UNIVERSIDAD POLITÉCNICA SALESIANA

SEDE QUITO

CARRERA DE INGENIERÍA AUTOMOTRIZ

DISEÑO DE UNA INTERFAZ PARA EL MONITOREO DEL BANCO DIDÁCTICO FUNCIONAL DE DIRECCIÓN CON ASISTENCIA ELÉCTRICA DE LA CARRERA DE INGENIERÍA AUTOMOTRIZ DE LA UNIVERSIDAD POLITÉCNICA SALESIANA, QUITO, CAMPUS SUR

Trabajo de titulación previo a la obtención del
Título de Ingeniero Automotriz

AUTORES: INGRID ESTEFANÍA GALARZA BERNITA
DAVID JOSUÉ ROMERO ALDAS
TUTOR: CARLOS ALBERTO CARRANCO QUIÑÓNEZ

Quito, Ecuador
2024

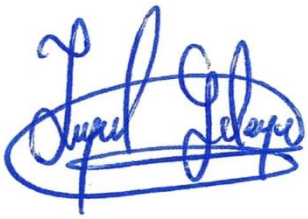
CERTIFICADO DE RESPONSABILIDAD Y AUTORÍA DEL TRABAJO DE TITULACIÓN

Nosotros, Ingrid Estefanía Galarza Bernita con documento de identificación N° 1752005791 y David Josué Romero Aldas N° 1850027423 manifestamos que:

Soy el autor y responsable del presente trabajo; y, autorizo a que sin fines de lucro la Universidad Politécnica Salesiana pueda usar, difundir, reproducir o publicar de manera total o parcial el presente trabajo de titulación.

Quito, 02 de agosto del 2024

Atentamente,



Ingrid Estefanía Galarza Bernita
1752005791

David Josué Romero Aldas
1850027423

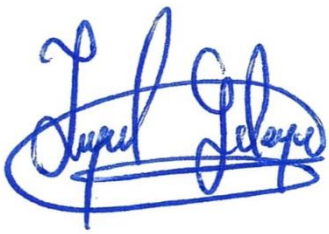
CERTIFICADO DE CESIÓN DE DERECHOS DE AUTOR DEL TRABAJO DE TITULACIÓN A LA UNIVERSIDAD POLITÉCNICA SALESIANA

Nosotros, Ingrid Estefanía Galarza Bernita con documento de identificación No. 1752005791 y David Josué Romero Aldas con documento de identificación No. 1850027423, expresamos nuestra voluntad y por medio del presente documento cedemos a la Universidad Politécnica Salesiana la titularidad sobre los derechos patrimoniales en virtud de que somos autores del Proyecto Técnico: "Diseño de una interfaz para el monitoreo del banco didáctico funcional de dirección con asistencia eléctrica de la Carrera de Ingeniería Automotriz de la Universidad Politécnica Salesiana, Quito, Campus Sur", el cual ha sido desarrollado para optar por el título de Ingenieros Automotrices, en la Universidad Politécnica Salesiana, quedando la Universidad facultada para ejercer plenamente los derechos cedidos anteriormente.

En concordancia con lo manifestado, suscribimos este documento en el momento que hacemos la entrega del trabajo final en formato digital a la Biblioteca de la Universidad Politécnica Salesiana

Quito, 02 de agosto del 2024

Atentamente,



Ingrid Estefanía Galarza Bernita
1752005791

David Josué Romero Aldas
1850027423

CERTIFICADO DE DIRECCIÓN DEL TRABAJO DE TITULACIÓN

Yo, Carlos Alberto Carranco Quiñónez con documento de identificación 1713629564, docente de la Universidad Politécnica Salesiana, declaro que bajo mi tutoría fue desarrollado el trabajo de titulación: DISEÑO DE UNA INTERFAZ PARA EL MONITOREO DEL BANCO DIDÁCTICO FUNCIONAL DE DIRECCIÓN CON ASISTENCIA ELÉCTRICA DE LA CARRERA DE INGENIERÍA AUTOMOTRIZ DE LA UNIVERSIDAD POLITÉCNICA SALESIANA, QUITO, CAMPUS SUR, realizado por Ingrid Estefanía Galarza Bernita con documento de identificación N° 1752005791 y por David Josué Romero Aldas con documento de identificación N° 1850027423, obteniendo como resultado final el trabajo de titulación bajo la opción: Proyecto Técnico que cumple con todos los requisitos determinados por la Universidad Politécnica Salesiana.

Quito, 02 de agosto del 2024

Atentamente,

_____

Ing. Carlos Alberto Carranco Quiñónez, Msc
1713629564

DEDICATORIA

Esta tesis está dedicada a mis padres, cuyo amor y apoyo incondicional han sido mi guía en este camino académico. Su sacrificio y dedicación han sido mi inspiración constante. Agradezco profundamente su confianza en mí y por ser mi fuente de motivación. Este logro no habría sido posible sin ustedes, quienes me han enseñado el valor del esfuerzo y la perseverancia. Esta investigación es un tributo a su infinito amor y compromiso, que han sido el motor detrás de cada paso que he dado. Con todo mi corazón, dedico este trabajo a ustedes, mis padres queridos.

Ingrid Galarza

Dedico este trabajo principalmente a mi madre, Marisol Aldas, quien siempre supo apoyarme a lo largo de mi juventud. Aunque ya no está conmigo, este gran logro es en su honor. Ahora puedo alzar mi cabeza y ver al cielo con orgullo y decirle a mi madre que lo logré y que la amo. A mi padre, Cristian Romero, por su amor infinito y su constante apoyo durante toda mi carrera académica; sin él, no habría podido culminar mis estudios. A mi familia y amigos, por su aliento y comprensión en los momentos difíciles. A mis profesores, por su sabiduría y guía invaluable. Este trabajo está dedicado a todos aquellos que creyeron en mí y me motivaron a alcanzar mis metas.

David Romero

AGRADECIMIENTO

Quiero expresar mi sincero agradecimiento a mis padres, cuyo amor y respaldo constante han sido mi ancla a lo largo de esta travesía académica. A los ingenieros de la universidad, agradezco su valiosa orientación y conocimientos compartidos, que han enriquecido profundamente mi aprendizaje. A mis amigos, les doy las gracias por su colaboración, amistad y por los momentos de apoyo mutuo que hemos compartido. Este logro representa el esfuerzo conjunto de todos ustedes, quienes han dejado una huella significativa en mi experiencia universitaria. Agradezco sinceramente haber compartido este camino con personas tan inspiradoras y solidarias.

Ingrid Galarza

Agradezco a Dios por estar a mi lado en los momentos más duros de mi vida personal y estudiantil, brindándome fuerza y consuelo. A mi padre, Cristian Romero, por su incondicional apoyo y por ofrecerme su mano en los momentos más difíciles. A mi familia, por su amor y comprensión, y a mis compañeros por su amistad y colaboración. A mi novia, Leslie Miranda, quien nunca me dejó solo y fue mi refugio en los momentos de incertidumbre. Finalmente, dedico un agradecimiento especial a mi madre, Marisol Aldas. Aunque ya no está físicamente conmigo, su amor y apoyo siempre me guiaron. A todos ustedes, gracias por creer en mí y ser parte esencial de este logro. Sin su apoyo y confianza, no habría sido posible alcanzar esta meta.

David Romero

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

El documento trata sobre la creación de una interfaz para el seguimiento y control de un banco didáctico de dirección con asistencia eléctrica (EPS), utilizado en la enseñanza de la ingeniería automotriz. Se emplean componentes electrónicos como Arduino, MPU6050, MCP2515, y una pantalla Nextion, además de la comunicación CAN bus y Bluetooth, para desarrollar una interfaz educativa que permita a los estudiantes entender mejor los sistemas EPS.

La importancia de los sistemas EPS y sus componentes, como los sensores que monitorean el par de dirección, ángulo de giro del volante y velocidad del vehículo. La metodología incluye la integración de estos sensores con ESP32 y Arduino, y el diseño de la interfaz en la pantalla Nextion. El desarrollo del código necesario para integrar estos componentes y la visualización de datos es crucial, utilizando librerías y funciones específicas.

Se demuestra cómo la interfaz creada puede llegar a mejorar la precisión en la presentación de los datos del controlador PID de la velocidad del vehículo y verifica su exactitud con valores obtenidos de un escáner. Se destacan los valores proporcionados por el sensor MPU6050 en términos de ángulo de dirección, par de dirección y velocidad de giro del volante.

El documento también aborda problemas actuales de interfaces didácticas limitadas en términos de interactividad y representación gráfica de datos. Propone el desarrollo de una interfaz personalizada que facilite una mejor gestión y análisis de datos, optimizando la enseñanza y formación en ingeniería automotriz.

Destaca la necesidad de interpretar con precisión la trama de datos del CAN BUS para el mantenimiento efectivo de los sistemas automotrices modernos. La interfaz propuesta busca mejorar la interacción y comprensión de los usuarios, proporcionando una herramienta educativa para la captura, interpretación y aplicación de datos en tiempo real.

Palabras Claves: Dirección Electroasistida, PID, CAN Bus, HMI, Hyundai Elantra, OBDII

ABSTRACT

The document discusses the creation of an interface for the monitoring and control of an educational electric power steering (EPS) bench, used in the teaching of automotive engineering. Electronic components such as Arduino, MPU6050, MCP2515, and a Nextion display, along with CAN bus and Bluetooth communication, are employed to develop an educational interface that allows students to better understand EPS systems.

The importance of EPS systems and their components, such as sensors that monitor steering torque, steering wheel angle, and vehicle speed, is highlighted. The methodology includes the integration of these sensors with ESP32 and Arduino, and the design of the interface on the Nextion display. The development of the necessary code to integrate these components and visualize data is crucial, utilizing specific libraries and functions.

It is demonstrated how the created interface can improve the accuracy in the presentation of the vehicle speed PID controller data and verify its accuracy with values obtained from a scanner. The values provided by the MPU6050 sensor in terms of steering angle, steering torque, and steering wheel rotation speed are highlighted.

The document also addresses current issues of educational interfaces that are limited in terms of interactivity and graphical data representation. It proposes the development of a customized interface that facilitates better data management and analysis, optimizing the teaching and training in automotive engineering.

The need to accurately interpret the CAN bus data stream for effective maintenance of modern automotive systems is emphasized. The proposed interface seeks to improve user interaction and understanding, providing an educational tool for real-time data capture, interpretation, and application.

Keywords: Electro Assisted Steering, PID, CAN Bus, HMI, Hyundai Elantra, OBD II

INTRODUCCIÓN

La creación de una interfaz destinada al seguimiento y control de un banco didáctico funcional de dirección con asistencia eléctrica es fundamental para la enseñanza y aprendizaje en la ingeniería automotriz. Este proyecto aborda la implementación de una interfaz utilizando componentes electrónicos como Arduino, MPU6050, MCP2515 y la pantalla Nextion, además de la comunicación CAN bus y Bluetooth.

En el capítulo 1 se proporciona una base sólida sobre de los sistemas de dirección asistida eléctrica (EPS), sus componentes y el funcionamiento del bus CAN. Se detallan los elementos clave del sistema EPS, como los sensores que monitorizan el par de dirección, ángulo de giro del volante y la velocidad del vehículo. También se exploran los sistemas de diagnóstico y monitoreo en vehículos, como el OBD II y el PID, y se introducen conceptos fundamentales de giroscopios y acelerómetros.

La metodología se desarrolla en el capítulo 2 se detalla la ejecución de los elementos electrónicos y su interconexión. Se detalla la integración del sensor MPU6050 con el ESP32 en el banco didáctico, la obtención del PID de velocidad del vehículo a través de la conexión del Arduino Uno con el MCP2515, y el diseño de la interfaz en la pantalla Nextion.

El desarrollo del código se describe en el capítulo 3 y es necesario para la integración de los componentes y la visualización de datos. Se presentan las librerías y funciones utilizadas para la programación de la MPU6050, la CAN bus y la pantalla Nextion. También se incluye el código para la comunicación Bluetooth, destacando la importancia de la sincronización y la correcta transmisión de datos entre los dispositivos.

En el capítulo 4, se evalúan los resultados derivados de la implementación de la interfaz, incluyendo la presentación de los datos del controlador PID de la velocidad del vehículo y se verifica su precisión mediante comparaciones con valores obtenidos de un escáner. Además, se muestran los valores del sensor MPU6050 en términos de ángulo de dirección, par de dirección y velocidad de giro del volante, resaltando la fiabilidad del sistema de visualización en tiempo real.

La investigación demuestra cómo la integración de estos componentes electrónicos y la creación de una interfaz amigable pueden mejorar significativamente el proceso de enseñanza en el ámbito de la ingeniería automotriz, permitiendo a los estudiantes obtener una comprensión más profunda y práctica de los sistemas de dirección asistida eléctrica.

PROBLEMA

El desarrollo de interfaces personalizadas para bancos didácticos de dirección con asistencia eléctrica es esencial para optimizar la gestión y análisis de datos, lo que a su vez promueve una comprensión más profunda y una formación efectiva en ingeniería automotriz. Actualmente, las interfaces disponibles presentan varias deficiencias que limitan la interactividad y la representación gráfica de los datos, lo que dificulta la identificación rápida de problemas y puede resultar en tiempos de diagnóstico prolongados y costosos.

Estas limitaciones no solo afectan la eficiencia y seguridad del sistema de dirección con asistencia eléctrica, sino que también obstaculizan la enseñanza efectiva de conceptos relacionados con sistemas automotrices avanzados. Las interfaces existentes a menudo carecen de opciones de personalización, restringiendo la capacidad de adaptar la visualización de datos a las necesidades específicas de los usuarios. Esta falta de personalización impide que los estudiantes y técnicos puedan enfocarse en los aspectos más relevantes para sus tareas particulares.

Además, la carencia de herramientas analíticas robustas dificulta la realización de diagnósticos precisos de problemas en el sistema, lo que es crucial para el mantenimiento y la optimización de los sistemas automotrices modernos. La falta de una interfaz intuitiva y funcional también puede desmotivar a los estudiantes, limitando su capacidad de aprender y aplicar conocimientos prácticos en situaciones reales.

Por consiguiente, es esencial desarrollar interfaces personalizadas para los bancos didácticos de dirección con asistencia eléctrica. Estas interfaces deben facilitar una gestión y un análisis de datos eficaces, promoviendo una comprensión más profunda y una formación más efectiva en este campo vital para la ingeniería automotriz. Implementar una solución que aborde estas carencias mejorará tanto la calidad de la enseñanza como la eficiencia en la identificación y resolución de problemas técnicos, contribuyendo de manera significativa a la formación de ingenieros automotrices altamente capacitados (Pinguil Peñaranda, 2023).

Delimitación del problema.

El presente estudio se centra en el Banco Didáctico Funcional de Dirección con Asistencia Eléctrica, localizado en la Universidad Politécnica Salesiana, Campus Sur, en Quito. Este banco didáctico se utiliza para simular y entender el funcionamiento de la dirección asistida eléctricamente en vehículos automotores, brindando a los estudiantes una herramienta práctica para su formación en ingeniería automotriz.

El proyecto se enfoca específicamente en el diseño de una interfaz para la visualización de los datos obtenidos a través del protocolo CAN BUS del banco didáctico. Este protocolo es fundamental en la comunicación de los sistemas electrónicos del vehículo, permitiendo la transmisión de datos entre diferentes componentes del sistema de dirección asistida, sin embargo, la falta de información,

La relevancia de este proyecto radica en la necesidad de interpretar con precisión la trama de datos del CAN BUS, la cual es crucial para llevar a cabo tareas de reparación y ajuste de manera eficiente. La correcta interpretación de estos datos permite a los estudiantes y técnicos obtener información en tiempo real sobre el estado y rendimiento de los componentes del sistema de dirección asistida eléctricamente. Esto, a su vez, facilita la detección de fallos, la realización de ajustes necesarios y la optimización del funcionamiento del sistema.

Adicionalmente, la comprensión y manejo de estos datos son esenciales para el mantenimiento efectivo de los sistemas automotrices modernos. La interfaz propuesta no solo busca mejorar la interacción y comprensión de los usuarios con el sistema, sino también proporcionar una herramienta educativa que refuerce el aprendizaje práctico sobre la captura, interpretación y aplicación de datos en tiempo real.

Objetivo General.

Desarrollar una interfaz eficiente y funcional para monitorear el Banco Didáctico Funcional de Dirección con Asistencia Eléctrica, implementando una interfaz de usuario intuitiva y detallada para mejorar la experiencia y optimizar el aprendizaje en el entorno académico.

Objetivos Específicos.

- Desarrollar un sistema de adquisición de datos que permita recopilar información relevante del Banco Didáctico Funcional de Dirección con Asistencia Eléctrica.
- Diseñar una interfaz de usuario intuitiva y amigable utilizando la pantalla Nextion para visualizar los datos en tiempo real de manera clara y precisa.
- Implementar funcionalidades de análisis y diagnóstico en la interfaz diseñada para facilitar la interpretación de los datos obtenidos y mejorar la comprensión del funcionamiento del sistema de dirección con asistencia eléctrica.
- Desarrollar un sistema de ingeniería inversa que permita examinar la trama de datos de la red Can Bus para intercambiar información en los distintos nodos de la red.

CAPÍTULO 1
1. MARCO TEÓRICO

1.1. Sistema de Dirección Asistida Eléctrica

El sistema de dirección asistida eléctrica (EPS) es una innovación significativa en la tecnología automotriz, diseñada para mejorar la eficiencia, la comodidad y la seguridad de los vehículos. A diferencia de los sistemas hidráulicos tradicionales, el EPS emplea motores eléctricos y sensores, eliminando la necesidad de componentes hidráulicos pesados y complejos.

Esta tecnología ofrece numerosas ventajas, incluyendo una reducción del peso del vehículo, una mayor eficiencia en el consumo de combustible, y una respuesta de dirección más precisa. Además, el EPS permite ajustar la asistencia de dirección según las condiciones de manejo y se integra fácilmente con otros sistemas electrónicos del vehículo, como el control de estabilidad y los sistemas de asistencia al conductor, también reduce los costos de mantenimiento al eliminar componentes hidráulicos propensos a desgaste y fugas (Shimizu & Kawai, 1991) .

1.1.1. Funcionamiento del EPS

La ECU del EPS calcula la asistencia necesaria y envía señales al motor eléctrico, que puede estar montado en la columna de dirección o en el piñón. Este motor eléctrico, con una potencia típica entre 300 y 600 vatios, proporciona la fuerza necesaria para asistir al conductor en el giro del volante. La asistencia se ajusta dinámicamente: se incrementa a bajas velocidades para facilitar maniobras como el estacionamiento y se reduce a altas velocidades para mejorar el control y la estabilidad del vehículo (Zhan & Yan, 2018).

1.1.2. Elementos del Sistema de Dirección Asistida Eléctrica (EPS)

1.1.2.1.Volante de dirección

El volante es un componente esencial del vehículo que permite al conductor controlar su dirección, transmitiendo movimientos a las ruedas y facilitando los giros. Forma

parte del sistema de dirección junto con la columna de dirección, el piñón y cremallera, la caja de dirección y las barras de dirección. Además, puede incorporar controles para la bocina, limpiaparabrisas, control de crucero, sistema de audio y otros sistemas electrónicos. En vehículos modernos, incluye sistemas de seguridad como el airbag del conductor y tecnologías avanzadas como sensores de ángulo de dirección, mejorando la precisión y estabilidad del vehículo. Estos avances proporcionan una experiencia de conducción más segura y cómoda (Arrata Zambrano & Yoza Rodríguez, 2020).

1.1.2.2.Columna de dirección

La columna de dirección es un componente esencial del sistema de dirección de un vehículo, conectando el volante con el mecanismo de dirección y transmitiendo los movimientos del volante a las ruedas. Incluye el eje de dirección, cojinetes y sistemas de ajuste de altura y profundidad del volante. En vehículos modernos, incorpora sistemas de seguridad y asistencia como airbags, controles electrónicos y mecanismos antirrobo. Los diseños actuales incluyen sistemas de absorción de energía para minimizar el impacto en colisiones, así como sensores y actuadores que trabajan con la dirección asistida electrónica (EPS) para una dirección más precisa y menos fatigante. Esta evolución subraya la creciente importancia de la columna de dirección en la conducción y la seguridad vial (Northeast Auto Service, 2023).

1.1.2.3.Caja de dirección

El movimiento del volante se transmite a través del eje de dirección hasta la caja de dirección, donde se convierte en un movimiento lineal. Este movimiento lineal se transfiere mediante barras articuladas con rótulas a las bieletas o brazos de acoplamiento, que giran las ruedas alrededor de su eje.

Tipos de cajas o mecanismos de dirección

- **Tornillo sinfín y rodillo:** Usa un tornillo que engrana con un rodillo, convirtiendo el giro en movimiento lineal. Es duradero y fuerte.
- **Tornillo sinfín y dedo:** Un tornillo mueve un brazo con un dedo acoplado, proporcionando una dirección precisa.

- **Tornillo sinfín y tuerca con bolas circulantes:** Usa bolas entre el tornillo y la tuerca para reducir fricción, proporcionando una dirección más suave y eficiente.
- **Tornillo sinfín y tuerca:** Convierte el giro en movimiento lineal usando un tornillo que desplaza una tuerca a lo largo de su eje.
- **Tornillo sinfín y sector dentado:** Combina un tornillo con un sector dentado que mueve el brazo de dirección.
- **Cremallera de relación variable:** Similar al piñón y cremallera, pero con una relación que cambia según cuánto se gira el volante, proporcionando más control a bajas velocidades y estabilidad a altas velocidades.
- **Piñón y Cremallera:** Convierte el giro del volante en un movimiento de lado a lado para dirigir las ruedas. Es simple y eficiente, y se usa en la mayoría de los autos modernos (Quispe Mamani, 2015).

1.1.2.4.Neumático y rueda

Los neumáticos y las ruedas son esenciales para el sistema de dirección asistida eléctrica (EPS) debido a su papel en la transmisión de la dirección y la tracción del vehículo. Los neumáticos, que proporcionan contacto con la carretera, afectan el manejo, la estabilidad y la seguridad del vehículo. La calidad y el estado de los neumáticos influyen en la eficacia del EPS, ya que un buen agarre y una presión adecuada permiten respuestas precisas a las maniobras del conductor. Las ruedas traducen el movimiento lineal del EPS en cambios de dirección del vehículo. El EPS requiere fricción adecuada entre los neumáticos y la carretera para funcionar bien, y los neumáticos desgastados pueden reducir su efectividad, dificultando el manejo y aumentando el riesgo de accidentes (Nazaruddin et al., 2019).

1.1.2.5.Batería y Sistema Eléctrico

El sistema EPS depende de la batería y del sistema eléctrico del vehículo para funcionar. Proporciona la energía necesaria para el motor eléctrico y la ECU.

1.1.2.6.Conectores y Arnés de Cableado

Estos elementos aseguran que todos los componentes electrónicos del EPS estén correctamente conectados y puedan comunicarse entre sí.

Los elementos mencionados anteriormente se visualizan de la siguiente manera, como se muestra en la Figura 1. Banco didáctico funcional de dirección con asistencia eléctrica de la Universidad Politécnica Salesiana, Quito, Campus Sur.

Figura 1: Banco didáctico de dirección con asistencia eléctrica

Maqueta de dirección electrónica. Fuente: (Autores)

1.1.3. Componentes electrónicos del Sistema de Dirección Asistida Eléctrica

A continuación, se describen los principales componentes del EPS y su funcionamiento:

- **Motor MDPS**

  El motor de asistencia en sistemas de dirección asistida electrónicamente (EPS), como se muestra en la Figura 2. El motorizado (MDPS), es crucial para proporcionar la fuerza necesaria que ayuda al conductor a girar el volante. Controlado por la unidad de control de dirección, este motor ajusta la corriente y la tensión para asegurar un rendimiento óptimo. En configuraciones típicas, opera con hasta 24 amperios de corriente máxima y 12 voltios de corriente continua (DC), alcanzando velocidades de hasta 200 rpm, lo que permite una asistencia rápida y efectiva en diversas condiciones de manejo (Zhang & Wang, 2015).

Figura 2: Motor MDPS en el banco didáctico

Ubicación del motor MDPS en el banco didáctico funcional con asistencia eléctrica.

Fuente: (Autores)

- **Unidad de Control Electrónico del Sistema de Dirección Asistida Eléctrica (EPS)**

  La Unidad de Control Electrónico (ECU) es esencial en los sistemas de Dirección Asistida Eléctrica (EPS), actuando como el cerebro central. Esta unidad procesa datos de múltiples sensores para calcular la asistencia necesaria en tiempo real, controlando el motor eléctrico y ajustando la asistencia según el esfuerzo del conductor, la velocidad del vehículo y el ángulo de dirección. Utiliza señales de sensores como el torque, el ángulo del volante, la velocidad del vehículo y las revoluciones del motor para determinar la cantidad precisa de asistencia mediante algoritmos predefinidos (Criollo Cabrera & Álvarez Torres, 2023), como se muestra en la Figura 3. Además de ajustar la corriente del motor de dirección para generar el torque adecuado y facilitar el giro del volante, la ECU incorpora funciones avanzadas como la compensación de inercia, el retorno activo del volante y la amortiguación. Estas características mejoran la respuesta del sistema, la estabilidad y la seguridad durante la conducción (Zhang & Wang, 2015).

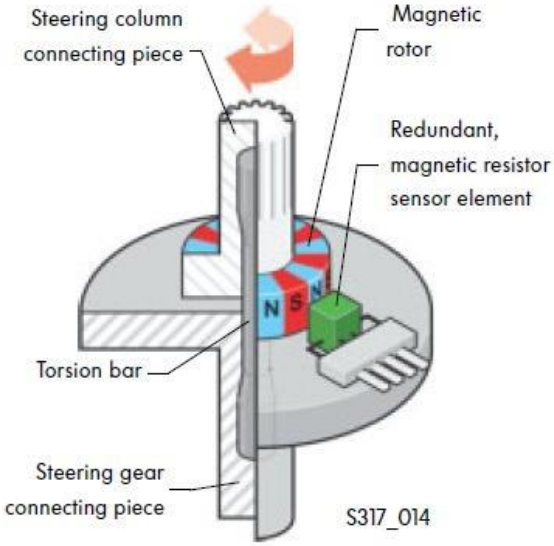
Figura 3: Módulo de dirección del banco

Ubicación del módulo de dirección en el banco didáctico funcional con asistencia eléctrica. Fuente: (Autores)

- **Sensor de Par de Dirección**

El EPS utiliza un sensor de par para medir el esfuerzo aplicado por el conductor al girar el volante. Este sensor convierte la fuerza en una señal eléctrica que se envía a la ECU. La ECU procesa esta información y determina la cantidad exacta de asistencia que debe proporcionar el motor eléctrico para optimizar la dirección, como se muestra en la Figura 4. Emplea curvas de características preestablecidas que consideran diversos parámetros como el par aplicado, la velocidad del vehículo, el voltaje, la corriente y la posición del rotor del motor. Estos parámetros son esenciales para calcular la corriente óptima requerida por el motor de asistencia, ajustándose en tiempo real para asegurar que las ruedas reciban la fuerza de dirección adecuada. Este ajuste dinámico mejora la precisión y la respuesta del sistema de dirección, proporcionando una experiencia de conducción más suave y segura (Shimizu & Kawai, 1991).

Figura 4: Sensor de par de dirección

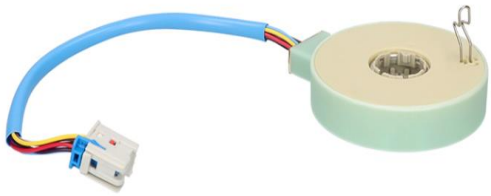


El gráfico muestra un sensor para dirección asistida, utilizando un rotor magnético y sensores para medir la torsión en la columna de dirección. Fuente: (Ingeniería y Mecánica Automotriz,2021)

- **Sensor de Ángulo de Dirección**

  El sensor de ángulo de dirección es crucial para medir y enviar la información sobre el giro de la transmisión de datos al módulo electrónico de control de la columna de dirección. Ubicado típicamente detrás del anillo retractor con el anillo colector del sistema de airbag, este sensor consta de un disco de codificación con anillos que incluyen múltiples parejas de barreras luminosas, como se muestra en la Figura 5. El anillo de incrementos, dividido en segmentos de 72° con almenas codificadas de manera variable, y el anillo de valores absolutos, explorado por seis parejas de barreras luminosas, permiten medir ángulos de hasta 1044°. El principio de funcionamiento se basa en la interrupción de la luz entre la fuente de luz y el sensor óptico, generando señales de tensión que la unidad de control interpreta para calcular el movimiento angular y determinar la posición inicial del volante (Quispe Mamani, 2015).

Figura 5: Sensor ángulo de dirección



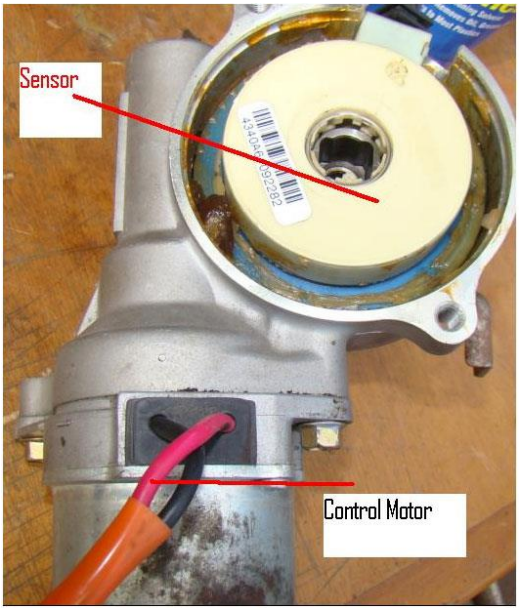
El gráfico muestra un sensor de ángulo de dirección con un conector eléctrico multicolor y un cable de conexión azul. Fuente: (Alfa e Parts, 2005)

- **Sensor de velocidad de giro del volante**

  El sensor de velocidad de giro del volante es un componente esencial en los sistemas de dirección asistida eléctrica (EPS), como se muestra en la Figura 6. Encargado de medir la rapidez con la que se gira el volante. Este sensor proporciona datos críticos a la unidad de control electrónico (ECU), que ajusta la cantidad de asistencia que el sistema EPS debe proporcionar, mejorando así la precisión y la respuesta de la dirección. El sensor generalmente funciona mediante principios ópticos o magnéticos; en el caso de los sensores ópticos, un disco codificado conectado al volante interrumpe un haz de luz, creando una serie de pulsos que se interpretan como velocidad angular. En los sensores magnéticos, el principio es similar, pero utiliza cambios en un campo magnético detectados por un sensor Hall. Estos datos permiten a la ECU determinar tanto la dirección como la velocidad del giro del volante, lo cual es crucial para la estabilidad y seguridad del vehículo, especialmente en maniobras rápidas o de emergencia (Jie & Yanling, 2021).

  La precisión de este sensor es vital para el rendimiento del EPS, ya que permite ajustes en tiempo real que facilitan una experiencia de conducción suave y segura (Badawy et al., 1999).

Figura 6: Sensor de velocidad de giro del volante



El gráfico muestra el sensor montado sobre un motor de control. Fuente: (Auto Avance, 2022)

- **Sensor de Velocidad del Vehículo**

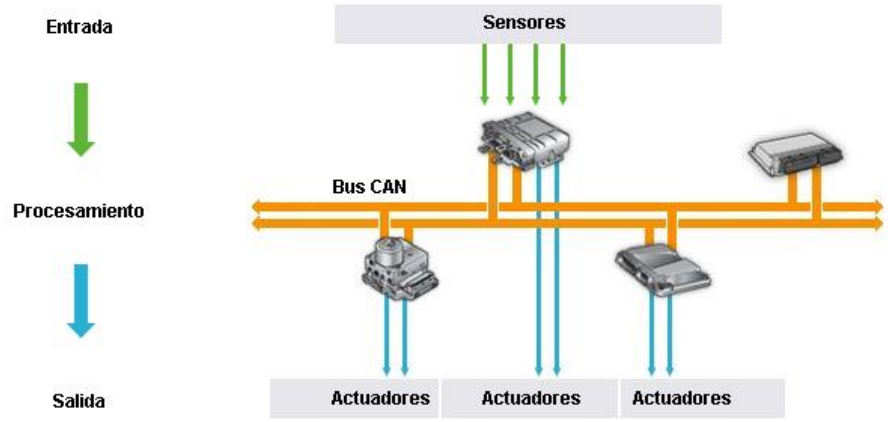
  El sensor de régimen del motor tiene la misión de determinar la cantidad de vueltas del cigüeñal por minuto. Esta importante magnitud de entrada se calcula en la unidad de control del motor y puede cuantificarse a partir de la señal del sensor inductivo de revoluciones del cigüeñal. El sensor de régimen del rotor es parte integral del motor para la dirección asistida electromecánica, mide la velocidad del rotor del motor eléctrico que asiste la dirección. Este sensor es crucial para el funcionamiento preciso del sistema de dirección asistida, ya que permite ajustar la cantidad de asistencia proporcionada en función de la velocidad del motor y las condiciones de manejo. La señal del sensor se procesa en la unidad de control del motor, lo que garantiza que la dirección asistida funcione de manera óptima, mejorando la seguridad y la comodidad del conductor al proporcionar la cantidad adecuada de asistencia en todo momento (Chafla Taquinga & Salinas Mejía, 2012).

1.2. Red CAN BUS

La red de área de controlador (CAN), como se muestra en la Figura 7. Es un tipo de comunicación en serie creado por Bosch en los años 80. Este sistema establece una

manera estándar de comunicarse de manera eficiente y confiable entre sensores, actuadores, controladores y otros nodos en tiempo real. Aunque empezó siendo utilizado principalmente por la industria automotriz, hoy en día se encuentra en una gran variedad de vehículos, como coches, camiones, barcos y hasta naves espaciales. Además, el protocolo CAN también es común en la automatización industrial y en otros sistemas de control conectados en red. En los vehículos, la red CAN conecta muchos módulos que interactúan con el entorno y manejan datos a diferentes velocidades. Algunas partes de la red tienen que procesar información rápidamente para cumplir con las normas de seguridad y emisiones, mientras que otras funcionan a una velocidad menor y están conectadas a elementos como luces e interruptores (Barona Zaldumbide & Terán Burgos, 2023).

Figura 7: Red Can Bus



El gráfico muestra la arquitectura de un sistema de comunicación en un vehículo basado en el Bus CAN. Fuente: (Auto Soporte, 2023)

1.2.1. Funcionamiento de la Red CAN BUS

La tecnología CAN Bus destaca por su fiabilidad en la comunicación, esencial para sistemas de tiempo real y alta integridad. Sus características clave incluyen:

- Ofrece herramientas para detectar errores en la transmisión y retransmitir automáticamente los datos erróneos, asegurando una comunicación confiable.

- Puede distinguir entre errores puntuales y fallos de nodo, desconectando los nodos defectuosos para evitar que el error sature la red.
- Los mensajes se priorizan y se garantizan tiempos de latencia, crucial para aplicaciones de tiempo real donde el tiempo de respuesta es vital.
- Asegura que los datos transmitidos sean consistentes, manteniendo la integridad de la información en todo momento.
- La red CAN permite una configuración flexible, permitiendo añadir o quitar nodos de forma dinámica sin afectar su funcionamiento. Soporta hasta 110 nodos en una red, facilitando su adaptación a diversas aplicaciones y necesidades

La tecnología CAN destaca por su fiabilidad en la comunicación, crucial para sistemas de tiempo real y alta integridad. Proporciona herramientas para detectar y retransmitir automáticamente datos erróneos, diferenciando entre errores puntuales y fallos de nodo, y desconectando los nodos defectuosos para evitar la saturación de la red. Además, prioriza mensajes y asegura tiempos de latencia, garantizando la consistencia de los datos. Permite una configuración flexible de la red, posibilitando la adición o eliminación dinámica de nodos sin afectar su funcionamiento, y soporta hasta 110 nodos (Freiberger et al., 2011)

1.2.2. Can High y Can Low

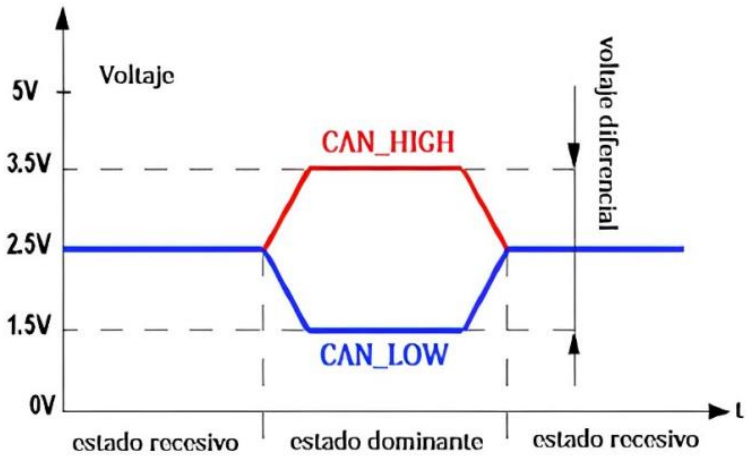
La red CAN (Controller Area Network) emplea un método de comunicación diferencial que se realiza a través de dos cables denominados CAN High y CAN Low.

Este enfoque diferencial es esencial para garantizar la robustez y la resistencia a interferencias del protocolo CAN, especialmente en los entornos automotrices adversos.

La transmisión de datos en el bus CAN se basa en la diferencia de voltaje entre CAN_H y CAN_L, donde los voltajes cambian en direcciones opuestas durante la transmisión. Cuando se envía un bit dominante (0 lógico), como muestra la Figura 8. El voltaje en CAN_H se incrementa (aproximadamente a 3.5V) mientras que el

voltaje en CAN_L disminuye (alrededor de 1.5V). Para un bit recesivo (1 lógico), ambos voltajes se estabilizan en un nivel intermedio de aproximadamente 2.5V (Ambrosio et al., 2018).

Figura 8: Red Can High y Can Low



El gráfico muestra las señales de la red CAN en sus estados CAN High y CAN Low. Fuente: (Barona Zaldumbide & Terán Burgos, 2023)

Esta técnica no solo permite una transmisión eficiente de datos, sino que también es eficaz para reducir las interferencias electromagnéticas (EMI), ya que las perturbaciones externas afectan a ambos cables de manera similar, permitiendo que el receptor interprete la diferencia de voltaje y cancele la interferencia común (Bosch, 1991).

La comunicación diferencial en la red CAN no solo reduce las interferencias, sino que también proporciona una gran robustez y fiabilidad gracias a sus mecanismos de detección y corrección de errores. Los controladores CAN tienen herramientas para identificar errores de paridad, errores de bits y otros fallos en la comunicación, lo que asegura la integridad de los datos transmitidos. Si se detecta un error, el sistema retransmite automáticamente el mensaje defectuoso, garantizando que la información importante se entregue de manera precisa y sin interrupciones (Feliciano Fuentes, 2019).

1.2.3. Comunicación con Otros Dispositivos

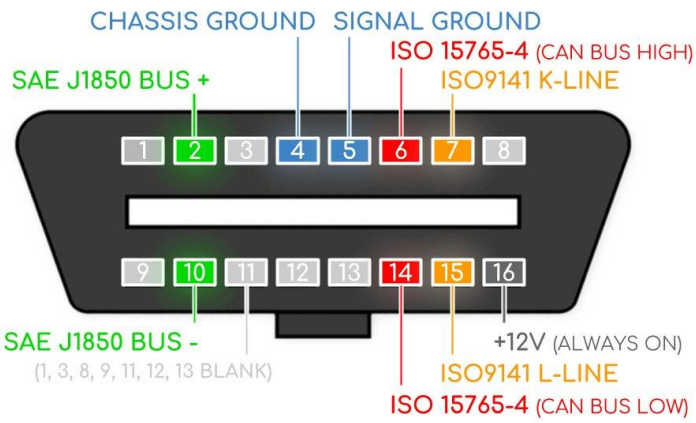
En el sistema OBD–II cuenta con cinco protocolos de comunicación, como se muestra en la Tabla 1. Un automóvil normalmente usa sólo uno de ellos como se muestra en la Figura 9. Estos son:

Tabla 1: Protocolos de OBD II

PROTOCÓLOS DE OBD II	
J1850 PWM	Es un tipo de modulación por ancho de pulso utilizado por Ford y Mazda.
J1850 VPW	Es otro tipo de modulación por ancho de pulso utilizado por General Motors y camiones ligeros
ISO91412	Es un protocolo antiguo utilizado en vehículos Chrysler, así como en vehículos europeos y asiáticos entre los años 2000 y 2004.
ISO142304	También conocido como KWP2000 (Keyword Protocol 2000), es comúnmente utilizado en vehículos a partir del año 2003.
ISO157654	Se refiere al protocolo CANBus, introducido por primera vez en 2004 y obligatorio en vehículos en Estados Unidos desde 2008 (Arroyo Núñez, n.d.).

En la tabla se muestra los protocolos de comunicación. Fuente: (OBDTester, 2023)

Figura 9: Pines de puerto OBD-II según los protocolos



El gráfico muestra el conector OBD-II Fuente: (Automotive Electronic Technology, 2022)

1.2.4. Byte

Un byte es una unidad de información digital que consta de 8 bits, donde cada bit puede tener un valor de 0 o 1. Esto permite que un byte represente 256 valores diferentes, desde 0 hasta 255. Los bytes son fundamentales en la informática y las telecomunicaciones, utilizados para almacenar y codificar datos, representar caracteres en sistemas como ASCII, y gestionar direcciones de memoria en computadoras. Además, son esenciales en la transmisión y procesamiento de datos, y forman la base para unidades de almacenamiento más grandes como kilobytes, megabytes y gigabytes (Patterson & Hennessy, 2016).

1.2.5. Hexadecimal

El sistema hexadecimal, o base 16, es un sistema de numeración utilizado en computación y programación que emplea 16 símbolos: los números del 0 al 9 y las letras A a F, donde A equivale a 10, B a 11, hasta F que equivale a 15 (Patterson & Hennessy, 2016).

El sistema hexadecimal es importante en la computación por varias razones:

- Convierte números binarios largos en formatos más manejables y legibles.

- Utilizado para representar direcciones de memoria de forma más concisa.
- Ayuda a los programadores a inspeccionar y manipular datos a bajo nivel.
- Es comúnmente usado para especificar colores en diseño web (Tanenbaum, 2009).

1.2.6. Trama de datos

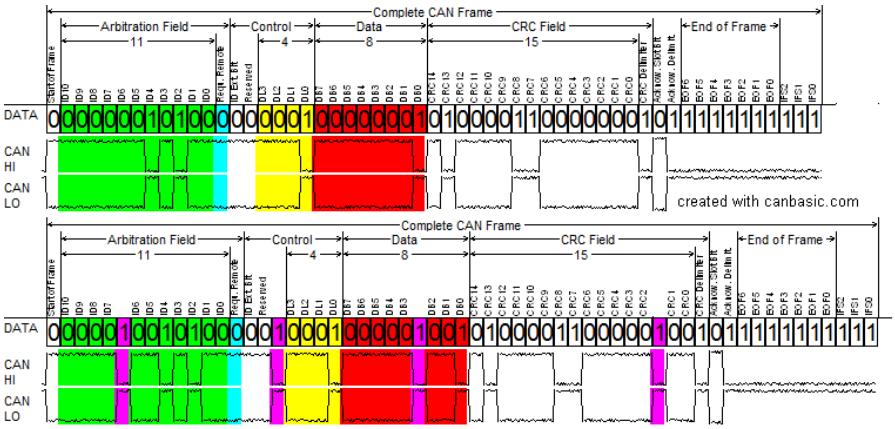
El proceso de envío de información en una red CAN, es capaz de enviar hasta 8 bytes. Se inicia con un identificador de 11 bits (formato estándar) o 29 bits (formato extendido).

La trama comienza con un bit dominante para sincronización seguido del identificador, que determina la relevancia de la trama en la red como se muestra en la Figura 10.

Las tramas con identificadores menores tienen mayor prioridad, sin embargo, el formato estándar incluye 11 bytes más 1 byte adicional para diferenciar entre tramas de datos y de petición remota (Tanenbaum, 2009).

El campo de control tiene 2 bytes reservados y 4 bytes que indican la longitud del campo de datos (08 bytes). El campo de CRC garantiza la integridad de los datos, seguido por un bit delimitador CRC, un campo de reconocimiento de 2 bytes (ACK), y el EOF con 7 bytes recesivos. Entre tramas, debe haber un espacio de 3 bytes recesivos (Martínez Requena, 2017).

Figura 10: Trama de datos



El gráfico muestra una trama de datos CAN completa, incluyendo campos de arbitraje, control, datos, CRC y fin de trama, con señales diferenciadas en CAN High y CAN Low. Fuente: (Researchgate, 2017)

1.2.7. Decodificación de la trama de datos

La decodificación de una trama de datos en una red CAN implica interpretar los distintos campos que forman la trama para extraer la información relevante. A continuación, se describe el proceso de decodificación.

Inicio de la Trama (Start of Frame SOF):

- Un solo bit dominante que marca el comienzo de la trama.

Campo de Identificador:

- Formato estándar (11 bits): Se utiliza para identificar la prioridad y la importancia de la trama en la red.
- Formato extendido (29 bits): Similar al formato estándar, pero con mayor longitud para permitir una mayor variedad de identificadores.

Bit RTR (Remote Transmission Request):

- Indica si la trama contiene datos (valor 0) o es una solicitud remota (valor 1).

Campo de Control:

- IDE (Identifier Extension): Indica si se usa el formato estándar (valor 0) o el extendido (valor 1).
- r0 (Reserved bit): Reservado para usos futuros.
- DLC (Data Length Code): 4 bits que indican el número de bytes de datos (0 a 8 bytes).

Campo de Datos:

- Contiene los datos de la trama, que pueden variar de 0 a 8 bytes según lo indique el DLC.

Campo de CRC (Cyclic Redundancy Check):

- Proporciona una verificación de integridad para los datos transmitidos.

Delimitador CRC:

- Un bit recesivo que sigue al campo de CRC.

Campo de Reconocimiento (ACK):

- 2 bits: uno dominante para la señal de ACK y uno recesivo como delimitador ACK.
- Indica que la trama ha sido recibida correctamente por al menos un nodo en la red.

Fin de la Trama (End of Frame EOF):

- 7 bits recesivos que indican el final de la trama.

Espaciado Intertrama (Intermission):

- 3 bits recesivos que deben estar presentes entre dos tramas consecutivas (Martínez Requena, 2017).

Este proceso asegura que los datos transmitidos se reciban y verifiquen correctamente, garantizando la integridad y el orden de las tramas en la red CAN (Martínez Requena, 2017).

1.3. Sistemas de Diagnóstico y Monitoreo en Vehículos

1.3.1. OBD II

El diagnóstico a bordo (OBD) y los códigos PID de OBD-II son esenciales en la industria automotriz moderna, facilitando la detección y corrección de problemas en vehículos mediante tecnología avanzada y estándares uniformes. Desde sus primeros usos en la década de 1980 hasta la obligatoriedad de OBD-II en 1994 en Estados Unidos, estos sistemas han evolucionado significativamente. OBD-II estandarizó el formato de mensajería y los protocolos de señalización eléctrica, mejorando la precisión del diagnóstico y permitiendo la comunicación en tiempo real entre vehículos y herramientas de escaneo (Concepcion, 2011).

1.3.2. PID

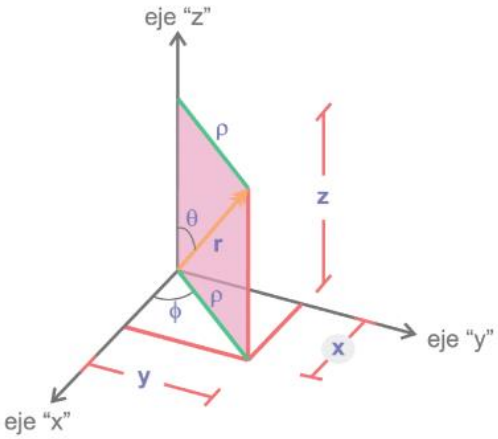
Los códigos PID (Parameter Identification) de OBD-II son herramientas clave en este proceso, permitiendo a los técnicos automotrices solicitar y recibir datos específicos del vehículo, como información del motor, sensores de oxígeno y presión de combustible. Estos códigos, definidos por el estándar SAE J1979, varían según el modo de operación y pueden ser personalizados por los fabricantes de vehículos para adaptarse a necesidades específicas de diagnóstico y monitoreo. La interpretación de la respuesta de los PID implica el uso de fórmulas y códigos binarios, proporcionando información detallada para identificar y resolver problemas mecánicos y electrónicos de manera eficiente (Rivera Fuentes, 2016).

1.4. Sistema de coordenadas

Un sistema de coordenadas es un marco de referencia utilizado para localizar puntos en el espacio o en un plano mediante números llamados coordenadas.

Los sistemas más comunes incluyen las coordenadas cartesianas, que utilizan ejes perpendiculares (X, Y y Z en el espacio tridimensional), como se muestra en la Figura 11. Especificar la posición de un punto mediante valores numéricos. Otros sistemas como las coordenadas polares, cilíndricas y esféricas emplean diferentes formas de medir las ubicaciones, utilizando ángulos y distancias radiales adicionales para representar los puntos. Estos sistemas son esenciales en disciplinas como matemáticas, física, ingeniería y geografía, facilitando la representación gráfica, el cálculo de distancias y la resolución de problemas geométricos y físicos de manera precisa y sistemática (Herrera Escudero, n.d.).

Figura 11: Coordenadas en el plano

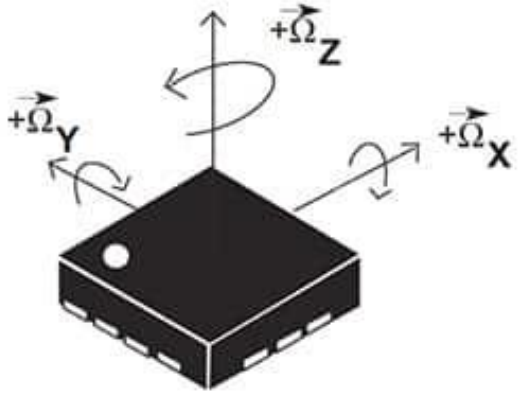


El gráfico muestra un sistema de coordenadas cilíndricas, donde se representan los ejes cartesianos x, y, z. Fuente: (Udoccz, 2024)

1.5. Giroscopio

Los giroscopios son dispositivos que utilizan el principio de conservación del momento angular, con un rotor giratorio que mantiene constante su eje de rotación a menos que se aplique un par externo, como se muestra en la Figura 12. Estos dispositivos son fundamentales para detectar la velocidad angular alrededor de un eje específico en relación con un sistema de referencia inercial. Son utilizados en una variedad de aplicaciones que requieren mantener la orientación precisa, como sistemas de navegación, estabilización de cámaras y control de vuelo de aeronaves (Sifuentes De La Hoya et al., 2023).

Figura 12: Ejes de giro del giroscopio

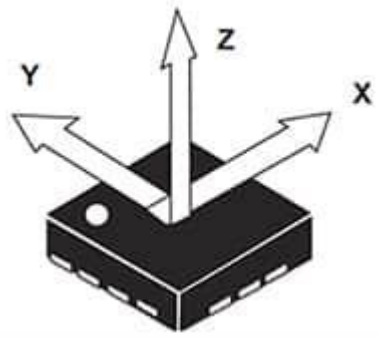


El gráfico muestra un giroscopio con sus tres ejes de rotación Fuente: (Digiker,2018)

1.6. Acelerómetro

El acelerómetro es un dispositivo esencial para medir la aceleración de un objeto, detectando tanto la aceleración estática, como la debida a la gravedad, y la dinámica, que incluye movimientos y vibraciones. Existen varios acelerómetros, cada uno diseñado para diferentes aplicaciones y niveles de precisión, como se muestra en la Figura 13. Utiliza tecnologías como el piezoeléctrico y el capacitivo: el primero genera carga eléctrica en respuesta a fuerzas aplicadas, mientras que el segundo mide cambios en la capacitancia debido al movimiento de estructuras internas. Este dispositivo tiene aplicaciones fundamentales en sistemas de navegación, estabilización de dispositivos móviles, y detección de caídas en dispositivos médicos, proporcionando datos cruciales para aplicaciones de seguimiento de movimiento y seguridad (Polizzi et al., 2020).

Figura 13: Ejes del acelerómetro



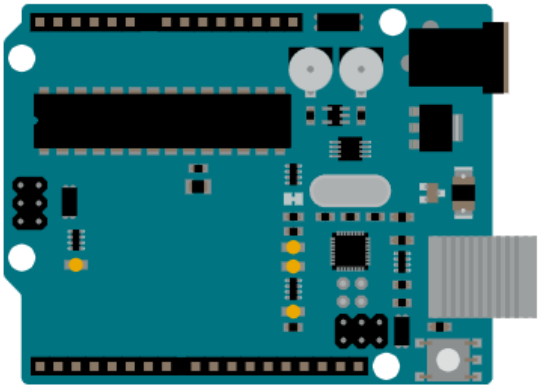
El gráfico muestra un acelerómetro con sus tres ejes de detección etiquetados como x, y, z. Fuente: (Digikey, 2018)

1.7. Arduino

Arduino UNO es una placa de microcontrolador versátil permite reemplazar el microcontrolador en caso de necesidad, proporcionando una plataforma robusta y accesible para estudiantes y entusiastas de la electrónica. Las características del Arduino se presentan en la Tabla 2. Este microcontrolador incluye múltiples pines digitales y analógicos para entradas y salidas, un oscilador de cristal para sincronización precisa, una

memoria flash para almacenamiento del código, y una EEPROM para guardar datos de manera no volátil

Figura 14: Arduino



El gráfico muestra una placa Arduino Uno, que incluye un microcontrolador ATmega.
Fuente: (Arduino Docs, 2024)

Tabla 2: Características del Arduino

Características	Descripción
Microcontrolador	ATmega328P
Frecuencia	16 MHz
Flash Memory	32 KB
Pines de Entrada/Salida Digitales	14 (6 de ellos PWM)
Pines de Entrada Analógica	**6**
Corriente Máxima por Pin I/O	40 mA
Tensión de Funcionamiento	5V
Tensión de Entrada (recomendada)	7-12V
Conector de Programación	USB Type B

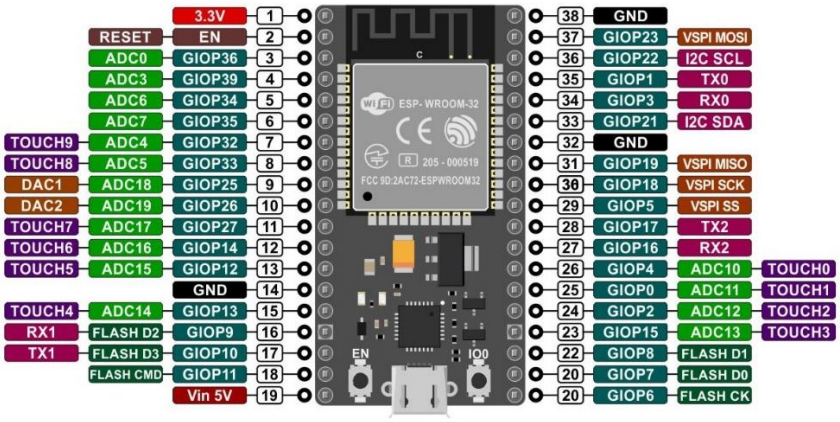
En la tabla se muestra las características generales. Fuente: (Arduino Docs, 2024)

1.8. ESP32

La ESP32 de Espressif Systems es un potente microcontrolador diseñado para aplicaciones IoT, equipado con CPU dual-core, interfaces de comunicación Wi-Fi y

Bluetooth, y características avanzadas de ahorro de energía, ideal para proyectos que demandan conectividad estable, procesamiento eficiente de señales y seguridad en la transmisión de datos (Beningo Jacob, 2020).

Figura 15: ESP32



El gráfico muestra una placa ESP32 con pines de alimentación y pines de interfaces como I2C, SPI y UART. Fuente: (Vasanza, 2021)

Las características de la placa ESP32 se detallan en la Tabla 3. Esta placa, basada en el microcontrolador ESP32, es conocida por su alta integración y capacidad de procesamiento. El ESP32 incluye un procesador dual-core Xtensa LX6, conectividad Wi-Fi y Bluetooth (Beningo Jacob, 2020)

Tabla 3: Características de la ESP32

Característica	Descripción
Procesador	Tensilica Xtensa X36
Nº bits	32
Nº Núcleos	2
Velocidad CPU	160 MHz-240 MHz
Alimentación	3 V- 3,6 V
Frecuencia reloj	40 MHz
Consumo	80 mA-225 mA (2.5 uA en deep sleep)
Wifi	802.11 b/g/n 2,4 ~ 2,5 GHz
Bluetooth	v4.2 BR/EDR

Característica	Descripción
Pines E/S	36

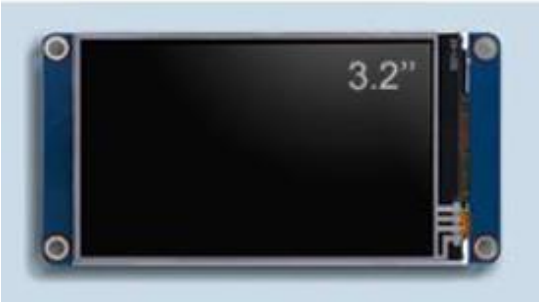
En la tabla se muestra las características generales. Fuente: (Ikiss, 2020)

1.9. Pantalla Nextion

Una pantalla Nextion es una interfaz hombre-máquina (HMI) avanzada que integra un microcontrolador y una pantalla táctil para simplificar el desarrollo de interfaces gráficas, como muestra en la Figura 16.

Utiliza un editor de software propio para diseñar interfaces sin necesidad de programación compleja, permitiendo la incorporación de botones, gráficos y texto. La comunicación con otros dispositivos se realiza mediante una interfaz serie UART, facilitando la integración con microcontroladores como Arduino y ESP32. Las pantallas Nextion varían en tamaño desde 2.4 hasta 7 pulgadas, con resoluciones que van desde 320x240 hasta 800x480 píxeles. Además, incluyen memoria flash interna para almacenar los diseños de interfaz y un procesador dedicado para gestionar la lógica de la pantalla, aliviando la carga del microcontrolador principal. Su versatilidad y facilidad de uso las hacen ideales para aplicaciones en automatización, electrodomésticos inteligentes y sistemas embebidos (Venegas Álvarez, 2023)

Figura 16: Pantalla Nextion



El gráfico muestra una pantalla táctil Nextion de 3.2 pulgadas, utilizada para interfaces gráficas en proyectos electrónicos. Fuente: (Repositorio,2020)

Las características de la pantalla Nextion se presentan en la Tabla 4. Este dispositivo incluye una interfaz de usuario HMI (Human Machine Interface) que permite una

comunicación eficiente entre el usuario y el sistema. La pantalla Nextion ofrece una resolución de alta definición, capacidad táctil capacitiva o resistiva, y una interfaz UART para una fácil integración con microcontroladores y otros dispositivos (Venegas Álvarez, 2023)

Tabla 4: Características de la Pantalla Nextion

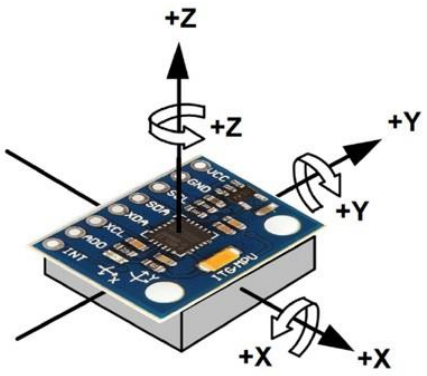
Característica	Descripción
Tamaño de Pantalla	De 2.4 a 7 pulgadas
Resolución	Desde 320x240 hasta 800x480 píxeles
Tipo de Pantalla	TFT LCD táctil resistiva o capacitiva
Memoria RAM	3584 bytes
Procesador	ARM Cortex-M0
Voltaje de Operación	5V
Consumo de Energía	~85 mA a 5V

En la tabla se muestra las características generales de la placa. Fuente: (Farnell, 2022)

1.10. MPU6050

El MPU6050 es un sensor MEMS que combina un acelerómetro de tres ejes y un giroscopio de tres ejes en un solo chip, fabricado por InvenSense, como se muestra en la Figura 17. Es popular en aplicaciones como drones, robots, estabilización de cámaras y dispositivos de realidad virtual debido a su capacidad de proporcionar datos precisos de aceleración y rotación (Alcalde Susi, 2019)

Figura 17: Ejes de un giroscopio



El gráfico muestra un sensor de seis ejes que combina un acelerómetro de tres ejes y un giroscopio de tres ejes. Fuente: (Mechatronics, 2023)

Una de las principales ventajas del MPU6050 es su integración de múltiples sensores en un solo chip, reduciendo el tamaño del dispositivo y el costo del hardware como se presenta en la Tabla 5. Sin embargo, el sensor puede ser susceptible al ruido y a la deriva, especialmente en el giroscopio, lo que puede afectar la precisión de las mediciones (Blanco González, 2021)

Tabla 5: Características del giroscopio

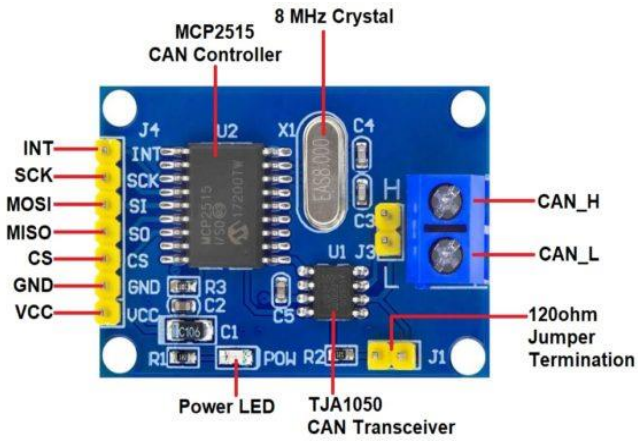
CARACTERÍSTICAS	**DESCRIPCIÓN**
Tipo de sensor	MEMS (Micro-Electro-Mechanical System)
Rangos de Medición del Acelerómetro	±2g, ±4g, ±8g, ±16g
Rangos de Medición del Giroscopio	±250, ±500, ±1000, ±2000 grados por segundo
Interfaz de Comunicación	I2C
Dirección I2C	Configurable, por defecto 0x68
Rango de Voltaje de Operación	2.375V a 3.46V
Consumo de Corriente	3.9 mA en modo activo, 5 μA en modo de espera
Frecuencia de Muestra	Hasta 1 kHz
Sensores Adicionales	Sensor de temperatura interno
Dimensiones	4x4x0.9 mm (LGA-20)
Pines Principales	VCC, GND, SCL, SDA, AD0, INT

En la tabla se muestra las características Generales. Fuente: (Solectro, 2024)

1.11. MCP2515

El MCP2515, un controlador de bus CAN independiente desarrollado por Microchip Technology, como se muestra en la Figura 18. Destaca por su capacidad para facilitar una comunicación robusta y eficiente entre múltiples nodos en sistemas automotrices e industriales. Cumple con las normas CAN 2.0A y CAN 2.0B, soportando velocidades de hasta 1 Mbps y proporcionando un buffer de recepción dual con filtros y máscaras de aceptación para mejorar la gestión del tráfico de datos. Utiliza el protocolo SPI para la comunicación con el microcontrolador, integrando funciones avanzadas de diagnóstico, manejo de errores y filtrado de mensajes por identificadores específicos, características cruciales para entornos ruidosos. Sin embargo, presenta limitaciones en la gestión de altas velocidades y tráfico intenso en redes congestionadas, aspectos importantes a considerar para aplicaciones más demandantes de comunicación CAN. (Urvina Barrionuevo & Chávez Pico, 2015)

Figura 18: Componentes de la placa Arduino Uno



El gráfico muestra un módulo controlador CAN basado en el MCP2515 y el transceptor CAN TJA1050.Fuente: (Howelectronics, 2023)

Las características del MCP2515 se detallan en la Tabla 6. Este controlador de red CAN (Controller Area Network) incluye una interfaz SPI (Serial Peripheral Interface) para una comunicación eficiente con microcontroladores, un conjunto completo de registros para la configuración y el monitoreo, y un búfer de recepción con capacidad para dos mensajes completos

Tabla 6: Características de la placa Arduino Uno

Característica	Descripción
Tipo de Dispositivo	Controlador de bus CAN (Controller Area Network)
Fabricante	Microchip Technology
Protocolo de Comunicación	SPI (Serial Peripheral Interface)
Compatibilidad CAN	Cumple con CAN 2.0A (11 bits de identificación) y CAN 2.0B (29 bits de identificación)
Velocidad de Comunicación	Hasta 1 Mbps
Señalización CAN	Utiliza líneas CANH (CAN High) y CANL (CAN Low) para la comunicación diferencial
Voltajes Típicos	CANH: 2.5V a 3.5V en estado dominante, 1.5V a 2.5V en estado recesivo CANL: 2.5V en estado recesivo
Buffers de Transmisión	3 buffers de transmisión con solicitud de transmisión y banderas de control
Pines Principales	VCC, GND, SCK, SI, SO, CS, INT (Opcional), RX0BF, RX1BF (Configurable como salidas de interrupción)
Dimensiones	Variante de encapsulado: SOIC y PDIP

En la tabla se muestra las características generales de la placa. Fuente: (Farnell, 2024)

1.12. HC-06

Es un dispositivo que permite conectar de forma inalámbrica para proyectos con Arduino, microcontroladores u otros dispositivos a un smartphone, computadora o dispositivo móvil a través de Bluetooth.

Figura 19: HC-06

El gráfico muestra módulo Bluetooth que permite la comunicación inalámbrica mediante una interfaz serial UART. Fuente: (Megatronica, 2024)

Los parámetros del módulo HC-06 se pueden configurar mediante comandos AT. La placa incluye un regulador de 3.3V, lo que permite alimentar el módulo con un voltaje entre 3.6V y 6V. Este módulo es el complemento ideal para proyectos.

CAPÍTULO II

2. METODOLOGÍA

2.1 Implementación del sensor MPU6050

2.1.1. Conexión ESP32 – MPU6050

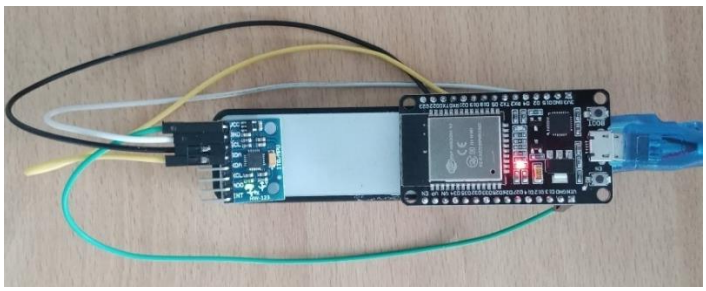
El sensor MPU6050 es implementado con la intención de obtener los valores de velocidad de giro, ángulo de dirección y el par de dirección del volante, la conexión de los pines de la MPU6050 con la ESP32 se visualiza en la Tabla 7 y la Figura 20, requiriendo un valor de entrada de 5V que será alimentado con un POWERBANK.

Tabla 7*:* ESP32 - MPU6050

Pines de la MPU6050	**Pines de la ESP32**
VCC	5V
GND	GND
SCL	22
SDA	21

Pines de la placa ESP32 - MPU6050. Fuente: (Autores)

Figura 20: ESP32 – MPU6050 - POWERBANK



Conexión ESP32 – MPU6050. Fuente: (Autores)

Sin embargo, para implementar el sensor MPU6050 en el Banco Didáctico Funcional de Dirección con Asistencia Eléctrica es importante tomar en cuenta la posición del sensor, la que será ubicada de forma horizontal, como se muestra en la Figura 21. Los ejes nos muestran los valores en tiempo real haciendo uso del giroscopio y acelerómetro implementado en el sensor.

Figura 21: MPU6050 en el volante de dirección

Ubicación de la MPU6050 en el volante de dirección. Fuente: (Autores)

2.1.1.1.Datos para la MPU6050

La ubicación del sensor MPU6050 es de manera horizontal para que el eje nos proporcione el dato de la velocidad angular de forma clara, por lo que no es necesario realizar alguna operación para obtener este dato. Sin embargo, nos ayudará a obtener las demás variables mediante ecuaciones. Para hallar dichas variables se requiere de los datos que muestra la Tabla 8 (Anónimo, 2022):

Tabla 8: Datos del volante

Dato	Valor

Masa	1.6 kg
Diámetro	0,38 m
Radio	0,19 m

Datos para la MPU6050. Fuente: (Autores)

En base a los datos obtenidos del volante que muestra la Tabla 8 planteamos las siguientes ecuaciones que son mencionadas en la Tabla 9. Estas ecuaciones y los datos obtenidos del volante son esenciales para entender y diseñar el sistema de dirección vehicular, garantizando una respuesta precisa y controlada.

La ecuación 1 calcula la velocidad lineal y se representa en metros por segundo, lo que es fundamental para evaluar y optimizar el comportamiento del sistema de dirección.

$$v = \omega * r \qquad Ec.(1)$$

v: velocidad lineal.
ω: velocidad angular
r: radio del volante

La ecuación 2 de la fuerza inercial representa la resistencia del volante a los cambios rotacionales, es crucial para la estabilidad y control del vehículo.

$$I = \frac{1}{2} * m * r^2 \qquad Ec.(2)$$

I: fuerza inercial
m: masa del volante.
r: radio del volante.

La ecuación 3 de la aceleración angular es vital para calcular el torque del volante, asegurando que el sistema de dirección responda con precisión a las órdenes del conductor.

$$\alpha = \frac{d\omega}{dt} \qquad Ec.(3)$$

α: velocidad lineal.

ω: velocidad angular

t: tiempo

La ecuación 4 del par de dirección analiza si el sistema de dirección se adapta correctamente a las maniobras del conductor, asegurando que el volante no sea demasiado sensible ni difícil de girar.

$$\tau = I * \alpha \qquad Ec.(4)$$

τ: par de dirección

I: fuerza inercial

α: velocidad lineal.

2.2. PID de la velocidad del vehículo

2.2.1. Conexión del Arduino Uno – MCP2515.

Para obtener el PID de la velocidad del vehículo se establece la comunicación con el módulo de dirección del Banco Didáctico Funcional de Dirección con Asistencia Eléctrica mediante la conexión física del Arduino Uno y la MCP2515. La Tabla 9 detalla los pines específicos para la conexión entre estos dos componentes, como se muestra en la Figura 22. Ilustra cómo deben estar conectados.

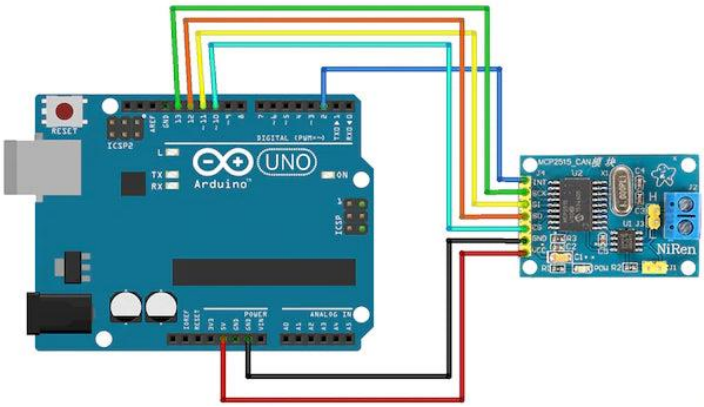
Tabla 9 Pines de la tarjeta MCP2515.

Pines de la MCP2515	Pines del Arduino Uno
INT	P2
SCK	P13

Pines de la MCP2515	Pines del Arduino Uno
SI	P11
SO	P12
CS	P10
GND	GND
VSS	5 V

Pines de la tarjeta MCP2515. Fuente: (Autores)

Figura 22: Conexión del Arduino Uno – MCP2515

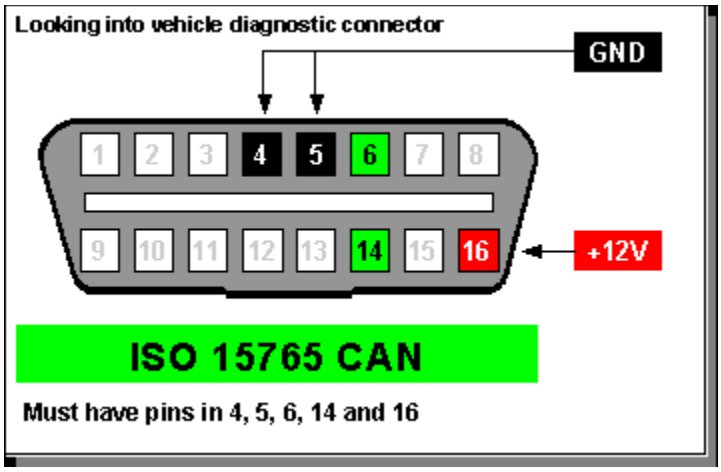


La gráfica muestra la conexión entre un Arduino Uno y un módulo CAN MCP2515. Fuente: (Autores)

2.2.2. Diseño del puerto OBD II

La comunicación con la CAN BUS del Banco Didáctico se realiza mediante el conector OBD II, siguiendo el protocolo ISO 15765, como se muestra en la Figura 23. Muestra que los pines 6 (CAN HIGH), 14 (CAN LOW), 4 y 5 (GND), y 16 (12V) son cruciales para obtener la información emitida por la CAN BUS.

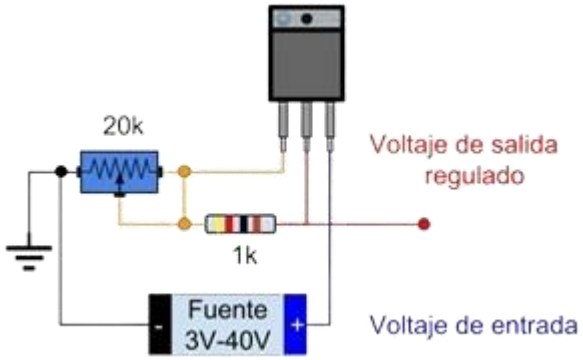
Figura 23: Pines de OBD II según el protocolo ISO15765



El gráfico muestra el conector OBD-II según el protocolo ISO 15765 CAN.
Fuente: (Research,2024)

Los pines 6 y 14 se conectan a las líneas de CAN HIGH y CAN LOW que son proporcionadas en la MCP2515, mientras que los pines 4 y 16 alimentan al Arduino Uno a través de un regulador de voltaje que proporciona una entrada estable de 5V, como se muestra en la Figura 24. Ilustra la conexión, usando los colores amarillos para CAN HIGH, azul para CAN LOW, rojo para la alimentación de 12V y negro para GND. Sin embargo, como se ha mencionado en secciones anteriores al Arduino Uno y la MCP2515 trabaja con la alimentación de 5V se utiliza un regulador de voltaje LM317T realizando el circuito, como se muestra en la Figura 24 usando un potenciómetro de precisión y una resistencia de 1k.

Figura 24: Circuito de un regulador de voltaje



El gráfico muestra un circuito de regulación de voltaje utilizando un regulador ajustable LM317. Fuente: (HetPro, 2024)

2.3. Visualización en la pantalla NEXTION

2.3.1. Conexión de pantalla Nextion – Arduino Uno

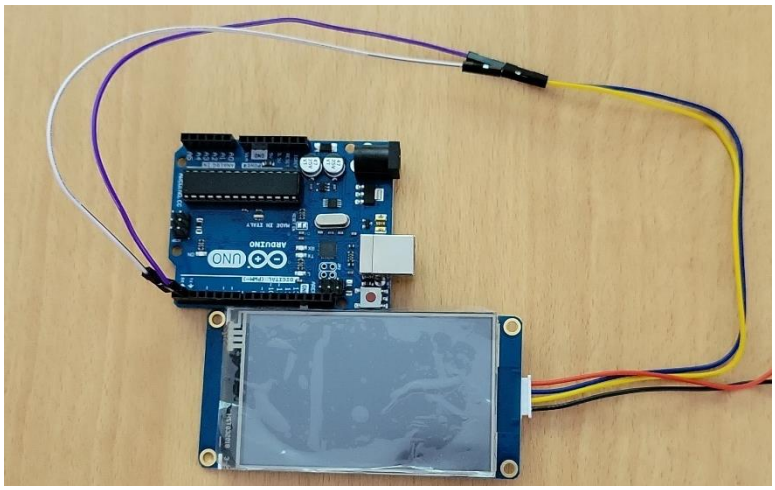
Conectar una pantalla Nextion con un Arduino Uno es un proceso que permite establecer una comunicación a través de un puerto serie (UART). Para la conexión, se deben conectar los pines TX (transmisión) y RX (recepción) de la pantalla Nextion a los pines RX y TX de la Arduino Uno, respectivamente. Además, la pantalla Nextion necesita alimentación la que será obtenida mediante un regulador de voltaje obtenido del circuito de la Red Can Bus, como se indica en la Tabla 10 y como se muestra en la Figura 25.

Tabla 10: Pantalla Nextion – Arduino Uno

Pines de la pantalla Nextion	Pines de la Arduino Uno
VCC	5V
GND	GND
Rx	Tx
Tx	Rx

Pines de la pantalla Nextion – Arduino Uno. **F**uente: (Autores)

Figura 25: Pantalla Nextion – Arduino Uno



Conexión de pantalla Nextion – Arduino Uno. **F**uente: (Autores)

2.3.2. Diseño de interfaz en Nextion

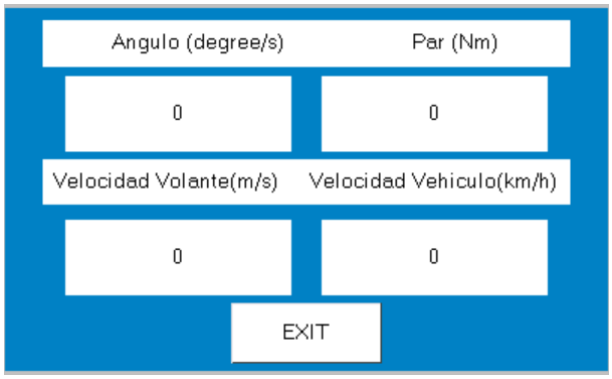
El software de Nextion permite diseñar graficas de forma intuitiva con componentes y funciones de programación para realizar interfaces personalizadas. Para empezar a editar en Nextion Editor realizamos una portada como se muestra en la Figura 26, destacando su claridad y atractivo en la representación de datos y controles.

Figura 26: Conexión de pantalla Nextion – Arduino Uno

Conexión de pantalla Nextion – Arduino Uno. Fuente: (Autores)

Se implemento un botón en el que su función es dirigir a una página en la que se muestran los valores que cada variable va a transmitir como indica la Figura 27, allí podemos visualizar que se encuentran en un valor 0.

Figura 27: Ventanas de toma de datos



Ventanas de toma de datos de cada variable. Fuente: (Autores)

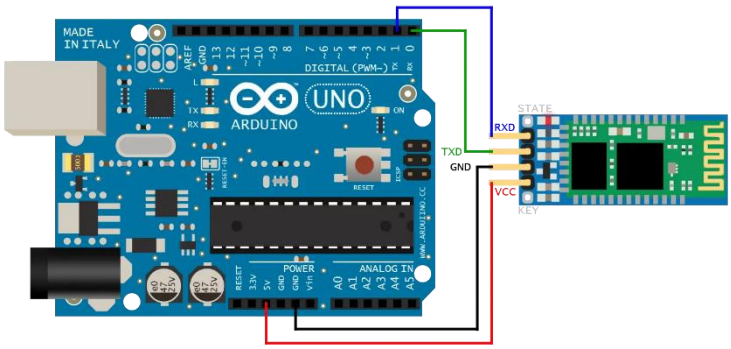
El archivo .tft se carga en la pantalla Nextion para implementar la interfaz de usuario y la lógica de interacción diseñada en el Nextion Editor. Este archivo contiene toda la información necesaria para que la pantalla muestre la interfaz y responda a las entradas del usuario según lo programado. El proceso implica compilar el proyecto en el Nextion Editor para generar el archivo .tft, que luego se transfiere a la pantalla Nextion mediante una tarjeta SD. Una vez transferido, la pantalla carga el archivo y ejecuta la interfaz, permitiendo la interacción de acuerdo con la lógica definida.

2.4. Comunicación bluetooth con Arduino Uno y ESP32

2.4.1. Conexión Arduino Uno y HC-06

Según la Figura 28 el Arduino Uno se conecta al HC-06 mediante los pines GND Y 5V para alimentación, y los pines TX y RX para la comunicación serial. Esto permite al Arduino Uno establecer una conexión Bluetooth para la comunicación inalámbrica con otros dispositivos, como la ESP32 o cualquier otro equipo compatible.

Figura 28: Conexión del Arduino Uno – HC-06

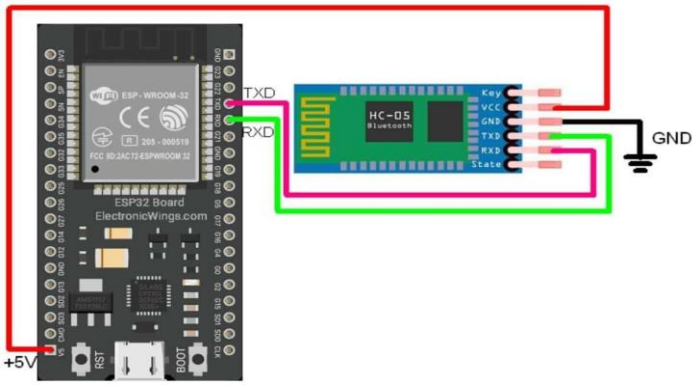


Fuente: (Autores)

El gráfico muestra la conexión de un módulo Bluetooth HC-05 a un Arduino Uno, esto permite la comunicación serial inalámbrica entre el Arduino y dispositivos externos mediante. Fuente: (Autores)

La ESP32 se conecta al HC-06 de manera similar como se muestra en la Figura 29, utilizando pines de alimentación (5V o 3.3V según las especificaciones), GND para tierra común, y pines específicos GPIO para TX y RX. Esta configuración permite que la ESP32 establezca también una conexión Bluetooth, facilitando la comunicación inalámbrica con dispositivos como el Arduino Uno u otros sistemas compatibles.

Figura 29: Conexión de ESP32 – HC06



Conexión de Bluetooth HC-06 y HC-05 Android Arduino. Fuente: (HetPro, 2024)

2.4.2. Identificación del Maestro – Esclavo

En esta configuración, la ESP32 actúa como maestro, aprovechando su capacidad para controlar y gestionar el sensor MPU6050, encargándose de capturar y procesar datos del ángulo de dirección y otras variables relevantes. Además, maneja la comunicación Bluetooth y transmite activamente los datos recolectados al Arduino Uno, supervisando todo el proceso de transferencia de información.

El Arduino Uno, por su parte, funciona como esclavo, recibiendo los datos enviados por la ESP32 a través de la conexión Bluetooth. También administra la comunicación con el módulo CAN Bus para obtener información específica del vehículo, como la velocidad del motor, respondiendo a las solicitudes y comandos de la ESP32. Esta distribución de roles garantiza una gestión eficiente y coordinada de los datos entre los diferentes componentes del sistema.

CAPÍTULO III

3. DISEÑO DE LA INTERFAZ

3.1. Código de la MPU6050

3.1.1. Librerías y funciones para la MPU6050

Para diseñar la interfaz de adquisición de datos y control, implementé un código específico para el sensor MPU6050. Este sensor es fundamental para obtener las mediciones de aceleración y velocidad angular necesarias para analizar el comportamiento del sistema.

Figura 30: Librerías para la MPU6050

```
#include "I2Cdev.h"
#include "MPU6050.h"
#include "Wire.h"

MPU6050 sensor;
```

Fuente: (Autores)

Aquí se incluyen las librerías esenciales para la comunicación I2C (`Wire.h`), la abstracción de dispositivos I2C (`I2Cdev.h`) y el control del sensor MPU6050 (`MPU6050.h`). La dirección del sensor puede variar entre 0x68 y 0x69 dependiendo del estado del pin AD0, siendo 0x68 la dirección predeterminada.

En la Figura 31 se declaran las variables globales para almacenar los valores en bruto (RAW) del acelerómetro (`ax`, `ay`, `az`) y del giroscopio (`gx`, `gy`, `gz`). También se definen constantes físicas, como la sensibilidad del giroscopio, la masa, el diámetro, el radio y la inercia del volante. Se inicializan variables adicionales para el ángulo de giro, la velocidad angular, el tiempo (`timer`), la compensación del giroscopio (`gyroX_offset`), la velocidad (`Velocidad`) y el par de giro (`Torque`).

Figura 31: Constantes y variables para la MPU6050

```
// Valores RAW (sin procesar) del acelerometro y giroscopio en los ejes x,y,z
int ax, ay, az;
int gx, gy, gz;
//---------------------------------------------------

const float sensibilidad = 20; // Sensibilidad del giroscopio en grados/segundo
const float masa = 1.6.; // Masa del volante en kg
const float diametro = 0.38; // Diámetro del volante en metros
const float radio = diametro / 2; // Radio del volante en metros
const float inercia = 0.5 * masa * radio * radio; // Inercia del volante en kg·m²

float anguloY = 0; // Ángulo de giro en el eje Y
float velocidadangularY = 0; // Velocidad angular en el eje Y
unsigned long timer;
float gyroY_offset = 0; // Offset inicial del giroscopio
float p1 = 0;
float Velocidad = 0;
float Torque = 0; // Par de giro del volante
```

Variables globales para **c**ódigo de MPU6050. Fuente: (Autores)

Las variables eventInterval1, previousTime1, eventInterval2 y previousTime2 se utilizan para controlar la ejecución de acciones a intervalos de tiempo específicos dentro del programa, detallando los valores en la Figura 32.

Figura 32: Código de MPU6050

```
unsigned long eventInterval1 = 100;
unsigned long previousTime1 = 0;

unsigned long eventInterval2 = 100;
unsigned long previousTime2 = 0;
```

Variables de tiempo para código de MPU6050. Fuente: (Autores)

La función setup() se ejecuta solo una vez al inicio del programa y es crucial para realizar las configuraciones iniciales necesarias antes de que el bucle principal (loop()) comience a ejecutarse repetidamente. En este caso, se configura la comunicación serial, se definen los pines para la comunicación I2C, se inicializa el sensor MPU6050, se calculan los ajustes (offsets) del giroscopio y se registra el

tiempo inicial en microsegundos. Estas configuraciones son esenciales para asegurar que el sensor esté listo y correctamente configurado para la recolección y procesamiento de datos durante la ejecución del programa, como se muestra en la Figura 33.

Específicamente, el código inicializa el sensor MPU6050 en una ESP32 con mpu.begin(), calcula los offsets del giroscopio con mpu.calcGyroOffsets(true), y registra el tiempo en microsegundos con timer = micros(). Estas preparaciones son vitales para obtener lecturas precisas y garantizar la sincronización temporal en aplicaciones de medición y control de movimiento.

Figura 33: Configuración en setup

```
void setup() {
  Serial.begin(115200);
  Wire.begin(21, 22); // Configuración de los pines SDA y SCL en la ESP32

  Serial.println("Inicializando MPU6050...");
  mpu.begin();
  mpu.calcGyroOffsets(true);
  timer = micros();
}
```

Fuente: (Autores)

Dentro de la función `loop`, se ejecuta el ciclo principal del programa, donde se realizan las siguientes acciones:

En la Figura 34 se indica como se obtienen los valores RAW del acelerómetro y giroscopio del sensor MPU6050.

Figura 34: Valores RAW

```
sensor.getAcceleration(&ax, &ay, &az);
sensor.getRotation(&gx, &gy, &gz);
```

Fuente: (Autores)

Los valores RAW se convierten a unidades más comprensibles, como metros por segundo al cuadrado (m/s²) para las aceleraciones y grados por segundo (°/s) para las velocidades angulares.

Para calcular el tiempo transcurrido desde la última lectura se realiza como se indica en la Figura 35.

Figura 35: Tiempo transcurrido

```
// Calcular el tiempo transcurrido desde la última lectura
unsigned long currentTime = micros();
float elapsedTime = (currentTime - timer) / 1000000.0; // Convertir a segundos
timer = currentTime;
```

Fuente: (Autores)

Las líneas de la Figura 36 y Figura 37 líneas de código calculan la velocidad angular y el torque utilizando datos del giroscopio, convirtiendo grados a radianes y aplicando restricciones de valor.

Figura 36: Cálculos de velocidad angular

```
//------------------------------------------------
angularVelocityX = (gx_deg_s / sensitivity) * (PI / 180);
Torque = inertia * angularVelocityX;
```

. Fuente: (Autores)

Figura 37: Restricciones de la velocidad angular

```
if (gx_deg_s > 700) gx_deg_s = -700;
if (gx_deg_s < -700) gx_deg_s = 700;
```

Fuente: (Autores)

En la Figura 38 se muestra un fragmento de código que calcula la velocidad como la diferencia entre dos valores de ángulo consecutivos (`angleY` y `p1`). Luego, verifica periódicamente si ha pasado un intervalo de tiempo (`eventInterval2`) desde la última impresión de datos. En caso afirmativo, imprime la velocidad actual, el ángulo actual y el torque por la consola serial. Esto es esencial para el monitoreo en tiempo real y la captura precisa de datos en aplicaciones de control de movimiento y análisis dinámico.

Figura 38: Velocidad del volante

```
Velocidad = angleY - p1;

if (millis() - previousTime2 >= eventInterval2) {
  Serial.print("Velocidad: ");
  Serial.print(Velocidad);
  Serial.print(" | Ángulo Y: ");
  Serial.print(angleY);
  Serial.print(" | Par de giro: ");
  Serial.println(Torque);
  previousTime2 = millis();
}
```

La figura 38 indica las líneas para la impresión de datos de la velocidad del volante.

Fuente: (Autores)

3.2. Código de la CAN BUS

3.2.1. Librerías y funciones para la MCP2515

Este código emplea la biblioteca CAN, como se muestra en la Figura 39, para comunicarse con un vehículo a través del bus CAN y leer la velocidad del vehículo utilizando el protocolo OBD-II. La biblioteca CAN habilita la comunicación CAN

en Arduino. La variable `useStandardAddressing` decide entre direcciones estándar (0x7E8) o extendidas (0x18DAF110) para filtrar mensajes. Con `CAN.begin(1000E3)`, se inicia el bus CAN a 1 Mbps. Los filtros aseguran que solo se procesen los mensajes pertinentes en el bus CAN. La inclusión de la biblioteca CAN habilita la comunicación y la segunda línea define una constante para determinar si se usa direccionamiento estándar o extendido en el bus CAN.

Figura 39: Librerías y función void setup para Can Bus

```
#include <CAN.h>

const bool useStandardAddressing = true;

void setup() {
  Serial.begin(9600);
  Serial.println("CAN OBD-II engine RPM");
  // start the CAN bus at 500 kbps
  if (!CAN.begin(1000E3)) {
    Serial.println("Starting CAN failed!");
    while (1);
  }

  // add filter to only receive the CAN bus ID's we care about
  if (useStandardAddressing) {
    CAN.filter(0x7e8);
  } else {
    CAN.filterExtended(0x18daf110);
  }
}
```

Librerías y configuración inicial del código. Fuente: (Autores)

La biblioteca CAN permite la comunicación CAN en Arduino. La opción `useStandardAddressing` determina si se utilizan direcciones estándar (0x7E8) o extendidas (0x18DAF110) para filtrar mensajes. La función `CAN.begin(1000E3)` inicia el bus CAN a 1 Mbps. Los filtros aseguran que solo se procesen los mensajes relevantes en el bus CAN.

En el fragmento de código mostrado en la Figura 40, se envía una solicitud CAN para obtener la velocidad actual del vehículo. El código espera la respuesta correcta basada en la longitud del mensaje, el modo y el PID, y luego asigna la velocidad leída a la variable `vel` para su posterior procesamiento o visualización.

Figura 40: Función void loop para CanBus.

```
void loop() {
  if (useStandardAddressing) {
    CAN.beginPacket(0x7df, 8);
  } else {
    CAN.beginExtendedPacket(0x18db33f1, 8);
  }
  CAN.write(0x02); // number of additional bytes
  CAN.write(0x01); // show current data
  CAN.write(0x0d); // Velocidad del vehículo
  CAN.endPacket();

  // wait for response
  while (CAN.parsePacket() == 0 ||
         CAN.read() < 3 ||          // correct length
         CAN.read() != 0x41 ||      // correct mode
         CAN.read() != 0x0d);       // correct PID

  float vel = (CAN.read());

  Serial.print("Velocidad del vehículo = ");
  Serial.println(vel);

  delay(1000);
}
```

Las líneas muestran la programación final de Can Bus. Fuente: (Autores)

3.3. Código para la pantalla Nextion

3.3.1. Librerías y funciones para la pantalla Nextion

Se incluyen las librerías necesarias para la comunicación con la pantalla Nextion (Nextion.h y SoftwareSerial.h) y para la comunicación CAN (CAN.h), como en la Figura 41

Figura 41: Librerías para Nextion

```
#include <Nextion.h>
#include <CAN.h>
#include <SoftwareSerial.h>
```

Fuente: (Autores)

En la Figura 42 define una instancia de SoftwareSerial (nexSerial) para la comunicación con la pantalla Nextion en los pines 2 (RX) y 3 (TX) y se declaran objetos NexText (n0, n1, n2, n3) para los componentes de texto en la pantalla Nextion que mostrarán los valores.

Figura 42: Comunicación con Nextion

```
SoftwareSerial nexSerial(2, 3); // RX, TX para la comunicación con Nextion

NexText n0 = NexText(1, 1, "n0");
NexText n1 = NexText(1, 2, "n1");
NexText n2 = NexText(1, 3, "n2");
NexText n3 = NexText(1, 4, "n3");
```

Fuente: (Autores)

Primero, se inicia la comunicación serial con la pantalla Nextion, que es una pantalla táctil usada para mostrar información al usuario. Para hacerlo, se llama a la función `nexInit()`, que inicializa los objetos Nextion necesarios para la comunicación. Luego, se establece la comunicación con el bus CAN (Controller Area Network) a una velocidad de 1 Mbps utilizando la función `CAN.begin(1000E3) `. Además, se

configura un filtro para recibir solo los mensajes CAN con el ID específico de 0x7e8, lo que significa que únicamente se recibirán los mensajes que contengan ese ID y se ignorarán los demás. Esto se logra mediante la función `CAN.filter(0x7e8)`. De esta manera, el sistema queda preparado para recibir y procesar los datos del CAN Bus, como se muestra en la Figura 43.

Figura 43: Función del void setup para Nextion

```
void setup() {
  Serial.begin(9600);
  nexSerial.begin(9600);
  nexInit();

  if (!CAN.begin(1000E3)) {
    Serial.println("Starting CAN failed!");
    while (1);
  }

  CAN.filter(0x7e8); // Filtro para recibir solo el ID específico de CAN
}
```

Fuente: (Autores)

En este sistema de monitoreo, se reciben datos del bus CAN, como la velocidad del vehículo, que se almacenan en variables específicas. Posteriormente, valores como el ángulo de dirección, el par de dirección y la velocidad de giro del volante se actualizan con datos obtenidos del CAN o de otros sensores. Estos valores se convierten en cadenas de caracteres utilizando sprintf() y se utilizan para actualizar los textos de los componentes de texto en la pantalla Nextion mediante setText().

Figura 44: Función void loop

```
void loop() {
  // Leer datos del CAN Bus
  if (CAN.parsePacket() > 0) {
    while (CAN.available()) {
      int len = CAN.packetLen();
      byte data[len];
      CAN.readBytes(data, len);

      if (len >= 3 && data[0] == 0x41 && data[1] == 0x0d) {
        velocidadVehiculo = data[2]; // Ejemplo de lectura de velocidad del vehículo desde CAN
        // Puedes implementar la lectura de otros datos aquí según los PIDs correspondientes
      }
    }
  }
```

Fuente: (Autores)

En la sección final, como se observa en la Figura 45, se presenta el código utilizado para actualizar los valores en una pantalla Nextion utilizando Arduino. Inicialmente, se declara un buffer de caracteres llamado `buffer`, con capacidad para almacenar hasta 20 caracteres, destinado a contener las cadenas formateadas. Posteriormente, se emplean múltiples llamadas a la función `sprintf` para convertir valores numéricos como `anguloDireccion`, `parDireccion`, `velocidadGiroVolante` y `velocidadVehiculo` en cadenas con una precisión de dos decimales (`"%.2f"`). Estas cadenas se asignan a los componentes de texto (`n0`, `n1`, `n2`, `n3`) en la pantalla Nextion mediante la función `setText()`.

Para asegurar que los datos mostrados en la interfaz gráfica reflejen los valores más actualizados obtenidos del bus CAN, se incorpora un retardo de 1000 milisegundos (1 segundo) antes de repetir el proceso de actualización. Este enfoque garantiza que los datos visualizados se mantengan sincronizados con las lecturas en tiempo real de los sensores, mejorando así la precisión y la utilidad de la interfaz para la monitorización de variables críticas del sistema.

Figura 45: Impresión de valores en Nextion

```
  // Actualizar los valores en la pantalla Nextion
  char buffer[20];

  // Convertir valores a cadenas para enviar a la pantalla Nextion
  sprintf(buffer, "%.2f", anguloDireccion);
  n0.setText(buffer);

  sprintf(buffer, "%.2f", parDireccion);
  n1.setText(buffer);

  sprintf(buffer, "%.2f", velocidadGiroVolante);
  n2.setText(buffer);

  sprintf(buffer, "%.2f", velocidadVehiculo);
  n3.setText(buffer);

  delay(1000); // Actualizar cada segundo o según sea necesario
}
```

Fuente: (Autores)

3.4. Código de la comunicación Bluetooth

3.4.1. ESP32 (Maestro)

Para que la ESP32 sea maestro se implementan las librerías que se muestran en la Figura 46 donde se indica que <BluetoothSerial.h> es la que Incluye la biblioteca BluetoothSerial, que permite la comunicación Bluetooth en el ESP32 mientras que BluetoothSerial SerialBT crea rea una instancia del objeto BluetoothSerial para la comunicación Bluetooth.

Figura 46: Librerías ESP32 (maestro)

```
#include <BluetoothSerial.h>

BluetoothSerial SerialBT;
```

Librerías para la comunicación Bluetooth de la ESP32 (maestro). Fuente: (Autores)

La función `setup` se ejecuta una vez al inicio, configurando la comunicación serial a 115200 baudios y el Bluetooth como "ESP32_Master". Además, inicializa la comunicación I2C en los pines 21 y 22, envía un mensaje al monitor serial, inicializa el sensor MPU6050, establece el offset del giroscopio en el eje X y almacena el tiempo actual en microsegundos.

Figura 47: Función setup con bluetooth

```
void setup() {
  Serial.begin(115200);
  SerialBT.begin("ESP32_Master");
  Wire.begin(21, 22);

  Serial.println("Inicializando MPU6050...");
  sensor.initialize();
  sensor.setXGyroOffset(gyroX_offset);
  timer = micros();
}
```

Fuente: (Autores)

Además, se crea la línea explicada en la Figura 48 la cual crea una cadena de datos con la velocidad, el ángulo y el torque.

Figura 48: Cadena de datos

```
String data = String(Velocidad) + "," + String(angleX) + "," + String(Torque);
SerialBT.println(data);
```

Fuente: (Autores)

3.4.2. Arduino Uno (Esclavo)

En el código mostrado en la Figura 49, se observa que la declaración #include <SoftwareSerial.h> incorpora la librería SoftwareSerial, esencial para habilitar la comunicación serial en cualquier pin digital del Arduino. Además, la línea SoftwareSerial BTSerial(10, 11) crea un objeto BTSerial utilizando los pines 10 y 11 para la recepción (RX) y transmisión (TX) respectivamente.

Figura 49: Librería para bluetooth de Arduino Uno (esclavo)

```
#include <SoftwareSerial.h>

SoftwareSerial BTSerial(10, 11); // RX, TX
```

Fuente: (Autores)

Dentro del loop principal de mi programa, se ha implementado el siguiente bloque de código como se observa en la Figura 50. Primero, utilizo la condición `if (BTSerial.available())` para verificar si hay datos disponibles en el puerto serial Bluetooth. Cuando hay datos disponibles, utilizo `String data = BTSerial.readStringUntil('\n');` para leer estos datos hasta encontrar un carácter de nueva línea (`'\n'`) y los guardo en la variable `data`. Finalmente, se utiliza `Serial.println("Received data: " + data);` para imprimir "Received data: " seguido de los datos recibidos en el monitor serial, permitiéndome visualizar los datos obtenidos a través del Bluetooth en el monitor serial del Arduino.

Figura 50: Trasmisión de datos por bluetooth

```
if (BTSerial.available()) {
  String data = BTSerial.readStringUntil('\n');
  Serial.println("Received data: " + data);
}
```

Fuente: (Autores)

CAPÍTULO IV

4. RESULTADOS

4.1. Resultados en el Banco Didáctico Funcional de Dirección con Asistencia Eléctrica.

4.1.1. Resultados CAN BUS.

Utilizando Arduino junto con el módulo MCP2515, se estableció una conexión sólida para capturar datos en tiempo real del sistema de control del vehículo. Este paso no solo demuestra nuestra capacidad de integrar tecnología, sino que también nos permite evaluar la precisión y consistencia de los datos obtenidos.

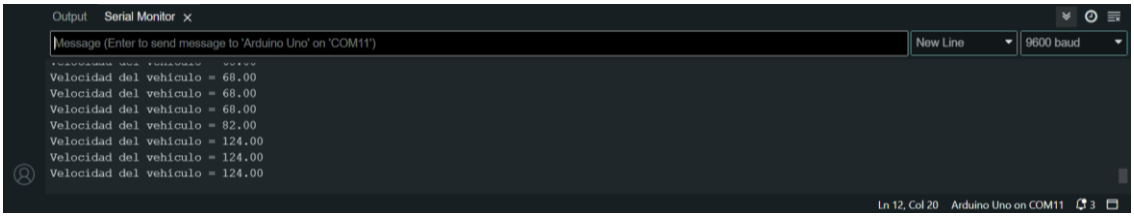
Para obtener solamente el PID se realizó la conexión mostrada en la Figura 51 la cual detalla la función de la MCP2515

Figura 51: Obtención de la velocidad del vehículo en el banco didáctico

Fuente: (Autores)

Para obtener los datos, se utilizó el ID 7e8 ya que es el que establece la comunicación CAN, se accedió al PID 0x0D, que corresponde a la velocidad del vehículo, ubicado en el byte 3 del mensaje CAN con el ID 0x7DF. La figura 40 muestra la conversión directa del valor hexadecimal a decimal de la velocidad del vehículo, eliminando la necesidad de usar fórmulas adicionales.

Figura 52: Velocidad del vehículo



Datos en Arduino IDE de la velocidad del vehículo. Fuente: (Autores)

Se compararon los datos del Arduino IDE con los del scanner para verificar su precisión. En la Figura 52 se muestran los valores capturados por el Arduino IDE durante la prueba del PID de velocidad del vehículo. Esta comparación se realizó utilizando el scanner como referencia estándar para validar los resultados. Los valores clave relacionados con la velocidad del vehículo fueron evaluados para determinar la exactitud del sistema. Los valores relevantes relacionados con la velocidad del vehículo para determinar la precisión del sistema implementado en la Figura 53 el scaner nos mostró el mismo valor que el obtenido por el Arduino Ide.

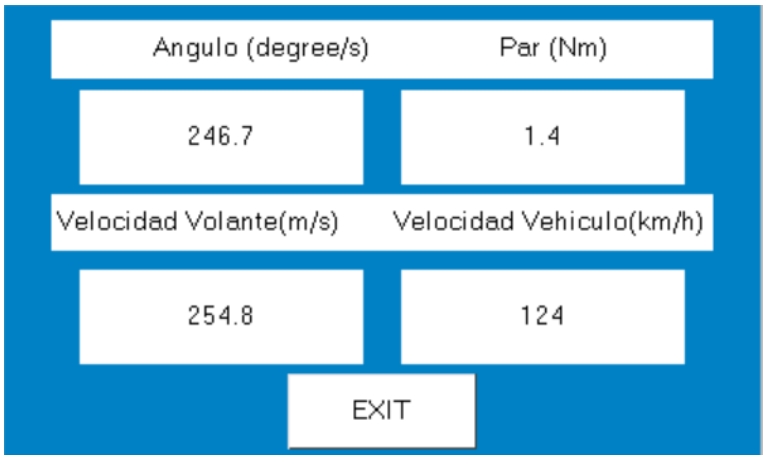
Figura 53: Valor de la velocidad del vehículo

Datos obtenidos del vehículo con el scanner del Valor de la velocidad. Fuente: (Autores)

4.2. Visualización en la pantalla Nextion.

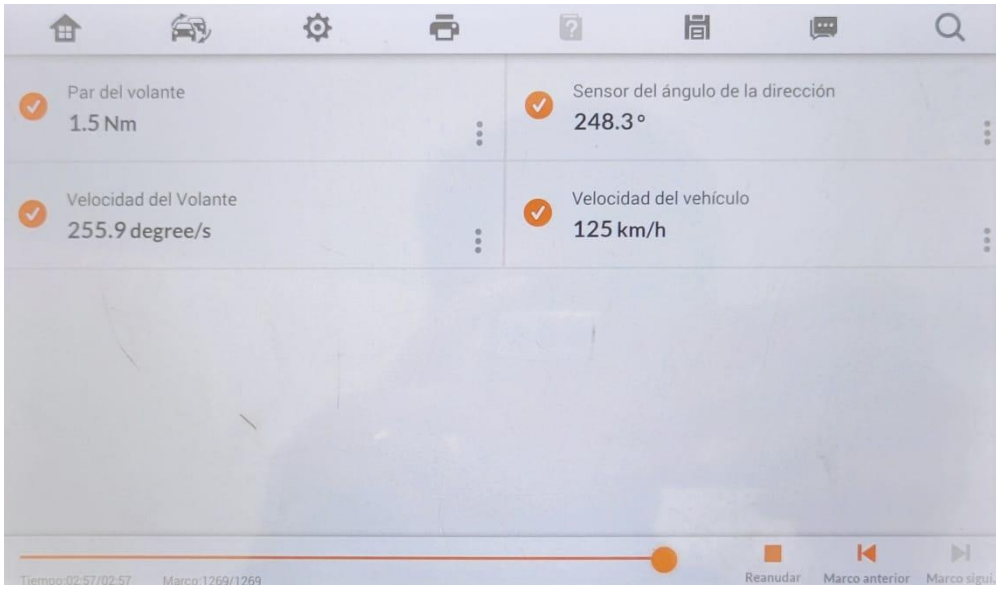
Para comprobar los valores que se obtiene del banco didáctico funcional de dirección con asistencia eléctrica, se observó la respuesta del sensor ante diferentes posiciones del volante, asegurando que los valores mostrados correspondieran a la ubicación real del volante. En la Figura 54 se presentan los datos visualizados en el Arduino IDE, mientras que la Figura 55 muestra los valores correspondientes obtenidos del scanner.

Figura 54: Valores visualizados en la Pantalla Nextion



Ángulo de dirección, Par de dirección, Velocidad de giro del volante y Velocidad del vehículo. Fuente: (Autores)

Figura 55: Valores del volante obtenidos en el scaner.



Fuente: (Autores)

Al comparar los datos se muestra una variación de datos entre el escáner automotriz y la pantalla Nextion puede ser causada por la falta de sincronización en los tiempos de muestreo, interferencias electromagnéticas, ruido en la comunicación, desviaciones del sensor no compensadas, y diferencias en el procesamiento de datos. O simplemente puede ser causada por una mala calibración en el sensor MPU6050 puede descalibrarse debido a impactos físicos, cambios ambientales o montaje incorrecto.

4.3. Resultados en un vehículo con Dirección con Asistencia Eléctrica.

4.3.1. PID de la velocidad del vehículo Honda Civic.

Para la veracidad de la interfaz ser realizo la implementación en un vehículo de marca Honda y modelo Civic del año 2001 ya que para realizar la prueba verificamos el protocolo que maneja el vehículo, el protocolo ISO lo cual al realizar la toma de datos el código para CanBus que se utilizó en el banco didáctico si se logró establecer la comunicación con el mismo ID (7e8) lo que nos proporcionó el valor directo de la velocidad del vehículo y el PID (0x0D) de la velocidad del vehículo es general para la mayoría de las marcas de vehículos.

Primero conectamos la interfaz de CANBUS para observar lo que nos registra el Arduino Ide, en la figura 55 nos muestra el valor obtenido de la CanBus y en la figura 56 se visualiza el velocímetro.

Figura 56: Vehículo Honda Civic

Dato obtenido de la CANBUS del vehículo Honda Civic. Fuente: (Autores)

Figura 57: Velocímetro del vehículo Honda Civic en CANBUS.

Fuente: (Autores)

4.3.2. MPU6050 en el vehículo Honda Civic

El giroscopio es ubicado de la misma manera horizontalmente como la figura 58 sin embargo en el código se tuvo que modificar la velocidad a 115200.

Sin embargo, hay que modificar los datos del volante quedando las variables como indica la tabla 11, ya que son diferentes valores que nos muestra el volante de Honda y quedando en el código como indica la figura 58.

Tabla 11: Vehículo Honda Civic

Dato	Valor
Masa	2.2 kg
Diametro	0,38 m
Radio	0,19 m

Datos del volante del Vehículo Honda Civic. Fuente: (Autores)

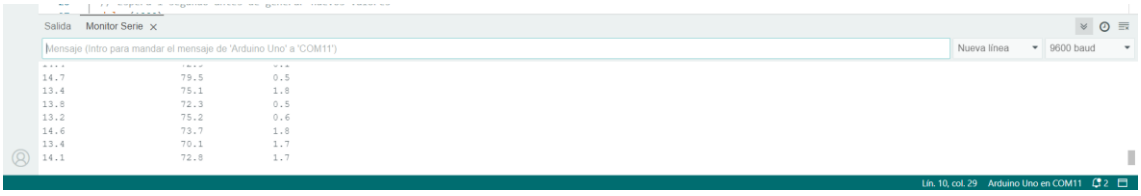
Figura 58: Constantes del vehículo Honda Civic en la MPU6050

```
// Valores RAW (sin procesar) del acelerometro y giroscopio en los ejes x,y,z
int ax, ay, az;
int gx, gy, gz;
//---------------------------------------------------------

const float sensibilidad = 20; // Sensibilidad del giroscopio en grados/segundo
const float masa = 2.2.; // Masa del volante en kg
const float diametro = 0.38; // Diámetro del volante en metros
const float radio = diametro / 2; // Radio del volante en metros
const float inercia = 0.5 * masa * radio * radio; // Inercia del volante en kg·m²

float anguloY = 0; // Ángulo de giro en el eje Y
float velocidadangularY = 0; // Velocidad angular en el eje Y
unsigned long timer;
float gyroY_offset = 0; // Offset inicial del giroscopio
float p1 = 0;
float Velocidad = 0;
float Torque = 0; // Par de giro del volante
```

Fuente: (Autores)

Después de establecer la MPU6050 en el volante, modificar las constantes y compilar los datos de la MPU6050 se indica en la figura 57, se llegan a visualizar los datos que muestra la comparamos que hay una variación poco notoria y esto puede aparecer por daños en el sensor de velocidad, algún problema con el tacómetro o simplemente la comunicación con la computadora.

Figura 59: Vehículo Honda Civic con MPU6050



Visualización del serial Monitor en el vehículo Honda Civic, Fuente: (Autores)

4.3.3. Visualización de la pantalla Nextion en el vehículo Honda Civic

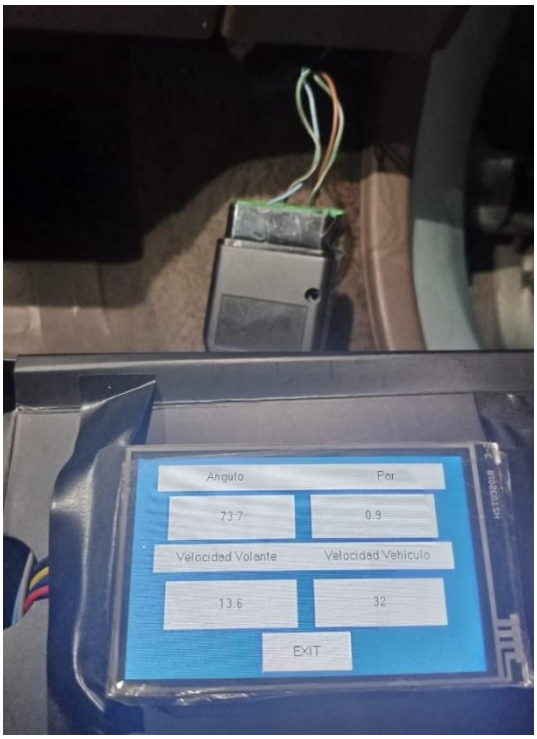
Al haber comprobado los datos anteriores en el vehículo y después de realizar la modificación en el código de la MPU6050 se conecta, los modem bluetooth, Arduino, a pantalla vemos las conexiones de la interfaz que indica en la figura 59.

Los datos que se muestran en la pantalla Nextion son los que se indica en la figura 59, de esta manera concluimos que es posible obtener datos de otro vehículo siempre y cuando cumpla el requisito de tener OBDII, y realizando la modificación en el código de la MPU6050 el cual aparece en Anexo1, así como el código del CanBus y la pantalla Nextion, como Anexo 2 y 3 respectivamente.

Figura 60: Conexión previa para conectar la interfaz al vehículo Honda Civic

Fuente: (Autores)

Figura 61: Interfaz conectada al vehículo Honda Civic



Fuente: (Autores)

Figura 62: Velocímetro del vehículo Honda Civic

Fuente: (Autores)

4.4. Comparación entre las pruebas realizadas en el Banco Didáctico Funcional de Dirección con Asistencia Eléctrica y el vehículo Honda Civic.

Los resultados obtenidos tanto del banco didáctico funcional de dirección con asistencia eléctrica como del vehículo Honda Civic del año 2001 muestran variaciones entre los datos registrados por el escáner y los mostrados en la pantalla Nextion. Estas discrepancias pueden ser atribuidas a varios factores, incluyendo falta de sincronización, interferencia electromagnética, ruido ambiental, desviaciones en la calibración de los sensores, diferencias en el procesamiento de datos, así como condiciones ambientales variables o una instalación incorrecta de los componentes.

Aunque los resultados no fueron idénticos, los patrones generales observados fueron similares, indicando que el sistema opera dentro de parámetros aceptables y proporciona datos mayormente precisos. Sin embargo, se recomienda realizar un análisis más detallado para identificar las causas específicas de las diferencias encontradas y así mejorar la consistencia y exactitud de los datos recolectados.

Aunque tanto el banco didáctico funcional como el vehículo Honda Civic demostraron que el sistema de dirección asistida eléctrica funciona conforme a lo esperado, es crucial reconocer las limitaciones en la precisión de los datos recopilados. Estas limitaciones deben ser consideradas para una interpretación adecuada de los resultados.

CONCLUSIONES

- Las variaciones entre los datos del banco didáctico funcional y del vehículo Honda Civic del año 2001 resaltan la influencia significativa de factores externos como la interferencia electromagnética y el ruido ambiental en la precisión de los sistemas de dirección asistida eléctrica.
- Se recomienda identificar y mitigar estos factores mejoraría la consistencia y exactitud de los datos recolectados.
- Las discrepancias específicas entre el escáner y la pantalla Nextion muestran que es necesaria la calibración precisa de los sensores y un procesamiento de datos uniforme.
- La implementación de la interfaz para el banco didáctico de dirección con asistencia eléctrica, utilizando componentes como Arduino, MPU6050, MCP2515 y la pantalla Nextion, ha demostrado ser fundamental para mejorar la comprensión y el aprendizaje en ingeniería automotriz, especialmente en lo que respecta a la interpretación de datos del protocolo CAN BUS.
- La integración de funcionalidades de análisis y diagnóstico en la interfaz ha permitido una evaluación precisa del rendimiento dinámico del vehículo, destacando la importancia de la captura y visualización de datos en tiempo real para la detección eficiente de fallos y la optimización del sistema de dirección asistida eléctricamente

RECOMENDACIONES

- Se recomienda realizar un análisis más profundo para determinar las causas específicas de las diferencias encontradas entre los datos recopilados y las mediciones del scanner, y así mejorar la consistencia y exactitud de los datos obtenidos.
- La inclusión de herramientas analíticas en la interfaz facilitaría la realización de diagnósticos precisos de problemas en el sistema, lo cual es crucial para el mantenimiento y optimización de los sistemas automotrices.
- Es recomendable ampliar la funcionalidad de la interfaz para incluir. el monitoreo de otros parámetros relevantes del sistema, como la temperatura, presión y el estado del sistema de alimentación ya que la maqueta es alimentada de esta manera.
- La inclusión de elementos de interactividad, como gráficos dinámicos y animaciones, mejoraría la experiencia del usuario y facilitaría la comprensión de la información.
- Para garantizar el correcto funcionamiento de la interfaz a largo plazo, se requiere establecer un plan de mantenimiento que incluya actualizaciones periódicas del código, la revisión de los componentes y la corrección de posibles errores.

REFERENCIAS BIBLIOGRÁFICAS.

Alcalde Susi, A. (2019). *Diseño e implementación de una plataforma para teleoperación y control de sistemas dinámicos*. Universidad Politécnica de Madrid.

Alfa e Parts. (n.d.). *SENSOR ÁNGULO DIRECCIÓN - Alfa e-Parts*. Retrieved June 25, 2024, from https://alfaeparts.com/productos/sensor-angulo-direccion

Ambrosio, R., Flores Méndez, J., Guzmán, P., & Sánchez Gaspariano, L. A. (2018). *Red de área de controlador (CAN-bus) de bajo costo para aplicaciones en un vehículo eléctrico*.

Anónimo. (2022, October 30). *Velocidad angular*. https://espanol.libretexts.org/Educacion_Basica/Trigonometria/02%3A_Ratios_trigonométricos/2.05%3A_Radianes/2.5.05%3A_Velocidad_angular

Arrata Zambrano, R. B., & Yoza Rodríguez, F. I. (2020). *Implementación y conversión del sistema de dirección vehicular manual a eléctricamente asistida*.

Badawy, A., Zuraski, J., Bolourchi, F., & Chandy, A. (1999, March 1). *Modeling and Analysis of an Electric Power Steering System*. https://doi.org/10.4271/1999-01-0399

Barona Zaldumbide, J. L., & Terán Burgos, J. N. (2023). *Diseño de un dashboard para el banco de sensores, actuadores y unidades de control, modelo pmk – tl – loo4 "hyundai grandeur" para monitoreo y control basado en red can bus*. Universidad Politécnica Salesiana.

Beningo Jacob. (2020, January 21). *Cómo seleccionar y usar el módulo ESP32 con Wi-Fi/Bluetooth adecuado para una aplicación de IoT industrial*. https://www.digikey.com/es/articles/how-to-select-and-use-the-right-esp32-wi-fi-bluetooth-module

Blanco González, V. J. (2021). *Puesta a punto de un sistema inalámbrico de bajo coste para análisis vibratorios: MPU6050 y módulo ESP8266.* Universidad de Sevilla.

Bosch, R. (1991, September). *CAN Specification.* https://www.bosch-semiconductors.com/ip-modules/can-protocols/

Chafla Taquinga, L. Á., & Salinas Mejía, M. S. (2012). *Construcción e implementación de un tablero didáctico del sistema de dirección asistida eléctricamente (EPS) para la Escuela de Ingeniería Automotriz.* [Escuela Superior Politécnica de Chimborazo]. http://dspace.espoch.edu.ec/handle/123456789/2284

Concepcion, M. (2011). *Estrategias de Sistemas Automotrices OBD-II*. Concepcion Mandy.

Feliciano Fuentes, L. G. (2019). *Identificación y control de parámetros de clúster de instrumentos automotriz mediante Red Can*. BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA.

Freiberger, S., Albrecht, M., & Käufl, J. (2011). Reverse Engineering Technologies for Remanufacturing of Automotive Systems Communicating via CAN Bus. *Journal of Remanufacturing*, *1*(1), 6. https://doi.org/10.1186/2210-4690-1-6

Herrera Escudero, A. (n.d.). *Sistemas de coordenadas en 3D*. Retrieved June 21, 2024, from https://www.udocz.com/apuntes/781852/sistemas-de-coordenadas-en-3d-geometria-analitica

Jie, C., & Yanling, G. (2021). Research on Control Strategy of the Electric Power Steering System for All-Terrain Vehicles Based on Model Predictive Current Control. *Mathematical Problems in Engineering*, *2021*, 1–15. https://doi.org/10.1155/2021/6642042

Martínez Requena, A. (2017). *Introducción a CAN bus: Descripción, ejemplos y aplicaciones de tiempo real*.

Nazaruddin, N., Zainuri, F., Siregar, R., Heryana, G., Adhitya, M., & Sumarsono, D. (2019). Electric power steering: an overview of dynamics equation and how it's developed for large vehicle. *IOP Conference Series: Materials Science and Engineering*, *673*, 012112. https://doi.org/10.1088/1757-899X/673/1/012112

Northeast Auto Service. (2023, August). *Automotive Steering Systems: An Overview*. https://www.autorepairindy.com/blog/driving-the-point-home-understanding-your-vehicles-steering-system/

Patterson, D. A., & Hennessy, J. L. (2016). *Computer Organization and Design: The Hardware/Software Interface*. Morgan kaufmann.

Pinguil Peñaranda, J. L. (2023). *IMPLEMENTACIÓN DE UN SISTEMA DE MONITOREO DE RED CAN BASADO EN IDENTIFICADORES PARA UN VEHÍCULO ELÉCTRICO RENAULT KANGOO ZE*. Universidad del Azuay.

Polizzi, J.-P., Fain, B., & Maspero, F. (2020). *Accelerometer*. Elsevier.

Quispe Mamani, E. (2015). *Diseño y construcción de una maqueta funcional de sistema de dirección electroasistida electricamente (EPS).*

Rivera Fuentes, P. (2016, December 13). *PIDs OBD-II*. https://es.scribd.com/user/75340377/Pablo-Rivera-Fuentes

Shimizu, Y., & Kawai, T. (1991). Development of Electric Power Steering. In *JOURNAL OF PASSENGER CARS* (Vol. 100).

Sifuentes De La Hoya, E., Javier, F., Aguilera, E., De Dios, J., Ruiz, C., Carreón, A. E. Q., Martín, J., & Aceves, S. (2023). DESARROLLO DE UN GIROSCOPIO DIGITAL INALÁMBRICO CON UN SENSOR TMR Y UN CIRCUITO DE INTERFAZ DIRECTA. In *Tecnológico Nacional de México en Celaya Pistas Educativas* (Vol. 44, Issue 144). http://itcelaya.edu.mx/ojs/index.php/pistas

Tanenbaum, A. (2009). *Modern operating systems*. Pearson Education, Inc.

Urvina Barrionuevo, K. R., & Chávez Pico, D. A. (2015). *Sistema electrónico de control y monitoreo vehicular a distancia mediante tecnología inalámbrica*. Universidad Técnica de Ambato.

Venegas Álvarez, D. A. (2023). *Diseño e implementación de un panel de control y monitoreo digital para activación de funciones y lectura de datos en tiempo real del vehículo Chevrolet Gemini*. Universidad de las Fuerzas Armadas.

Zhan, Z., & Yan, Y. (2018). Design of a steering stabilizer based on CAN bus. *AIP Conference Proceedings, 1955*. https://doi.org/10.1063/1.5033683

Zhang, S., & Wang, Y. (2015). *Study of Electric Power Steering System*.

ANEXOS.

Anexo1.

Código de ESP32 maestro

```
#include "Wire.h"
#include "I2Cdev.h"
#include "MPU6050.h"
#include <BluetoothSerial.h>

MPU6050 sensor;
BluetoothSerial SerialBT;

int ax, ay, az;
int gx, gy, gz;
const float sensitivity = 20;
const float mass = 1.6;
const float diameter = 0.38;
const float radius = diameter / 2;
const float inertia = 0.5 * mass * radius * radius;

float angleX = 0;
float angularVelocityX = 0;
unsigned long timer;
float gyroX_offset = 0;
float p1 = 0;
float Velocidad = 0;
float Torque = 0;

void setup() {
  Serial.begin(115200);
  SerialBT.begin("ESP32_Master");
  Wire.begin(21, 22);

  Serial.println("Inicializando MPU6050...");
  sensor.initialize();
  sensor.setXGyroOffset(gyroX_offset);
  timer = micros();
}

void loop() {
  sensor.getAcceleration(&ax, &ay, &az);
  sensor.getRotation(&gx, &gy, &gz);

  unsigned long currentTime = micros();
  float elapsedTime = (currentTime - timer) / 1000000.0;
  timer = currentTime;

  angularVelocityX = (gx / sensitivity) * (PI / 180);
  Torque = inertia * angularVelocityX;
  Velocidad = gx;

  String data = String(Velocidad) + "," + String(angleX) + "," + String(Torque);
  SerialBT.println(data);
```

```
  Serial.print("Velocidad: ");
  Serial.print(Velocidad);
  Serial.print(" | Ángulo X: ");
  Serial.print(angleX);
  Serial.print(" | Par de giro: ");
  Serial.println(Torque);

  delay(100);
}
```

Anexo2.

Código de Arduino Esclavo

```
#include <CAN.h>
#include <SoftwareSerial.h>
SoftwareSerial BTSerial(10, 11);
void setup() {
  Serial.begin(9600);
  BTSerial.begin(9600);
  Serial.println("CAN OBD-II engine RPM");
  if (!CAN.begin(1000E3)) {
    Serial.println("Starting CAN failed!");
    while (1);
  }
  CAN.filter(0x7e8);
}
void loop() {
  if (BTSerial.available()) {
    String data = BTSerial.readStringUntil('\n');
    Serial.println("Received data: " + data);
  }
  CAN.beginPacket(0x7df, 8);
  CAN.write(0x02);
  CAN.write(0x01);
  CAN.write(0x0d);
  CAN.endPacket();
  while (CAN.parsePacket() == 0 ||
         CAN.read() < 3 ||
         CAN.read() != 0x41 ||
         CAN.read() != 0x0d);
  float vel = CAN.read();
  Serial.print("Velocidad del vehículo = ");
  Serial.println(vel);
  delay(1000);
}
```

Anexo3.

Código de Nextion

```
#include <SoftwareSerial.h>
// Configuración de pines
SoftwareSerial nextion(10, 11); // RX, TX para Nextion

float angulo = 0;
float par = 0;
float velVol = 0;
float velVeh = 0;
void setup() {
  // Inicializar la comunicación serie
  Serial.begin(9600);
  nextion.begin(9600);
  esp32Serial.begin(9600);
  // Mensaje inicial
  Serial.println("Sistema iniciado");
}
void loop() {
  // Verificar si hay datos disponibles desde la ESP32
  if (esp32Serial.available()) {
    // Leer los datos de la ESP32
    String data = esp32Serial.readStringUntil('\n');
    parseData(data);
    // Enviar datos a la pantalla Nextion
    sendToNextion("angulo", angulo);
    sendToNextion("par", par);
    sendToNextion("velvol", velVol);
    sendToNextion("velveh", velVeh);
  }
}
void parseData(String data) {
  // Asumiendo que los datos están en el formato: "angulo:valor;par:valor;velvol:valor;velveh:valor"
  int anguloIndex = data.indexOf("angulo:") + 7;
  int parIndex = data.indexOf("par:") + 4;
  int velVolIndex = data.indexOf("velvol:") + 7;
  int velVehIndex = data.indexOf("velveh:") + 7;
  angulo = data.substring(anguloIndex, data.indexOf(";", anguloIndex)).toFloat();
  par = data.substring(parIndex, data.indexOf(";", parIndex)).toFloat();
  velVol = data.substring(velVolIndex, data.indexOf(";", velVolIndex)).toFloat();
  velVeh = data.substring(velVehIndex).toFloat();
}
void sendToNextion(String component, float value) {
  // Construir el comando Nextion
  String command = "page1." + component + ".val=" + String(value);
  nextion.print(command);
  nextion.write(0xFF);
  nextion.write(0xFF);
  nextion.write(0xFF);
}
```