera). The two batteries that you use the prototype allow it to have an autonomy of 20 minutes which allows it to cover quite large areas.

Although it is true that in the market, there are robots similar to the one presented in this report, as *TURTLEBOT3 WAFFLE PI*, they have higher prices (around USD 1300) and do not have the versatility that our proposed model. Our prototype can be easily configured for multiple and complex scenarios, all at a cost of approximately USD 500. Therefore, it is a very useful tool with great applications within the engineering world. For example, it can be used as an architecture assistant since it allows to reconstruct a place in an easy, fast and practically automatic way. In the same way, it can be used as an exploration vehicle in places of difficult access, where GPS positioning is of great importance. However, for outdoor aplications a new mechanical design would be required.

Acknowledgment

This work has been funded with Poli-Grant 16931. We are grateful to the Electrical and Electronic Engineering department of Universidad San Francisco de Quito for giving us the opportunity to highlight our skills and aptitudes by carrying out this research work.

References

[1] T. Luettel, M. Himmelsbach, and H.-J. Wuensche, "Autonomous ground vehicles—concepts and a path to the future," *Proceedings of the IEEE*, vol. 100, no. Special Centennial Issue, pp. 1831–1839, 2012.

[2] J. d. León and A. Barrientos, "Estudio de los patrones de marcha para un robot hexápodo en tareas de búsqueda y rescate," *Actas de las XXXVIII Jornadas de Automática*, 2017.

[3] R. C. Leegstra, M. De Paula, I. Carlucho, F. J. Solari, A. F. Rozenfeld, G. G. Acosta, B. V. Menna, J. Roberto, L. M. Arrien, H. J. Curti, *et al.*, "Macábot: Prototipo de vehículo autónomo de superficie (asv)," *Revista Tecnología y Ciencia*, no. 36, pp. 142–154, 2019.

[4] J. Maddox, "Improving driving safety through automation, presentation at the congressional robotics caucus," *National Highway Traffic Safety Administration*, 2012.

[5] J. E. Stranden, "Autonomous driving of a small-scale electric truck model with dynamic wireless charging," Master's thesis, Norwegian University of Science and Technology , Faculty of Engineering , Department of Engineering Cybernetics, 6 2019.

[6] J. Izquierdo-Reyes, L. A. Curiel-Ramirez, R. Bustamante-Bello, and R. A. Ramirez-Mendoza, "Perspective of autonomous driving in mexico," in *2018 International Conference on Mechatronics, Electronics and Automotive Engineering (ICMEAE)*, pp. 144–146, IEEE, 2018.

[7] SLAMTEC, *RPLIDAR A1*. SLAMTEC, 1 ed., 8 2016. Development Kit User Manual.

[8] Genius, *Eye 312 Web Cam*. Genius, 1 ed. Data sheet.

[9] U-blox, *Neo 6M*. U-blox, 7 ed., 12 2012. Data sheet.

[10] Ozzmaker, *BerryIMU Quick Start Guide*. Ozzmaker, 1 ed., 2019. Quick Start Guide.

[11] ELECFreaks, *Ultrasonic Ranging Module HC - SR04*. ELECFreaks. Data sheet.

[12] C. Fairchild and T. L. Harman, *ROS Robotics By Example: Learning to control wheeled, limbed, and flying robots using ROS Kinetic Kame*. Packt Publishing Ltd, 2017.

[13] H. Nobuhara, W. Pedrycz, and K. Hirota, "Relational image compression: optimizations through the design of fuzzy coders and yuv color space," *Soft Computing*, vol. 9, no. 6, pp. 471–479, 2005.

[14] S. Blum, D. Kong, A. Levy, B. Nainani, J. Roth, J. Oster, and G. Cohen, "Data synchronization across multiple devices connecting to multiple data centers," Apr. 24 2018. US Patent 9,955,444.

[15] TIMOTOR, *TGP01D-A130 Gear Motor*. TIMOTOR. Data sheet.

ANEXO B: VEHÍCULO AUTÓNOMO CON NAVEGACIÓN BASADA EN SENSOR LIDAR - TUTORIAL

Vehículo Autónomo con navegación basada en sensor LIDAR

Tutorial

Elaborador por: Christian Hernández y Daniel Paredes
Fecha de actualización: diciembre 2020

Contenido

1 Resumen

Este tutorial te proporcionara las herramientas necesarias para desarrollar un robot basado en ROS. Debes tener en consideración el hardware utilizado y su compatibilidad en caso de hacer modificaciones. Todo el entorno es desarrollado en ROS 1.

2 Descripción de Hardware

Para el desarrollo de este proyecto se utilizaron los siguientes dispositivos.

1. Raspberry 3B
2. Slamtech RPLIDAR A2M8
3. BerryIMU v2

4. Controlador Motores L298N (x2)
5. Motores DC TGP01D-A130 (x4)
6. GPD ublox neo-6m
7. WebCam
8. Baterías Li-Po de 3 celdas (x2)
9. DC-DC Buck Converter LM2596
10. Micro SD (16gb mínimo)
11. Modulo Wifi usb

Los dispositivos listados pueden variar si se cambia la versión de Raspberry o se ocupa otra plataforma para reproducir este proyecto.

Es importante recalcar que la versión 3B de raspberry tiene modulo Wifi incluido, sin embargo, dicho modulo no funcionaba en el dispositivo que teníamos a disposición por lo que añadimos un módulo Wifi USB.

Todos los dispositivos y módulos fueron montados en un chasis genérico.

3 Conexiones

3.1 GPIO

¿Qué es GPIO?

General Purpose Input Output (GPIO) es un sistema de entrada y salida de propósito general que permite lectura y escrituras tanto analógicas como digitales, el numero de pines de GPIO disponibles varía dependiendo la tarjeta que se ocupe para el procesamiento, en este caso un raspberry 3B, hay que tomar en cuenta que si se utiliza una versión anterior a la 3B, es posible que la cantidad de pines y su disposición cambie.

Existen 2 formas de numerar los pines de la Raspberry Pi, en modo GPIO o en modo BCM.

En el modo GPIO, los pines se numeran de forma física por el lugar que ocupan en la placa y en el modo BCM, los pines se numeran por la correspondencia en el chip Broadcom.

En la siguiente Imagen se muestra la equivalencia entre GPIO y BMC, la más utilizada es GPIO que es con la que trabajamos en el proyecto.

BOARD	GPIO		GPIO	BOARD
01	3.3v DC Power		DC Power 5v	02
03	GPIO02 (SDA1 , I²C)		DC Power 5v	04
05	GPIO03 (SCL1 , I²C)		Ground	06
07	GPIO04 (GPIO_GCLK)		(TXD0) GPIO14	08
09	Ground		(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)		(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)		Ground	14
15	GPIO22 (GPIO_GEN3)		(GPIO_GEN4) GPIO23	16
17	3.3v DC Power		(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)		Ground	20
21	GPIO09 (SPI_MISO)		(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)		(SPI_CE0_N) GPIO08	24
25	Ground		(SPI_CE1_N) GPIO07	26
27	ID_SD (I²C ID EEPROM)		(I²C ID EEPROM) ID_SC	28
29	GPIO05		Ground	30
31	GPIO06		GPIO12	32
33	GPIO13		Ground	34
35	GPIO19		GPIO16	36
37	GPIO26		GPIO20	38
39	Ground		GPIO21	40

La figura muestra la conexión de los pines GPIO a los controladores de motor, la manera correcta de alimentar el proyecto y conexión serial a algunos periféricos.

3.2 USB

- RPLIDAR A2M8
- Modulo GPS

Nota: La conexión para el módulo GPS puede ser Serial UART, pero recomendamos usar un módulo puente UART-USB debido a que este no genera problemas con la comunicación por bluetooth.

3.3 Alimentación

Se requiere 2 baterías Li-Po de 3 celdas. La primera energiza los controladores de motor y la otra alimenta el resto del sistema, es decir, al raspberry y todos sus periféricos.
Una batería de 3 celdas tiene un voltaje nominal de 11.1 V (3,7v por celda) y en el caso específico del proyecto, usamos baterías con capacidad de 1000 mAh.

El voltaje máximo que se puede suministrar a un raspberry es de 5v (±0.25v) y una corriente máxima de 2.5A, para lograr este valor utilizamos un conversor DC-DC que suministre un voltaje que se puede fijar a 5.15v. Se utiliza ese voltaje debido a que todos los periféricos conectados provocan una caída de voltaje significativa. Con 5.15v aseguramos que al correr todos los nodos del programa el voltaje en la entrada se mantenga en 5v.

La conexión se realiza desde la batería al módulo donde se regula el voltaje de salida usando un multímetro y luego se conecta a los pines de 5v y GND más cercano (revisar el diagrama de GPIO para encontrar los pines).

Es importante colocar el conversor apartado del resto de dispositivos ya que su temperatura sube mucho y podría estropear componentes.

4 Guía Rápida de instalación

Se puede instalar la versión final del proyecto, totalmente funcional.

Se debe copiar el archivo con extensión .img que esta almacenado en la tarjeta micro sd (incluida en los materiales del proyecto) al escritorio o algún directorio de un computador y seguir los pasos de la sección 5.1 y 5.2.

Esto permite explorar el proyecto y conocer sus características y limitaciones, sin embargo, no es un sustituto al resto del tutorial ya que te saltas todo el aprendizaje de ROS, Linux.

5 Puesta en marcha

5.1 Descarga RaspBerry Pi imager y Ubuntu Mate

RaspberryPi Imager es un software que te permite insertar/preinstalar el sistema operativo en la memoria sd, se puede ocupar alternativas como Win32Diskimager y funcionan correctamente, pero la ventaja con el software de raspberry es que puedes descargar otros sistemas operativos desde la misma app, sin embargo, ya que ocupamos una versión de Ubuntu hecha específicamente para procesadores basados en arquitectura ARM, debemos descargar el OS desde la página de Ubuntu.

Puedes descargar directamente el software desde este enlace:

https://downloads.raspberrypi.org/imager/imager_1.4.exe

Debido a que la versión de Ubuntu mate que usamos en este proyecto no es la más actual, es poco difícil encontrarla en página de Ubuntu, este enlace te envía al repositorio de

versiones antiguas y la que recomendamos elegir es ubuntu-mate-16.04-desktop-armhf-raspberry-pi, la descarga puede ser directa o por Torrent.

https://releases.ubuntu-mate.org/archived/xenial/armhf/

una vez descargada, debes descomprimir el archivo, trabajaremos con el archivo con extensión .img

5.2 Escritura de OS en micro sd

Conecta tu memoria micro sd (mínimo 16gb recomendado) y abre el software raspberry Pi imager, seleccionar su dispositivo de almacenamiento y en sistema operativo elige la opción de **formateo** y luego click al botón WRITE.

Una vez finalizado, se repite el proceso, pero ahora seleccionas en sistema operativo la opción **USE CUSTOM** y seleccionas el archivo .img que descargaste y Click en write.

Una vez terminado el proceso, expulsa la micro sd del computador e insértala en el raspberry.

5.3 Configuración inicial

Una vez insertada la memoria microSD en la raspberry, se debe conectar a un monitor, ratón, teclado y a internet por medio de cable ethernet. Si no dispones de todos los dispositivos puedes acceder por ssh.

Realiza la configuración inicial de usuario y contraseña. una vez que te encuentres en el escritorio de Linux, Habilita la opción de inicio de sesión automático y por último, ejecuta estos comandos.

(Cada uno se debe ejecutar después de que el anterior haya terminado su trabajo.)

```
sudo apt-get update
sudo apt-get upgrade
sudo reboot
```

5.4 VNC Server

Se utiliza este programa para controlar de manera remota el raspberry con un entorno gráfico. Esto evita que se necesite conectar periféricos como ratón y teclado.

Se inicia un servidor que nos da control total del escritorio por medio de un computador Windows, MAC e incluso a través de aplicaciones móviles en Android o IOS.

Primero se descarga el instalador (archivo con extensión .deb) que puedes ejecutar a través del instalador de paquetes predeterminado de Ubuntu.

https://www.realvnc.com/en/connect/download/vnc/

En esta página seleccionas la plataforma raspberry y descargas el archivo. Una vez descargado se abre directamente con el instalador.

También es necesario crear una cuenta realvnc para conseguir la licencia gratuita.

Al terminar la instalación del paquete, abre el terminal y ejecuta este comando.

```
Vnclicensewiz
```

Se abrirá una ventana de configuración donde es necesario iniciar sesión con la cuenta recién creada y aceptar términos y condiciones.

Luego ejecuta estos comandos:

```
sudo systemctl start vncserver-x11-serviced.service
sudo systemctl enable vncserver-x11-serviced.service
```

al terminar, aparecerá un icono en la barra de tareas de color blanco.

Ejecutar el siguiente comando

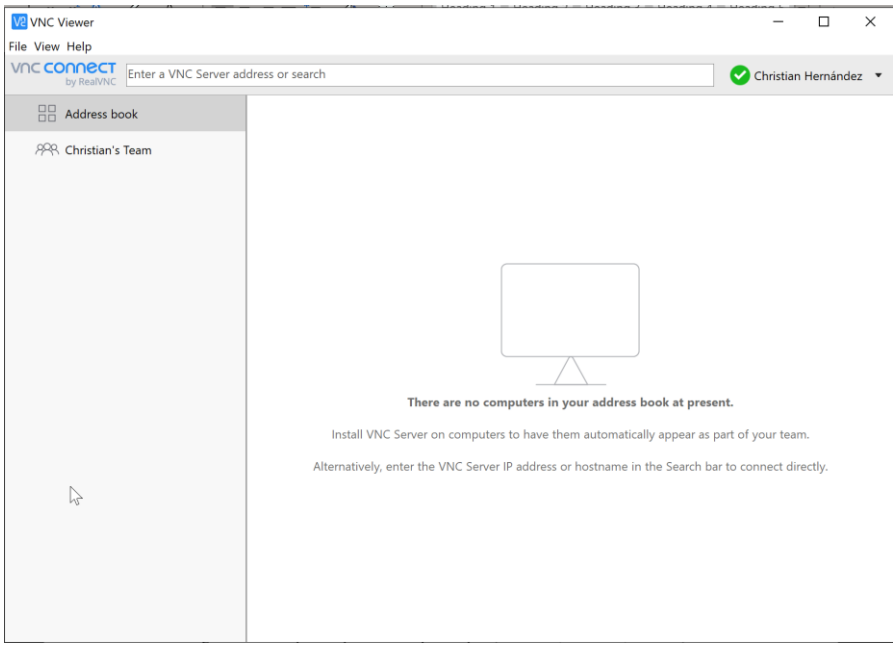
```
sudo reboot
```

Ahora es necesario instalar el cliente VNC que accederá al raspberry de forma remota.

Descarga el software para la plataforma que ocupes (Windows, MACOS, LINUX).

https://www.realvnc.com/en/connect/download/viewer/

una vez instalado se puede ver esta ventana



Para una funcionalidad completa, inicia sesión con tu cuenta de RealVnc. Este servicio actualiza la ip asignada al Raspberry por el router automáticamente para que no sea necesario escribir las credenciales en cada conexión.

Click en usuario-equipo (Christian’s Team) y debe aparecer el raspberry configurado. Solo hace falta dar click e ingresar el usuario administrador y contraseña.

Aparecerá el escritorio y lo podrás controlar.

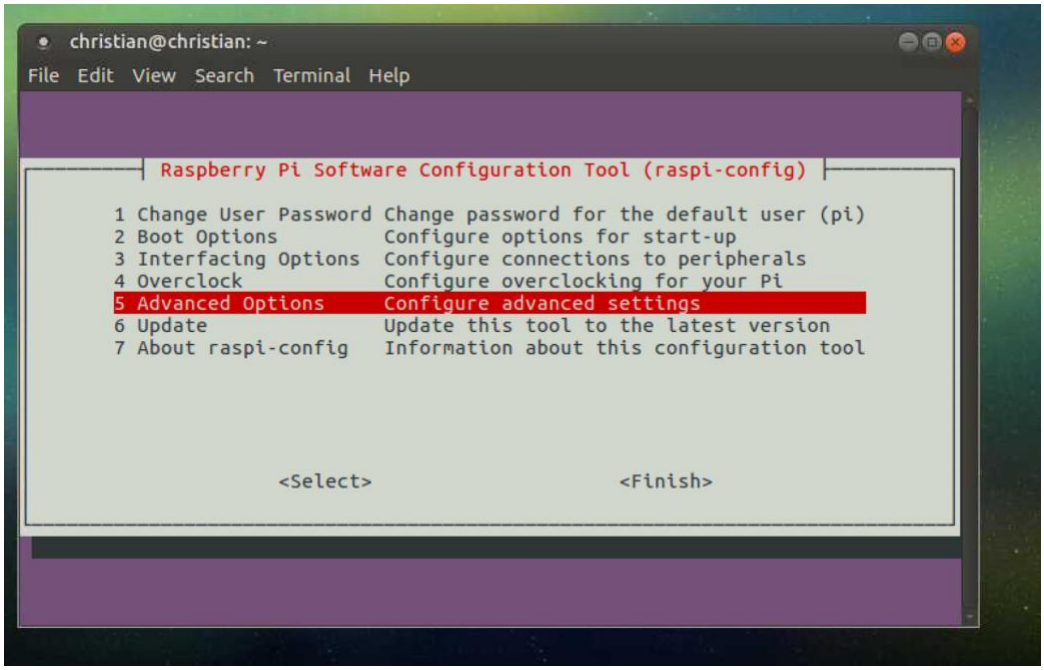
Si no quieres iniciar sesión en RealVnc viewer o lo vas a usar en un dispositivo aparte, puedes escribir la dirección IP del Raspberry en la barra de la figura anterior y te pedirá las credenciales de inicio de sesión.
Para conocer el IP asignado puedes ocupar el CMD de Windows o instalar la aplicación **Fing** para Androiid y IOS que escanea y muestra Ip de cada dispositivo conectado a tu red.

Después de este punto ya no es necesario que se mantenga conectado el monitor, mouse y teclado ya que se controlara de manera remota, es importante recalcar que VNC server solo funciona si el raspberry está conectado a la red por lo que el **cable ethernet o la conexión wifi deben estar activas y enlazadas**.

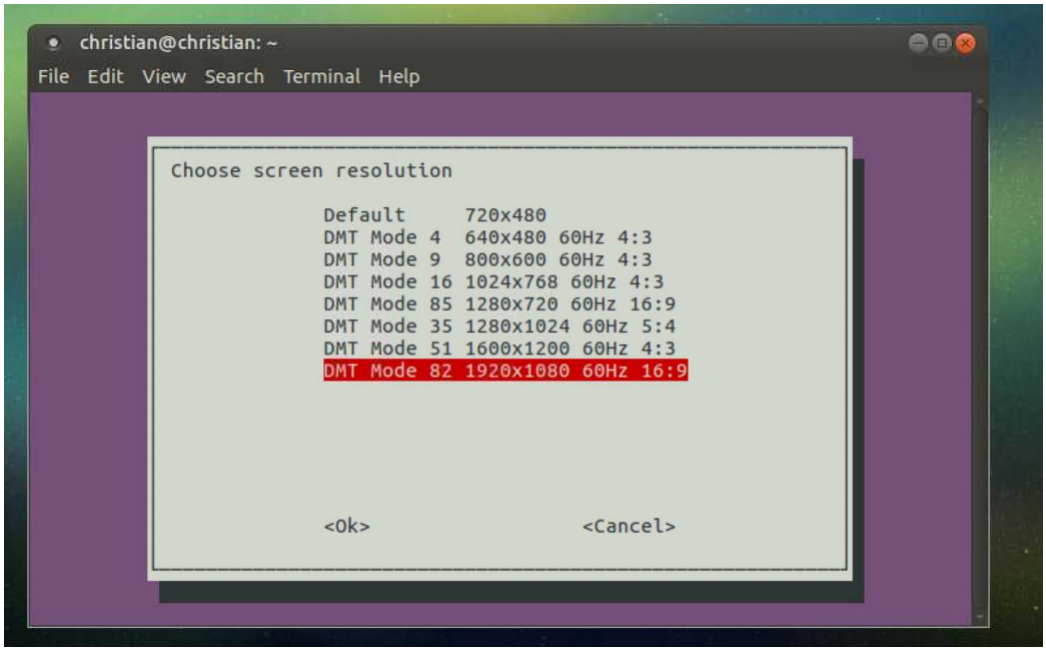
Como configuración adicional, se define la resolución fija para transmitir por el servidor. Usualmente esta se define por la resolución del monitor, pero si el dispositivo inicia sin un monitor, se configura por defecto con la mínima resolución.
Ejecutar estos comandos

```
sudo apt-get install raspi-config
sudo raspi-config
```

Aparecerá esta Ventana, con el teclado seleccionamos **advanced options** y luego **resolution**.



Puedes elegir la resolución que quieras, entre más, mejor.



Al finalizar te pedirá que reinicies la máquina.

5.5 Conexión wifi (opcional)

Esta configuración aplica solo si usas exactamente el mismo raspberry que usamos durante el proyecto. Nuestra unidad tenía un desperfecto de hardware por lo que el wifi no es reconocido.

Si usas otro raspberry (incluso el mismo modelo) y el wifi funciona correctamente (se conecta a la red) **puedes saltar esta sección**.

Conecta el módulo USB WIFI al raspberry y ejecuta estos comandos

```
sudo wget http://downloads.fars-robotics.net/wifi-
drivers/install-wifi -O /usr/bin/install-wifi
sudo chmod +x /usr/bin/install-wifi
sudo install-wifi -h
sudo install-wifi
sudo reboot
```

Al reiniciar, las funciones del wifi deben estar completamente habilitadas. Estos comandos funcionan con cualquier módulo wifi que ocupes ya que buscará en un repositorio, los drivers necesarios para tu wifi (modulo - hardware) dependiendo del kernel que utilices.

6 ROS

6.1 Instalación

Robot Operating System es un conjunto de librerías y herramientas de visualización y software que proporciona servicios diseñados para un clúster de computadoras y manejo de dispositivos de bajo nivel. Es compatible con Linux, Windows (tiene soporte limitado). Para más información sobre ROS puedes visitar su página web donde encontraras tutoriales y documentación, también puedes leer el proyecto de titulación asociado a este tutorial, DISEÑO E IMPLEMENTACIÓN DE UN VEHICULO AUTÓNOMO CON NAVEGACIÓN BASADA EN SENSORES LIDAR.

Antes de instalar ROS ejecuta estos comandos, sirven para instalar el software **htop** para monitoreo de recursos y **gedit** que es un excelente editor de texto.

```
sudo apt-get install gedit
sudo apt-get install htop
```

Esta sección del tutorial fue tomada de la página oficial de ROS, puedes acceder por este enlace: http://wiki.ros.org/kinetic/Installation/Ubuntu

Cada comando debe ejecutarse en orden, a diferencia del tutorial original, en este solo se muestran los comandos necesarios para que ros funcione en esta distribución de Linux y con las configuraciones necesarias.

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
```

```
sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80'
--recv-key C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654

sudo apt-get update

sudo apt-get install ros-kinetic-desktop-full

echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc

source ~/.bashrc

sudo apt install python-rosdep python-rosinstall python-
rosinstall-generator python-wstool build-essential

sudo apt install python-rosdep

sudo rosdep init

rosdep update
```

6.2 Crear un espacio de trabajo (workspace)

Un espacio de trabajo es un lugar donde se alojan los paquetes instalados y creados y de donde se van a ejecutar los nodos en el futuro. Es recomendable usar un espacio de trabajo por proyecto.

Ejecuta estos comandos.

```
$ source /opt/ros/kinetic/setup.bash
```

Ahora, procedemos a crear el workspace llamado catkin

```
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws/
$ catkin_make
```

Utilizamos el comando **catkin_make** para compilar paquetes en este workspace. Al ejercutarlo por primera vez, se creará un enlace CMakeLists.txt dentro de la carpeta 'src'. Además, dentro del directorio se crearan las carpetas 'build' y 'devel', y en esta última se pueden encontrar diferentes archivos setup.*sh que superponen el workspace con el entorno del sistema. Una vez terminados estos pasos, hay que generar un nuevo archivo setup.*sh

```
$ source devel/setup.bash
```

Esta línea es extremadamente importante, con ella defines en ROS con que workspace trabajar. Se debe ejecutar la línea dentro del directorio catkin_ws, caso contrario no funcionará. Si te olvidas de ejecutar la línea antes de correr los nodos, ros no iniciará. Solo debe ejecutarse una vez por inicio de sistema.

Para terminar, debemos de asegurarnos de que el workspace esté superpuesto por el script de configuración, para lo cual, debemos comprobar que la variable de entorno ROS_PACKAGE_PATH incluya el directorio en el que se encuentra.

```
$ echo $ROS_PACKAGE_PATH
```

```
El resultado será este
```

```
/home/youruser/catkin_ws/src:/opt/ros/kinetic/share
```

6.3 Crear un paquete de ROS

El primer paso debemos de dirigirnos al directorio de origen donde generamos el workspace de catkin.

```
$ cd ~/catkin_ws/src
```

Después, utilizamos el comando catkin_create_pkg para crear un nuevo paquete que se llamara ‘beginner_tutorials’ el cual dependerá de std_msgs, roscpp y rospy:

```
$ catkin_create_pkg beginner_tutorials std_msgs rospy roscpp
```

Con esto se generará una carpeta ‘beginner_tutorials’, la cual contendrá los archivos package.xml y CMakeLists.txt. Estos archivos poseen información brindada en catkin_create_pkg.

catkin_create_pkg necesita que le brindemos un ‘package_name’ y opcionalmente una lista de condiciones o dependencias del mismo paquete.

```
# catkin_create_pkg <package_name> [depend1] [depend2] [depend3]
```

Ahora, necesitamos crear los paquetes en el workspace catkin

```
$ cd ~/catkin_ws
$ catkin_make
```

Una vez hecho esto, debemos de añadir el workspace en el entorno de ROS, para lo cual debemos de obtener el archivo de configuración generado

```
$ . ~/catkin_ws/devel/setup.bash
```

7 Códigos del proyecto y ejecución de nodos

En este punto, nuestro sistema ya es capaz de correr el proyecto completo con un par de comandos. Todos los códigos de encuentran en la memoria SD entregada junto con este documento. Son scripts de Python que controlan el funcionamiento de los motores, la lectura del control por bluetooth y también scripts con extensión .launch en lenguaje HTML que definen la comunicación de nodos y parámetros de inicialización del proyecto.

Todos los códigos están comentados y son fáciles de entender por lo que recomendamos leerlos, ejecutarlos y cambiar valores de los parámetros para entender perfectamente su funcionamiento.

Para poder ejecutar el programa primero debes conectar el mando por bluetooth, prender la alimentación de los motores y asegurarse que la cámara y el LIDAR este conectados por USB.

Para ejecutar la versión final del proyecto debemos usar estos comandos:

```
Cd ~/catkin_ws
Source /devel/setup.bash
sudo chmod 666 /dev/ttyLIDAR
roslaunch vehiculoauto mapeo_manual.launch
```

8 Paquetes adicionales

Se usaron 2 paquetes en el proyecto que pueden ser reemplazados en caso de cambio de hardware, utilidad de proyecto o conveniencia del grupo de trabajo.

Hector_SLAM paquete para mapeo y localización simultaneas y pygame para la lectura del control por bluetooth son utilizados en este proyecto por las actuales limitaciones de hardware, sin embargo, podrías ser cambiadas por paquetes similares.

Recomendamos leer la documentación sobre cada uno para mas detalles sobre las ventajas y desventajas sobre otras alternativas.

Para instalarlas usa estos comandos:

PYGAME:

```
sudo apt-get install python-pygame
```

Hector SLAM

Opcion 1, instalar con el gestor de paquetes apt:

```
sudo apt-get install ros-kinetic-hector-slam
```

tener en cuenta la distribución <kinetic>, en caso de ser diferente debes cambiar al nombre de la distribución de ros que utilizas.

El método 1 puede fallar debido que no todos los paquetes en todas las versiones pueden ser instaladas a través de apt. En ese caso debemos compilarlo desde la fuente.

Opción 2, compilación desde la fuente:

```
source ~/catkin_ws/devel/setup.bash
cd ~/catkin_ws/src
git clone https://github.com/tu-darmstadt-ros-pkg/hector_slam.git
catkin build
```

IMPORTANTE – la instalación de hector slam toma muchísimo tiempo, dependiendo de la versión de raspberry puede tomar más de 2 horas.