UNIVERSIDAD SAN FRANCISCO DE QUITO USFQ

Colegio de Ciencias e Ingeniería

Diseño e implementación de un sistema de automatización para un vehículo eléctrico mediante el uso de un clúster jerárquico de dos niveles

Dennis Andrés Parra Serrano

Daniel Sebastián Paredes Monge

Ingeniería Electrónica

Trabajo de fin de carrera presentado como requisito

para la obtención del título de

Ingeniero Electrónico

Quito 23 de mayo de 2021

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 sistema de automatización para un vehículo eléctrico mediante el uso de un clúster jerárquico de dos niveles

Dennis Andrés Parra Serrano

Daniel Sebastián Paredes Monge

René Játiva Espinoza, Ph.D.

Quito 23 de mayo de 2021

© DERECHOS DE AUTOR

Por medio del presente documento certifico que he leído todas las Políticas y Manuales de la Universidad San Francisco de Quito USFQ, incluyendo la Política de Propiedad Intelectual USFQ, y estoy de acuerdo con su contenido, por lo que los derechos de propiedad intelectual del presente trabajo quedan sujetos a lo dispuesto en esas Políticas.

Asimismo, autorizo a la USFQ para que realice la digitalización y publicación de este trabajo en el repositorio virtual, de conformidad a lo dispuesto en la Ley Orgánica de Educación Superior del Ecuador.

Nombres y apellidos: Daniel Sebastián Paredes Monge

Código: 00137219

Cédula de identidad: 1727179143

Lugar y fecha: Quito, 23 de mayo de 2021

Nombres y apellidos: Dennis Andrés Parra Serrano

Código: 00133633

Cédula de identidad: 1726224585

Lugar y fecha: Quito, 23 de mayo de 2021

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UNPUBLISHED DOCUMENT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 automatizado un vehículo eléctrico de dimensiones 1x2x0.30 metros a través de la implementación de un sistema de navegación a control remoto, el cual también brinda la posibilidad de observar el entorno del prototipo mediante el uso de sensores y cámaras. Todas las funciones necesarias para controlar el vehículo se procesan mediante un clúster de dos niveles conformado por un Raspberry Pi 4B y un Intel NUC 8i5beh, enlazados en red a través de las herramientas de ROS. La versatilidad del software permite que nuestro diseño pueda cumplir distintas tareas, como bien puede ser el mapeo de zonas o el transporte de materiales u objetos únicamente configurando el software.

Palabras clave: Vehículo eléctrico, Clúster, Mapeo laser, Sensores de precisión, ROS.

ABSTRACT

An electric vehicle with dimensions 1x2x0.30 meters has been automated through the implementation of a remote-controlled navigation system, which also provides the possibility of observing the prototype environment using sensors and cameras. All the functions necessary to control the vehicle are processed by a two-tier cluster made up of a Raspberry Pi 4B and an Intel NUC 8i5beh, linked in a network through ROS tools. The versatility of the software allows our design to fulfill different tasks, such as the mapping of zones or the transport of materials or objects only by configuring the software.

Keywords: Electric vehicle, Cluster, Laser mapping, Precision sensors, ROS.

Índice de contenido

Índice de figuras

Índice de tablas

Introducción

La evolución de los vehículos eléctricos ha avanzado a un paso acelerado durante los últimos años, ya que ha cambiado la forma en la que las personas viajan y transportan cosas a distintos lugares. También, gracias a los avances tecnológicos en otras áreas, como por ejemplo el internet de las cosas y la constante disminución del tamaño de procesadores y computadores, tenemos la disponibilidad de muchas puertas para la generación de nuevas ideas y oportunidades para el avance de tecnologías ya existentes.

En esta etapa del proyecto, la visión es utilizar una pequeña computadora lo suficientemente potente y colocarla en un vehículo eléctrico comercialmente diseñado para la recreación de niños de 3-4 años, y modificarlo para proporcionarle cierta autonomía. Este vehículo se maneja remotamente con un dualshock 4, que es un control de ps4. Para esto, se utiliza un clúster jerárquico de dos niveles conformado por un Raspberry Pi 4B y un Intel NUC 8i5beh, los cuales manejarán por separado el sistema de potencia y el de control respectivamente. Al utilizar estos recursos, el procesamiento de información que fluye a través de todo el sistema resulta optimizado, dando un grado de versatilidad para que en el prototipo se puedan implementar nuevas funciones sin que el rendimiento se vea afectado, ya que se optimiza cada proceso que realiza el carrito entre ambos dispositivos.

Respecto a la automatización del vehículo, se utilizaron dos motores que controlan su movimiento en cuatro direcciones (izquierda, derecha, adelante y atrás) proporcionándole libertad de desplazamiento sobre los cuatro ejes. Para lograr este funcionamiento se utilizaron dos drivers con puentes H integrados que ayudan a controlar la cantidad de potencia que se entrega a los motores respectivamente y mediante las señales de los GPIO (General Purpose Input/Output, Entrada/Salida de Propósito

General) del Raspberry se manipula la velocidad de estos motores. Al usar este tipo de drivers no es necesario calcular la cantidad exacta de voltaje y corriente que necesitan los motores ya que estos componentes administran por si solo los recursos necesarios para la conducción del vehículo, sin embargo, para manipular la velocidad de los motores con mayor facilidad colocamos reguladores de voltaje (step-down) que pueden llegar a regular la velocidad de los motores según el voltaje que venga desde la batería, ya que la cantidad de revoluciones por minuto (RPM) a las cuales trabajan estos motores es directamente proporcional al nivel de voltaje que se está suministrando.

El trabajo descrito en este proyecto se centra específicamente en dos aspectos del diseño y construcción del vehículo eléctrico: el clúster de dos niveles que se encarga de controlar los sistemas de navegación y su infraestructura, y el diseño de control de los motores y cómo se lo puede controlar a través de una microcomputadora como un Raspberry Pi. Al dividir el proyecto en estas dos partes se debe asegurar el sistema de movimiento de los motores en primer lugar como prioridad y una vez establecido dicho control, podemos pasar a utilizar la minicomputadora NUC que es mucho más potente para que utilice los sensores y una cámara 3D para brindar datos e imagen acerca del entorno del vehículo.

Tanto el hardware como el software del carrito están pensados para que haya un nivel alto de modularidad en el diseño y que éste tenga la facilidad de mejorar cualquier ámbito interno o externo del carrito dotándole a su vez de un alto nivel de escalabilidad, con el fin de realizar una continua investigación y seguir aumentando su nivel de autonomía.

Marco Teórico

Clúster:

Un clúster es un conjunto de computadoras que trabajan paralelamente mediante una conexión en red, la cual permite que en varios aspectos puedan considerarse como un sistema operativo único. En la actualidad, se utilizan varios dispositivos de la misma gama para diseñar un clúster con el fin de reducir los tiempos de carga en cada dispositivo y aumentar el rendimiento de sus respectivos procesos.

Un claro ejemplo son los clústeres basados en Raspberry pi como el de la Figura 1, ya que estos son computadoras con un nivel de procesamiento bastante limitado, por lo que al juntar más de uno, se puede dividir el procesamiento para que todos trabajen conjuntamente y se puedan generar resultados de manera más rápida.

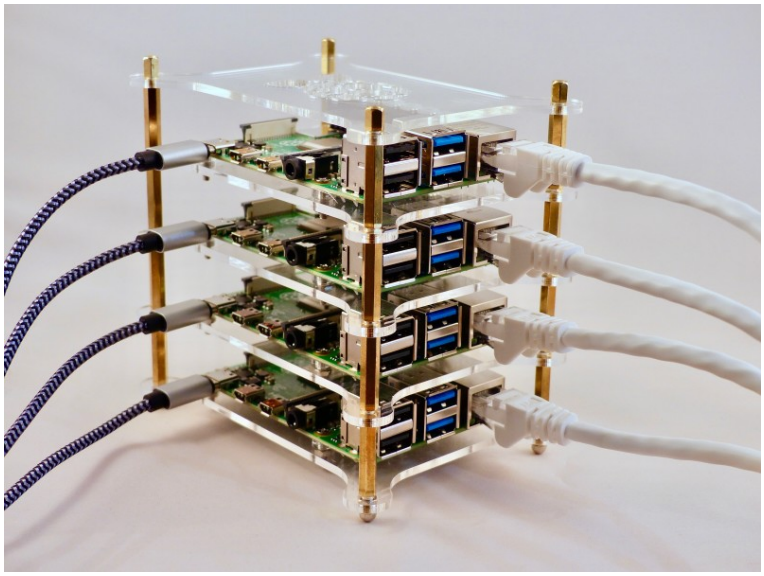


Figura 1: Arquitectura de un clúster

En este tipo de clúster, todas las computadoras están conectadas mediante cable de red a un mismo módulo de conexión en red, sea un switch o un router. Una vez hecho esto, se instala el mismo sistema operativo a todos los dispositivos del clúster. En estos casos suele seleccionarse Raspbian al ser un software bastante versátil, o Ubuntu que es

una distribución de Linux enfocada al desarrollo de aplicaciones o procesos complejos, como se muestra en la Figura 2.

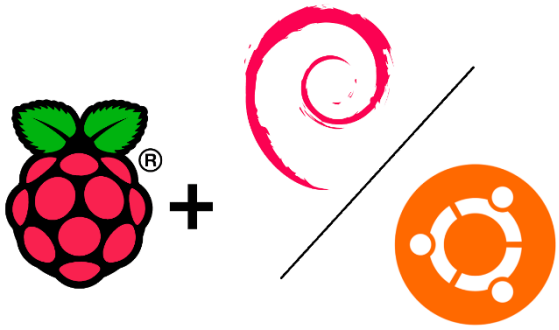


Figura 2: Raspberry + Raspbian/Ubuntu

Luego de terminar con la instalación de los OS de cada “sub-computador”, debemos asignar direcciones IP estáticas a cada uno de ellos, con el fin de dirigir la información hacia un destino deseado. El uso de IPs dinámicas no se recomienda puesto que la dirección que identifica a cada dispositivo por separado podría variar al momento de encender el módulo de conexión en red.

Para Ubuntu, la dirección IP estática se asigna desde el centro de control como se muestra en la Figura 3. Dentro de la opción de configuración avanzada de redes, seleccionamos como tipo de conexión cableada, y ponemos dentro de los ajustes de IPv4 no sea automático sino manual, es ahí donde procedemos a configurar la IP que va a identificar el dispositivo seleccionado

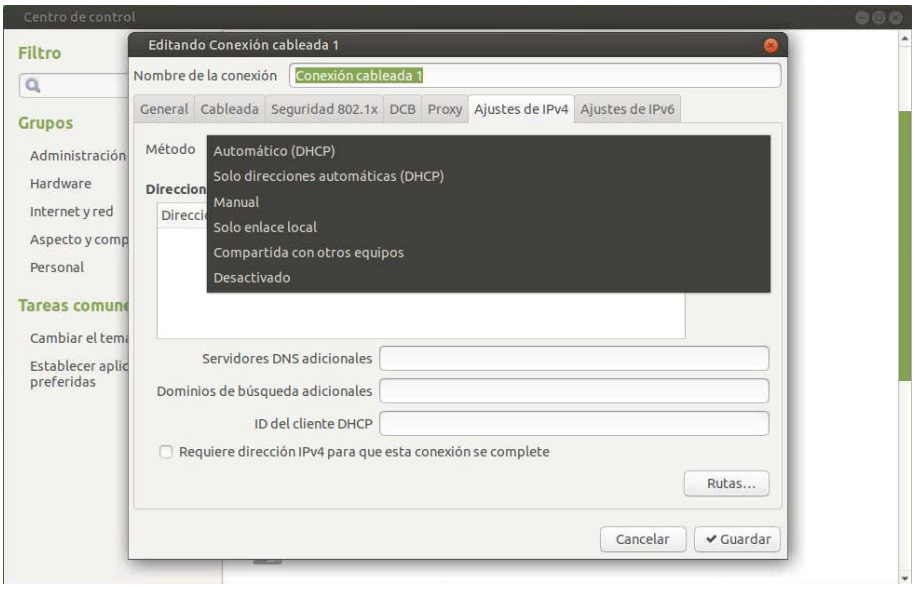


Figura 3: Centro de control de conexiones dentro de Ubuntu Mate 20.04

Una vez hecho esto, debemos de configurar lo que se llama “nodo master”, el cual es el dispositivo que va a manipular al resto, siendo que los otros se llamen “nodos slaves”. Este tipo de configuración hace posible la división de procesos, ya que el master redirige cada tarea hacia uno o más slaves, haciendo que cada sub-computador del clúster esté realizando alguna tarea paralelamente con los demás, y los resultados son mostrados en el nodo master ya que ese sería también como la central de control donde podemos ver que se encuentra haciendo cada computador que compone el clúster. Esta relación se ilustra en la Figura 4.

Figura 4: Enlace entre master y slave

Para nuestro proyecto decidimos replantear este concepto y diseñar un clúster de dos niveles, compuesto por dos computadoras que sean distintas pero que de igual manera puedan dividirse los procesos y aplicaciones mutuamente para formar un sistema operativo mucho más potente. Estamos hablando de un OS (sistema operativo) compuesto por el procesamiento en paralelo de un Intel NUC y un Raspberry pi 4 como el de la Figura 5. Para este enlace, utilizamos ROS y dividimos el OS en dos. La primera parte es el sistema de control y procesamiento de información que corre en el NUC, y la segunda parte es el sistema de potencia y manejo de los motores que corre en el Raspberry gracias a sus GPIO.

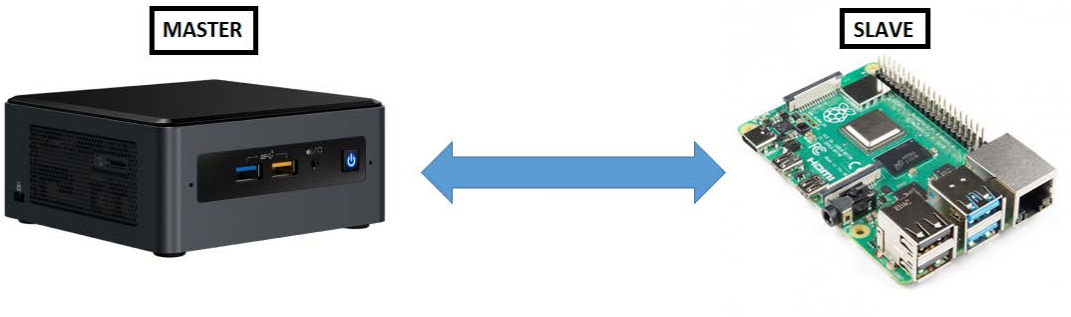


Figura 5: Disposición master y slave del prototipo

ROS:

Robot Operating System (ROS) es un entorno de trabajo que conlleva distintas librerías dirigidas para el diseño de software de robots. Dependiendo de los requerimientos, ROS puede abastecer a cualquier usuario de distintas herramientas para simplificar la tarea de diseñar las diferentes mecánicas que puede desempeñar un robot. Su logotipo se muestra en la Figura 6.

Figura 6: Logotipo de Robot Operating System

ROS es considerado un “meta-sistema operativo” ya que posee ciertas características propias de un OS como la abstracción de hardware, control de dispositivos de bajo nivel, administración de paquetes, entre otros. Pero, a pesar de esto, ROS debe instalarse como una aplicación de un sistema operativo para poder funcionar. Para la facilidad del usuario, ROS es compatible con distintas distribuciones de Linux, Mac OS, y de manera experimental para Windows.

El funcionamiento de ROS conlleva la división de los procesos del sistema en nodos que se comunican a través de tópicos. Éstos pueden estar escritos en distintos lenguajes de programación, ya que ROS permite la versatilidad de poder realizar la transmisión de datos e información entre nodos sin importar si ambos no se encuentran bajo la misma programación. Los nodos pueden ser de dos tipos, si un nodo es Publisher, significa que emite información a cualquier nodo desde la salida de su script, y si un nodo es Subscriber, quiere decir que recibe información de la salida de un nodo y la utiliza como entrada para su script. (Hernandez, Barbosa, Paredes, & Játiva, 2020)

Es importante recalcar que todos los nodos son manipulados desde el ROS_MASTER, el cual gestiona los caminos a los que debe dirigirse la información de cada nodo respectivamente, este se activa una vez iniciemos ROS en nuestro OS. A continuación, en la Figura 7 se muestra un esquema jerárquico sobre su funcionamiento.

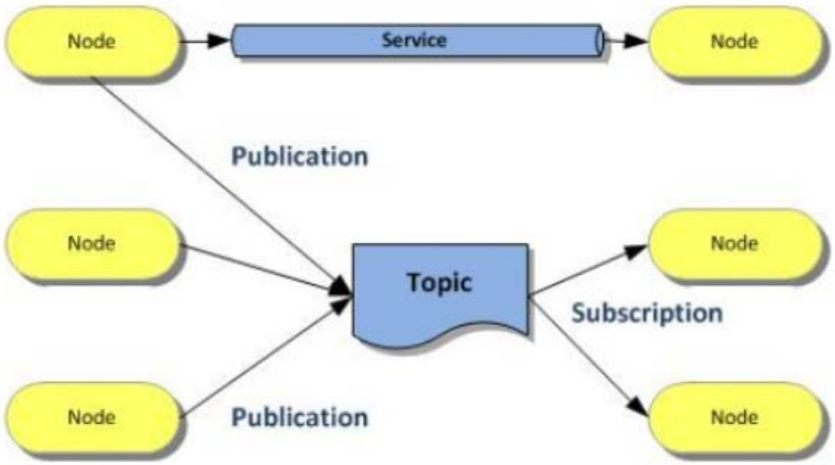


Figura 7: Esquema de funcionamiento de los nodos de ROS

Ahora que introducimos ROS como el entorno de diseño para nuestro prototipo, procedemos a explicar los nodos que hacen posible la funcionalidad de nuestro vehículo, y de manera paralela explicamos sobre como utilizamos el método de ROS Network para generar la comunicación entre el Intel NUC y el Raspberry pi 4.

Rplidar_ROS

El rplidarNode es un driver o controlador destinado para el RPLIDAR con la finalidad de tomar todo el muestreo que realiza este sensor, a través de su aplicación SDK, y convertirlo en información procesable para ROS. Este nodo fue creado por Slamtec, empresa que desarrolla los sensores tipo láser como el sensor lidar A2 que usamos en este proyecto. El resultado de este nodo se muestra en la Figura 8.

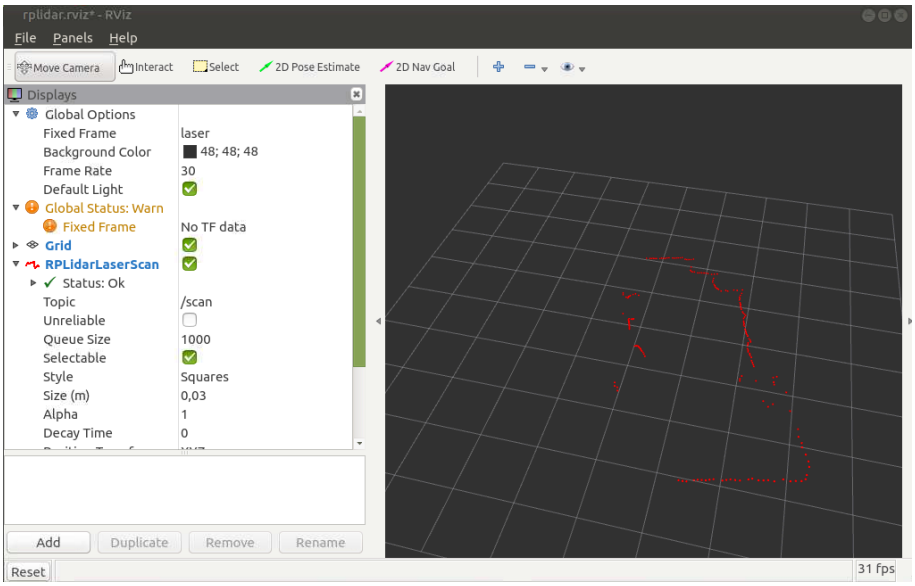


Figura 8: Visualización en Rviz del funcionamiento del nodo rplidar

Como bien podemos observar, al tratarse de un sensor omnidireccional de 360 grados, vemos que el láser permite marcar todos los límites del entorno por el que se encuentra atravesando el sensor. Esta visualización la vemos desde la herramienta de visualización 3D de ROS llamada Rviz, que es parte de los paquetes de la instalación.

Hector SLAM

Este nodo contiene un algoritmo que utiliza el muestreo y posición de un sensor láser para proporcionar una reconstrucción 2D de un entorno. En este caso, utiliza la información del sensor LIDAR A2 para que los límites que marcaba en el caso anterior sirvan como molde para la generación de un mapa con el entorno donde se encuentra el sensor, podemos ver de igual manera este mapeo con la herramienta Rviz en la Figura 9.

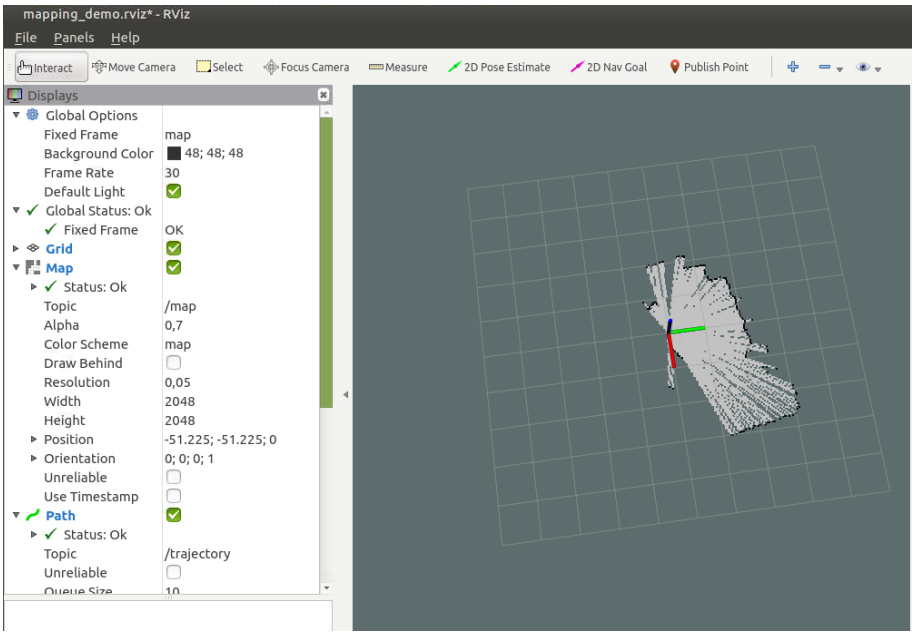


Figura 9: Visualización en rviz del funcionamiento del nodo SLAM

ZED_ROS_wrapper

Se trata de un nodo creado por StereoLabs, empresa que desarrolla cámaras 3D, que permite el funcionamiento de la cámara ZED 2 mediante ROS, además de proporcionar información sobre la odometría visual, mapa de profundidad, seguimiento de posición e incluso datos IMU (Inertial Measurement Unit), gracias a que esta cámara contiene un sensor que incluye acelerómetro, barómetro y giroscopio que trabajan paralelamente para la recolección de información sobre el entorno de la cámara.

Sin embargo, este sensor es únicamente funcional si se tiene a disposición una GPU, ya que este nodo trabaja mediante CUDA, una plataforma de cómputo en paralelo, creada por NVIDIA, que permite aprovechar la potencia de cálculo de una tarjeta gráfica para realizar distintas aplicaciones, como sería en nuestro caso el procesamiento de imagen y video que puede realizar la cámara y convertirlo a información manipulable mediante ROS. Por lo tanto, en esta etapa del proyecto se dispuso de la cámara sin anexarla a ROS. Una vista tomada con esta cámara se muestra en la Figura 10.

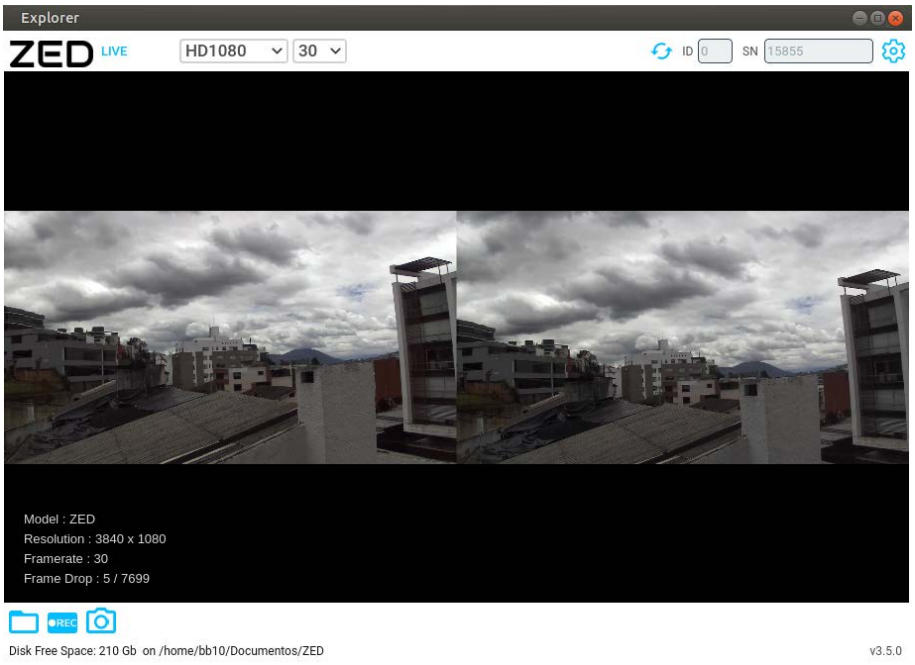


Figura 10: Toma de pantalla con la visualización de la cámara 3D

ROS NETWORK

Este método trata de utilizar el ROS_MASTER_URI para configurar distintos dispositivos bajo un mismo master. Esto se puede dar sin importar la distribución de ROS que estemos utilizando (Kinetic, Melodic o Noetic).

ROS es un entorno que está pensado para la informática distribuida, ya que un nodo no hace suposiciones sobre el camino que debe tomar para ejecutarse, sino que es capaz de correrse atravesando el mismo camino que recorre otro nodo paralelamente. Esta configuración es sencilla ya que únicamente se requiere de dos cosas: primero, que todos los dispositivos estén configurados bajo el mismo “maestro”, es decir que las aplicaciones que se ejecutan en los dispositivos estén ejecutándose en una misma red, y a su vez de pueda visualizar dentro del dispositivo master, en nuestro caso en el NUC; y, segundo, una conectividad bidireccional entre cada máquina y el master, para lo cual se debe

reconocer la dirección IP de cada dispositivo con el fin de que el ROS_MASTER_URI pueda identificar donde se encuentra ejecutándose cada nodo.

Diseño del prototipo

Para el desarrollo de este proyecto, en primer lugar se desarrolló un diagrama de bloques modular en el cual se indican relaciones que debe tener cada componente del vehículo y que se muestra en la Figura 11.

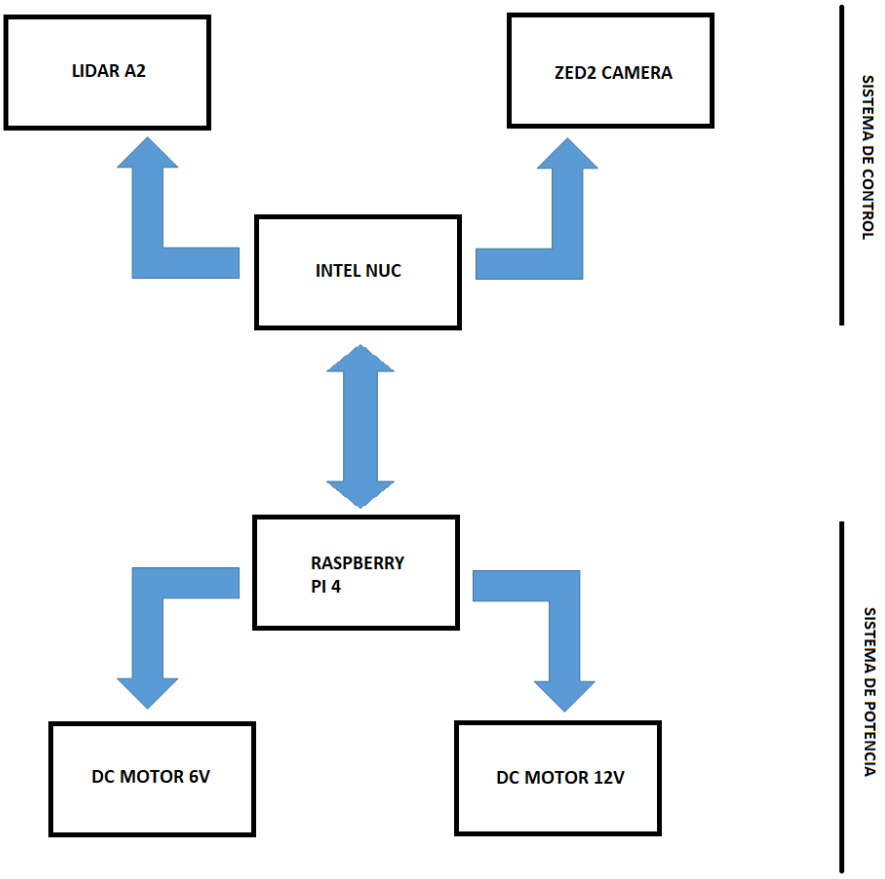


Figura 11: Diagrama de bloques del sistema del prototipo

Una vez considerado el orden y número de conexiones de cada componente a partir del diagrama relacional, se adquirieron los componentes necesarios para su implementación. A continuación, se enumeran y describen los materiales utilizados.

Lista de materiales

- **BMW i8 Spyder Toy Rollplay**

Este es un carro eléctrico diseñado para la conducción de niños menores a 6 años, el cual sirvió como base para el desarrollo del proyecto. De este juguete, se utilizaron sus motores internos, ruedas, sistema de tracción y carrocería. En esta última se instalaron todos los componentes necesarios para la navegación automática o semiautomática del vehículo, tales como el Raspberry, el NUC, el sensor Lidar, etc. Cabe recalcar que uno de sus motores internos fue modificado en pro del desarrollo de este proyecto para tener un mejor desempeño en la potencia que se utiliza en el movimiento del carro.

- **Motor RS 500**

Motor utilizado para realizar el movimiento hacia adelante y atrás del vehículo. Para su correcto funcionamiento necesita de 6 V a 1.12 A, los cuales son suministrados desde las baterías. El motor está conectado solamente a la rueda posterior derecha ya que el sistema de tracción propio del vehículo es suficiente para mover toda la carrocería con solo esta disposición del motor. La hoja de características de este componente se encuentra en la sección de anexos.

- **Motor reductor 25Ga370 370 de 12V**

Motor utilizado para realizar el movimiento de la dirección (izquierda y derecha) del vehículo. Para su correcto funcionamiento necesita de 12 V a 1.12 A, los cuales son suministrados desde las baterías. El motor es asistido por un mecanismo de engranajes propios del carro eléctrico que está conectado a las ruedas delanteras. La hoja de características de este componente se encuentra en la sección de anexos.

- **Motor shield driver VNH2SP30-30 de canal simple**

Driver que permite el acoplamiento del sistema de control, mediante los puertos GPIO del Raspberry Pi, los motores y la fuente de alimentación. Integran un puente H que permite cambiar la polarización de los motores y controlar su sentido de rotación. Al ser un driver no necesita de un sistema de potencia controlado externo ya que suministra la energía suficiente que requieren los motores. En consecuencia, con este diseño se pueden agregar motores más potentes de hasta 16 V a 14 A, lo que permitiría una mayor potencia en el movimiento y también una mayor capacidad de carga. La hoja de características de este componente se encuentra en la sección de anexos.

- **Step Down XL4005**

Conversores DC-DC utilizado para alimentar con el voltaje (19V) y la corriente (5A) correcta a la computadora Intel NUC 8i5beh mediante las baterías utilizadas. La hoja de características de este componente se encuentra en la sección de anexos.

- **Step Down LM2596**

Conversores DC-DC utilizado para alimentar los drivers de los motores que necesitan un voltaje de 6 V y 12 V, y también para alimentar la microcomputadora Raspberry Pi que necesita 5 V. Cada uno de estos conversores está conectado a una batería específica que suministra energía a los motores y a las computadoras. La hoja de características de este componente se encuentra en la sección de anexos.

- **Raspberry Pi 4 Computer Modelo B 4GB RAM**

Microcomputadora para el control de los motores gracias a sus puertos GPIO de fácil acceso. Esta computadora es parte del clúster que esta exclusivamente encargada de enlazar las ordenes enviadas por el usuario mediante el DualShock 4 Wireless Controller con los motores encargados de mover el vehículo libremente. Al realizar esta única tarea,

el rendimiento de este proceso es optimizado lo que resulta en acciones más rápidas por parte del carro eléctrico y con un menor riesgo de fallo inesperado del sistema de control.

- **Intel NUC 8i5beh**

Mini PC utilizada para el control de los sistemas de navegación y de reconocimiento del entorno gracias a su gran capacidad de procesamiento y su tamaño compacto adaptado a las necesidades de este proyecto. Esta computadora controla el sensor Lidar para el mapeo de la zona, la cámara 3D y el módulo GPS y también administra todos los programas en los que se puede visualizar remotamente toda la información que se recolecta de los componentes antes mencionados.

- **Wireless N Router TL-WR741ND**

Router utilizado para crear una red local Wi-Fi (sin conexión a internet) para conectar las computadoras utilizadas y controlarlas remotamente. Este componente permite al vehículo tener una mayor libertad en su utilización ya que se lo puede llevar a cualquier sitio y utilizar todas sus funciones sin la necesidad de tener una conexión a internet lo cual amplia significativamente la cantidad de usos que se le puede dar a este autómata. Es únicamente limitado por el rango del router.

- **RPLIDAR A2**

Escáner de laser a rango con tecnología infrarroja que permite mapear el entorno a través de su tecnología basada en el principio de rango de triangulación láser. Este componente ayuda al vehículo y a su usuario a tener una mejor comprensión del entorno por el que navega y a hacer un mapa en dos dimensiones que puede utilizarse posteriormente en diversas aplicaciones como por ejemplo, la conducción completamente automática del vehículo.

- **ZED2 Camera**

Esta cámara implementa la visión del conductor virtual el vehículo en un rango de 120°. El punto de este componente es imitar la visión humana para analizar las imágenes obtenidas y utilizar estos datos en diferentes funciones interesantes como la detección de objetos 3D, sensores de profundidad, rastreo de posiciones, etc. Al ser un hardware creado específicamente para el desarrollo de aplicaciones, su utilidad potencial no puede pasar desapercibida. Para esta etapa del proyecto, esta cámara se utiliza sencillamente para captura y streaming de video del entorno por el cual navega el vehículo.

- **Batería LIPO 6s de 22V**

Batería de Litio y polímero de 6 celdas utilizada para alimentar la computadora Intel NUC 8i5beh. Esta batería es de gran utilidad en este proyecto ya que al ser compacta y recargable resulta ideal.

- **Batería LIPO 3s de 11.7V**

Batería de Litio y polímero de 3 celdas utilizada para alimentar la computadora Raspberry Pi. Esta batería es de gran utilidad en este proyecto ya que al ser compacta y recargable resulta ideal.

- **Baterías de plomo y ácido**

Baterías secas utilizadas para alimentar los motores ya que suministra corrientes lo suficientemente grandes para administrar correctamente la energía durante el arranque de los motores.

- **Conectores XT60**

Conectores utilizados en las conexiones de las baterías LiPo hacia sus respectivos componentes debido a su resistencia a corrientes altas (de hasta 60 A).

- **Conectores EC5**

Conectores utilizados en las conexiones de las baterías secas ya que se realizó una adaptación en la cual están conectadas en serie pero a la vez necesitan una manera rápida y simple de desconectarse para poder recargarlas.

- **Cables AWG 14**

Cables utilizados para conectar las baterías con las computadores y motores ya que pueden resistir una gran cantidad de corriente sin dañarse.

- **Jumpers de protoboard**

Cables utilizados para conectar los GPIO del Raspberry con los drivers encargados del movimiento del motor.

- **DualShock 4 Wireless Controller**

Control inalámbrico de la consola de videojuegos Playstation 4 utilizado como interfaz física para que el usuario pueda controlar el movimiento del vehículo. Este componente se conecta mediante una conexión bluetooth a la computadora Raspberry Pi y es controlado mediante el programa “ds4drv” para registrar todos los comandos que el usuario realiza y los convierte en acciones de los motores.

Esquema de conexiones

En la figura 12 se observa la disposición de cables y las conexiones entre componentes según lo detallado en la sección anterior. Cabe recalcar que todo este hardware se ha montado en el chasis del carro eléctrico y que los cables están conectados tal y como se muestra en la figura. Para evitar el cruce de cables y mantener organizado todo el hardware, el carrito dispone de un montaje de distintas piezas de acrílico que separan las distintas partes como son los reguladores de los drivers, el GPIO del Raspberry de su fuente de alimentación, etc. Esto se evidenciará más adelante en la sección del prototipo final.

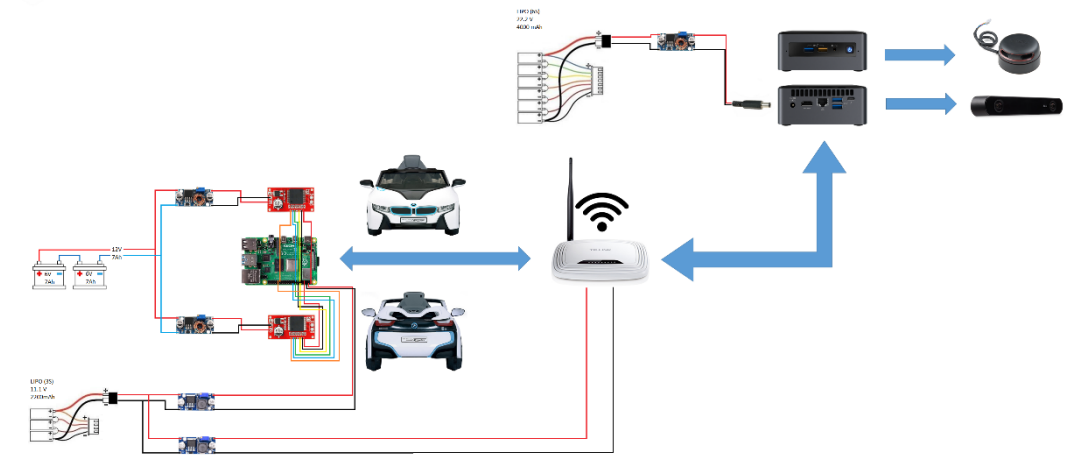


Figura 12: Diagrama de conexiones interna y externas del vehículo

Diseño y funcionamiento del sistema de control de movimiento

Conexión de los motores con el driver VNH2SP30-30

Una vez detallados todos los componentes utilizados, su función y sus conexiones es pertinente explicar cómo fue diseñado el sistema de control del vehículo autónomo.

El carro eléctrico originalmente vino de fábrica con un motor de 6V alimentado por dos baterías de 6V a 3A conectadas en serie[1] y acoplado a la rueda posterior derecha que se accionaba mediante un pedal controlado por el usuario dentro del vehículo. Inicialmente, se desconectaron los cables del motor que lo conectaban con el pedal para conectarlo con el driver VNH2SP30-30 en las terminales OUTA y OUTB, como se muestra en la figura 13.

Figura 13: Driver VNH2SP30 Parte Frontal

Una vez conectado el motor al driver se procedió a conectar los puertos correspondientes a los GPIO del Raspberry Pi que permiten la transmisión de señales

[1] El sistema de alimentación se explicará posteriormente

como se muestra en Ilustración 4. Para la activación de los motores en uno u otro sentido se utilizan los puertos INA e INB y para controlar la potencia que los motores reciben mediante el puerto PWM. Se debe tomar en cuenta que el puerto EN siempre va a estar activo mientras se quiera controlar el motor en cualquier dirección (para este proyecto, una vez que se inicia todo el sistema de control, el puerto EN siempre está energizado). Por último, los puertos de 5V y GND del driver se conectan respectivamente con los puertos GPIO homónimos. En la figura 14 se puede observar la ubicación de los puertos utilizados.

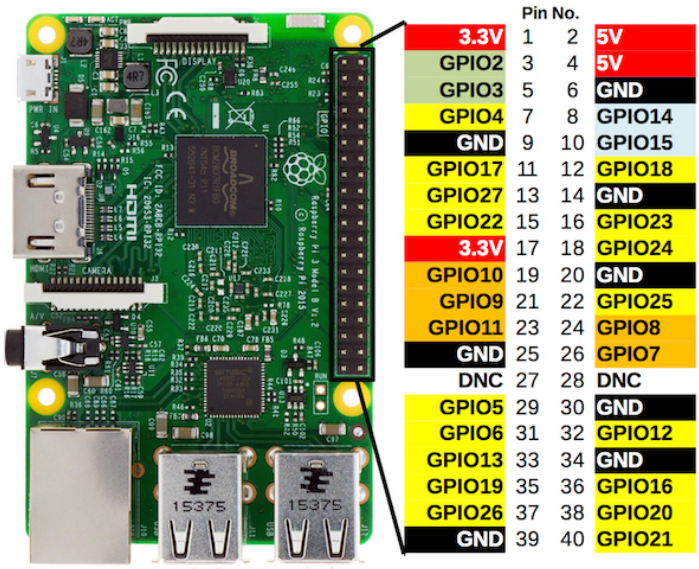


Figura 14: Ubicación de los pines GPIO de un Raspberry Pi

Para finalizar las conexiones de este motor se conectaron los puertos positivo y negativo de la batería seca con los puertos de PWR + y – que se pueden observar en la parte superior de la figura 15, respectivamente.

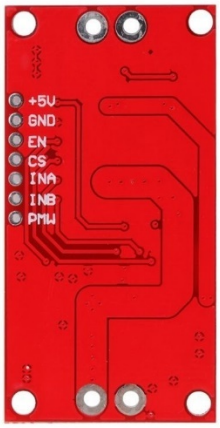


Figura 15: Driver VNH2SP30 Parte Trasera

Para controlar la dirección (izquierda y derecha) del vehículo se utilizó el sistema de movimiento mecánico del volante y se lo adaptó con un motor de 12V para que funcione eléctricamente y también pueda ser controlado remotamente. Este motor funciona con un sistema de engranajes que transmiten el movimiento de rotación, la dirección y la energía hacia una barra que mueve ligeramente las ruedas posteriores para cambiar la dirección de movimiento del auto mientras se está moviendo adelante o hacia atrás.

El sistema de conexión del manejo de la dirección es similar al utilizado en el motor mencionado anteriormente. Se conectan las baterías secas en serie a los puertos respectivos de un segundo driver VNH2SP30-30 y también se conectan los puertos de los motores en las entradas OUTA y OUTB del driver. La única diferencia de conexión respecto al motor anterior son los pines GPIO a los puertos de control.

En la Tabla 1 se muestra la conexión respectiva de los pines GPIO con los puertos respectivos del driver VNH2SP30-30.

Puerto	Motor (adelante y atrás)	Motor dirección (derecha e izquierda)
+5V	2	4
GND	6	9
EN	11	12
INA	13	16
INB	15	18
PWM	29	31

Tabla 1: Conexiones de puertos GPIO con los drivers VNH2SP30-30 respectivos

Fuente de alimentación para los motores

Debido a que los dos motores utilizados requieren voltajes nominales diferentes (de 6V y 12V) y además la batería seca incluida originalmente en el carro eléctrico desde la fábrica suministraba tan solo 6V a 7Ah se decidió añadir otra batería seca con las mismas especificaciones y conectarlas en serie para mantener la capacidad del conjunto 7 Ah pero con un voltaje mayor (de 12V). La conexión de las baterías es tal y como se muestra en la figura 16.

Como se mencionó anteriormente los motores requieren voltajes de 6V y 12V y la fuente de alimentación con baterías en serie suministra 12V, se utilizaron dos conversores de voltaje DC-DC LM2596 (cada uno ajustado al voltaje de su respectivo motor) como medio de seguridad para que los drivers reciban el voltaje exacto y lo distribuyan hacia los motores sin riesgo de que éstos reciban más voltaje del necesario y se quemen.

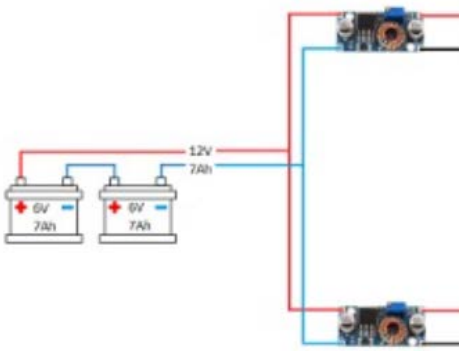


Figura 16: Conexión de las baterías de los motores

Puesta en marcha

Sistema de potencia y control de motores

Una vez conectados los drivers a los motores, a la fuente de alimentación y al Raspberry Pi se desarrolló el software para controlar el vehículo por medio de control remoto.

En primer lugar, se instaló el software “ds4drv” que permite que el enlace bluetooth del control Dualshock 4 con el Rapsberry Pi. Una vez instalado se enlazo el control con el Raspberry Pi.

Para poder registrar las acciones realizadas por el usuario mediante el control Dualshock 4 se usa un conjunto de módulos para Python llamado Pygame el cual se utiliza comúnmente para desarrollar video juegos en sistemas operativos Linux. A través de este conjunto de librerías se puede procesar las señales recibidas por el control remoto y utilizarlas como señales que se pueden procesar como órdenes para controlar los GPIO del Raspberry Pi, y, en consecuencia, los motores del vehículo.

Ahora se procede a describir el script con el cual los motores funcionan. En primer lugar, se importan todas las librerías de Python que se van a utilizar:

```
import pygame
import RPi.GPIO as GPIO
import time
import rospy
```

Luego se inicializa a este script como un nodo de ROS para que pueda trabajar en conjunto con otros scripts necesarios para el funcionamiento completo del vehículo

```
rospy.init_node("controlmanual")
```

Se utiliza el comando GPIO.cleanup() para asegurarse que todos los pines del GPIO se encuentren apagados para que no emitan una señal innecesaria que pueda alterar el funcionamiento del vehículo.

También se definen los pines de acuerdo con su puerto respectivo en el driver VNH2SP30-30 como se mostró en la Tabla 1.

```
ENS1=11
ENS2=12
INA1=13
INB1=15
INA2=16
INB2=18
PWM1=29
PWM2=31
```

Una vez declarados los pines se los inicializa y define como puertos de GPIO y también se les asigna un estado lógico (1 o 0, high o low, prendido o apagado). Como se mencionó anteriormente al momento de iniciar el script de control del vehículo los pines de EN siempre van a estar en 1 lógico hasta que se interrumpa o finalice el programa. Cabe recalcar que los pines INA e INB de cada motor van a empezar apagados pero cambiaran su estado dependiendo de las órdenes recibidas por el usuario mediante el control remoto como se observará posteriormente.

```
GPIO.setmode(GPIO.BOARD)


GPIO.setup(INA1,GPIO.OUT)
GPIO.setup(INB1,GPIO.OUT)
GPIO.setup(INA2,GPIO.OUT)
GPIO.setup(INB2,GPIO.OUT)
GPIO.setup(ENS1,GPIO.OUT)
GPIO.setup(ENS2,GPIO.OUT)
GPIO.setup(PWM1,GPIO.OUT)
GPIO.setup(PWM2,GPIO.OUT)


GPIO.output(ENS1,GPIO.HIGH)
GPIO.output(ENS2,GPIO.HIGH)
GPIO.output(INA1,GPIO.LOW)
GPIO.output(INB1,GPIO.LOW)
GPIO.output(INA2,GPIO.LOW)
GPIO.output(INB2,GPIO.LOW)
```

Los pines que controla el PWM, es decir, la potencia total que reciben los motores se necesita inicializarlos como se muestra a continuación

```
freq=100
p1=GPIO.PWM(PWM1,freq)
p2=GPIO.PWM(PWM2,freq)
```

Y también se necesita declarar cuanta potencia va a entregar a los motores con la variable pwm. En este proyecto se utilizó el 100% de la potencia, es decir, los motores funcionan a su máxima capacidad.

```
pwm=100

p1.start(pwm)
p2.start(pwm)
```

Ahora se definen las funciones que energizan los pines que controlan los cuatro tipos de movimiento del vehículo adelante (motor1F()), atrás (motor1B()), izquierda (motor2L()) y derecha (motor2R()). Estas funciones se utilizarán posteriormente.

```
def motor1F():
    GPIO.output(INB1,True)
    GPIO.output(INA1,False)


def motor2L():
    GPIO.output(INB2,GPIO.HIGH)
    GPIO.output(INA2,GPIO.LOW)


def motor2R():
    GPIO.output(INA2,GPIO.HIGH)
    GPIO.output(INB2,GPIO.LOW)
```

La siguiente sección del código se centra en crear un programa y una ventana donde se vincule las órdenes emitidas por el control Dualshock 4 con el programa desarrolla y además pueda visualizarse que acciones se están realizando.

```
# colores interfaz pygame
BLACK = pygame.Color('black')
WHITE = pygame.Color('white')


# configuracion de impresion de datos
class TextPrint(object):
    def __init__(self):
        self.reset()
        self.font = pygame.font.Font(None, 20)

    def tprint(self, screen, textString):
        textBitmap = self.font.render(textString, True, BLACK)
        screen.blit(textBitmap, (self.x, self.y))
        self.y += self.line_height

    def reset(self):
        self.x = 10
        self.y = 10
        self.line_height = 15

    def indent(self):
        self.x += 10

    def unindent(self):
        self.x -= 10



pygame.init()

screen = pygame.display.set_mode((500, 700))

pygame.display.set_caption("My Game")

done = False

clock = pygame.time.Clock()

pygame.joystick.init()

textPrint = TextPrint()
a=1
```

Finalmente, en un loop while infinito a través de las funciones de la librería pygame, se registran los botones que se presionan en el control remoto y dependiendo del botón presionado, el script realiza una función en específico. En este proyecto se utilizan 5 funciones que se controlan a través de los botones adelante, atrás, izquierda, derecha y el pad central del control Dualshock 4. Las funciones que activan estos botones se utilizan para mover al carro en la dirección respectiva y con el botón del pad central del control se finaliza el programa y se apaga el sistema de control de movimiento.

```
# -------- Main Program Loop -----------
while a==1:

    # Possible joystick actions: JOYAXISMOTION, JOYBALLMOTION, JOYBUTTONDOWN,
    # JOYBUTTONUP, JOYHATMOTION
    for event in pygame.event.get(): # User did something.
        if event.type == pygame.QUIT: # If user clicked close.
            done = True # Flag that we are done so we exit this loop.
        elif event.type == pygame.JOYBUTTONDOWN:
            print("Joystick button pressed.")
        elif event.type == pygame.JOYBUTTONUP:
            print("Joystick button released.")

    # DRAWING STEP

    screen.fill(WHITE)
    textPrint.reset()

    # Get count of joysticks.
    joystick_count = pygame.joystick.get_count()

    textPrint.tprint(screen, "Number of joysticks: {}".format(joystick_count))
    textPrint.indent()

    # For each joystick:
    for i in range(joystick_count):
        joystick = pygame.joystick.Joystick(i)
        joystick.init()

        textPrint.tprint(screen, "Joystick {}".format(i))
        textPrint.indent()

        # Get the name from the OS for the controller/joystick.
        name = joystick.get_name()
        textPrint.tprint(screen, "Joystick name: {}".format(name))

        # Usually axis run in pairs, up/down for one, and left/right for
        # the other.
        axes = joystick.get_numaxes()
        textPrint.tprint(screen, "Number of axes: {}".format(axes))
        textPrint.indent()

        for i in range(axes):
            axis = joystick.get_axis(i)
            textPrint.tprint(screen, "Axis {} value: {:>6.3f}".format(i, axis))
        textPrint.unindent()
```

```
buttons = joystick.get_numbuttons()
textPrint.tprint(screen, "Number of buttons: {}".format(buttons))
textPrint.indent()

for i in range(buttons):
    button = joystick.get_button(i)
    buttonp = joystick.get_button(13)
    textPrint.tprint(screen,
                     "Button {:>2} value: {}".format(i, button))
textPrint.unindent()

hats = joystick.get_numhats()
textPrint.tprint(screen, "Number of hats: {}".format(hats))
textPrint.indent()

# Hat position. All or nothing for direction, not a float like
# get_axis(). Position is a tuple of int values (x, y).
for i in range(hats):
    hat = joystick.get_hat(i)
    textPrint.tprint(screen, "Hat {} value: {}".format(i, str(hat)))
textPrint.unindent()

textPrint.unindent()


# Desde aqui va el codigo de control de los motores

        if hat==(0,1):
                textPrint.tprint(screen, "adelante")
                motor1F()


 # Movimiento lateral izquierdo

         elif hat==(-1,0):
                 textPrint.tprint(screen, "izquierda")
                 motor2L()
 # Movimiento lateral derecho

         elif hat==(1,0):
                 textPrint.tprint(screen, "Derecha")
                 motor2R()


 elif buttonp == 1:
          textPrint.tprint(screen, "GPIO CLEARED")
          GPIO.cleanup()
```

```
else:

        GPIO.output(INA1,GPIO.LOW)
        GPIO.output(INB1,GPIO.LOW)
        GPIO.output(INA2,GPIO.LOW)
        GPIO.output(INB2,GPIO.LOW)
```

Puesta en marcha del sistema de control de movimiento

Una vez explicado cómo funciona todo el sistema de control de movimiento, se puede inicializar esta función del vehículo.

En primer lugar, se debe vincular el control Dualshock 4 con el Raspberry Pi mediante bluetooth. Para comprobar que el control se encuentra correctamente vinculado, la luz led ubicada en la parte de arriba del control debe ser de un color azul intenso.

Luego se debe abrir una terminal en donde se debe escribir el comando: roscore. Dejar que la terminal despliega toda la información y minimizar la ventana de esa terminal para que se ejecute en segundo plano como se muestra en la Figura 17.

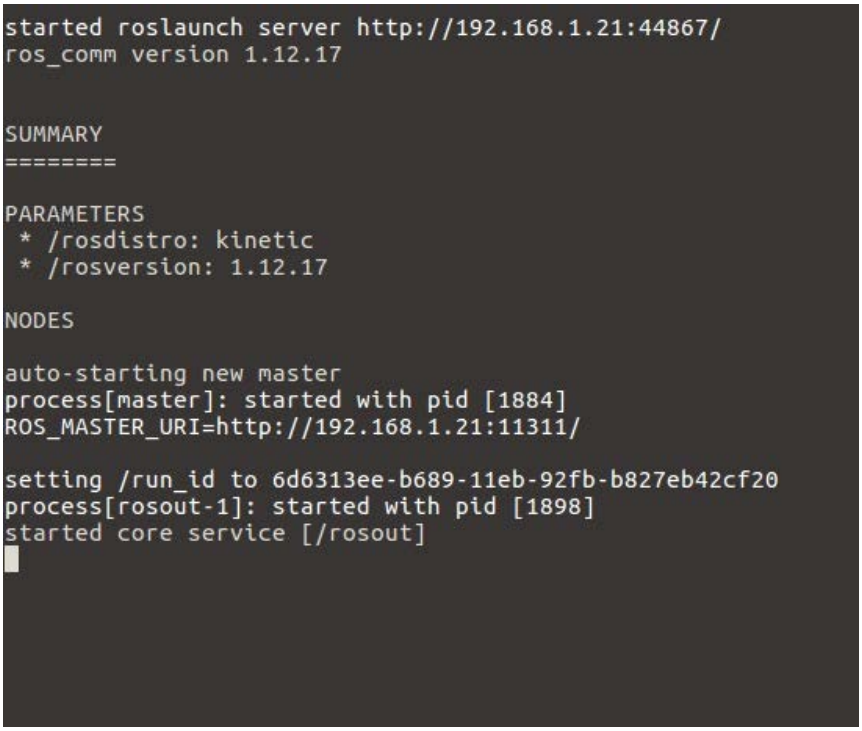


Figura 17: Roscore en funcinamiento

Finalmente, en otra terminal se debe ejecutar el comando: python controlmanual.py. Este comando ejecutará todo el script antes descrito y desplegará una ventana la cual muestra los botones del control que están presionados y la función que el script está realizando en ese momento como se muestra en la Figura 18. Es decir, el programa estará completamente funcional y el vehículo se moverá de acuerdo con las acciones del usuario.

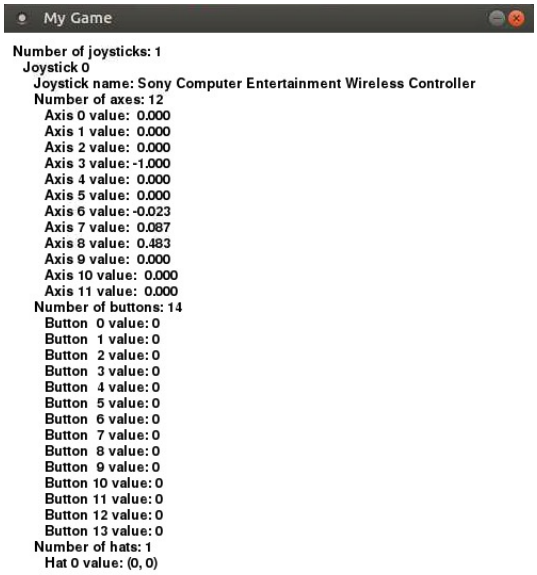


Figura 18: Programa de control de movimiento

Para finalizar con el control del movimiento simplemente se presiona el botón el pad del control y todos los programas se cerrarán.

Sistema de navegación y visualización del entorno

Una vez tengamos listo el sistema de los motores, debemos de abrir otra terminal y correr las siguientes líneas de comando para correr la parte de los sensores

```
cd ~/catkin_ws
source devel/setup.bash
```

```
sudo chmod 666 /dev/ttyUSB0

roslaunch bb10_auto mapeo_manual,launch
```

Con esto lo que hacemos es ubicar nuevamente el workspace para que el ROS master sepa donde trabajar, chmod es un comando para brindarle permisos de escritura y lectura al USB en el que se conectar el LIDAR, y el roslaunch es para ejecutar el archivo launch que corre paralelamente el nodo rplidar con el hector slam. Aparte de ello, si queremos visualizar la parte frontal del vehículo con la cámara, debemos de abrir otra línea de comandos y ejecutar lo siguiente:

```
cd /usr/local/zed/tools

./ZED_Explorer
```

Con esto nos dirigimos a la carpeta donde se instalaron los diferentes archivos de stereo labs, abrimos el visualizador de la cámara ZED2. Lamentablemente al no disponer de una GPU dentro de ninguna de las computadoras incorporadas, no podemos incorporar la ejecución del visualizador en el script de launch de ROS ya que en este caso debería trabajar no desde el visualizador sino desde el nodo, y aquel como ya se mencionó anteriormente, únicamente funciona con CUDA y este a su vez funciona si detecta la GPU.

Prototipo Final

A continuación, en la Figuras 19 - 25 se muestran imágenes de la incorporación final de todos los componentes con el prototipo.

Figura 19: Vision posterior del prototipo

Figura 20: Visión posterior del prototipo

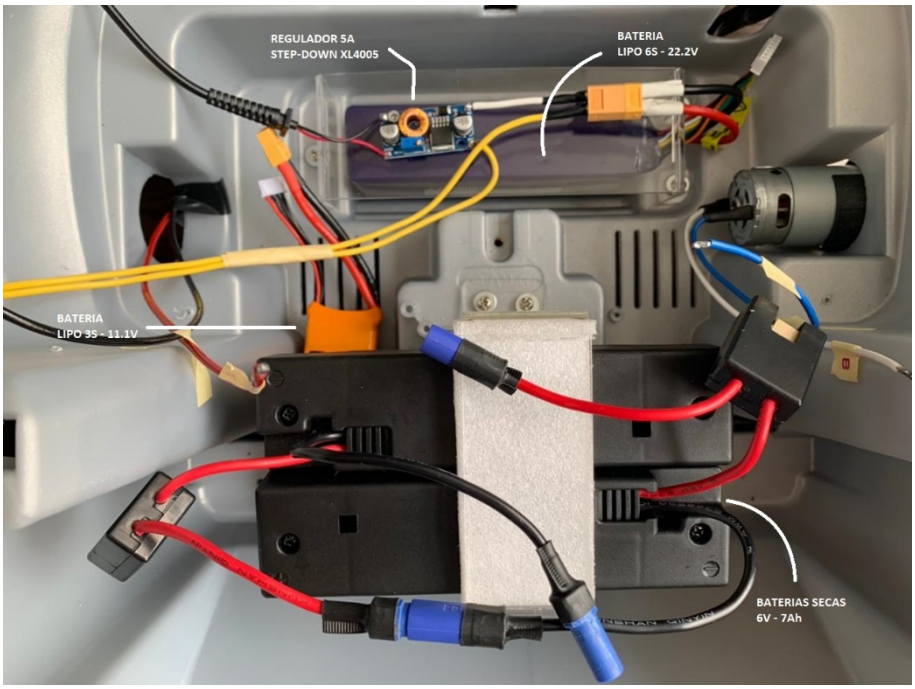


Figura 21: Sistema de alimentación del prototipo

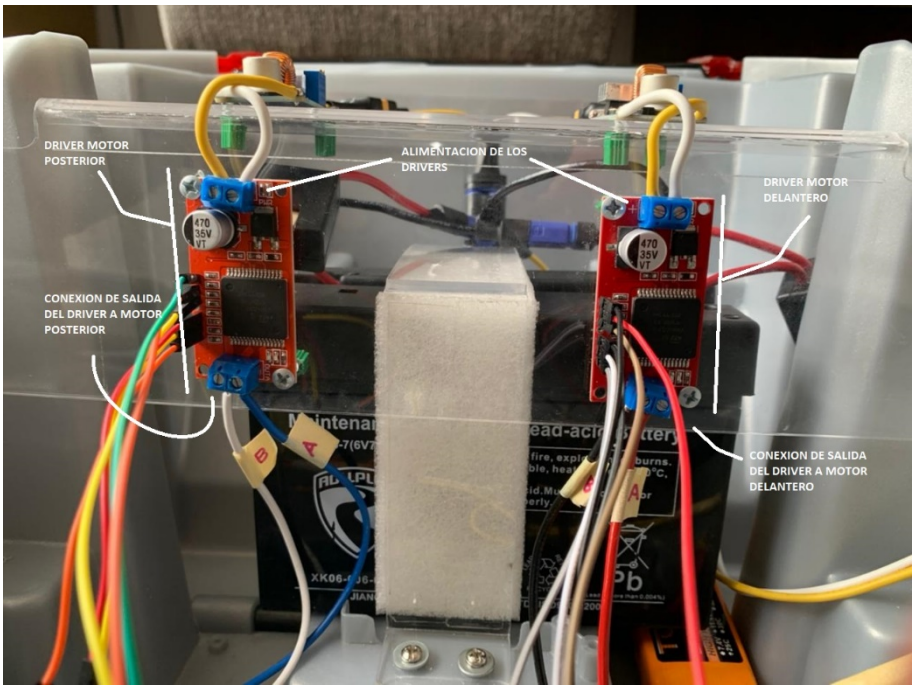


Figura 22: Disposición de los drivers

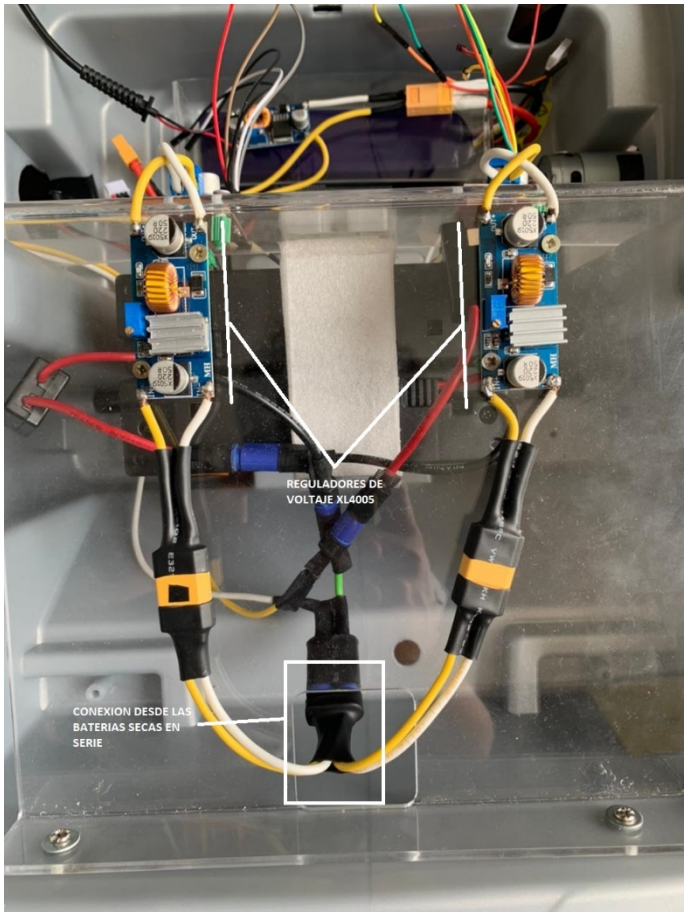


Figura 23: Disposición de los reguladores de voltaje

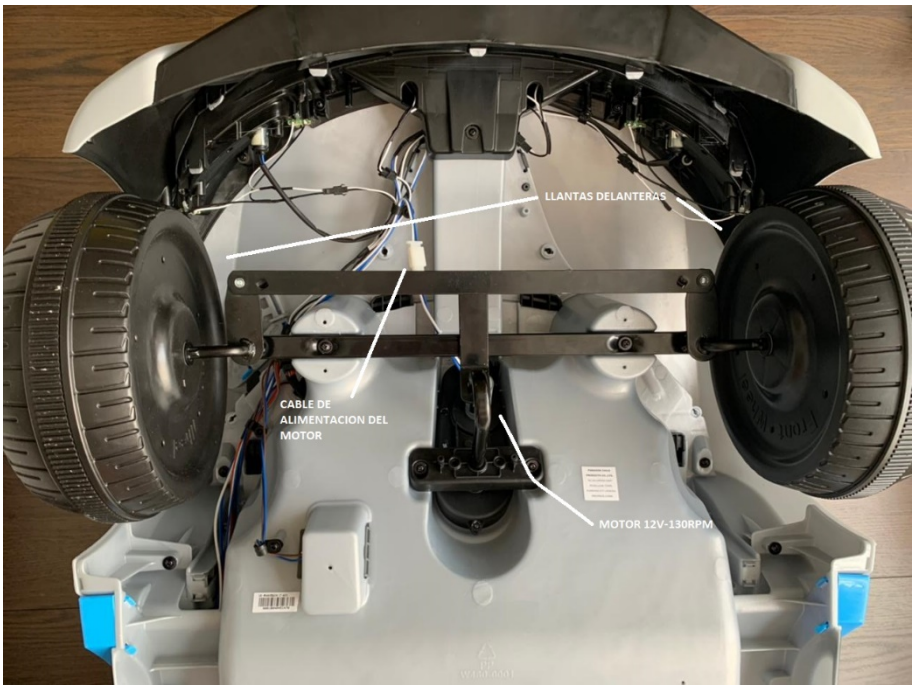


Figura 24: Parte frontal del vehiculo

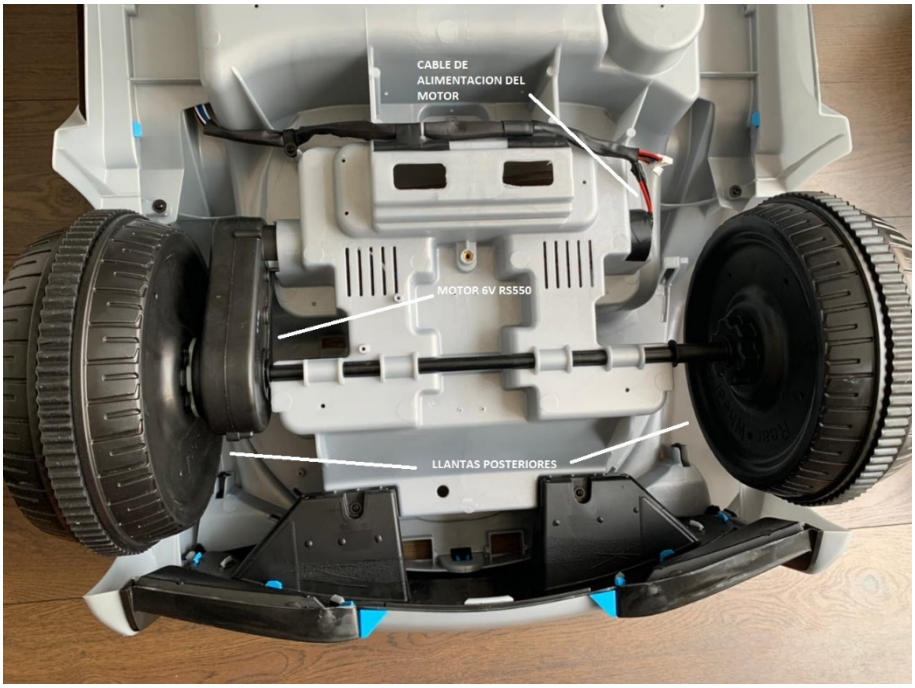


Figura 25: Parte posterior del vehículo

Conclusiones:

El objetivo principal del proyecto era el de implementar un sistema de control que fuera capaz de brindarnos un manejo efectivo del vehículo eléctrico elegido. A lo largo del proceso, se usaron distintas herramientas que facilitaron el diseño e implementación de este. Existen muchas formas más de armar un prototipo similar; sin embargo, se escogió la que se ha mostrado debido a su diseño modular y a la versatilidad que provee al vehículo para su escalabilidad en el futuro.

Como bien sabemos ROS es un entorno orientado al diseño de robots, por lo que explorar esta herramienta podría beneficiar a futuro para aumentar el grado de autonomía del prototipo, y en futuro hasta llegue a tener un grado cinco de autonomía, donde el carrito únicamente requerirá de la dirección a donde deseamos hacerlo llegar, y este emprenderá la ruta más óptima hacia dicho destino.

Otra de las herramientas que pudimos explorar es la cámara ZED2, esta cámara en primera instancia no parece tan potente. Sin embargo, tras analizar sus componentes vimos una amplia gama de aplicaciones que se le pueden dar, como bien puede ser el trabajar paralelamente con el sensor lidar y generar un mapeo en tres dimensiones, además de que este podría brindarnos más información sobre el entorno del vehículo gracias a su sensor IMU. O también podría implementarse en la cámara un software que sea capaz de darnos el reconocimiento de personas o quizá el de señales de tránsito como un STOP o un semáforo que haga posible que el carrito sepa actuar ante esas señales.

Finalmente, los objetivos de esta etapa del proyecto fueron cumplidos y se dispone de un nuevo vehículo automatizado, de una nueva plataforma sobre la cual continuar los desarrollos, de forma de conseguir en el futuro mediato de un vehículo completamente autónomo, similar a los de la marca TESLA que ya cuentan con estas funcionalidades.

Referencias

ArnoldBail. (24 de Diciembre de 2014). *Raspberry Pi + ROS.* Obtenido de slideshare: https://www.slideshare.net/ArnoldBail/robotics-and-ros

Evans, P. (2020). *Build a Raspberry Pi cluster computer*. Obtenido de TheMagPi: https://magpi.raspberrypi.org/articles/build-a-raspberry-pi-cluster-computer

Hernandez, C., Barbosa, J., Paredes, D., & Játiva, R. (2 de Diciembre de 2020). *Design and Implementation of an autonomous vehicle with LIDAR-based navigation.* Obtenido de IEEE Xplore: https://ieeexplore.ieee.org/document/9359423

J.Pomeyrol. (24 de Mayo de 2011). *Experiencia UBUNTU*. Obtenido de MuyLinux: https://www.muylinux.com/2011/05/24/participa-en-la-encuesta-de-experiencia-ubuntu/

Javier. (11 de Febrero de 2012). *MYSQL MASTER -- SLAVE*. Obtenido de El mundo en bits: https://www.elmundoenbits.com/2012/11/mysql-master-slave.html#.YKUB9qhKhPZ

Nunez, J. (5 de Junio de 2020). *ROS Kinetic y OpenCV en Raspberry PI*. Obtenido de CostaRicaMakers: https://costaricamakers.com/?p=2861

Anexos

Escalabilidad y Posibles Desarrollos Futuros

El desarrollo de este proyecto fue planteado como una base para el perfeccionamiento y mejora de un sistema completamente autónomo con diferentes funciones que puedan ayudar a realizar tareas de la vida cotidiana, en la industria o en la investigación científica de una forma más rápida y sencilla.

Entre las aplicaciones a futuro que se pueden realizar en este vehículo se encuentran:

- **Mejora en el sistema de control del movimiento**

El sistema de control de movimiento del vehículo se encuentra explicado detalladamente en anteriores secciones. Con base a todos los procedimientos realizados, se puede hacer mejoras a los motores cambiándolos a unos más potentes para que el carro pueda llevar más peso, lo que implicaría si se agregaran componentes que añadieran funcionalidades a este autómata.

En cambio, en la parte del software, al ser sencillo de entender y desarrollar, también se le puede poner ciertas funcionalidades para que el sistema de control tenga mayor variedad en sus sistema de funcionamiento como puede ser un cambio en el valor PWM para cambiar las velocidades del auto, un protocolo para que el carro se mueva de cierta forma por un cierto tiempo al presionar solo un botón o incluso un sistema de movimiento completamente automatizado sin el uso del control remoto solamente utilizando los datos recolectados por el mapeo del entorno.

- **Camara ZED2**

Este dispositivo es utilizado en este proyecto para realizar un streaming de video de las zonas por donde circula el vehículo y “ver” lo que ve en tiempo real. Pero este

hardware, al poseer sensores 3D sofisticados, se lo puede sacar más provecho generando software de análisis espacial para que tenga más funcionalidades y variadas como: detección de objetos, mapeo de profundidad de una zona, rastreo de objetos, etc.

Este componente facilita la creación y el desarrollo de estos softwares ya que puede ser interconectado con varias librerías third-party como: ROS, Matlab, OpenCV, Unity, CUDA (NVIDIA), etc. Al tener librerías asociadas en programas de desarrollo esta cámara se puede utilizar para crear muchas aplicaciones en diferentes ámbitos.

Hay que aclarar que para utilizar estas funciones de este componente se necesita conectar una GPU con puerto Thunderbolt a la computadora NUC para que pueda procesar todos estos programas apropiadamente.

- **Sistema de energía**

La autonomía del vehículo depende en gran parte de sus fuentes de energía (que son varias baterías conectadas a diferentes componentes) por lo que si se quiere mejorar la misma se necesita encontrar una forma de energía unificada y autosuficiente para el uso deseado del carro. Es decir, se podría implementar una sola batería en incluso un sistema de energía recargable y renovable como celdas solares, para que el vehículo dependa solamente de una fuente de alimentación que no solo disminuya el espacio y la carga utilizada en la carrocería, sino que también mejore el tiempo de uso de este autómata para funciones que realice funciones que necesiten un largo tiempo de funcionamiento.

Este proyecto fue ideado como un sistema en constante evolución el cual necesita de más fases en las cuales se vayan agregando y mejorando los sistemas modulares del este vehículo para que pueda tener varias funcionalidades e incluso crear sistemas diferentes basados en una base de desarrollo como la que es presentada en este proyecto.

Datasheet Motor 6V RS550

RS-550PC/VC

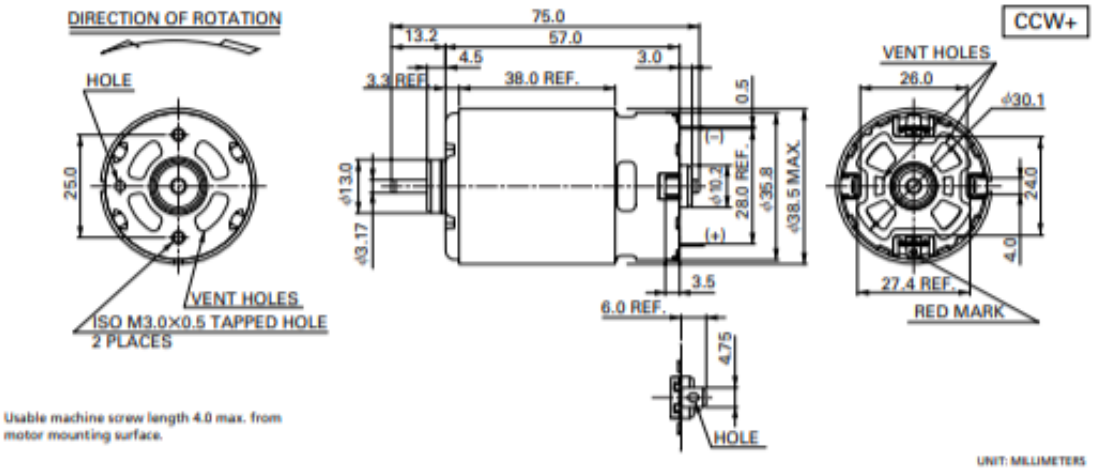
MABUCHI MOTOR
Carbon-brush motors

OUTPUT : 5.0W~100W (APPROX) WEIGHT : 255g (APPROX)

Typical Applications **Cordless Power Tools :** Drill / Cordless Garden Tool / Air Compressor
Toys and Models : Ride-on Toy

MODEL		VOLTAGE		NO LOAD		AT MAXIMUM EFFICIENCY				STALL		
		OPERATING RANGE	NOMINAL	SPEED	CURRENT	SPEED	CURRENT	TORQUE	OUTPUT	TORQUE		CURRENT
				r/min	A	r/min	A	mN·m	g·cm	W	mN·m	g·cm	A
RS-550PC-7527	(*1)	6.0~14.4	12V CONSTANT	18200	1.15	16130	8.97	47.8	488	80.7	421	4292	70.0
RS-550VC-7527	(*1)	6.0~14.4	14.4V CONSTANT	19800	1.30	17620	10.5	64.7	660	119	588	5994	85.0

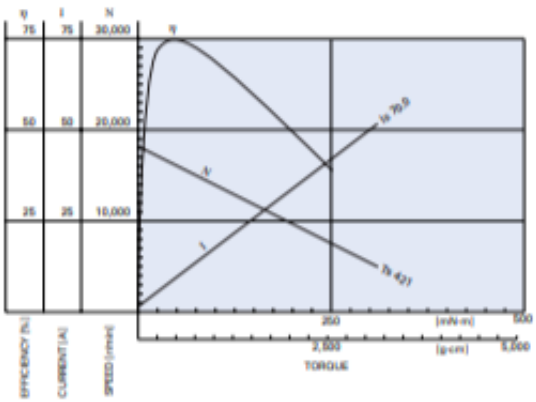
(*1) CCW shifted commutation (CCW+)



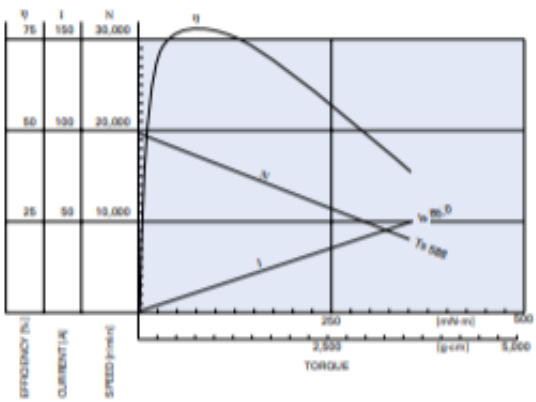
Usable machine screw length 4.0 max. from motor mounting surface.

UNIT: MILLIMETERS

RS-550PC-7527 12.0V



RS-550VC-7527 14.4V



MABUCHI MOTOR CO. LTD. **Headquarters** 430 Matsuhidai, Matsudo-shi, Chiba-ken, 270-2280, Japan Tel:81-47-384-0623 Fax:81-47-385-2026 (Sales Dept.)

Datasheet Motor 12V IT-25GA370

ITO MOTOR

IT-25GA370

Applications

Robot/ Measuring instrument /Audio equipment /Medical instrument
Atuomatic window curtain/ Healthful equipment/ Electronic game
Liquid crystal display screen/ Dryer drying machine/ Toilet paper dispens

Model

Gearmotor Specifications

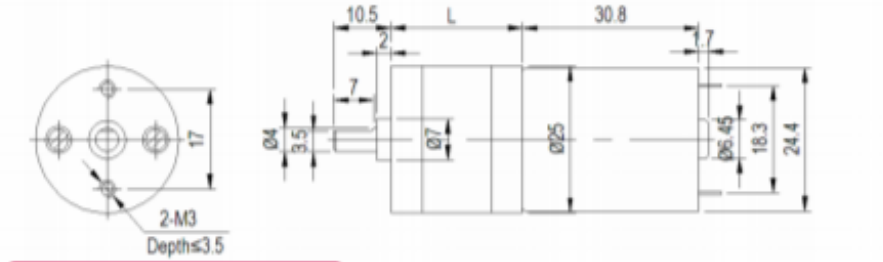
Gear ratio i	10-21	26-47	59-103	130-227	286-499
Length of gearbox(mm)	19	21	23	25	27

Gear Motor Specifications

	Voltage	No-load		Rated				Stall	
	Rated	Speed	Current	Speed	Current	Torque	Power	Torque	Curren
	VDC	r/min	A	r/min	A	Kg.cm	W	Kg.cm	A
IT-25GA370-47	12	120	0.05	100	0.2	0.9	0.95	6.5	1.2
IT-25GA370-	12	54	0.05	46	0.2	1.9	0.9	13.5	1.2
IT-25GA370-	12	85	0.1	65	0.4	1.6	1.1	7	1.5
IT-25GA370-	12	22	0.03	18	0.1	2	0.37	11	0.46
IT-25GA370-	12	38	0.1	30	0.5	3.2	1	15	1.5
IT-25GA370-	12	25	0.05	20	0.25	1.8	0.85	15	1.2
IT-25GA370-	12	64	0.2	50	0.6	8	4.2	15	1.5
IT-25GA370-	12	11	0.05	9	0.25	8.5	0.8	15	1.2
IT-25GA370-47	6	130	0.08	100	0.4	6.5	1	6.5	2
IT-25GA370-	6	55	0.08	45	0.4	2	0.95	12.5	2
IT-25GA370-	6	26	0.08	20	0.35	4.5	0.9	15	2
IT-25GA370-	6	16	0.08	12	0.35	7	0.85	15	2
IT-25GA370-	6	12	0.08	10	0.35	8	0.85	15	2

1kg.cm=103gf.cm=9.8N.cm=0.098 N.m=13.89oz.in=0.868Lb.in=0.07233Lb.ft

Dimensions



Feature

- Encoder possible / Internal capacitors or varister
- Voltage and speed adjustment by winding change
- Modification of shaft configuration (length,flat, hole, knurling, rear shaft etc.)

Datasheet Regulador DC 5A

XLSEMI

Datasheet

5A 300KHz 32V Buck DC to DC Converter **XL4005**

Features

- Wide 5V to 32V Input Voltage Range
- Output Adjustable from 0.8V to 30V
- Maximum Duty Cycle 100%
- Minimum Drop Out 0.6V
- Fixed 300KHz Switching Frequency
- 5A Constant Output Current Capability
- Internal Optimize Power MOSFET
- High efficiency
- Excellent line and load regulation
- TTL shutdown capability
- EN pin with hysteresis function
- Built in thermal shutdown function
- Built in current limit function
- Built in output short protection function
- Available in TO-263 package

Applications

- LCD Monitor and LCD TV
- Digital Photo Frame
- Set-up Box
- ADSL Modem
- Telecom / Networking Equipment

General Description

The XL4005 is a 300KHz fixed frequency PWM buck (step-down) DC/DC converter, capable of driving a 5A load with high efficiency, low ripple and excellent line and load regulation. Requiring a minimum number of external components, the regulator is simple to use and include internal frequency compensation and a fixed-frequency oscillator.

The PWM control circuit is able to adjust the duty ratio linearly from 0 to 100%. An enable function, an over current protection function is built inside. When short protection function happens, the operation frequency will be reduced from 300KHz to 60KHz. An internal compensation block is built in to minimize external component count.

TO263-5L

Figure1. Package Type of XL4005

Pin Configurations

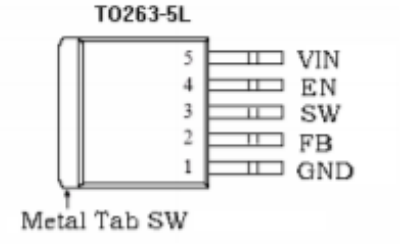


Figure2. Pin Configuration of XL4005 (Top View)

Table 1 Pin Description

Pin Number	Pin Name	Description
1	GND	Ground Pin. Care must be taken in layout. This pin should be placed outside of the Schottky Diode to output capacitor ground path to prevent switching current spikes from inducing voltage noise into XL4005.
2	FB	Feedback Pin (FB). Through an external resistor divider network, FB senses the output voltage and regulates it. The feedback threshold voltage is 0.8V.
3	SW	Power Switch Output Pin (SW). SW is the switch node that supplies power to the output.
4	EN	Enable Pin. Drive EN pin high to turn on the device, drive it low to turn it off.
5	VIN	Supply Voltage Input Pin. XL4005 operates from a 5V to 32V DC voltage. Bypass Vin to GND with a suitably large capacitor to eliminate noise on the input.

Function Block

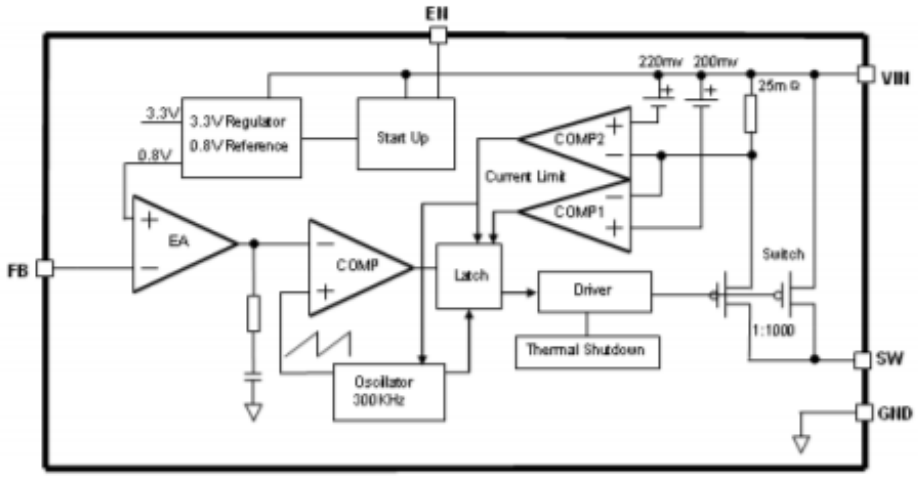


Figure3. Function Block Diagram of XL4005

Typical Application Circuit

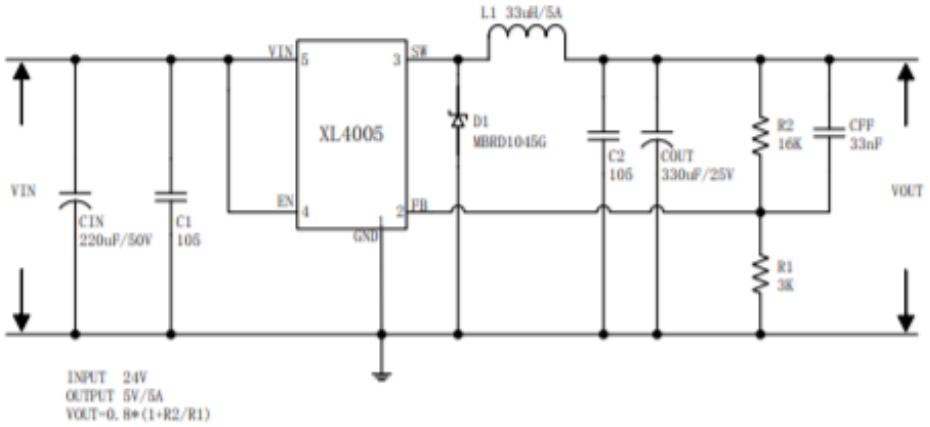


Figure4. XL4005 Typical Application Circuit (24V~5V/5A)

Ordering Information

Order Information	Marking ID	Package Type	Packing Type Supplied As
XL4005E1	XL4005E1	TO263-5L	800 Units on Tape & Reel

XLSEMI Pb-free products, as designated with "E1" suffix in the par number, are RoHS compliant.

Absolute Maximum Ratings（Note1）

Parameter	Symbol	Value	Unit
Input Voltage	Vin	-0.3 to 35	V
Feedback Pin Voltage	V_{FB}	-0.3 to Vin	V
EN Pin Voltage	V_{EN}	-0.3 to Vin	V
Output Switch Pin Voltage	V_{Output}	-0.3 to Vin	V
Power Dissipation	P_D	Internally limited	mW
Thermal Resistance (TO263) (Junction to Ambient, No Heatsink, Free Air)	R_{JA}	30	℃/W
Operating Junction Temperature	T_J	-40 to 125	℃
Storage Temperature	T_{STG}	-65 to 150	℃
Lead Temperature (Soldering, 10 sec)	T_{LEAD}	260	℃
ESD (HBM)		2000	V

Note1: Stresses greater than those listed under Maximum Ratings may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operation is not implied. Exposure to absolute maximum rating conditions for extended periods may affect reliability.

XL4005 Electrical Characteristics

T_a = 25℃;unless otherwise specified.

Symbol	Parameter	Test Condition	Min.	Typ.	Max.	Unit
System parameters test circuit figure4						
VFB	Feedback Voltage	Vin = 5V to 32V, Vout=5V Iload=0.5A to 5A	0.776	0.8	0.824	V
Efficiency	ŋ	Vin=12V ,Vout=5V Iout=5A	-	90	-	%

Electrical Characteristics (DC Parameters)

Vin = 12V, GND=0V, Vin & GND parallel connect a 220uf/50V capacitor; Iout=500mA, T_a = 25℃; the others floating unless otherwise specified.

Parameters	Symbol	Test Condition	Min.	Typ.	Max.	Unit
Input operation voltage	Vin		5		32	V
Shutdown Supply Current	I_{STBY}	V_{EN}=0V		60	200	uA
Quiescent Supply Current	I_q	V_{EN} =2V, V_{FB} =Vin		3	5	mA
Oscillator Frequency	Fosc		240	300	360	Khz
Switch Current Limit	I_L	V_{FB} =0		8		A
EN Pin Threshold	V_{EN}	High (Regulator ON) Low (Regulator OFF)		1.4 0.8		V
EN Pin Input Leakage Current	I_H	V_{EN} =2V (ON)		1	15	uA
	I_L	V_{EN} =0V (OFF)		1	15	uA
Max. Duty Cycle	D_{MAX}	V_{FB}=0V		100		%

Test Circuit and Layout guidelines

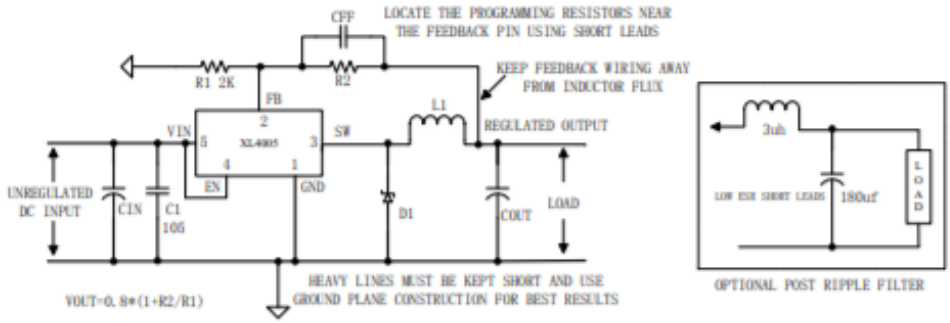


Figure5. Standard Test Circuits and Layout Guides

Select R1 to be approximately 2K, use a 1% resistor for best stability.

C1 and CFF are optional; in order to increase stability and reduce the input power line noise, CIN and C1 must be placed near to VIN and GND;

For output voltages greater than approximately 10V, an additional capacitor CFF is required. The compensation capacitor is typically between 100 pf and 33 nf, and is wired in parallel with the output voltage setting resistor, R2. It provides additional stability for high output voltage, low input-output voltages, and/or very low ESR output capacitors, such as solid tantalum capacitors.

CFF=1/(31*1000*R2); This capacitor type can be ceramic, plastic, silver mica, etc. (Because of the unstable characteristics of ceramic capacitors made with Z5U material, they are not recommended.)

Typical System Application for 24V ~ 12V/4A Version

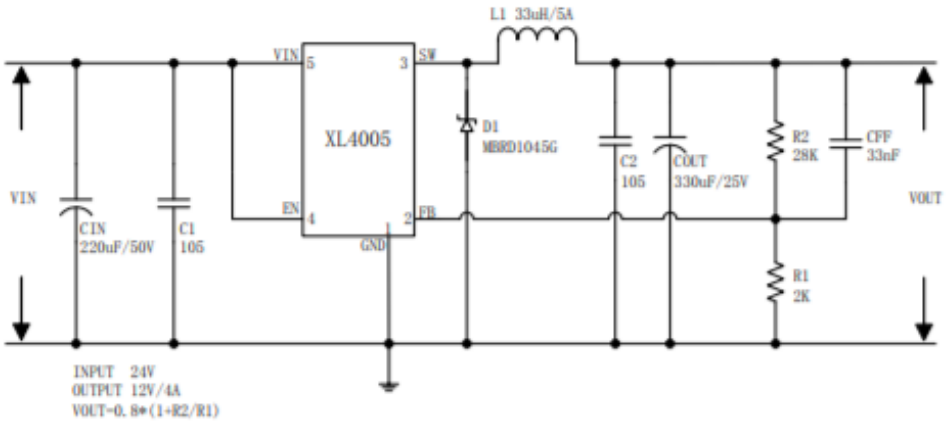


Figure6. XL4005 System Parameters Test Circuit (24V ~ 12V/4A)

Typical System Application for 24V ~ 5V/5A

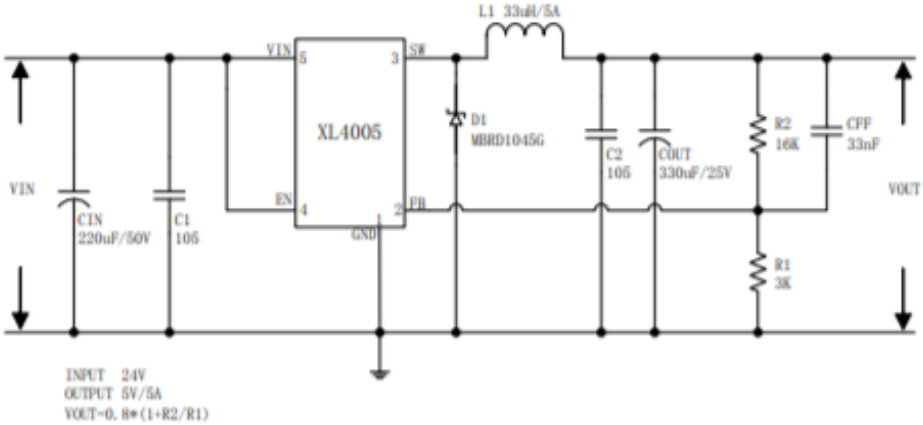
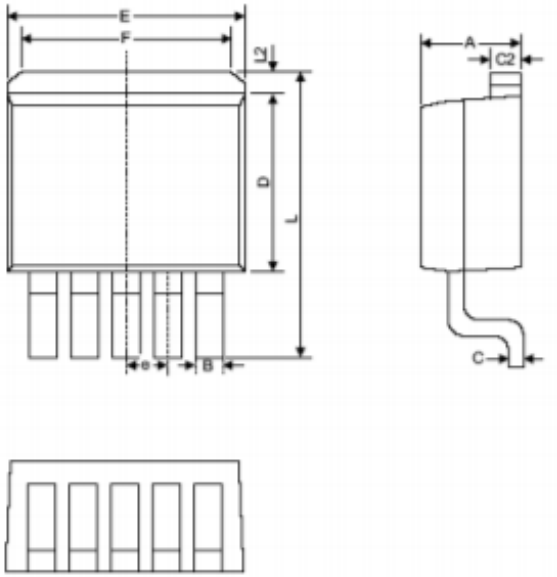


Figure7. XL4005 System Parameters Test Circuit (24V ~ 5V/5A)

Package Information

TO263-5L



Symbol	Dimensions In Millimeters		Dimensions In Inches	
	Min	Max	Min	Max
A	4.06	4.83	0.160	0.190
B	0.71	1.02	0.030	0.040
C	0.36	0.64	0.014	0.025
C2	1.14	1.40	0.045	0.055
D	8.39	9.65	0.330	0.380
E	9.78	10.54	0.385	0.415
e	1.55	1.85	0.061	0.073
F	6.36	7.36	0.250	0.290
L	13.95	15.37	0.549	0.605
L2	1.12	1.42	0.044	0.056

Important Notice

XLSEMI reserve the right to make modifications, enhancements, improvements, corrections or other changes without notice at any time. XLSEMI does not assume any liability arising out of the application or use of any product described herein; neither does it convey any license under its patent rights, nor the rights of others. XLSEMI assumes no liability for applications assistance or the design of Buyers' products. Buyers are responsible for their products and applications using XLSEMI components. To minimize the risks associated with Buyers' products and applications, Buyers should provide adequate design and operating safeguards. XLSEMI warrants performance of its products to the specifications applicable at the time of sale, in accordance with the warranty in XLSEMI's terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent XLSEMI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

For the latest product information, go to www.xlsemi.com.