BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA

FACULTAD DE CIENCIAS DE LA ELECTRÓNICA

MAESTRÍA EN CIENCIAS DE LA ELECTRÓNICA
OPCION EN AUTOMATIZACIÓN

Interfaz Gráfica de Usuario de un cluster de instrumentos automotriz y su interacción con hardware de bajo costo

T E S I S

PRESENTADA PARA OBTENER EL TITULO DE:

MAESTRO EN CIENCIAS DE LA ELECTRONICA
OPCION EN AUTOMATIZACION

PRESENTA

ING. SERGIO JOSUÉ ORTIZ HERNÁNDEZ

DIRECTORES DE TESIS

DR. LUIS ABRAHAM SÁNCHEZ GASPARIANO

DR. JOSÉ ELIGIO MOISÉS GUTIÉRREZ ARIAS

Puebla, Puebla, Agosto 2025

*Becario SECIHTI

BUAP

Benemérita Universidad Autónoma de Puebla

Facultad de Ciencias de la Electrónica

Maestría en Ciencias de la Electrónica
Opción en Automatización

Interfaz Gráfica de Usuario de un cluster de instrumentos automotriz y su interacción con hardware de bajo costo

Alumno: **Ing. Sergio Josué Ortiz Hernández**

Directores de Tesis: **Dr. Luis Abraham Sánchez Gaspariano (FCE-BUAP)**
Dr. José Eligio Moisés Gutiérrez Arias (FCE-BUAP)

Puebla, Puebla. 12/24.

Agradecimientos

Quiero agradecer a mis padres Sergio Sergio Ramón Ortiz León y Leticia Blandina Hernández Cobos por apoyarme de manera incondicional en todas mis decisiones, sin mis padres no podría haber conseguido toda esta trayectoria, y todo este proyecto de vida los amo mucho papás.

También quiero agradecer a Ana Laura García Ramos por todo el amor y apoyo que he recibido como compañera y pareja en mi vida, gracias también por esa maravillosa confianza y motivación que me das en todas nuestras bonitas aventuras gracias por este hermoso proyecto de vida juntos te amo mucho.

Agradezco a mis asesores de tesis el Dr. Luis Abraham Sánchez Gaspariano y al Dr. José Eligió Moises Gutierrez Arias por toda esta trayectoria, preparación, paciencia y amistad. Quiero agradecer a todos mis profesores durante mi preparación en maestría, a la Dra. Amparo, al Dr. Sergio, a la Dra. Aurora, Al Dr. Jorge y al Dra. Josefina, muchas gracias por compartir su tiempo, y maravillosas enseñanzas que estoy seguro me servirán para toda la vida.

Agradezco a mis hermanos Emmanuel y Gabriel; al igual, a mi primo Julio, por todo lo bueno que han hecho mi. Quiero agradecer a mis compañeros de maestría que en todo este tiempo nos hemos vuelto grandes amigos y estoy seguro de que grandes colegas. Compartí momentos increíbles muchas gracias a Raul, David, Uriel, Victor, Diego, Luis, Gengis, Abril, Gabriel e Iki. Quiero darle un merito especial a los alumnos de la carrera de sistemas automotrices, ya que gracias a ellos pudimos ir desarrollando este proyecto para poder tener un trabajo de calidad.

Quiero agradecer a SECIHTI por la beca recibida para poder realizar mis estudios de Maestría, ya que sin su apoyo este estudio no podría haberse llevado a cabo.

Finalmente, agradezco a dios por la salud y virtud que me da para hacer lo que más me gusta en este mundo.

Resumen

La presente tesis aborda el diseño e implementación de una interfaz gráfica de usuario (GUI) para la simulación de un clúster de instrumentos automotriz, integrándola con hardware de bajo costo para establecer una red de comunicación vehicular. El objetivo principal del proyecto fue desarrollar una solución accesible y didáctica, orientada a la educación y la investigación en el campo de los sistemas automotrices y las redes vehiculares.

Se logró construir un prototipo funcional que consta de cuatro nodos interconectados a través de un bus CAN. Estos nodos, basados en microcontroladores ESP32 y transceptores CAN, fueron configurados para simular la captura y transmisión de datos críticos como el nivel de combustible, la temperatura del motor y parámetros del tren motriz. Uno de los nodos actuó como un gateway, encargado de recibir los datos del bus CAN y enviarlos a la GUI desarrollada en Python (Tkinter) a través de comunicación serial.

Las pruebas realizadas con estudiantes de ingeniería en sistemas automotrices demostraron la eficacia del sistema, confirmando la correcta funcionalidad de la red CAN y la visualización precisa de los datos en tiempo real en la GUI. Un hallazgo significativo fue la viabilidad económica de esta propuesta, la cual representa una alternativa de muy bajo costo en comparación con los equipos de laboratorio comerciales, facilitando así el acceso a la formación práctica en tecnologías automotrices. En síntesis, esta tesis valida la aplicación de software de código abierto y hardware accesible para crear un entorno de aprendizaje práctico y eficiente en el ámbito de las interfaces y redes vehiculares.

Definiciones

- Interfaz Gráfica de Usuario (GUI): Programa que usa objetos gráficos para la interacción usuario-máquina.
- Cluster de Instrumentos Automotriz: Panel que muestra información vital del vehículo al conductor (velocidad, combustible, etc.).
- Hardware de Bajo Costo: Componentes electrónicos económicos y disponibles, como ESP32 y transceptores CAN, para proyectos accesibles.
- Bus CAN (Controller Area Network): Estándar de comunicación vehicular para interconectar dispositivos sin un host central.
- Nodos (en una red CAN): Dispositivos individuales conectados al bus CAN que envían o reciben mensajes.
- Gateway: Dispositivo que conecta dos redes o protocolos, usado aquí para enlazar la red CAN con la GUI.
- ESP32: Microcontrolador de bajo costo con Wi-Fi y Bluetooth, empleado para construir los nodos CAN.
- Transceptor CAN: Componente que adapta señales lógicas del microcontrolador a señales físicas del bus CAN.
- Tkinter: Librería estándar de Python para desarrollar interfaces gráficas de usuario (GUIs).

Índice general

Agradecimientos II

Resumen III

Definiciones IV

1. Introducción 1
- 1.1. Interfaces automotrices . . . 1
  - 1.1.1. Cuadros de Instrumentos . . . 1
  - 1.1.2. Diagnóstico a bordo . . . 8
- 1.2. Redes automotrices . . . 9
- 1.3. Interfaces gráficas de usuario automotrices en la educación . . . 20
- 1.4. Objetivos y plan del trabajo del proyecto de Tesis . . . 24
  - 1.4.1. Objetivo general . . . 24
  - 1.4.2. Objetivos específicos . . . 24

2. Justificación 25
- 2.1. Simulación de Hardware y Software en Bucle . . . 25
  - 2.1.1. Fundamentos de HIL y SIL . . . 25
  - 2.1.2. Ventajas del HIL y SIL . . . 26
- 2.2. Instrumentos Virtuales de Vehículos: Estudio de Caso del Head-Up display (HUD) . . . 27
- 2.3. Herramientas de entrenamiento automotriz, caso de estudio multipuerto de inyección electrónica de combustible . . . 28

3. Sistemas electrónicos en el automóvil 30
- 3.1. Unidades de control electrónico en el vehículo . . . 30
  - 3.1.1. Arquitectura de las ECU´s . . . 30
  - 3.1.2. Funcionalidad y operación . . . 31
  - 3.1.3. Sistemas de diagnostico . . . 31
  - 3.1.4. Rol en la red vehicular . . . 31
  - 3.1.5. Avances tecnológicos . . . 32
  - 3.1.6. Importancia en el futuro automotriz . . . 32
- 3.2. Sensores y actuadores automotrices, un enfoque sistémico . . . 32
  - 3.2.1. Sensores y actuadores en el sistema de seguridad . . . 32

3.2.2. Sensores y actuadores en el sistema de confort y conveniencia . . . 33
3.2.3. Sensores y actuadores en el tren motriz . . . 34
3.3. Sensores automotrices considerados en el proyecto de Tesis . . . 34
3.3.1. Sensor ECT . . . 34
3.3.2. Sensor APP . . . 39
3.3.3. Sensor CKP . . . 41
3.3.4. Sensor de nivel de gasolina . . . 42
3.4. Actuadores automotrices considerados en el protecto de Tesis . . . 44
3.4.1. Motor de Corriente Continua (DC) RS550 . . . 44

4. Integración de Software y Hardware de código abierto para la Implementación de una Red Vehicular 47
4.1. Interfaces gráficas en MATLAB Y Python . . . 47
4.1.1. APP Designer . . . 47
4.1.2. Tkinter y Qt . . . 51
4.1.3. App Designer vs Tkinter . . . 53
4.2. Desarrollo del Cuadro de Instrumentos Automotriz Virtual . . . 54
4.2.1. Implementación en App Designer . . . 54
4.2.2. Implementación en Tkinter . . . 56
4.2.3. Integración de Comunicación Serial . . . 59
4.2.4. Generación del Instalador de la GUI . . . 61
4.3. Soluciones CANBUS para hardware de código abierto . . . 62
4.3.1. Tarjetas Arduino . . . 62
4.3.2. ESP32 . . . 62
4.3.3. Shields CAN Bus de SparkFun/Seeedstudio . . . 62
4.3.4. Módulo MCP2515 . . . 62
4.3.5. Transceptor SN65HVD230 . . . 62
4.3.6. Selección del Hardware . . . 62
4.4. Red de 4 nodos con sensores y actuadores automotrices . . . 63
4.4.1. Proceso de Desarrollo . . . 64
4.4.2. Nodo de Nivel de Combustible . . . 66
4.4.3. Nodo de Temperatura . . . 70
4.4.4. Nodo del Tren Motriz . . . 76
4.4.5. Implementación de la Red CAN . . . 85
4.4.6. Montaje de la Red de Cuatro Nodos . . . 93
4.4.7. Análisis de trama de datos CAN . . . 98
4.4.8. Comparación de Costos de Soluciones . . . 101

5. Conclusiones 103
5.1. Contribuciones Originales . . . 104
5.2. Trabajo a futuro . . . 104

Apendice A: Recursos del Proyecto 106

Apendice B: Manual de Usuario **107**
5.3. Introducción . . . 107
5.3.1. Descripción del Programa . . . 107
5.4. Requisitos del Sistema . . . 108
5.5. Instalación del Programa . . . 108
5.6. Configuración del Nodo (Gateway) . . . 109
5.6.1. Requisitos del Nodo . . . 109
5.6.2. Programar el Nodo . . . 109
5.6.3. Conectar el Nodo . . . 110
5.7. Uso del Programa . . . 111
5.8. Solución de Problemas . . . 112
5.9. Licencia . . . 113
5.10. Contacto . . . 113
5.11. Créditos . . . 113

Bibliografía **114**

Capítulo 1

Introducción

1.1. Interfaces automotrices

Las interfaces automotrices han evolucionado de manera significativa en las últimas décadas, transformando la experiencia de conducción y optimizando la interacción entre el conductor y el vehículo. Estas interfaces abarcan desde los controles físicos y analógicos hasta las pantallas táctiles y sistemas avanzados de realidad aumentada.

1.1.1. Cuadros de Instrumentos

A lo largo de los años y conforme se ha incrementado la complejidad de los automóviles, se generó la necesidad de conocer el estado de algunos elementos que intervienen en el funcionamiento del automóvil, por lo cual fueron apareciendo ciertos medidores que a lo largo del tiempo fueron evolucionando hasta llegar al cuadro de instrumentos que conocemos actualmente.

El registro mas antiguo de un cuadro de instrumentos data de 1923, en el modelo Ford T; el cual, únicamente contaba con un amperímetro que servía para conocer el nivel de carga de la batería; dicho elemento ya no se ve en los cuadros de instrumentos actuales. De igual manera, como la velocidad del modelo T era muy baja, el velocímetro no venia en los modelos masivos, pero existe registro que algunos de estos modelos tienen incorporados el tácometro [1]. Esta simplicidad de diseño del Ford T se puede observar en la figuras 1.1a y 1.1b respectivamente.

(a) Velocimetro y amperimetro del año 1914.$Fuente$: $https$: $//es.wikipedia.org/wiki/Ford_T$.

(b) Tablero original del año 1924.$Fuente$: $https$: $//www.fastlanecars.com/vehicles/3092/1924-ford-model-t$.

Figura 1.1: Tableros de instrumentos del modelo Ford T.

A partir de 1932 muchos automóviles tenían tableros de instrumentos más complejos, los cuales ya contaban con medidores de temperatura, nivel de gasolina, tácometro, amperímetro, entre otros medidores, como se nota en la figura 1.2, los cuales ya se iban asemejando al cuadro de instrumentos actual, aunque aún venían con elementos que posteriormente se desecharían [1].

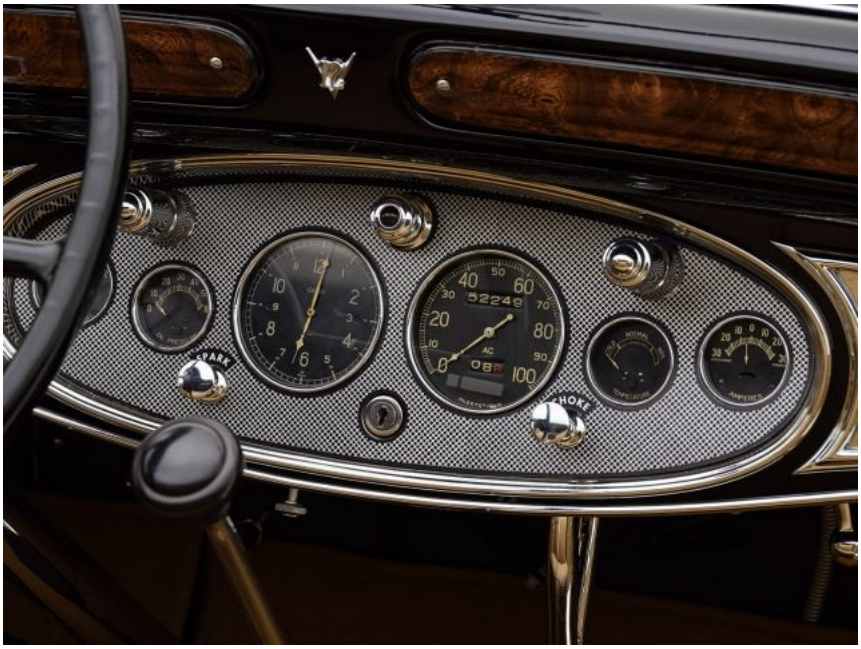


Figura 1.2: Cuadro de instrumentos de un cadillac v12 convertible del año 1931. $Fuente: https://hymanltd.com/vehicles/5914-1931-cadillac-v12-convertible-coupe/.$

Entre los años de 1940 a 1980 el tablero de instrumentos fue evolucionando con base en el estilo de cada época; adicional a esto, la aparición del Diodo Emisor de Luz, LED por sus siglas en inglés, beneficio a los tableros de dicha época; ya que se usaron los LEDs para mejorar la visualización en las noches, como se puede observar en la figura 1.3 .

Figura 1.3: Tablero con LED de Chevy Camaro de 1970-1981. $Fuente$: $https$: $//www.ebay.com/itm/1970-1981-Chevy-Camaro-Gauge-Instrument-Cluster-LED-bulb-upgrade-70-81-/202536168968$.

De 1990 al 2000 la comodidad del usuario se volvió primordial, por lo cual se busco una estandarización en los cuadros de instrumentos procurando contar información relevante acerca del vehículo;lo cual, desencadeno en la ubicación actual de los mismos, que es en la parte posterior del volante, para que la información sea más accesible para el conductor [1], tal y como se muestra en la figura 1.4.

Figura 1.4: Cadillac Seville 1993. $Fuente$: $http$: $//smclassiccars.com/cadillac/539522-1993-cadillac-seville-touring-sts-only-13k-miles-runs-and-drives-great.html$.

En la actualidad, los cuadros de instrumentos cuentan con una unidad de control electrónico(ECU por sus siglas en ingles), la cual gestiona la información recibida de los diversos sensores dentro del vehículo a través de redes multiplexadas desde otras ECU, para después procesar dicha información y enviar mensajes al conductor a través de indicadores dentro del mismo cuadro de instrumentos, los cuales se pueden clasificar de la siguiente forma [2]:

- Indicador Luminoso.
- Reloj analógico.
- Reloj electrónico o display.
- Indicadores acústicos.

En la actualidad, se cuenta con una nueva tendencia para los cuadros de instrumentos, en la cual se busca que el tablero se encuentre dentro del rango de visión del conductor. Esto para evitar que tenga que bajar la vista para poder verificar algunos datos del automóvil y así reducir la cantidad de accidentes, ya que una de las causas de accidentes, además del exceso de velocidad, son las distracciones. Para evitar esto, se busca implementar el sistema de visualización cabeza arriba o Head up Display(HUD) dentro de los automóviles, como lo explica Hernández del Arco [3]. En la figuras 1.5a y 1.5b se muestran algunos HUD que ya están en desarrollo.

(a) Huawei HUD. *Fuente : https : //www.moztako.click/2021/09/huawei − apresenta − tecnologia − hud − ar − para.html.*

(b) Xiaomi HUD.*Fuente : https : //www.lavanguardia.com/andro4all/android/xiaomi − carrobot − hud − pantalla − coche.*

Figura 1.5: Visualizadores cabeza arriba.

Los cuadros de instrumentos pueden variar su diseño dependiendo de la marca del vehículo. A continuación en la figura 1.6 se muestra un cuadro de instrumentos con algunos indicadores.

1. Tacómetro electrónico
2. Indicador de sobrecalentamiento
3. Lampara de baja presión
4. Indicador de faro antiniebla
5. Indicador de luz de carretera
6. Indicador de giro a la izquierda
7. Pantalla digital
8. Indicador de problema en frenos
9. Indicador de giro a la derecha
10. Indicador de frenos ABS
11. Indicador del sistema de control de motor
12. Velocímetro
13. Indicador mal funcionamiento del motor
14. Indicador de cinturón no abrochado
15. Indicador para pisar el freno
16. Indicador de gasolina
17. Indicador de airbags defectuosas
18. Indicador de puertas abiertas
19. Indicador de problemas con el ESP
20. Botón para restablecer el kilometraje
21. Botón para ajustar la hora
22. Indicador de dirección asistida
23. Indicador de batería descargada

Figura 1.6: Cuadro de Instrumentos VW Polo.$Fuente$: $https$: $//carsclick.ru/volkswagen/obzor/pro-pribornuju-panel-folksvagen-polo/$

Además, algunos cuadros de instrumentos van equipados con ordenador de a bordo (OBD), el cual ofrece una información más detallada a través de uno o varios displays, que pueden coincidir o no con el display del cuadro de instrumentos [2]. En la figura 1.7 se muestra una linea del tiempo de lo mencionado anteriormente.

Figura 1.7: Linea del tiempo del cuadro de instrumentos.

1.1.2. Diagnóstico a bordo

El diagnóstico a bordo, conocido comúnmente como OBD-II (On-Board Diagnostics II), es un sistema esencial en los vehículos modernos que desempeña un papel fundamental en la monitorización y el mantenimiento de los motores de combustión interna. El OBD-II es una tecnología que ha revolucionado la forma en que los técnicos automotrices y los propietarios de vehículos pueden rastrear y solucionar problemas relacionados con el rendimiento del automóvil.

Orígenes de OBD I

Para comprender completamente la importancia del OBD-II, es necesario retroceder en el tiempo. A finales de la década de 1960, los problemas de contaminación del aire y la creciente complejidad de los motores de automóviles impulsaron la necesidad de un sistema de diagnóstico más avanzado. Esto dio lugar al OBD-I, la primera generación de diagnóstico a bordo, que permitía a los mecánicos acceder a datos básicos del motor y emitía códigos de error simples.

Sin embargo, la verdadera revolución se produjo en 1996, cuando se introdujo el OBD-II en todos los vehículos fabricados en los Estados Unidos. Este sistema estableció estándares mucho más rigurosos, exigiendo a los vehículos que cumplan con regulaciones más estrictas de emisiones y que proporcionen una mayor cantidad de información de diagnóstico. Además, el OBD-II se convirtió en un estándar global, lo que permitió a los fabricantes de vehículos y a los técnicos automotrices de todo el mundo acceder a información estandarizada y mejorar la capacidad de diagnosticar problemas de manera más eficiente [4].

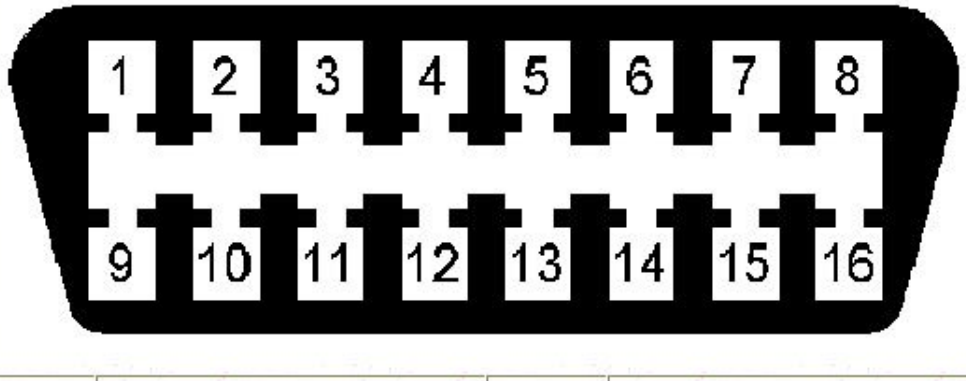
El OBD-II ofrece numerosos beneficios a propietarios de vehículos y profesionales de la industria automotriz. Algunos de estos beneficios incluyen:

1. **Detección temprana de problemas:** El OBD-II puede identificar problemas potenciales antes de que se conviertan en fallas graves, lo que permite una reparación o mantenimiento preventivo oportuno.

2. **Reducir las emisiones contaminantes:** El monitoreo constante de las emisiones de escape ayuda a reducir la contaminación ambiental y garantiza que los vehículos cumplan con los estándares de emisiones.

3. **Ahorro de tiempo y dinero:** Los códigos de diagnóstico facilitan la identificación precisa de problemas, lo que acelera el proceso de reparación y reduce los costos de mano de obra.

La figura 1.8 muestra un conector OBD II.



PIN	DESCRIPTION	PIN	DESCRIPTION
1	Vendor Option	9	Vendor Option
2	J1850 Bus +	10	j1850 BUS
3	Vendor Option	11	Vendor Option
4	Chassis Ground	12	Vendor Option
5	Signal Ground	13	Vendor Option
6	CAN (J-2234) High	14	CAN (J-2234) Low
7	ISO 9141-2 K-Line	15	ISO 9141-2 Low
8	Vendor Option	16	Battery Power

OBD-II Connector and Pinout

Figura 1.8: Conector y Pines de Salida del OBD II.$Fuente$: $https$: $//carros.narkive.es/tgeFj8fV/potencia-obd-ii-cuando-la-llave-no-esta-en-ignicion$

1.2. Redes automotrices

Conforme pasaban los años, la necesidad de comunicación dentro de los vehículos fue generando problemas a la hora de interconectar todos los cables; por lo cual, la industria automotriz tuvo que desarrollar un protocolo de comunicación que redujera la cantidad de cableado dentro del automóvil, ya que estos generaban mucho peso al igual que hacia mas complicado el entendimiento de las conexiones. En 1980 comenzaron los esfuerzos para generar un protocolo que fuera capaz de cubrir las necesidades de comunicación que existían en ese tiempo. En respuesta a esta necesidad se crearon varios protocolos de comunicación, tales como el protocolo de Red de Conexión Local (LIN), el de transporte de sistemas orientado a medios (MOST), FlexRay y el de red de área del controlador(CAN).

En 1983 Robert Bosh GmbH desarrollo el protocolo CAN Bus el cual se enfoco prin-

cipalmente a las aplicaciones en las cuales la comunicación en tiempo real es critica entre unidades de control para la gestión del motor,transmisión, frenos, etc. [5]. En la figura 1.9, se muestra la historia del protocolo CAN.

Figura 1.9: Historia CAN Bus.

Las redes CAN pueden ser de alta velocidad(CAN HS), cuya velocidad de transmisión es de 500 Kb/s a 1 Mb/s;o de baja velocidad(CAN LS), que tiene una velocidad de hasta 250 Kb/s. El CAN HS, también conocido como CAN tracción, se utiliza para aplicaciones donde se requiere un alto nivel de seguridad, como el sistema de frenos ABS, el sistema de bolsas de aire, etc. Por otro lado, el CAN LS o CAN confort se utiliza principalmente en aplicaciones de confort; por ejemplo, el limpiaparabrisas, los elevadores de las ventanas, etc.

Estandarización

El modelo de interconexión de sistemas abiertos(OSI por sus siglas en inglés) es una arquitectura para la comunicación entre computadores, la cual fue desarrollada por la Organización Internacional de Estandarización(ISO) con el objetivo de ser el marco de referencia para el desarrollo de protocolos.El modelo considera siete capas [6], las cuales se muestran en la figura 1.10 .

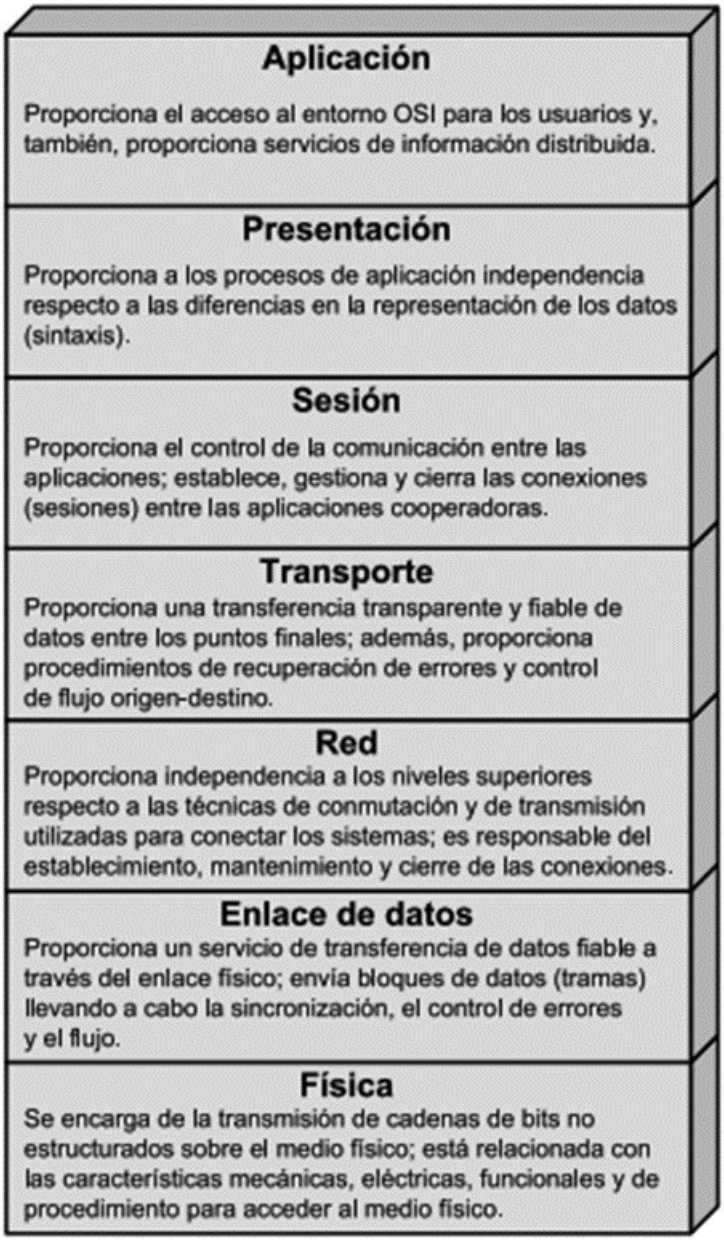
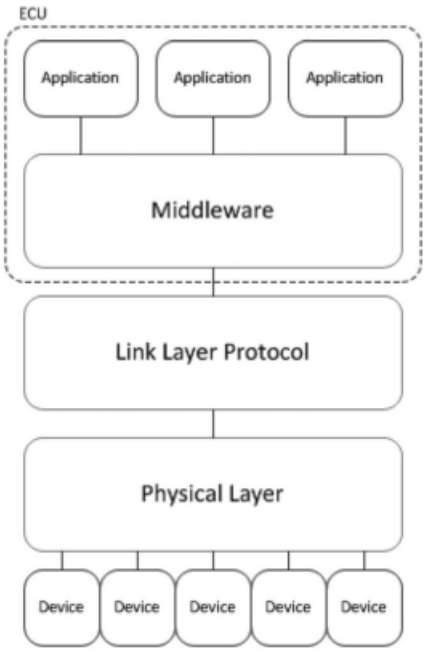


Figura 1.10: Capas del modelo OSI. Fuente: [6]

El protocolo CAN BUS esta definido por 3 de las 7 capas del modelo OSI, las cuales incluyen: Capa Física, Capa de Datos y Capa de Aplicación.Estas, se muestran en la Figura 1.11.



Layered view of an automotive communication network.

Figura 1.11: Arquitectura del protocolo CAN según modelo OSI. Fuente: [7]

El estándar ISO 11898 *Vehículos de carreta-Red de área de controlador(CAN)* describe la arquitectura CAN por medio de capas del modelo ISO/OSI. Específicamente, la Capa Física y la Capa de Enlace de Datos para transmisiones de hasta un 1 Mbit/s.

La parte 1 y 2 corresponden conjuntamente a la primera versión de la norma:

- **ISO 11898-1**-Parte 1: Capa de Enlace de Datos y señalización
- **ISO 11898-2**-Parte 2: Unidad de acceso medio de alta velocidad

Posteriormente se desarrollaron las siguientes partes:

- **ISO 11898-3**-Parte 3: Interfaz de baja velocidad, tolerante a fallas, dependiente del medio
- **ISO 11898-4**-Parte 4: Comunicación activada por tiempo
- **ISO 11898-5**-Parte 5: Unidad de acceso medio de alta velocidad con modo de bajo consumo

Además de la familia ISO 11898, existen otros estándares para CAN de otras organizaciones de estandarización. Por ejemplo la Sociedad de Ingenieros Automotrices (SAE) con el estándar J2284-1 [5].

Capa física

La capa física del CAN Bus esta compuesta por:

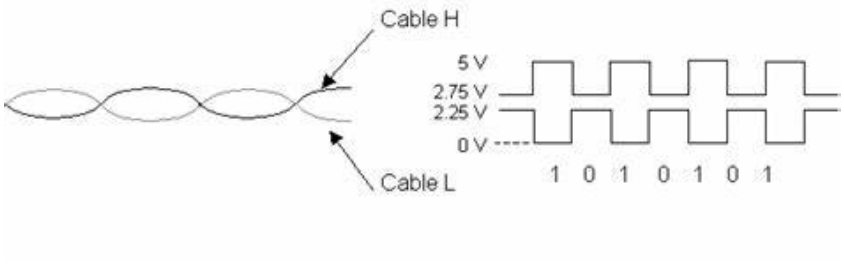
- Controlador
- Transceptor
- Cable de Bus de red o de datos

A continuación, se describe cada elemento que conforma la capa física de CAN.

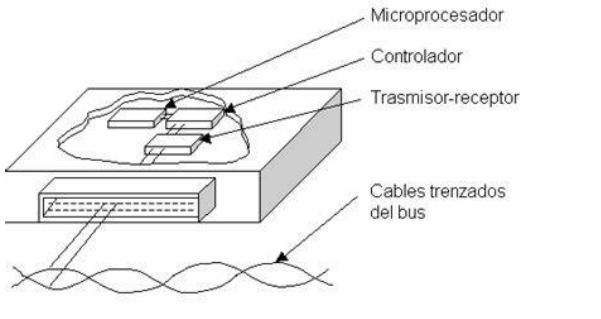
Controlador Es el elemento que se encarga de comunicar el microprocesador con el transmisor-receptor; ya sea para enviar información al microprocesador o enviarla al bus de datos mediante el transceptor. El controlador determina la velocidad de transmisión de datos y trabaja con valores de tensión bajos.

Transceptor Es un transmisor y un receptor; el cual, tiene la misión de *recibir y de transmitir los datos*, además de acondicionar y preparar la información para que pueda ser usada por los controladores. Esta formado por un filtro y uno o varios comparadores.

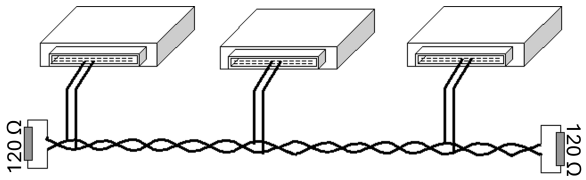
Cable de Bus de datos Son los canales a través de los cuales fluye la información; el cual, cuenta con un enlace multipunto y una transmisión Full-Duplex para la transmisión de los datos.Los datos se transmiten a través de un par trenzado que se conecta entre las unidades principales de la red CAN, esto es para reducir la interferencia eléctrica en la transmisión de datos; esto se debe, a que la transmisión de datos es en base a la diferencia de voltaje que se tiene entre los dos cables del par trenzado en donde la tensiones suelen oscilar entre 1.5 y 2.5 V en el cable CAN L y entre 2.5 y 3.5 V en el CAN H. La figura 1.12a muestra como trabaja una red CAN por medio de diferencia de voltaje, la figura 1.12b la composición del nodo CAN y la figura 1.12c muestra la composición de una Red CAN.



(a) Diferencial de voltaje



(b) Nodo CAN



(c) Red CAN

Figura 1.12: Elementos de una red CAN. $Fuente: https://slideplayer.es/slide/35272/$

Para comprender mejor la función de cada componente físico de una red CAN se muestran a continuación los pasos seguidos durante un envío de datos a través de una linea CAN:

1. Los sensores envían la información a su centralita correspondiente.
2. El microprocesador de la centralita trata esta información y la envía al controlador. Este a su vez la pasa al transceptor
3. El transceptor transforma la información digital recibida en señales eléctricas y la vuelca en el bus de datos.
4. El resto de centralitas reciben el mensaje y deciden si les interesa o no. Envían una confirmación de recepción del mensaje al bus de datos.
5. Las centralitas interesadas en el mensaje lo aceptan, lo procesan y deciden si ignorarlo o no.

Capa de enlace de datos

La trama de datos de CAN esta codificada usando el método Non Return Zero (NRZ) con bits de relleno esto tiene como ventaja que no se requiere un ancho de banda grande para poder transmitir. De igual manera, esto ayuda a que la lectura de las información sea mejor ya que se puede adaptar a los tiempos de reloj de cada nodo.En el método de relleno el transmisor puede enviar 5 bits con el mismo valor consecutivos y el siguiente bits a enviar es un bit con el valor inverso; por lo cual el receptor lee los 5 bits y el siguiente lo descarta.

Dentro del protocolo CAN se tienen 4 tipos de tramas, las cuales son las siguientes:

- Trama de datos
- Trama de error
- Trama remota
- Trama de sobrecarga

La trama de datos transporta los datos del mensaje mientras que las otras tramas son para contención de fallos, activación y sincronización.

Trama de datos Existen dos tipos de tramas de datos(Figura 1.13):

- CAN 2.0A (11 bits de identificación)
- CAN 2.0B (29 bits de identificación)

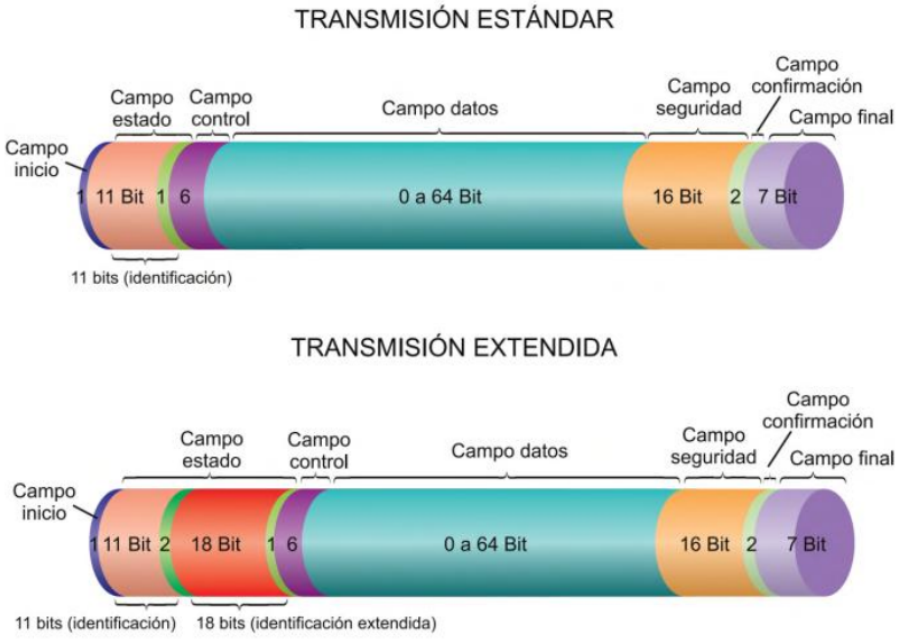


Figura 1.13: Estructura de la trama de datos de CAN.Fuente: [2]

El CAN 2.0A permite hasta 2,048 direcciones lógicas diferentes(0-2,047), esto quiere decir que solo podemos generar máximo 2,048 mensajes utilizando esta especificación. Mientras que el CAN 2.0B puede generar hasta 536,870,912 mensajes diferentes. Las especificación CAN 2.0A está conformada por diferentes campos [5]:

- **Inicio de trama (1 bit):** Marca el inicio de la trama utilizando un bit dominante; es decir un bit 0.
- **Campo de Arbitraje (12 bits):** Este campo contiene la identificación del mensaje además de generar la prioridad del mismo. El identificador de 11 bits va seguido del bit RTR el cual identifica si el mensaje contiene datos(bit 0) o no contiene datos(bit 1).
- **Campo de Control (6 bits):** El primer bit del campo de control sirve para identificar si existe una extensión en el identificador (IDE). Si se tiene una especificación CAN 2.0A el bit IDE es 0, por lo contrario si el bit IDE es 1 la especificación sera CAN 2.0B. El siguiente bit r0 es reservado. Los últimos 4 bits la longitud del campo de datos(DLC).
- **Campo de Datos (0-8 bytes):** Contiene los datos del mensaje correspondiente.
- **Campo CRC (16 bits):** Contiene la suma de comprobación de los bits anteriores de la trama.La suma de comprobación CRC de 15 bits de longitud es únicamente usada para detectar fallos, no para corregir errores.
- **Campo de Reconocimiento (2 bits):** En este campo el transmisor envía dos bits recesivos (1) y espera que el bit que se encuentra el slot ACK cambie a un bit dominante (0), lo cual indica que el mensaje ha sido recibido por al menos un receptor.
- **Fin de trama (7 bits):** Este campo tiene 7 bits recesivos seguidos lo cual rompe la codificación utilizada. Lo cual avisa el fin de la trama.

En la especificación 2.0B los campos que cambian su estructura son el campo de Arbitraje y el primer bit del campo de control.

- **Campo de Arbitraje (32 bits):** El campo se divide en 3 partes. Los primeros 11 bits son los mas significativos(MSBs) seguido de 2 bits recesivos: bit de sustituto de respuesta remota (SRR) y bit indicador de extensión de Identificación (IDE). Posteriormente se encuentra los 18 bits menos significativos(LSBs) del campo. Por ultimo se encuentra el bit RTR.
- **Campo de Control (6 bits):** A diferencia de la especificación 2.0A en este campo empieza con dos bits recesivos r1 y r0. Los últimos 4 bits al igual que la especificación 2.0A contiene la longitud del campo de datos.

Trama de error Un principio del CAN es detectar la mayor cantidad de posibles errores y tratarlos dentro del circuito integrado del CAN. Cualquier error que sea detectado

por un nodo dentro de la red es notificado a los demás nodos. Después de la notificación del error todos los participantes de la red descartan el mensaje actual; todo esto, es posible gracias a la trama de error que al igual que el final de la trama de datos rompe las reglas del bit de relleno. La estructura de la trama de error se muestra en la figura 1.14.

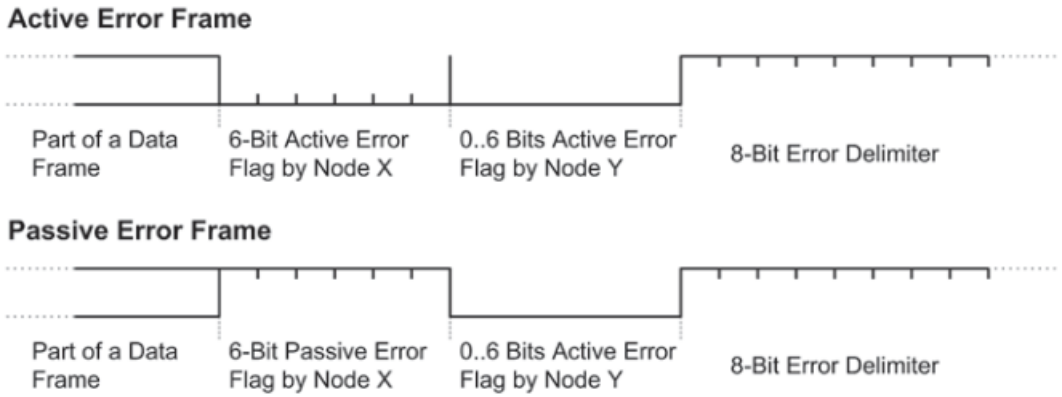


Figura 1.14: Estructura Trama de Error. Fuente: [5].

El manejo del error esta habilitado para identificar 5 diferentes tipos de error

- **Error de Bit:** Aparece cuando el bit transmitido no recibe el mismo valor que ha sido enviado. Excluyendo el campo de arbitraje y la ranura de confirmación.
- **Error de Relleno:** Aparece cuando mas de 5 bits consecutivos tienen el mismo valor. Sin tomar en cuenta el fin de trama y el espacio entre tramas.
- **Error de Comprobación:** Aparece cuando la suma de comprobación calculada es diferente a la suma de comprobación recibida
- **Error de Formulario:** Cuando existe una violación en el formato de trama
- **Error de acuse de recibo:** Cuando un transmisor no recibe el bit de confirmación en la ranura ACK.

Trama remota Alternativamente a los mensajes que comúnmente se mandan por el bus, el CAN permite la opción que un destinatario de una información especifica pueda solicitar datos actuales al emisor a través de esta trama. La cual tiene como su bit recesivo RTR y su trama de datos no contiene datos(independientemente del DLC). La estructura de la trama remota se muestra en la figura 1.15. Como se puede notar es muy parecida a la estructura CAN 2.0A Y 2.0B a diferencia de su campo de arbitraje. La trama remota debe ser enviada con el mismo DLC que su correspondiente trama de datos. Si mas de un nodo CAN iniciara simultáneamente tramas remotas con el mismo identificador, esta destruiría a la otra.

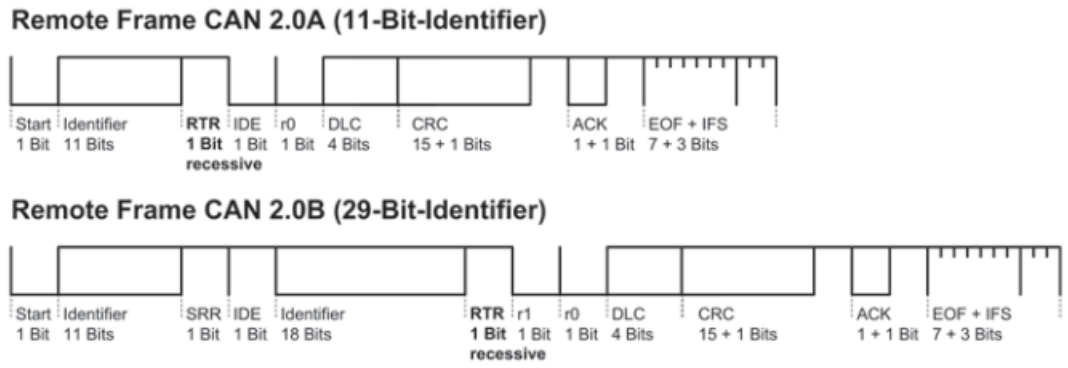


Figura 1.15: Estructura Trama Remota. Fuente: [5].

Trama de sobrecarga Otro medio de manejo de excepciones es la trama de sobrecarga. La estructura de esta trama es exactamente la misma que la trama de error a excepción que en la trama de error se sobre escribe y destruye el mensaje mientras que en la trama de sobrecarga empieza exclusivamente en el espacio entre tramas(Véase en la Figura 1.16).

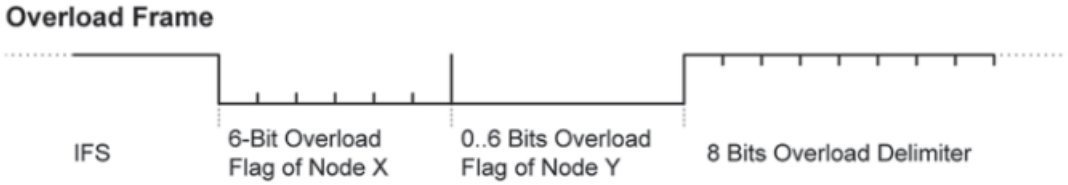


Figura 1.16: Estructura de la trama de sobrecarga.Fuente: [5].

Funcionamiento de la red CAN

El funcionamiento de la red CAN puede entenderse mejor a través de la descripción de sus procesos principales: la transmisión de mensajes, el arbitraje de acceso al bus y la recepción de datos por parte de los nodos.

Proceso de Transmisión y Recepción de Mensajes

Cuando un nodo desea enviar un mensaje, primero debe verificar si el bus está libre. Si el bus está ocupado por otro nodo, el nodo emisor debe esperar hasta que se libere. Una vez que el bus esté libre, el nodo puede comenzar a transmitir el mensaje. En este proceso, los mensajes se envían en tramas de datos que incluyen tanto la información que se quiere transmitir como el control necesario para garantizar que el mensaje llegue correctamente.

Durante la transmisión, puede ocurrir un arbitraje para determinar qué nodo tiene prioridad en el bus. Esto ocurre si varios nodos intentan transmitir al mismo tiempo. El protocolo CAN utiliza una jerarquía de prioridades basada en el ID de los mensajes, de

modo que el mensaje con el ID más bajo (mayor prioridad) será el primero en transmitirse. Este mecanismo de arbitraje es una de las características distintivas del protocolo CAN y garantiza que la red se mantenga eficiente incluso en sistemas con muchos nodos.

El Arbitraje en la Red CAN

El arbitraje es el proceso que asegura que solo un nodo pueda transmitir en el bus en un momento dado. Si dos nodos intentan transmitir simultáneamente, los bits del mensaje se comparan bit a bit. El nodo que tenga un bit de valor 0 (dominante) en una posición donde el otro nodo tenga un bit de valor 1 (recesivo) prevalecerá y continuará con la transmisión. El nodo que no tiene prioridad se detiene y vuelve a intentarlo cuando el bus esté libre.

Ciclo Continuo de Comunicación

El ciclo de comunicación en la red CAN es continuo. Los nodos no solo transmiten datos, sino que también monitorean el bus para recibir mensajes de otros nodos. Esto garantiza que todos los dispositivos estén sincronizados y puedan operar conjuntamente. Si el mensaje recibido es válido, el nodo lo procesa. De lo contrario, continuará monitoreando el bus hasta que se reciba un mensaje válido.

Para ilustrar mejor este proceso, a continuación se presenta un diagrama de flujo que muestra de manera visual el funcionamiento de la red CAN en cuanto a la transmisión de mensajes, arbitraje y recepción.

Diagrama de Flujo del Funcionamiento de la Red CAN

El siguiente diagrama de flujo(figura 1.17 describe de manera detallada el proceso de funcionamiento de una red CAN, desde la preparación de un mensaje hasta su transmisión y recepción:

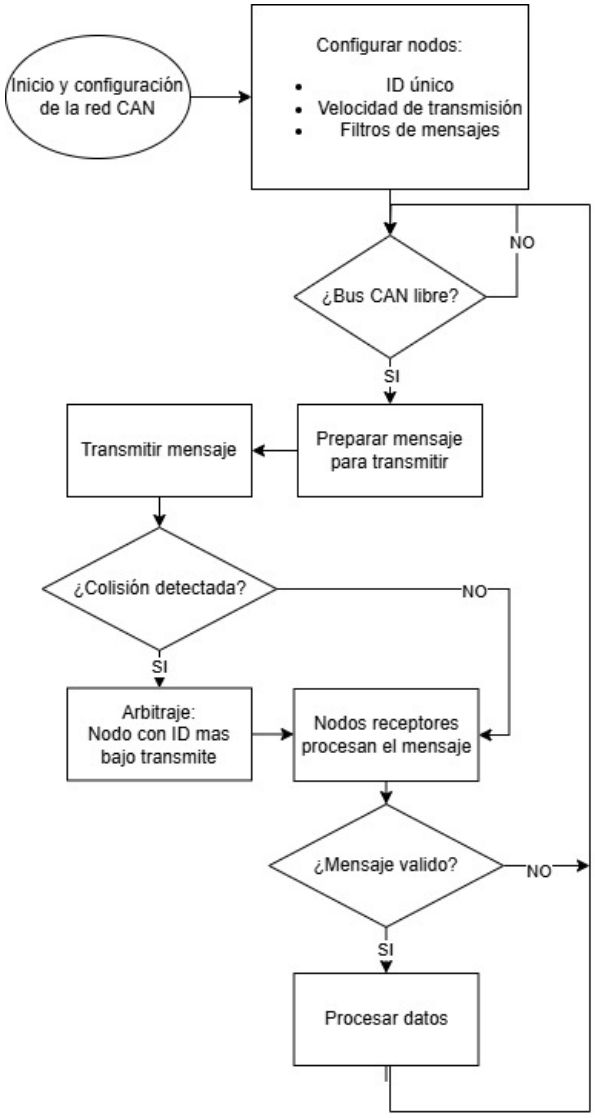


Figura 1.17: Diagrama de flujo del funcionamiento de la red CAN.

Este diagrama explica de forma detallada los pasos por los que pasa un mensaje desde que un nodo inicia la transmisión hasta que el mensaje es recibido y procesado por otros nodos. También muestra cómo se gestionan las situaciones de colisión mediante el proceso de arbitraje, y cómo los nodos gestionan el bus CAN de manera eficiente.

1.3. Interfaces gráficas de usuario automotrices en la educación

Los costos de funcionamiento de un laboratorio de interfaces y redes vehiculares para mejor enseñanza de conceptos de redes automotrices tales como el protocolo de

comunicación vehicular CAN BUS, la unidad de diagnóstico a bordo OBD2 y la interpretación de códigos de fallas, son muy altos. Tomando en cuenta que se debería de contar con al menos un panel de instrumentos por mesa de trabajo, cuyos costos oscilan entre los 2 mil pesos a 23 mil pesos, y al menos 3 nodos de comunicación de CAN BUS por mesa. El costo de 6 nodos CAN BUS con su respectivo hardware y software por parte del la empresa National Instruments ronda en la cantidad de 266 mil pesos., por lo cual el costo de una mesa de trabajo tomando en cuenta el tablero de instrumento mas barato tendría un costo aproximado de 267 mil pesos. En la figuras 1.18a y 1.18b se muestran dos elementos de una mesa de trabajo para la enseñanza de redes vehiculares.

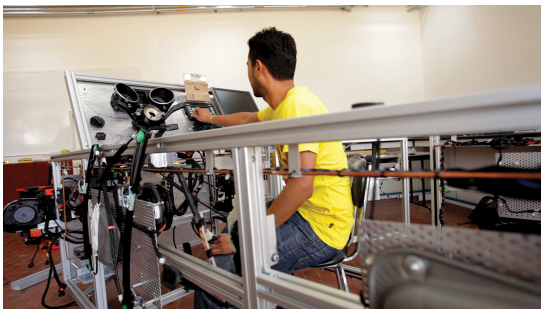
(a) Tablero de instrumetos Attitude.$Fuente$: $https$: $//articulo.mercadolibre.com.mx/MLM-748740508-cluster-tablero-instrumentos-attitude-15-19-aut-8100c312-_J\ M.$

(b) Modulo CAN BUS HS/FD NI.$Fuente$: $https$: $//www.ni.com/es-mx/shop/model/usb-8502.html$

Figura 1.18: Elementos de la mesa de trabajo.

El gasto anual de costos de mantenimiento de dicha infraestructura para un laboratorio rondaría mas de medio millón de pesos. Por lo cual puede ser un precio elevado para la implementación de dicho laboratorio. Las universidades grandes y, principalmente, las privadas, pueden costear este gasto; tan solo en Puebla, la Universidad de las Américas Puebla (UDLAP) y la Universidad Iberoamericana Campus Puebla cuentan con laboratorios (Figuras 1.19 y 1.20) con estas características.

(a) Vista general del Laboratorio.$Fuente$: $https://contexto.udlap.mx/impulso-a-la-manufactura-metalmecanica-mexicana/$

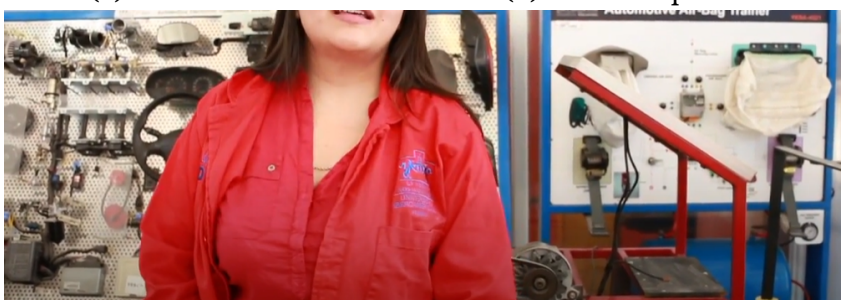


(b) Mesa de trabajo de redes vehiculares.$Fuente$: $https://contexto.udlap.mx/redes-automotrices/$.

Figura 1.19: Laboratorio automotriz UDLAP.

(a) Automovil.

(b) Medición por Escaner.



(c) Paneles de enseñanza de redes vehiculares y bolsas de aire

Figura 1.20: Laboratorio automotriz Ibero.$Fuente$: $https://www.youtube.com/watch?v=Z7WAl7WzUrM$

Sin embargo, universidades publicas, universidades privadas o centros de enseñanza técnica mas pequeños difícilmente pueden solventar el costo. Por ejemplo, en nuestra casa de estudios la Benemérita Universidad Autónoma de Puebla (BUAP), el laboratorio de Sistemas Automotrices (Figura 1.21) no cuenta con esta infraestructura a pesar de que una de las áreas de especialidad ofertadas para los estudiantes de la Ingeniería en Sistemas Automotrices es la de Comunicaciones Vehiculares. Otras instituciones donde

se imparte carreras afines al área automotrices incluyen a la Universidad Politécnica de Puebla(UPPue), la Universidad Tecnológica de Puebla (UTP) y la Universidad Popular Autónoma del Estado de Puebla (UPAEP) se muestran en la figura 1.22, por nombrar algunas que carecen de una infraestructura orientada hacia las comunicaciones vehiculares.

(a) Jetta

(b) Instrumentos

(c) Motores y auto

(d) Motor electrico

Figura 1.21: Laboratorio automotriz BUAP

(a) UPAEP.*Fuente : https : //upaep.mx/licenciaturas/ingenieria en − diseno − automotriz*

(b) UTPuebla.*Fuente : https : − //www.youtube.com/watch?v = jJY S4yV Fejg*

Figura 1.22: Otros Laboratorios automotrices de diferentes universidades

Hay en la literatura una gran cantidad de casos de éxito reportados en diversas áreas de la ingeniería como lo es el control, la instrumentación electrónica y los sistemas de comunicación digital [8–12].

1.4. Objetivos y plan del trabajo del proyecto de Tesis

1.4.1. Objetivo general

Implementar la interfaz gráfica de usuario de clúster de instrumentos de vehículo en algún lenguaje de programación, MATLAB o PyTHON, y su interacción con *hardware* de bajo costo que emule el panel de instrumentos de un vehículo real para su aplicación en la enseñanza de conceptos de interfaces y redes vehiculares.

1.4.2. Objetivos específicos

- Desarrollar una interfaz gráfica de usuario de un clúster de instrumentos automotriz utilizando el lenguaje de programación Python, la cual muestre las RPM del motor, la velocidad del vehículo, el nivel de combustible y la temperatura del motor.
- Implementar, con hardware de bajo costo, un sensor de nivel de líquido que emule el sistema de sensado del tanque de combustible y comunicar dicho nivel a la interfaz gráfica de usuario, para que el clúster de instrumentos virtual muestre el nivel de combustible.
- Implementar, con hardware de bajo costo, un tácometro que emule el sistema de sensado del cigüeñal del vehículo y comunicar las RPM, así como la velocidad de desplazamiento del auto, a la interfaz gráfica de usuario, para que el clúster de instrumentos virtual muestre la velocidad de giro del motor y la velocidad de desplazamiento del vehículo.
- Implementar, con hardware de bajo costo, un sistema de control de temperatura que emule el sistema de refrigeración del motor y comunicar dicho valor a la interfaz gráfica de usuario, para que el clúster de instrumentos virtual muestre la temperatura del motor.
- Comunicar los tres nodos anteriores: rpm-velocidad, temperatura de motor y nivel de tanque de combustible, mediante protocolo CAN BUS para su posterior comunicación con el clúster de instrumentos estando todos los nodos en red.

Capítulo 2

Justificación

2.1. Simulación de Hardware y Software en Bucle

Actualmente se busca reducir el tiempo y el costo que implican las pruebas en un ambiente real, por cual se ha buscado alternativas para poder realizar dichas pruebas. Algunas de estas técnicas son la simulación de **Hardware** y **Software** en bucle (HIL Y SIL por sus siglas en inglés), que han demostrado su eficiencia para reducir el tiempo de desarrollo y el costo del mismo.

2.1.1. Fundamentos de HIL y SIL

La simulación de HIL implica sustituir las partes de un sistema real por un modelo matemático de estas piezas. En el contexto automotriz, se puede evaluar una unidad de control electrónico utilizando un planta modelada (por ejemplo, motor, tanque de gasolina, frenos, etc.), esto ayuda a realizar pruebas del funcionamiento de la ECU sin necesidad de contar con el vehículo físico [13]. En la figura 2.1 se muestra un configuración de un sistema de HIL.

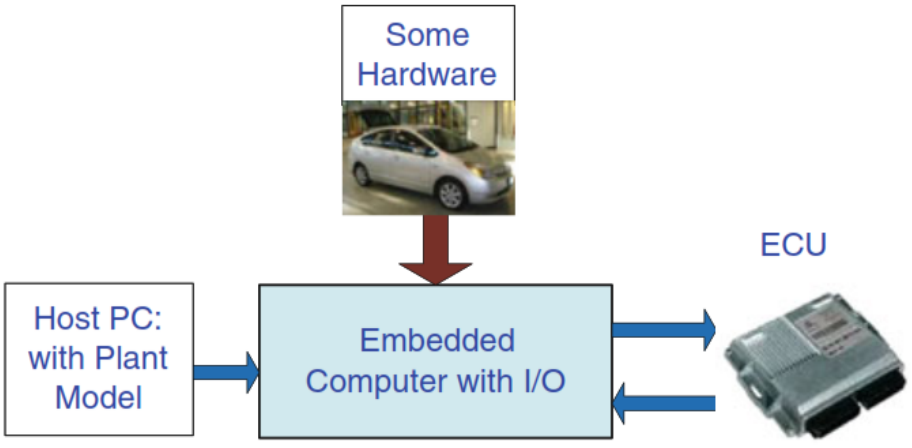


Figura 2.1: Configuración de un sistema HIL. Fuente: [13].

El SIL se utiliza para probar el software de un procesador real mediante simulación, logrando eficiencia operativa y brevedad de código en la fase inicial de desarrollo. El entorno de pruebas de SIL incluye sistemas de gestión de pruebas, ejecución de pruebas y gestión de versiones, lo cual permite garantizar una buena repetibilidad. En la figura 2.2 se muestra un modelo SIL típico. Si el software de la simulación SIL es capaz de simular un sistema real, la simulación SIL también puede simular la función de una simulación HIL.

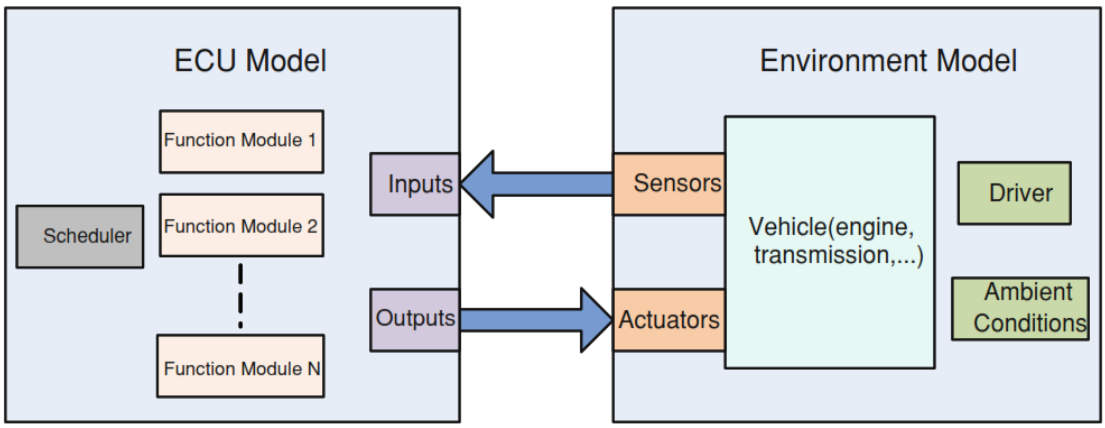


Figura 2.2: Test de modelo SIL. Fuente: [13]

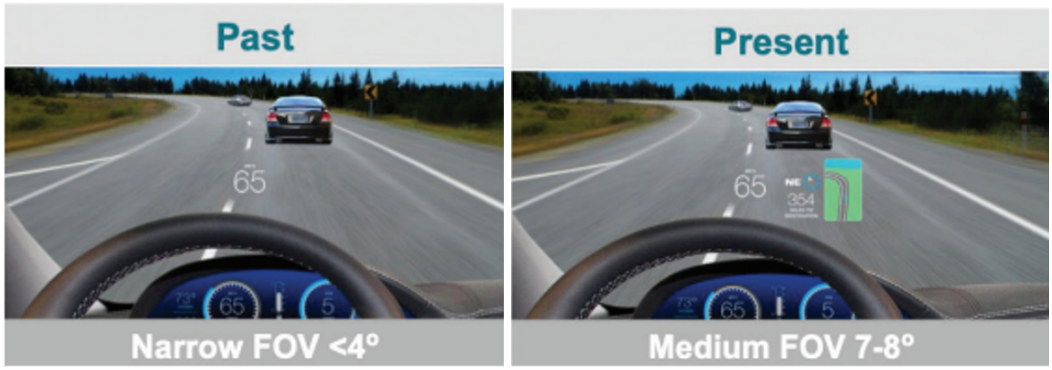
2.1.2. Ventajas del HIL y SIL

La simulación HIL se destaca como una estrategia eficiente en términos de costos, duración y seguridad, especialmente en situaciones con cronogramas ajustado y desarrollo acelerado. Esta técnica permite realizar pruebas incluso antes de la disponibilidad de prototipos, logrando un ahorro significativo de tiempo. Además desempeña un papel esencial en el desarrollo de componentes que llegan a contar con una interfaz gráfica de usuario.

La simulación HIL también se destaca por su capacidad para prevenir daños al evaluar situaciones potenciales, como el sobrecalentamiento en motores o el rendimiento del sistema anti bloqueo de frenos (ABS). Aunque HIL ofrece beneficios significativos en términos de errores y defectos de diseño, es importante destacar que SIL presenta ventajas en cuanto a agilidad y la ausencia de requisitos en tiempo real, lo que se traduce en una opción menos costosa. No obstante, es esencial señalar que SIL no puede reemplazar por completo a HIL, ya que no modela detalladamente los procesos de bajo nivel.

2.2. Instrumentos Virtuales de Vehículos: Estudio de Caso del Head-Up display (HUD)

El sector automotriz avanza hacia la electrificación, priorizando la seguridad del vial. Una innovación destacada es el parabrisas con Head-Up Display (HUD), donde un proyector a bordo transmite información clave directamente al parabrisas. Esto asegura que el conductor mantenga la vista en la carretera, mejorando el tiempo de reacción ante obstáculos y mejorando la seguridad. El HUD muestra datos como la velocidad y la navegación, eliminando la necesidad de apartar la vista hacia el panel de control. La fabricación del parabrisas implica laminar dos vidrios con un núcleo de butiral de polivinilo para compensar el ángulo de la imagen proyectada. Además, el HUD contribuye a una conducción más segura al mostrar información vital en el campo de visión del conductor, reduciendo las distracciones y mejorando la calidad de conducción en condiciones complejas de tráfico. Se destaca la futura evolución hacia HUD de realidad aumentada, impulsada por inteligencia artificial, que proporcionará información más amplia sobre los alrededores del automóvil. En resumen, el parabrisas con HUD ofrece beneficios significativos, mejorando la seguridad, la calidad de conducción y la concentración del conductor en la carretera [14].En la figura 2.3 se muestra la evolución que ha tenido el HUD a través del tiempo.



(a) Pasado. (b) Presente.

(c) Futuro

Figura 2.3: Evolución del HUD. Fuente: [12]

Por lo cual mediante las técnicas de simulación de HIL y SIL, se busca implementar una interfaz gráfica de usuario de un cuadro de instrumentos automotriz capaz de alinearse con las tendencias del HUD en el parabrisas. Al igual que se busca generar un sistema capaz de emular el funcionamiento de una red vehicular e inducir condiciones especificas para ser mostradas en nuestra interfaz.

2.3. Herramientas de entrenamiento automotriz, caso de estudio multipuerto de inyección electrónica de combustible

Dentro de la facultad de ciencias de la electrónica, específicamente en el laboratorio de sistemas automotrices podemos encontrar el entrenador del multipuerto de inyección, el MEGATECH MEG550G. El cual es un prototipo educativo que cuenta con varios sensores automotrices, al igual que una unidad de control electrónico. El principal objetivo que tiene este prototipo es mostrar el funcionamiento del sistema de inyección para vehículos de combustión interna. El entrenador se puede observar en la figura 2.4

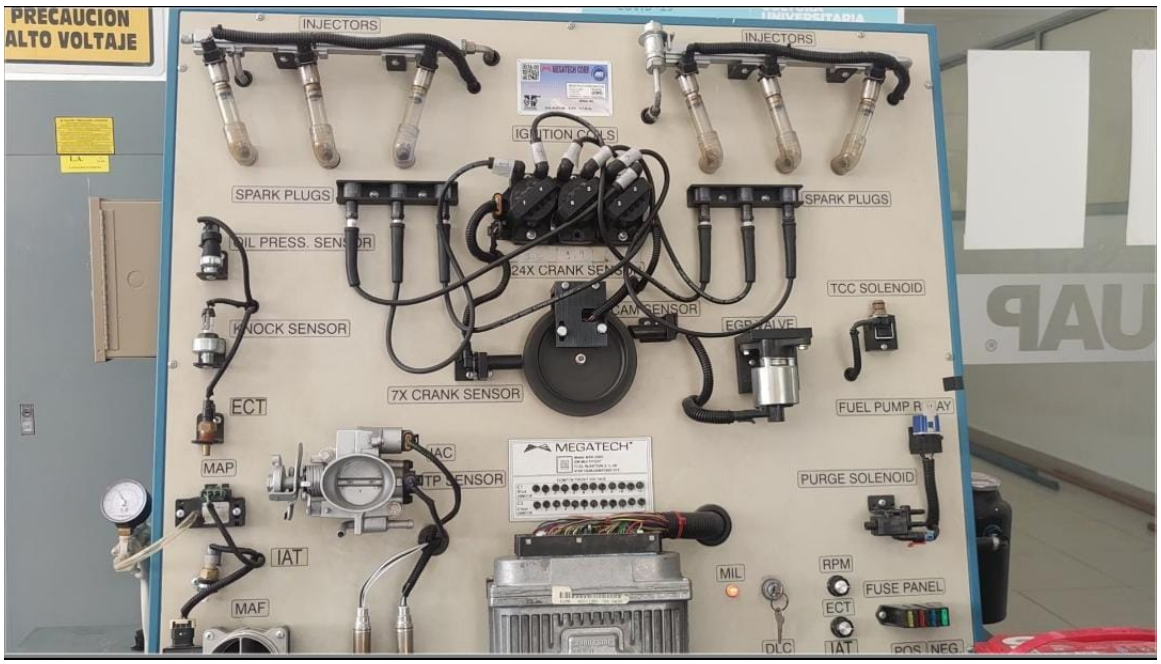
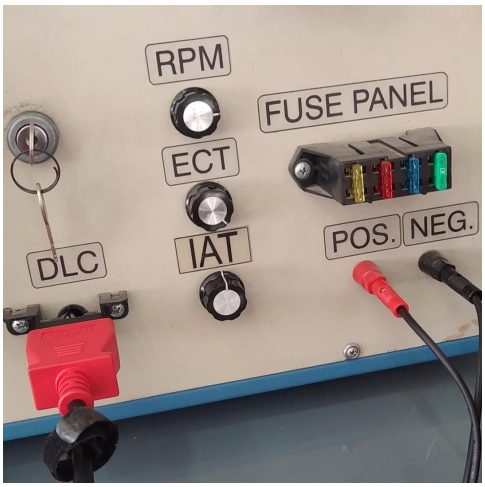


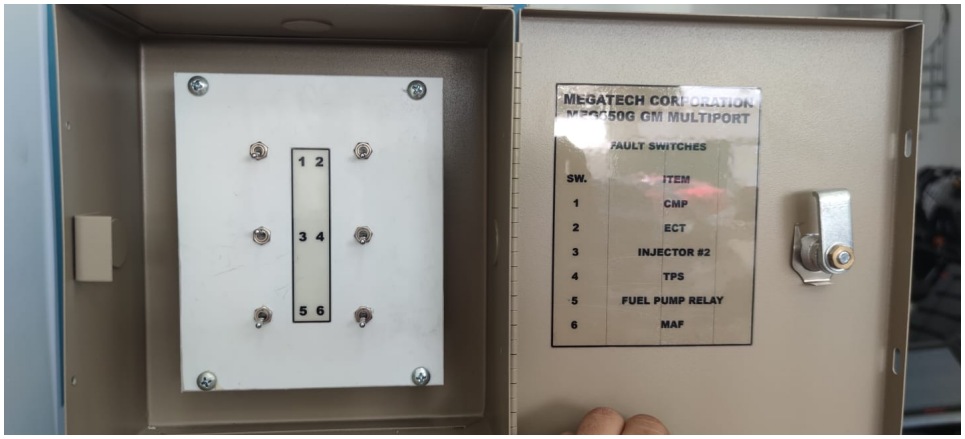
Figura 2.4: Entrenado de multipuerto de inyección MEG550G

El entrenador permite variar manualmente las señales de entrada de los sensores que lo componen; tales como el sensor de temperatura (ECT), el sensor del flujo de masa (MAF), el sensor de presión del colector (MAP), entre otros sensores. Al variar estas señales podemos generar fallas en el vehículo, las cuales se pueden observar con un escáner OBD II. De igual manera el entrenador cuenta con un tablero a un costado del mismo el cual también nos permite inducir otras fallas y poder observarlas en

el escáner. En la figura 2.5a se puede observar la conexión del escáner OBD II y en la figura 2.5 se visualiza la caja con las 6 fallas que podemos inducir.



(a) Entrada de OBD II en DLC



(b) Caja para inducir fallas

Figura 2.5: Elementos del MEG550G

Este entrenador nos ayudo a generar una visión general de los sensores automotrices de nuestro interés para poder observar como se desempeñan, y de esta manera poder implementarlos de una forma similar.

Capítulo 3

Sistemas electrónicos en el automóvil

3.1. Unidades de control electrónico en el vehículo

Las Unidades de Control Electrónico (ECU, por sus siglas en ingles) son componentes fundamentales en los vehículos modernos, responsables de procesar datos de sensores para optimizar el funcionamiento de diversos sistemas automotrices. Desde su introducción en la década de 1970 para gestionar la inyección electrónica de combustible, las ECU han evolucionado hasta convertirse en elementos esenciales que regulan desde el motor hasta sistemas avanzados de asistencia al conductor (ADAS) [15]. Este capitulo explora el diseño, la funcionalidad y los avances de las ECU, con un enfoque en su implementación utilizando hardware de bajo costo, como microcontroladores de código abierto, para aplicaciones de investigación.

3.1.1. Arquitectura de las ECU´s

Una ECU típica integra un microcontrolador, memoria para datos y firmware, interfaces de entrada para sensores y salidas para actuadores, como se ilustra en la figura 3.1. La arquitectura varia según su función especifica [16]:

- **ECU del motor:** Controla parámetros como la inyección de combustible, el encendido y el flujo de aire para maximizar la eficiencia y el rendimiento.
- **ECU de confort:** Gestiona sistemas como el climatizado, ventanas eléctricas y asientos ajustables para mejorar la experiencia del usuario.
- **ECU de seguridad:** Administra sistemas críticos como el frenado anti-bloqueo (ABS) y los airbags, garantizando la seguridad de los ocupantes.
- **ECU de comunicación:** Facilita la interconexión entre ECU mediante protocolos como CAN y FlexRay [17].

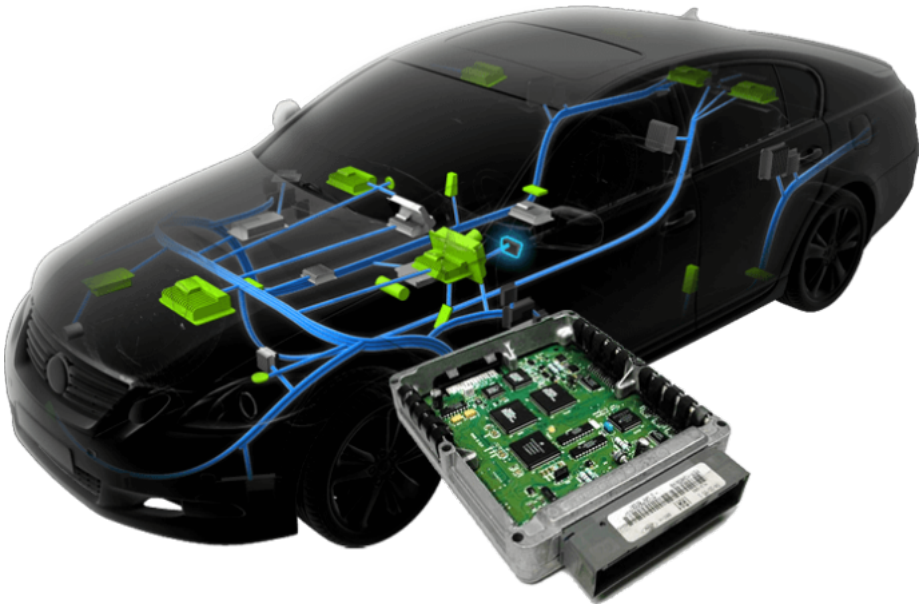


Figura 3.1: Unidad de control electrónico dentro del automóvil.

3.1.2. Funcionalidad y operación

Las ECU procesan señales analógicas y digitales de sensores, convertidas en datos mediante convertidores analógico-digitales (ADC). El firmware ejecuta algoritmos para interpretar estos datos y generar comandos hacia actuadores. Por ejemplo, en una ECU de motor, sensores como el de posición del cigüeñal y el de temperatura del aire de admisión permiten calcular la inyección de combustible, optimizando el rendimiento y reduciendo emisiones [18]. En este trabajo, se implementaran algoritmos de control en plataformas de bajo costo, utilizando sensores comerciales para garantizar la accesibilidad.

3.1.3. Sistemas de diagnostico

Las ECU integran capacidades de auto-diagnostico mediante estándares como OBD-II, que detectan y registran fallos, permitiendo acciones correctivas [19]. Los vehículos modernos también soportan actualizaciones remotas de software, mejorando la mantenibilidad [20]. En este proyecto, se exploraran interfaces OBD-II simuladas utilizando hardware de bajo costo para desarrollar sistemas de diagnostico accesibles.

3.1.4. Rol en la red vehicular

En vehículos modernos, las ECU forman una red interconectada que utiliza protocolos como CAN y Ethernet Automotriz para una comunicación rápida y confiable [21]. Por ejemplo, durante un frenado de emergencia, la ECU de frenos coordina con la ECU de transmisión para desactivar la aceleración, como se muestra en la figura 3.2.

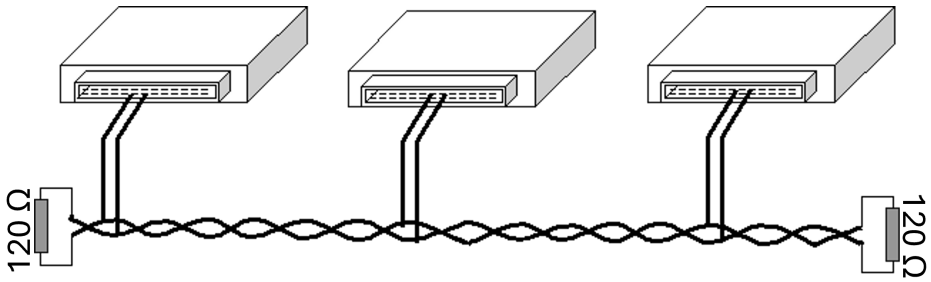


Figura 3.2: Diagrama conceptual de una red vehicular mostrando la interacción entre diferentes ECU.$Fuente: https://slideplayer.es/slide/35272/$

3.1.5. Avances tecnológicos

La evolución hacia vehículos eléctricos y autónomos ha impulsado innovaciones en las ECU [22]:

1. **Consolidación de ECU:** Integración de múltiples funciones en una sola unidad para reducir costos y complejidad.
2. **Procesadores avanzados:** Capaces de ejecutar algoritmos de aprendizaje automático para conducción autónoma.
3. **Seguridad cibernética:** Incorporación de medidas contra ciberataques, un aspecto critico en vehículos conectados [23].

3.1.6. Importancia en el futuro automotriz

Las ECU serán fundamentales en la transición hacia la movilidad eléctrica y la conducción autónoma, integrando datos en tiempo real para optimizar el rendimiento y la seguridad [24]. Este trabajo demuestra que estas funcionalidades pueden implementarse con hardware económico, facilitando la investigación académica.

3.2. Sensores y actuadores automotrices, un enfoque sistémico

Los sensores y actuadores son esenciales para monitorear y controlar sistemas automotrices, utilizando sensores comerciales para reducir costos y facilitar la experimentación [25].

3.2.1. Sensores y actuadores en el sistema de seguridad

Los sistemas de seguridad combinan sensores de impacto, radar, cámaras y LIDAR para detectar riesgos y activar actuadores como airbags y frenos hidráulicos [26]. El

sistema de bolsas de aire, mostrado en la figura 3.3, utiliza actuadores pirotécnicos para un despliegue rápido.

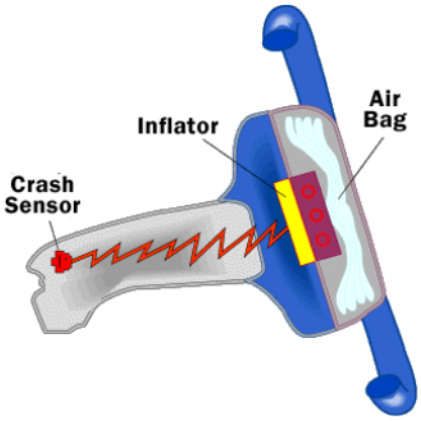


Figura 3.3: Sistema de bolsa de aire. Fuente: [27]

Además, los sistemas avanzados de frenado combinan sensores de radar, cámaras y tecnología LIDAR. Estos dispositivos detectan obstáculos en el camino, lo que permite a los actuadores hidráulicos aplicar presión al sistema de frenos para evitar colisiones. Esto se observa en tecnologías como el frenado automático de emergencia, una característica estándar en muchos vehículos modernos [28].

3.2.2. Sensores y actuadores en el sistema de confort y conveniencia

Los sensores de temperatura y calidad del aire, combinados con actuadores en sistemas de climatización, optimizan el confort del usuario [29], como se muestra en la figura 3.4. Sensores de presión en asientos ajustan la ergonomía mediante actuadores eléctricos [30].

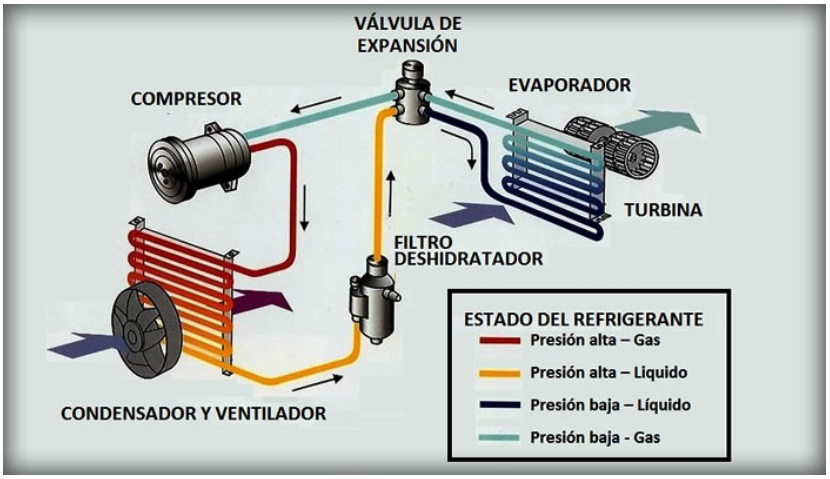


Figura 3.4: Esquema del sistema de climatización. Fuente: [31]

En el ámbito de los asientos, los sensores de presión detectan la presencia y la postura de los ocupantes, mientras que los actuadores eléctricos ajustan la posición del asiento para maximizar la ergonomía y seguridad. Estos sistemas, además de proporcionar confort, contribuyen al correcto funcionamiento de dispositivos como las bolsas de aire laterales [32].

3.2.3. Sensores y actuadores en el tren motriz

Sensores de presión y oxigeno monitorizan el flujo de combustible y la combustión, mientras que actuadores como inyectores de combustible optimizan la eficiencia [33]. En la figura 3.5 se muestra un sistema de inyección.

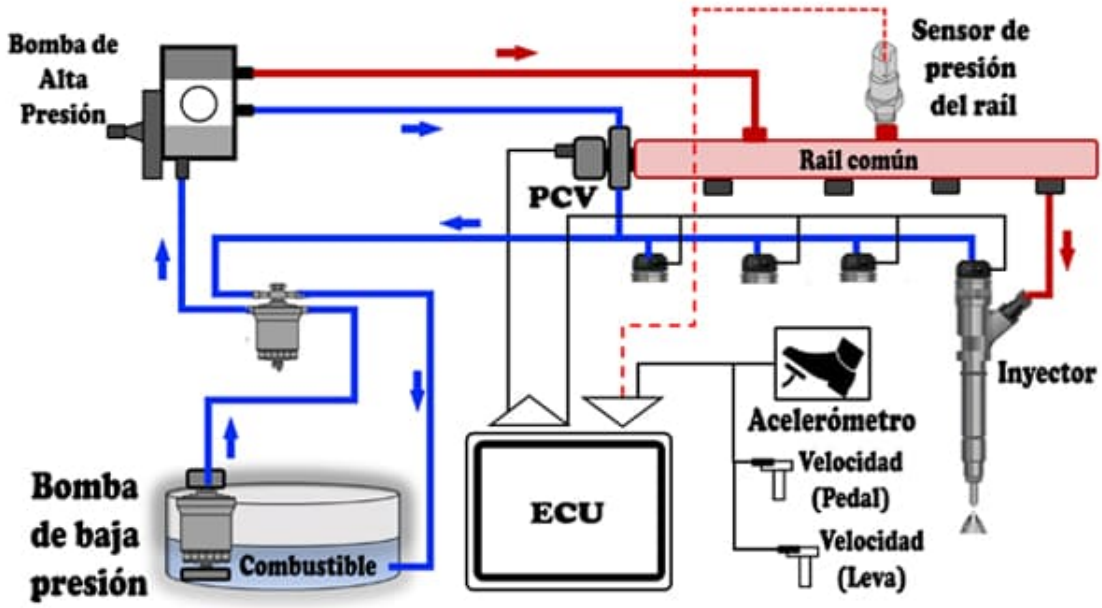


Figura 3.5: Sistema de inyección del automóvil. Fuente: [34]

3.3. Sensores automotrices considerados en el proyecto de Tesis

3.3.1. Sensor ECT

El sensor de temperatura del refrigerante del motor, conocido como ECT (Engine Coolant Temperature), es un componente esencial en los sistemas de gestión electrónica de motores de combustión interna. Su función principal es medir la temperatura del líquido refrigerante que circula por el bloque del motor y enviar esta información a la unidad de control electrónico (ECU). Con base en esta señal, la ECU ajusta parámetros

como la mezcla aire-combustible, el avance de encendido y el régimen de ralentí, con el objetivo de optimizar el rendimiento del motor, reducir emisiones contaminantes y mejorar la eficiencia térmica.

La figura 3.6 muestra un ejemplo típico de sensores ECT utilizados en aplicaciones automotrices.

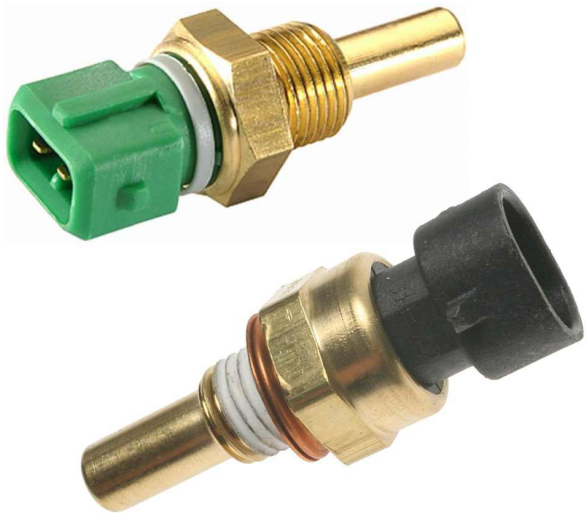


Figura 3.6: Sensor de temperatura ECT. Fuente: [35]

La mayoría de los sensores ECT utilizan termistores de coeficiente de temperatura negativo (NTC), cuya resistencia disminuye exponencialmente a medida que aumenta la temperatura [18]. Esta relación no lineal entre resistencia y temperatura se describe mediante la ecuación de Steinhart-Hart, presentada en la ecuación (3.1), cuya curva característica se muestra en la figura 3.7:

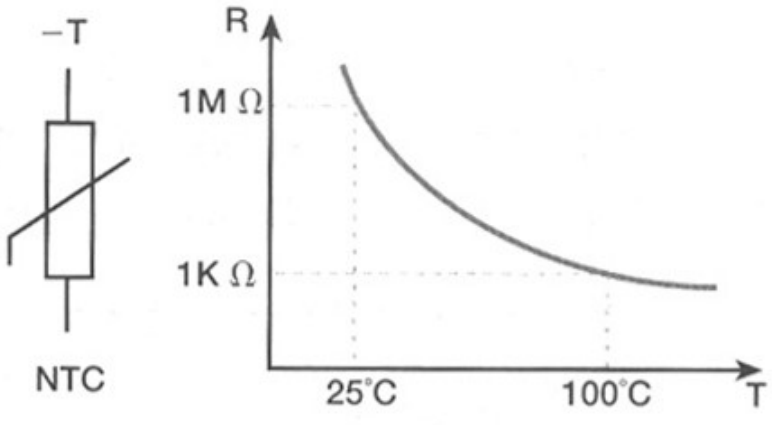


Figura 3.7: Símbolo y curva característica de un sensor NTC. Donde R es resistencia y T es temperatura. Fuente: [36]

$$\frac{1}{T} = A + B \cdot \ln(R) + C \cdot (\ln(R))^3 \tag{3.1}$$

Donde:

- T: Temperatura en Kelvin.
- R: Resistencia del termistor en ohmios.
- A, B, C: Coeficientes de calibración específicos del sensor.

¿Por qué se utilizan tres coeficientes?

La ecuación de Steinhart-Hart utiliza tres coeficientes (A, B y C) porque permite ajustar con alta precisión la curva de comportamiento de un termistor NTC en un amplio rango de temperaturas. Esta cantidad de coeficientes representa un equilibrio entre exactitud y simplicidad computacional. Con tres puntos de calibración, se puede interpolar de forma precisa sin necesidad de modelos más complejos o con más parámetros, lo que facilita su implementación en sistemas embebidos y aplicaciones automotrices.

Tipos de datos válidos para calibrar la ecuación

Para calcular los coeficientes de la ecuación de Steinhart-Hart (3.1), se utilizan tres pares de datos experimentales de resistencia y temperatura: (R_1, T_1), (R_2, T_2), (R_3, T_3). Los valores de resistencia y temperatura pueden ser de distintos tipos, siempre que se respeten las unidades y se mantenga la coherencia numérica:

- **Enteros**: Son comunes en tablas de fabricantes y permiten cálculos manuales más simples. Ejemplo: $R = 1000\ \Omega$, $T = 300\ K$.
- **Decimales**: Se obtienen de mediciones más precisas o interpolaciones. Ejemplo: $R = 1234.56\ \Omega$, $T = 298.15\ K$.
- **Fraccionarios**: Aunque menos frecuentes, pueden surgir en cálculos teóricos o simulaciones. Ejemplo: $R = \frac{5000}{3}\ \Omega$.

Es fundamental que:

- La temperatura esté expresada en Kelvin (K), no en grados Celsius.
- La resistencia esté expresada en ohmios (Ω).
- Los tres puntos estén distribuidos en el rango operativo del sensor.

Selección de puntos para calibración

La elección de los tres pares de datos (R_1, T_1), (R_2, T_2), (R_3, T_3) para calcular los coeficientes A, B y C de la ecuación (3.1) debe considerar restricciones específicas para garantizar la precisión y estabilidad en el cálculo de los coeficientes. Estas restricciones son cruciales porque la ecuación de Steinhart-Hart es altamente sensible a la distribución de los puntos debido a su naturaleza no lineal. Un mal posicionamiento de los puntos puede llevar a errores significativos en la interpolación, como sobreajustes en regiones locales o desviaciones en temperaturas extremas, lo que compromete la fiabilidad del modelo en aplicaciones reales como el monitoreo del ECT.

- **Cobertura amplia del rango térmico**: Los puntos deben representar temperaturas bajas, medias y altas dentro del rango operativo del sensor (por ejemplo, -40°C a 150°C para aplicaciones automotrices). Esto asegura que el modelo capture la curvatura no lineal completa del termistor NTC, evitando extrapolaciones inexactas fuera de los puntos calibrados. Si los puntos están demasiado cercanos, el coeficiente C (que maneja el término cúbico) puede volverse inestable, resultando en oscilaciones no físicas en la curva ajustada.
- **Precisión en la medición**: Las lecturas de resistencia y temperatura deben ser confiables y obtenidas con instrumentos calibrados, con errores mínimos (por ejemplo, ±0.1 % en resistencia y ±0.1 K en temperatura). Errores en las mediciones se propagan amplificados a través del logaritmo y el término cúbico en la ecuación (3.1), lo que puede distorsionar los coeficientes B y C. En la práctica, se recomienda usar múltiples mediciones promediadas en cada punto para reducir el ruido y mejorar la robustez del sistema de ecuaciones resultante.
- **Evitar valores extremos**: Para minimizar errores por saturación o comportamiento no lineal fuera del rango útil del termistor, los puntos no deben estar en los límites absolutos donde el sensor podría exhibir comportamientos anómalos (como resistencias infinitas en temperaturas muy bajas o cero en altas). En su lugar, seleccione puntos dentro del 10-90 % del rango operativo para asegurar que la matriz del sistema de ecuaciones sea bien condicionada y evite singularidades numéricas durante el cálculo de los coeficientes, como divisiones por valores cercanos a cero en las fórmulas de despeje.

Formulación del sistema de ecuaciones

Sustituyendo los tres pares de datos en la ecuación de Steinhart-Hart (3.1) se obtiene el siguiente sistema:

$$\frac{1}{T_1} = A + B \cdot \ln(R_1) + C \cdot (\ln(R_1))^3 \tag{3.2}$$
$$\frac{1}{T_2} = A + B \cdot \ln(R_2) + C \cdot (\ln(R_2))^3 \tag{3.3}$$
$$\frac{1}{T_3} = A + B \cdot \ln(R_3) + C \cdot (\ln(R_3))^3 \tag{3.4}$$

Para simplificar el sistema (3.2)-(3.4), se definen las siguientes variables:

$$\begin{aligned} L_1 &= \ln(R_1), \quad L_2 = \ln(R_2), \quad L_3 = \ln(R_3) \\ Y_1 &= \frac{1}{T_1}, \quad Y_2 = \frac{1}{T_2}, \quad Y_3 = \frac{1}{T_3} \end{aligned} \tag{3.5}$$

Reescribiendo el sistema (3.2)-(3.4) con estas variables (3.5):

$$Y_1 = A + B \cdot L_1 + C \cdot L_1^3 \tag{3.6}$$
$$Y_2 = A + B \cdot L_2 + C \cdot L_2^3 \tag{3.7}$$
$$Y_3 = A + B \cdot L_3 + C \cdot L_3^3 \tag{3.8}$$

Despeje explícito de los coeficientes

Para resolver el sistema (3.6)-(3.8), se eliminan términos comunes mediante restas entre ecuaciones:

$$Y_2 - Y_1 = B(L_2 - L_1) + C(L_2^3 - L_1^3) \tag{3.9}$$
$$Y_3 - Y_1 = B(L_3 - L_1) + C(L_3^3 - L_1^3) \tag{3.10}$$

Definimos las diferencias a partir de las ecuaciones (3.9) y (3.10):

$$\begin{aligned} \Delta Y_{21} &= Y_2 - Y_1, \quad \Delta Y_{31} = Y_3 - Y_1 \\ \Delta L_{21} &= L_2 - L_1, \quad \Delta L_{31} = L_3 - L_1 \\ \Delta L_{21}^3 &= L_2^3 - L_1^3, \quad \Delta L_{31}^3 = L_3^3 - L_1^3 \end{aligned} \tag{3.11}$$

El sistema reducido queda:

$$\Delta Y_{21} = B \cdot \Delta L_{21} + C \cdot \Delta L_{21}^3 \tag{3.12}$$
$$\Delta Y_{31} = B \cdot \Delta L_{31} + C \cdot \Delta L_{31}^3 \tag{3.13}$$

Despejamos los coeficientes a partir del sistema reducido (3.12)-(3.13):

$$C = \frac{\Delta Y_{31} \cdot \Delta L_{21} - \Delta Y_{21} \cdot \Delta L_{31}}{\Delta L_{31}^3 \cdot \Delta L_{21} - \Delta L_{21}^3 \cdot \Delta L_{31}} \tag{3.14}$$

$$B = \frac{\Delta Y_{21} - C \cdot \Delta L_{21}^3}{\Delta L_{21}} \tag{3.15}$$

$$A = Y_1 - B \cdot L_1 - C \cdot L_1^3 \tag{3.16}$$

Aplicación práctica

Una vez determinados los coeficientes A, B y C mediante las ecuaciones (3.14), (3.15) y (3.16), se puede utilizar la ecuación de Steinhart-Hart (3.1) para convertir cualquier valor de resistencia medido en una temperatura precisa. Este proceso de calibración es fundamental para garantizar la fiabilidad y exactitud del sensor ECT en aplicaciones automotrices, especialmente en condiciones de operación variables.

3.3.2. Sensor APP

El sensor de posición del pedal de aceleración (APP) es un componente clave en los vehículos modernos, especialmente aquellos con sistemas de aceleración electrónica (*drive-by-wire*). Este sensor detecta la posición del pedal de aceleración y envía esta información al módulo de control del motor, que ajusta la cantidad de combustible suministrada al motor y otros parámetros para controlar la velocidad del vehículo [37]. La figura 3.8 muestra una representación de dicho sensor.

Figura 3.8: Sensor de posición del pedal de aceleración (APP). Fuente: [38]

El funcionamiento del sensor APP se basa en el uso de potenciómetros lineales. Un potenciómetro lineal es un resistor variable cuya resistencia cambia linealmente con la posición de un deslizador. En el sensor APP, el pedal de aceleración está conectado a este deslizador, de modo que al presionar el pedal, el deslizador se mueve a lo largo del potenciómetro, modificando la resistencia [39].

El sensor APP suele incorporar dos potenciómetros lineales en paralelo, proporcionando redundancia para mejorar la seguridad, como se ilustra en la figura 3.9. Estos potenciómetros están conectados a una fuente de voltaje y a tierra, y la posición del deslizador determina la salida de voltaje, que es interpretada por el módulo de control del motor para determinar la posición del pedal.

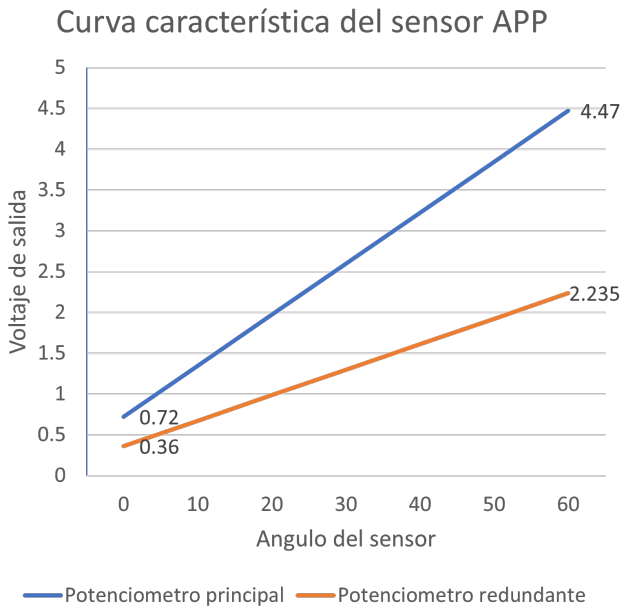


Figura 3.9: Curva característica del sensor APP.

El comportamiento del sensor APP se describe mediante las siguientes ecuaciones. La ecuación (3.17) modela la tensión de salida del sensor en función de la posición del deslizador:

$$V_{\text{out}} = V_{\text{in}} \times \frac{R_{\text{slider}}}{R_{\text{total}}} \tag{3.17}$$

donde:

- V_{out}: Tensión de salida del sensor (voltios).
- V_{in}: Tensión de entrada al sensor (voltios).
- R_{slider}: Resistencia entre el deslizador y el terminal de tierra del potenciómetro (ohmios).
- R_{total}: Resistencia total del potenciómetro (ohmios).

La ecuación (3.18) relaciona la posición del pedal con la resistencia del potenciómetro:

$$\text{Posición} = \frac{R_{\text{slider}}}{R_{\text{total}}} \times \text{Recorrido total del pedal} \tag{3.18}$$

donde:

- Posición: Posición del pedal de aceleración (porcentaje, donde 0 % es pedal sin presionar y 100 % es pedal totalmente presionado).
- Recorrido total del pedal: Distancia total que recorre el pedal desde la posición inicial hasta la máxima (en milímetros).

Las ecuaciones (3.17) y (3.18) ilustran cómo el sensor APP convierte la posición física del pedal en una señal eléctrica interpretable por el módulo de control del motor, permitiendo un ajuste preciso de la velocidad del vehículo.

3.3.3. Sensor CKP

El sensor de velocidad del cigüeñal es un componente esencial para monitorear la rotación del cigüeñal en motores de combustión interna. Su función principal es enviar señales al módulo de control del motor (ECM) para determinar la velocidad de rotación y la posición del cigüeñal, lo cual es crucial para la sincronización del encendido y la inyección de combustible. Este sensor suele basarse en el efecto Hall o en principios inductivos. En los sensores de efecto Hall, se emplean imanes para generar señales eléctricas proporcionales a las variaciones del campo magnético producidas por los dientes de la rueda fónica o el volante del cigüeñal [40].

El principio del efecto Hall, descubierto por Edwin Hall en 1879, permite a estos sensores medir la intensidad del campo magnético. Cuando un conductor está sometido a un campo magnético perpendicular a la corriente que lo atraviesa, se genera un voltaje transversal, conocido como voltaje Hall, proporcional a la intensidad del campo magnético y a la corriente. Esta señal se procesa para obtener información precisa sobre la posición y la velocidad de rotación del cigüeñal [41]. La ecuación que describe el voltaje Hall se presenta en la ecuación (3.19):

$$V_H = K \cdot B \cdot I \cdot d \tag{3.19}$$

Donde:

- V_H: Voltaje Hall generado en el conductor (en voltios).
- K: Constante de Hall, dependiente de las propiedades del material conductor.
- B: Densidad del campo magnético (en teslas).
- I: Corriente que atraviesa el conductor (en amperios).
- d: Espesor del conductor en la dirección perpendicular al campo magnético (en metros).

Los sensores de efecto Hall en vehículos generan señales de salida que permiten al ECM calcular la velocidad de rotación del cigüeñal mediante el conteo de variaciones en la señal producidas por los dientes de la rueda fónica, como se ilustra en la figura 3.10.

Figura 3.10: Sensor del cigüeñal basado en el efecto Hall. Fuente: [42]

El sensor de velocidad del cigüeñal basado en el efecto Hall está diseñado para generar una señal cuando los dientes de la rueda fónica interfieren con el campo magnético del sensor. Esta señal, típicamente una onda cuadrada, es procesada por el ECM para calcular las revoluciones por minuto (RPM) del motor y determinar la posición exacta del cigüeñal, esencial para el control preciso del motor [43]. La señal generada se convierte en una señal digital que permite medir la velocidad de rotación mediante técnicas de conteo de pulsos y cálculo del tiempo entre pulsos, como se muestra en la figura 3.11.

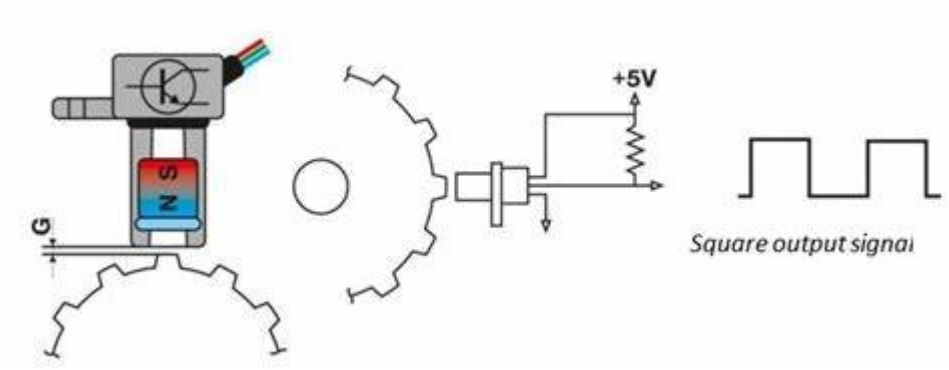


Figura 3.11: Señal cuadrada generada por el sensor del cigüeñal. Fuente: [42]

3.3.4. Sensor de nivel de gasolina

El sensor de nivel de gasolina en los vehículos es esencial para monitorear la cantidad de combustible en el tanque. Uno de los tipos más comunes de este sensor es el **sensor de resistencia variable**, que utiliza una boya flotante conectada a un potenciómetro. A medida que el nivel de combustible cambia, la boya se mueve hacia arriba o hacia abajo, lo que modifica la resistencia del potenciómetro y, en consecuencia, la

señal eléctrica enviada al sistema de medición del vehículo. Esta variación de resistencia se traduce en la lectura del nivel de combustible que se muestra en el tablero del vehículo [44].

El principio de funcionamiento del sensor de resistencia variable se basa en la ley de Ohm. Cuando la boya está en la parte superior (tanque lleno), la resistencia es baja, lo que genera una tensión más baja. Por otro lado, cuando el tanque está vacío, la boya se encuentra en la parte inferior, lo que aumenta la resistencia y genera una mayor tensión. Esta variación de la tensión se convierte en una señal que indica el nivel de combustible [45].

La resistencia total del circuito se describe mediante la ecuación (3.20):

$$R_{\text{total}} = R_{\text{pot}} + R_{\text{fija}} \tag{3.20}$$

Donde:

- R_{total} es la resistencia total del circuito.
- R_{pot} es la resistencia variable del potenciómetro, que cambia con la posición de la boya.
- R_{fija} es la resistencia fija en el circuito.

Cuando la boya está en la parte superior (tanque lleno), la resistencia del potenciómetro es mínima, resultando en una señal de salida baja. Cuando la boya está en la parte inferior (tanque vacío), la resistencia es máxima, generando una señal más alta. Esta variación de voltaje se procesa para mostrar el nivel de combustible en el tablero del vehículo [46].

Figura 3.12: Sensor de nivel de gasolina de Honda Fit. Fuente: [44].

El cálculo del nivel de combustible que se muestra en el cuadro de instrumentos del vehículo se obtiene a partir de la señal de voltaje proporcionada por el sensor. La ecuación (3.21) describe cómo convertir esta señal en un porcentaje que representa el nivel de combustible:

$$N_{\text{combustible}} = \frac{V_{\text{sensor}} - V_{\text{min}}}{V_{\text{max}} - V_{\text{min}}} \times 100 \tag{3.21}$$

Donde:

- $N_{\text{combustible}}$ es el nivel de combustible mostrado en el cuadro de instrumentos (en porcentaje, donde 0 % es tanque vacío y 100 % es tanque lleno).
- V_{sensor} es el voltaje medido por el sensor de nivel de combustible.
- V_{min} es el voltaje correspondiente al nivel mínimo del sensor (cuando el tanque está vacío).
- V_{max} es el voltaje correspondiente al nivel máximo del sensor (cuando el tanque está lleno).

Este proceso convierte la señal del sensor en una lectura de porcentaje que el conductor puede ver en el indicador de combustible en el tablero del vehículo [46].

3.4. Actuadores automotrices considerados en el protecto de Tesis

Los actuadores automotrices son componentes clave en los sistemas de control de vehículos, ya que realizan funciones mecánicas a partir de señales eléctricas. Estos actuadores se encuentran en aplicaciones como la dirección asistida, el control de frenos, la suspensión, y los sistemas de transmisión, entre otros. Son esenciales para la automatización y el confort en los vehículos modernos [47].

En esta tesis, no se utilizarán actuadores reales debido a limitaciones de hardware. En su lugar, se buscarán opciones para emular el comportamiento de estos actuadores, permitiendo realizar pruebas de control sin la necesidad de componentes costosos o difíciles de integrar.

3.4.1. Motor de Corriente Continua (DC) RS550

El motor **RS550**(figura 3.13) es un motor de corriente continua (DC) de 12V, comúnmente utilizado en aplicaciones que requieren alta velocidad y eficiencia en un tamaño compacto. Este motor se emplea en vehículos pequeños como coches de juguete y motocicletas eléctricas para niños, así como en robots pequeños debido a su capacidad para alcanzar altas revoluciones por minuto (RPM) [48].

Figura 3.13: Motor RS550. Fuente: [48]

Características Generales de los Motores DC

Los motores de corriente continua (DC) son ampliamente utilizados debido a su simplicidad y eficiencia. Estos motores convierten la energía eléctrica en energía mecánica mediante la interacción de un rotor y un campo magnético. Las características principales de los motores DC incluyen:

- **Control sencillo de velocidad:** La velocidad de rotación se puede ajustar fácilmente variando el voltaje aplicado.
- **Alta eficiencia:** Los motores DC son eficientes en aplicaciones que no requieren grandes cargas.
- **Versatilidad:** Son adecuados para una amplia gama de aplicaciones, desde robots hasta vehículos eléctricos.
- **Bajo costo de fabricación y mantenimiento:** Lo que los hace ideales para aplicaciones de bajo costo.

Especificaciones Técnicas del Motor RS550

El motor RS550 tiene las siguientes especificaciones técnicas:

- **Voltaje de operación:** 12V
- **Potencia de salida:** 35W - 55W
- **Velocidad de rotación:** 550 a 18,000 RPM
- **Dimensiones:** Longitud de 8 cm, diámetro de 3.7 cm

- **Peso:** 220 g
- **Eje del motor:** 0.3 cm

Este motor es ideal para aplicaciones que requieren alta velocidad y eficiencia en un espacio compacto [49].

Aplicaciones del Motor RS550

El motor RS550 es adecuado para vehículos de juguete eléctricos, robots pequeños, y vehículos eléctricos para niños. Su alta velocidad y eficiencia lo hacen perfecto para sistemas donde el espacio es limitado, pero la velocidad es crucial. Esta propiedad también lo convierte en una opción económica para simular actuadores de control en vehículos automatizados sin necesidad de hardware costoso. económica para simular actuadores de control en vehículos automatizados sin necesidad de hardware costoso.

Capítulo 4

Integración de Software y Hardware de código abierto para la Implementación de una Red Vehicular

Este capítulo detalla el desarrollo de una red vehicular utilizando hardware de código abierto, enfocándose en la integración de componentes de software y hardware para crear un cuadro de instrumentos virtual funcional. Se aborda el diseño de interfaces gráficas de usuario (GUIs) en MATLAB y Python, la implementación de una red CAN de cuatro nodos con sensores y actuadores, la generación de un instalador ejecutable para la GUI, y la creación de manuales para facilitar su uso y aprendizaje.

4.1. Interfaces gráficas en MATLAB Y Python

Existen varias herramientas disponibles para desarrollar interfaces gráficas capaces de interactuar con implementaciones físicas. Este proyecto explora App Designer de MATLAB (disponible en versiones posteriores a 2016b, reemplazando la herramienta GUIDE) y las bibliotecas Tkinter y PySide de Python. Estas opciones se discuten en detalle a continuación, con un enfoque en su aplicabilidad a interfaces de redes vehiculares.

4.1.1. APP Designer

La herramienta GUIDE (Graphical User Interface Development Enviromente) fue introducida en el año 2004 en la versión MATLAB 7, el cual desde entonces se volvió una excelente opción para diseñar y construir interfaces interactivas; cabe mencionar que dicha herramienta sera eliminada en nuevas versiones; por lo cual, la herramienta que se encuentra disponible para las nuevas versiones es App Designer. Las dos herramientas cuentan con varias similitudes ya que las dos cuentan con una ventana donde po-

demos colocar los elementos que se encuentran ya pre-diseñados y están disponibles para su uso. En la figura 4.1 se muestra el entorno de diseño de la herramienta GUIDE y en la figura 4.2 la de la herramienta App Designer.

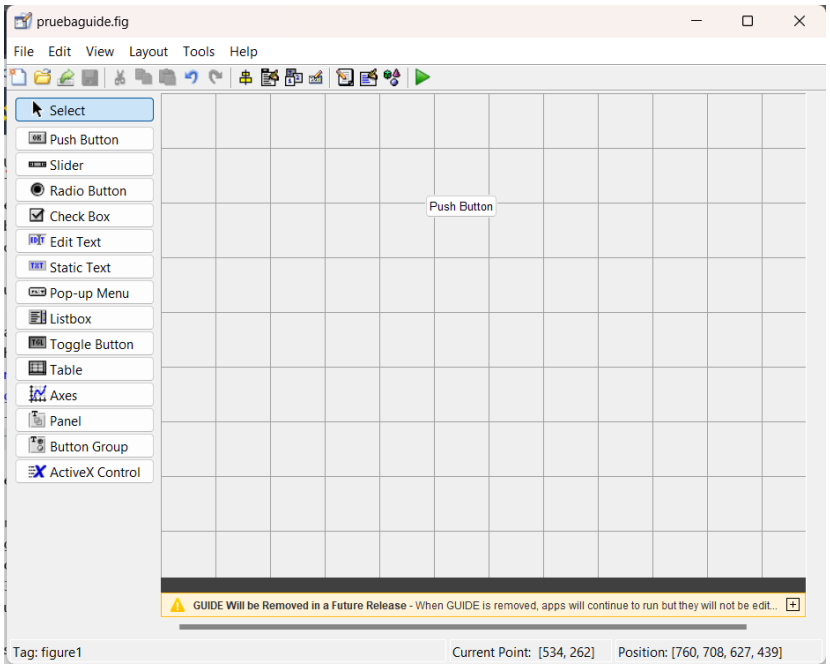


Figura 4.1: Entorno de diseño GUIDE

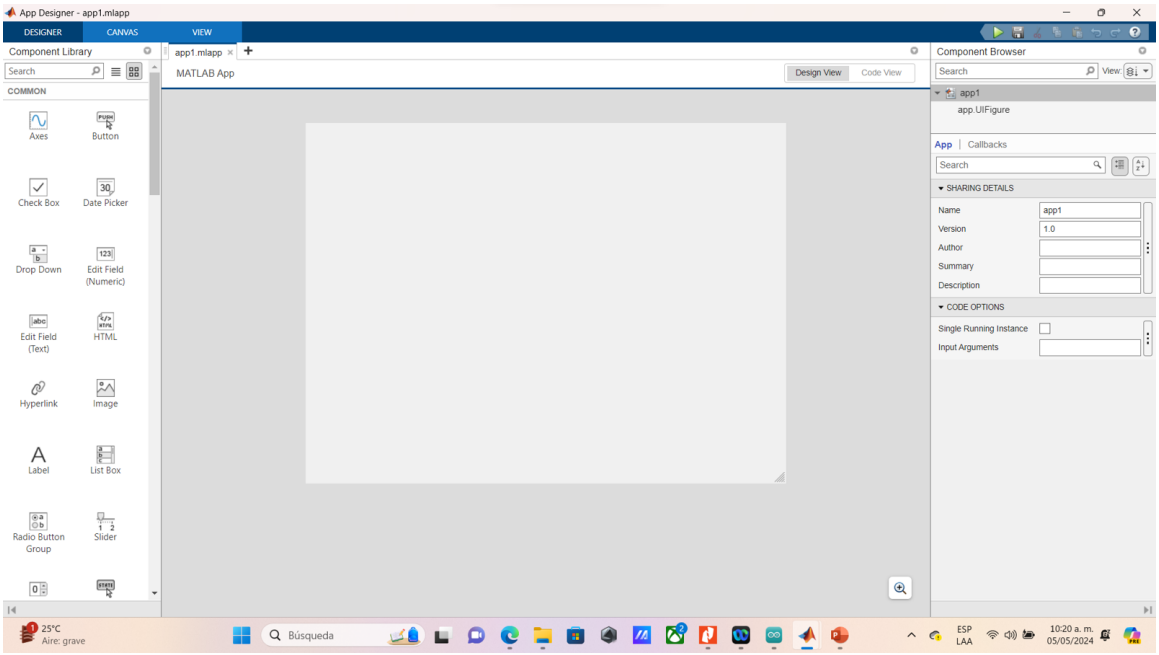
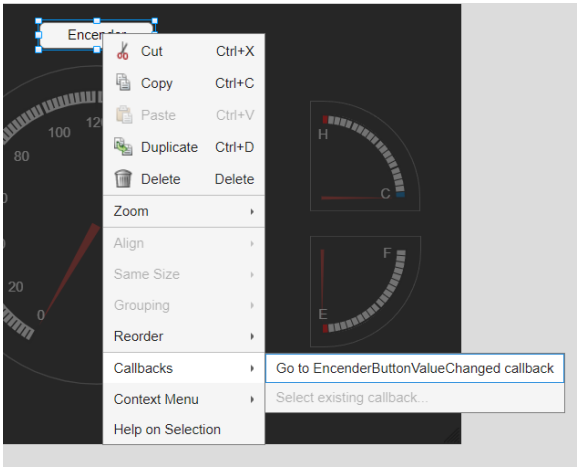


Figura 4.2: Entorno de diseño App Designer

Como se observa en las figuras antes mencionadas, las dos herramientas cuentan con similitudes ya que las dos cuentan con una ventana de diseño; una de las diferencias significativas es que la programación de los elementos es diferente. Cuando utilizamos la herramienta de GUIDE para desarrollar nuestra interfaz gráfica la programación asociada a nuestros objetos se realiza en un archivo *.m*, el cual se encuentra vinculado a nuestra interfaz gráfica la cual genera un archivo *.fig* el cual contiene los elementos gráficos de nuestro programa a diseñar.

Por otro lado la herramienta App Designer genera un único archivo el cual, esta representado por la extensión *.mlapp*. Esto se debe, a que dentro de la misma herramienta contamos con la venta de diseño; la cual, contiene los elementos gráficos y de igual manera, contamos con una venta de código en donde se tendrá contenida toda la programación; cabe mencionar, que una de las ventajas que tiene esta herramienta es que la programación que contiene el programa principal no se puede modificar, evitando así la eliminación del programa por algún descuido.

Para poder programar los objetos que se encuentran disponibles se tiene que generar un callback, el cual contendrá la programación de dicho elemento, el cual puede ser un botón, un slider, etc. Para poder generar un callback hay que hacer click derecho en el objeto a programas y posteriormente seleccionar el callback para poder programar; el cual, se crea en la ventana de código como se observa en la figuras 4.3a y 4.3b.



(a) Generación del callback del motor.

```
function EncenderButtonValueChanged(app, event)
    estado = app.EncenderButton.Value;
    if estado == true
        app.EncenderButton.Text = 'Apagar';
        app.Velocimetro.Visible = "on";
        app.Acelerometro.Visible = "on";
        app.NivelG.Visible ="on";
        app.Temperatura.Visible = "on";
        delete(instrfind({'Port'},{'COM11'}));
        device=serial('COM11','BaudRate',9600,'Terminator','CR/LF');
        fopen(device);
        i=1;
        while(1)
            dato=fscanf(device,'%f,%f\r\n');
            velocidad=dato(1);
            nivel=dato(2);
            app.Velocimetro.Value=velocidad;
            app.NivelG.Value=nivel;
            rpm = (velocidad*1000)/3.1416*0.4064;
        if rpm <7000
            app.Acelerometro.Value=rpm;
        elseif rpm<14000
            app.Acelerometro.Value=rpm-7000;
        elseif rpm<21000
            app.Acelerometro.Value=rpm-14000;
        elseif rpm<28000
            app.Acelerometro.Value=rpm-21000;
        else
```

(b) Programación para el botón.

Figura 4.3: Como programar y generar un callback en App Designer.

Entre los elementos que podemos utilizar dentro de la herramienta de App Designer se encuentran botones, deslizadores, válvulas, cuadros de texto, entre otros. Dichos elementos se pueden observar en la figura 4.4.

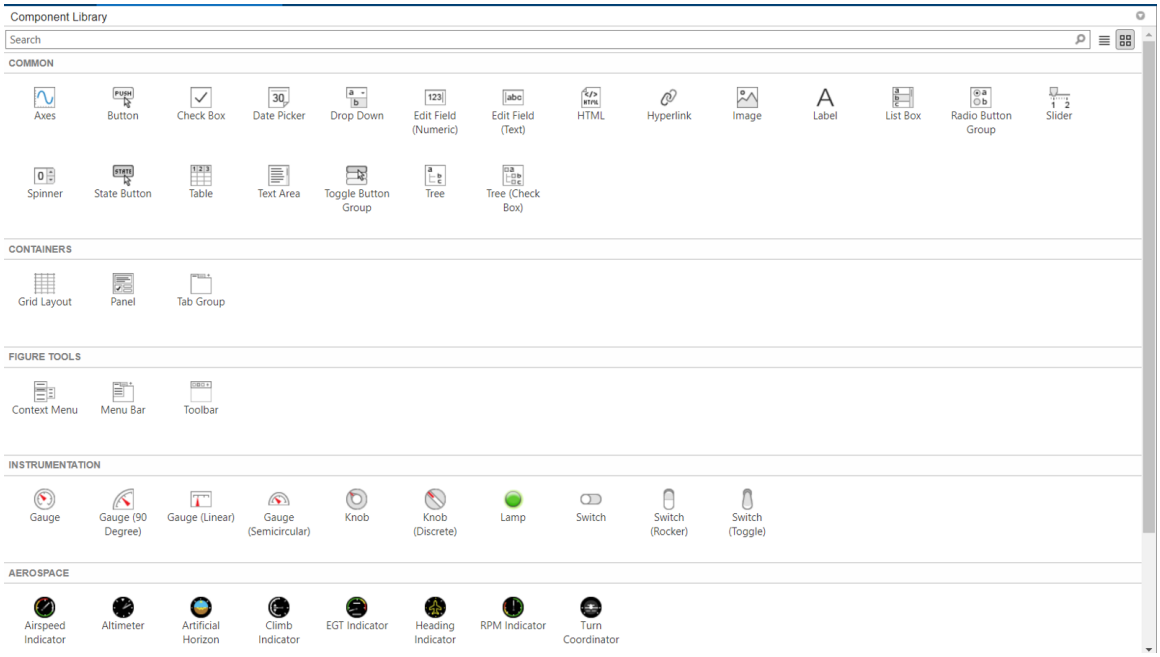
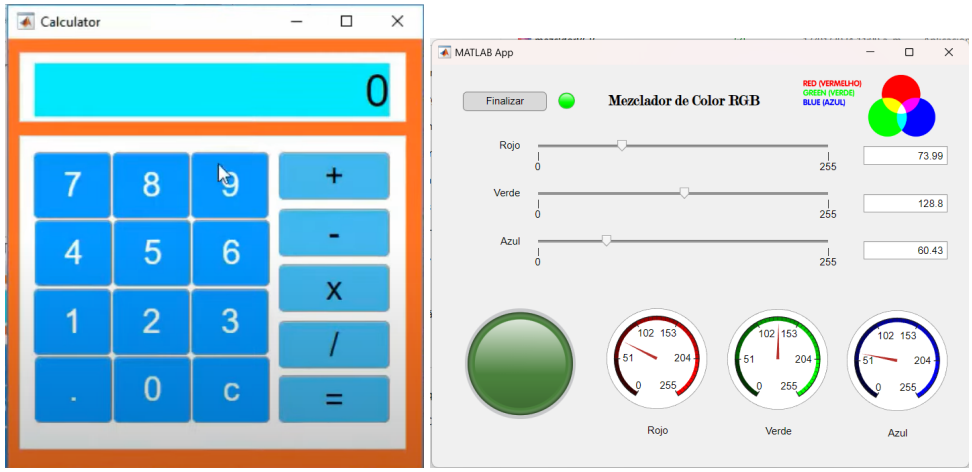


Figura 4.4: Biblioteca de componentes de App Designer.

Todos estos elementos nos ayudan a generar diferentes interfaces gráficas dependiendo del objetivo que se busca que cumplan; por ejemplo, podemos generar una calculadora o un mezclador de color. A continuación en la figuras 4.5a y 4.5b se muestran algunos ejemplos de interfaces gráficas diseñadas en App Designer y mas adelante mostraremos la interfaz gráfica del cuadro de instrumentos la cual ocuparemos para nuestra implementación.



(a) Calculadora. (b) Mezclador RGB.

Figura 4.5: Interfaces realizadas en App Designer.

4.1.2. Tkinter y Qt

Tkinter es la librería estándar para la creación de interfaces gráficas de usuario, la cual ya forma parte del entorno de programación de Python.Dicha librería esta basada en el lenguaje de programación Tcl, el cual incluye un marco de trabajo (framework) conocido como Tk. Este framework nos proporciona una biblioteca de controles gráficos(widgets) para el desarrollo de las interfaces [50].

El funcionamiento básico de Tkinter implica la creación de una ventana principal (o root window") utilizando la clase Tk(). Esta ventana actúa como el contenedor principal para todos los elementos de la GUI. A partir de ahí, se pueden agregar widgets (como botones, etiquetas, cuadros de texto, etc.) a la ventana principal como se puede observar en la figura 4.6 Para poder organizar los widgets dentro de la ventana principal tkinter ocupa un sistema de geometría llamado packing. El método pack() se utiliza para agregar un widget a la ventana y especificar cómo debe colocarse en relación con otros widgets.

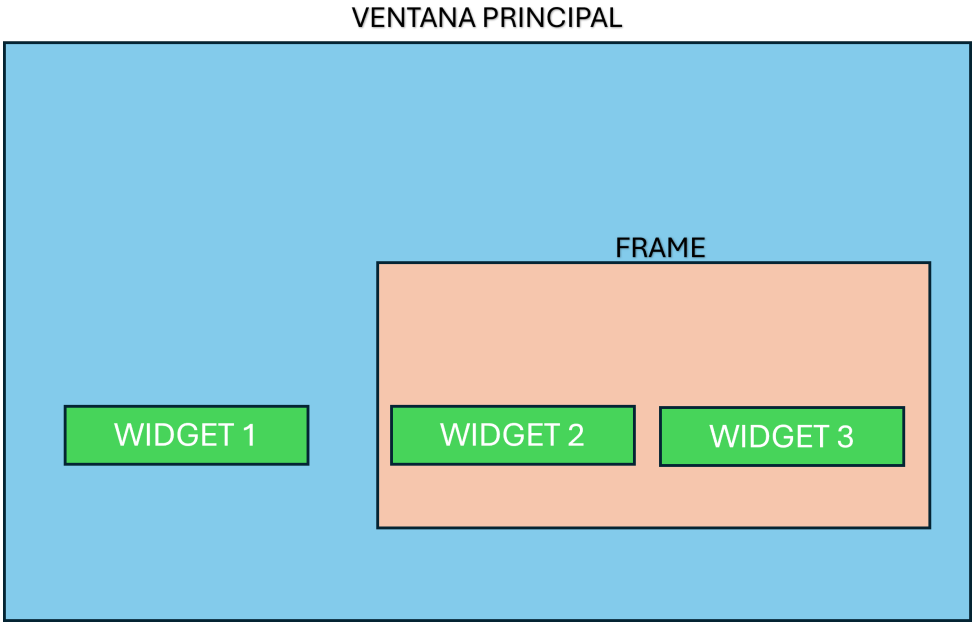
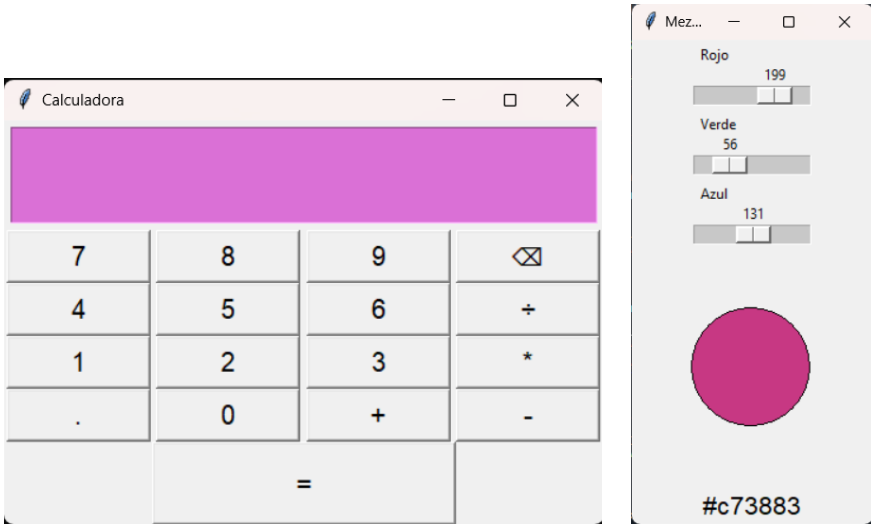


Figura 4.6: Forma esquemática de la Interfaz en tkinter.

Además de pack(), Tkinter también ofrece otros métodos de geometría, como grid() y place(), que permiten organizar los widgets de manera más precisa dentro de una ventana. También ofrece la capacidad de asociar funciones (llamadas çallbacks") con eventos de usuario, como hacer clic en un botón. Esto permite que la GUI responda a las acciones del usuario de manera interactiva. En la figura 4.7 (a) y (b) se muestran algunos ejemplos de interfaces realizadas en tkinter.



(a) Calculadora. (b) Mezclador RGB.

Figura 4.7: Interfaces en Tkinter.

Por otro lado; tenemos PyQT y Pyside, las cuales son dos bibliotecas que al igual que tkinter nos permiten diseñar GUI. Las dos bibliotecas antes mencionadas estan basadas en Qt de C++. Al igual que tkinter, estados herramientas ofrecen una gran variedad de widgets para desarrollar las GUI.

PyQt es desarrollado por Riverbank Computing y proporciona enlaces Python para la biblioteca Qt [51]. Se considera una biblioteca madura y completa, con una amplia gama de funcionalidades y una documentación detallada. PyQt está disponible en dos versiones principales: PyQt4 (para Qt4) y PyQt5 (para Qt5).

Pyside es otro conjunto de enlaces Python para Qt, desarrollado por The Qt Company y la comunidad Qt. Al igual que PyQt, PySide proporciona acceso a la funcionalidad de Qt desde Python. PySide tiene dos versiones principales: PySide para Qt4 y PySide2 para Qt5. Para utilizar PyQt o PySide, necesitamos instalar estas bibliotecas, ya que no se encuentran disponibles en el entorno estándar de Python.

Una de las ventajas que tiene Qt a diferencia de Tcl, es que Qt nos proporciona un entorno de diseño llamado Qt creator el cual se puede observar en la figura 4.8. En el cual podemos colocar elementos ya prediseñados, los cuales podemos programar; de igual manera podemos colocar contenedores en los que nosotros podemos crear nuestros propios componentes en base a código. Qt creator nos genera un archivo *.ui* el cual que posteriormente tendremos que convertir en un archivo *.py* para poder programar la interacción de los elementos con el usuario.

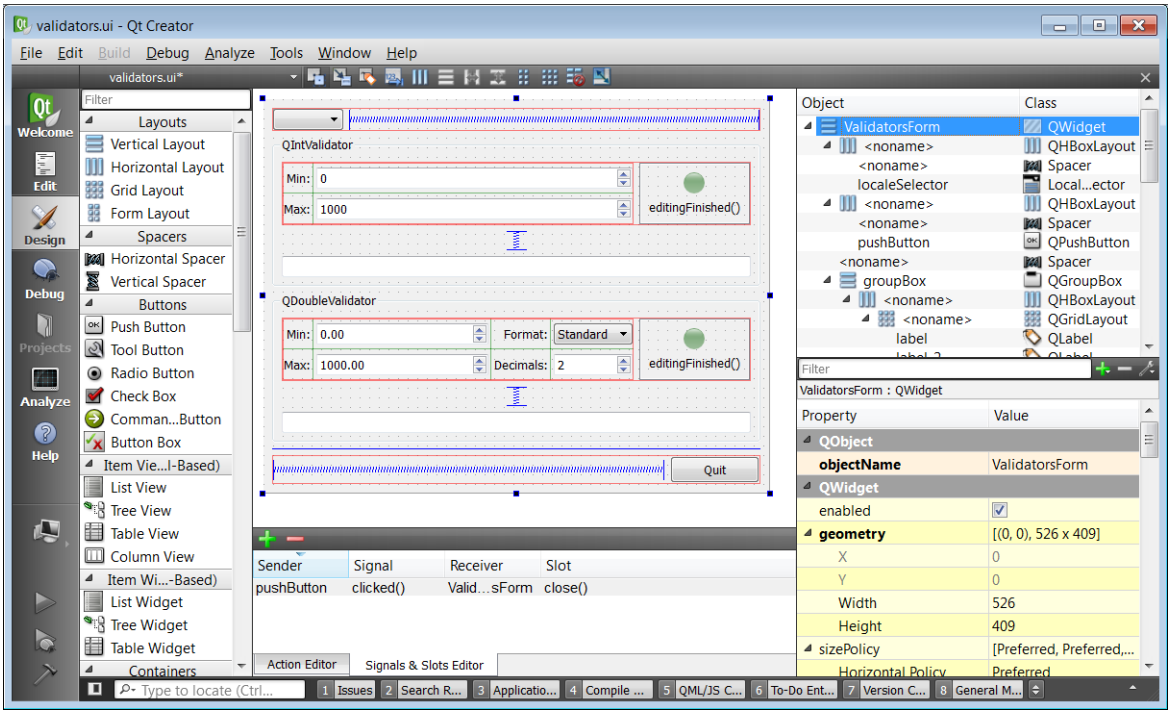


Figura 4.8: Interfaz de QT Creator

4.1.3. App Designer vs Tkinter

A continuación, se presenta una comparativa entre **App Designer** y **Tkinter** en el contexto del desarrollo de un **cuadro de instrumentos automotriz** que se comunicará con una **ESP32** mediante un puerto serial.

- **Facilidad de uso:**
  - **App Designer:** Alta, especialmente para usuarios de MATLAB.
  - **Tkinter:** Moderada, requiere más programación manual.
- **Costo:**
  - **App Designer:** Requiere una licencia de MATLAB.
  - **Tkinter:** Gratuito, incluido con Python.
- **Flexibilidad:**
  - **App Designer:** Menor, más limitado a MATLAB y sus funciones.
  - **Tkinter:** Alta, totalmente flexible y personalizable.
- **Interacción con hardware:**
  - **App Designer:** Limitada, requiere integración externa.

    - **Tkinter:** Muy eficiente, fácil integración con ESP32 usando `pyserial`.

- **Escalabilidad:**
    - **App Designer:** Alta, ideal para proyectos complejos de visualización y datos.
    - **Tkinter:** Moderada, adecuada para proyectos más simples.

- **Compatibilidad:**
    - **App Designer:** Requiere MATLAB y entorno específico.
    - **Tkinter:** Compatible con cualquier entorno Python y librerías estándar.

En conclusión, para el desarrollo de un **cuadro de instrumentos automotriz** que se comunique con una **ESP32** mediante un puerto serial, **Tkinter** es la opción ideal. Ofrece una mayor flexibilidad y es completamente gratuito, mientras que su integración con hardware como la ESP32 es mucho más directa y sencilla. Por el contrario, **App Designer**, aunque potente dentro del entorno MATLAB, presenta limitaciones en cuanto a la integración con hardware externo y requiere una licencia de MATLAB, lo que lo hace menos adecuado para proyectos fuera de dicho ecosistema.

4.2. Desarrollo del Cuadro de Instrumentos Automotriz Virtual

El cuadro de instrumentos se desarrolló utilizando App Designer (MATLAB) y Tkinter (Python). App Designer fue seleccionado por su integración moderna con MATLAB, mientras que Tkinter fue elegido por su facilidad de uso y soporte nativo de Python sin dependencias adicionales.

4.2.1. Implementación en App Designer

La interfaz de App Designer, mostrada en la Figura 4.9, utiliza deslizadores y botones para controlar válvulas, con imágenes precargadas que se activan según los valores de los sensores. Sin embargo, la flexibilidad de diseño de App Designer es limitada, ya que restringe a los usuarios a componentes predefinidos con opciones de personalización mínimas.

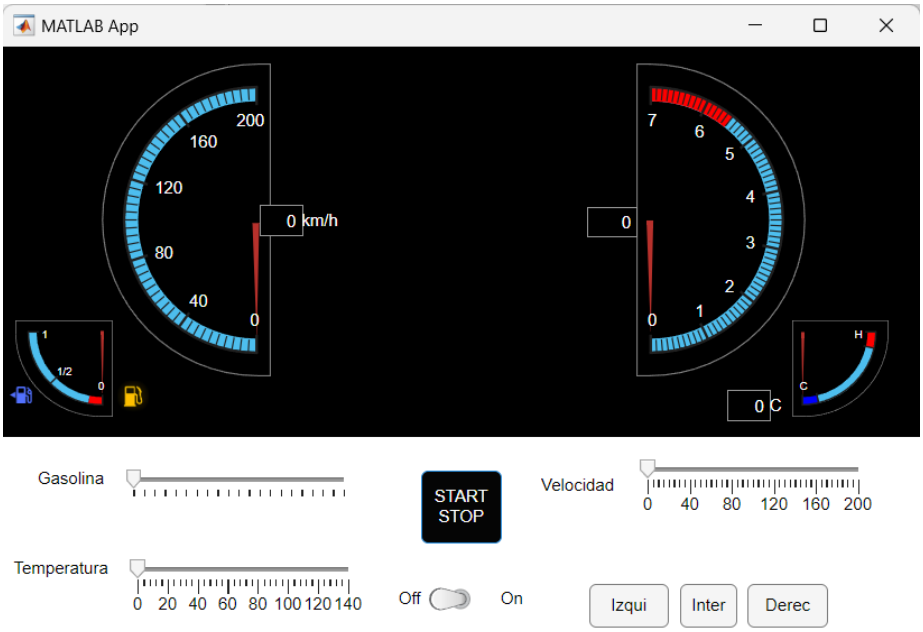


Figura 4.9: Cuadro de Intrumentos diseñado en App Designer

En cuestión de facilidad para la interacción con el hardware a implementar. En este caso la comunicación mediante puerto serial con el microprocesador a ocupar, dicha herramienta nos ayuda a realizar esta interacción de manera correcta y sencilla. En la figura 4.10 se muestra una interfaz la cual ya cuenta con la comunicación mediante el puerto serial, por lo cual la interfaz ya es controlada por los datos que llegan al microprocesador mediante los sensores que ocuparemos y que posteriormente son transmitidos a nuestra computadora.

Figura 4.10: Interfaz con interacción por puerto serial

4.2.2. Implementación en Tkinter

A diferencia de App Designer, que proporciona componentes predefinidos, Tkinter requiere construir los elementos gráficos desde cero. El proceso para diseñar las válvulas de velocidad, aceleración, nivel de gasolina y temperatura del motor se detalla en el diagrama de flujo de la Figura 4.11. Este diagrama muestra la secuencia para crear una ventana principal, generar óvalos y arcos para las válvulas, y posicionar el texto en ellas, especificando sus coordenadas y propiedades visuales como color de relleno, contorno y grosor.

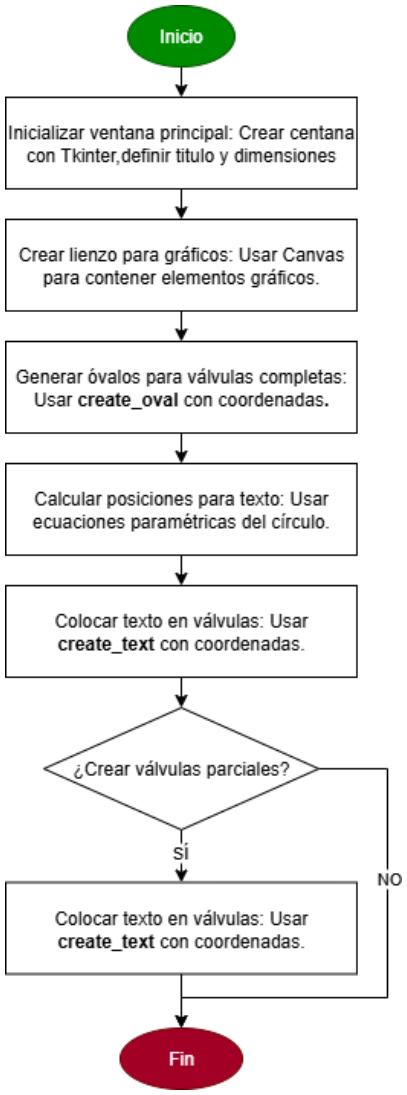


Figura 4.11: Diagrama de flujo para la creación de la interfaz en Tkinter

Para posicionar etiquetas numéricas alrededor de la circunferencia de un círculo que representa una válvula, se distribuyen 18 puntos uniformemente. Las coordenadas de estas etiquetas se calculan mediante las ecuaciones paramétricas presentadas en

(4.1) y (4.2), donde i es un índice que varía de 0 a 17:

$$x(i+1) = 600 - 155\text{sen}\left(\frac{(i+1)\cdot 2\pi}{18}\right) \tag{4.1}$$

$$y(i+1) = 300 - 155\cos\left(\frac{(i+1)\cdot 2\pi}{18}\right) \tag{4.2}$$

Estas ecuaciones derivan de las formas generales de un círculo de radio R centrado en (h, k), descritas en (4.3) y (4.4):

$$x(t) = h + R\cdot\cos(t) \tag{4.3}$$

$$y(t) = k + R\cdot\text{sen}(t) \tag{4.4}$$

donde t varía de 0 a 2π, se adaptan para generar 18 puntos igualmente espaciados alrededor del círculo. Aquí, el centro está en $(600, 300)$, el radio es 155, y el ángulo entre puntos consecutivos es $\frac{2\pi}{18}$.

El proceso de incorporación de deslizadores para controlar los medidores de las válvulas se detalla en el diagrama de flujo de la Figura 4.12. Este incluye la creación de deslizadores, la configuración de una función que actualiza las líneas de los medidores según los valores seleccionados y la visualización de los datos en la interfaz.

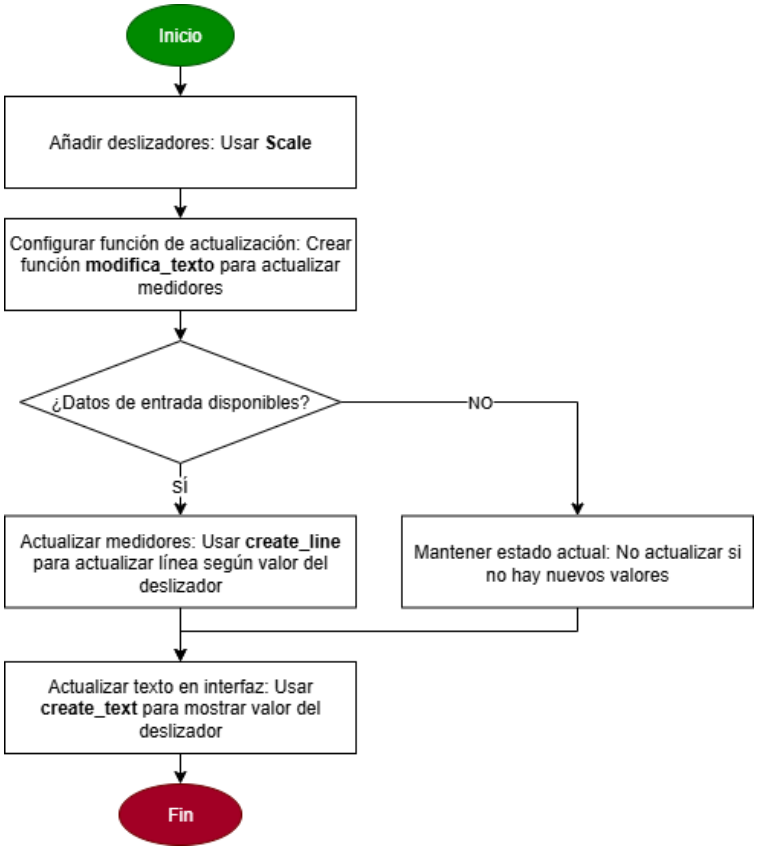


Figura 4.12: Diagrama de flujo para la incorporación de deslizadores en Tkinter

El resultado de la interfaz completa, con válvulas, medidores y deslizadores, se muestra en la Figura 4.13.

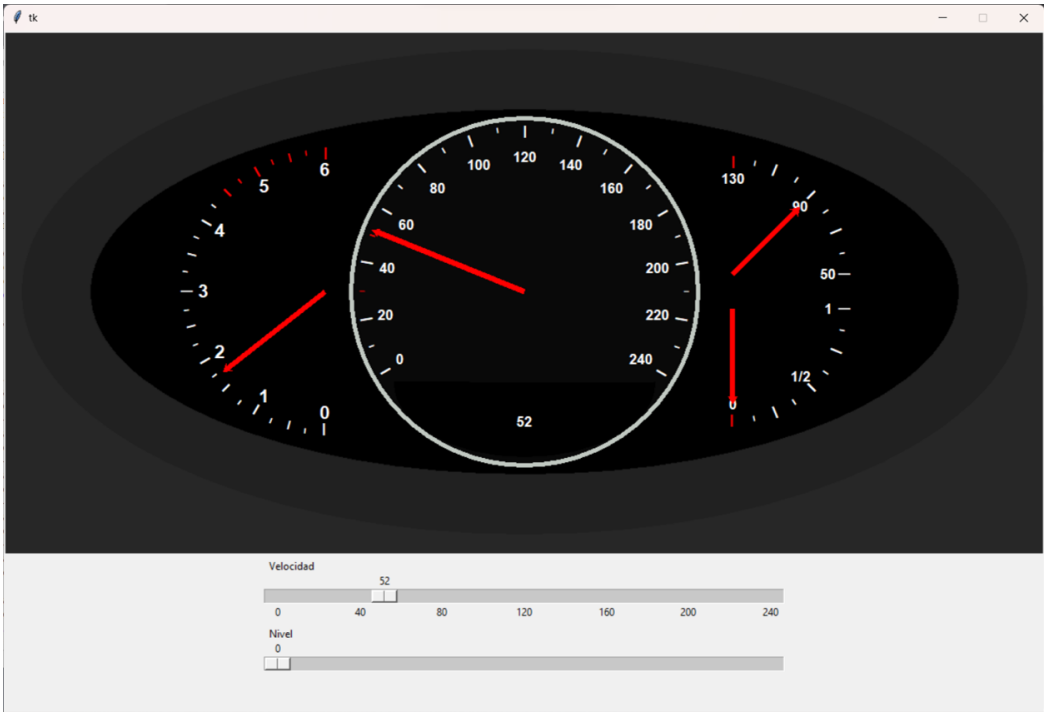


Figura 4.13: Interfaz del cuadro de instrumentos con deslizadores

El resultado final de la interfaz con todas las válvulas y medidores se observa en la Figura 4.14.

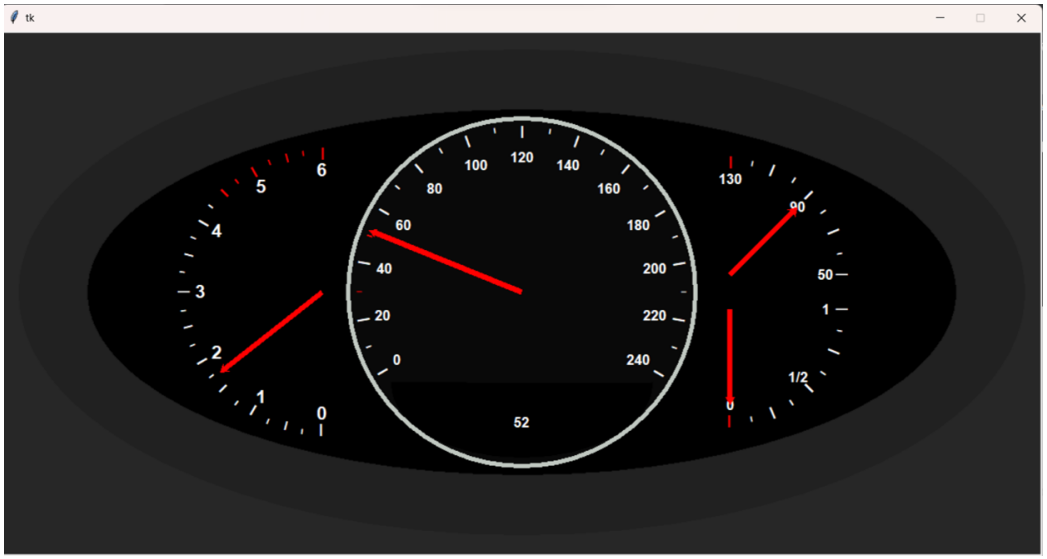


Figura 4.14: Cuadro de instrumentos de Kia Sorento en Tkinter

4.2.3. Integración de Comunicación Serial

La comunicación con hardware externo, como sensores o microcontroladores, se logra mediante el puerto serial, permitiendo actualizar la interfaz gráfica en tiempo real. Este proceso se describe en el diagrama de flujo de la Figura 4.15, que incluye la instalación de la biblioteca necesaria, la identificación y configuración del puerto COM, la lectura continua de datos y la actualización de los elementos gráficos con la información recibida.

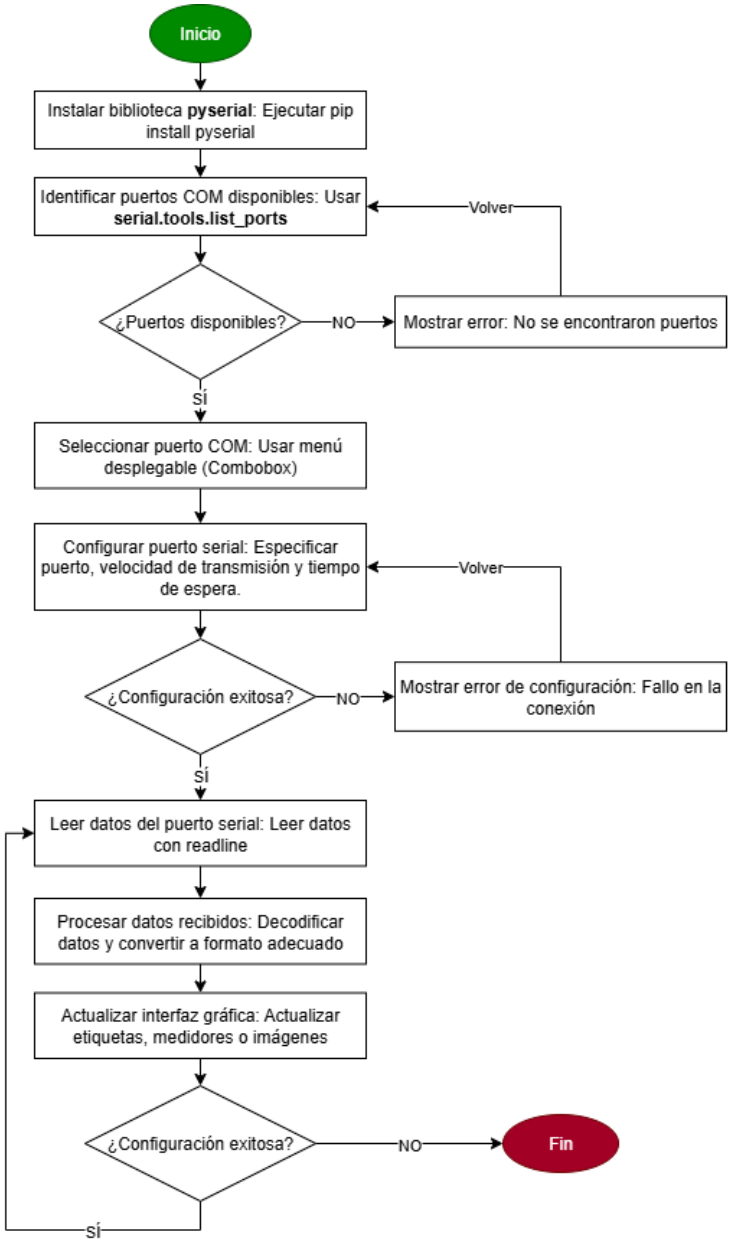


Figura 4.15: Diagrama de flujo para la integración de comunicación serial en Tkinter

Además, se implementa una ventana para seleccionar el puerto COM, como se muestra en la Figura 4.16. Una vez conectado, los datos se visualizan en la interfaz, que también puede incluir imágenes redimensionadas, como se observa en la Figura 4.17.

Figura 4.16: Ventana de inicio de la interfaz

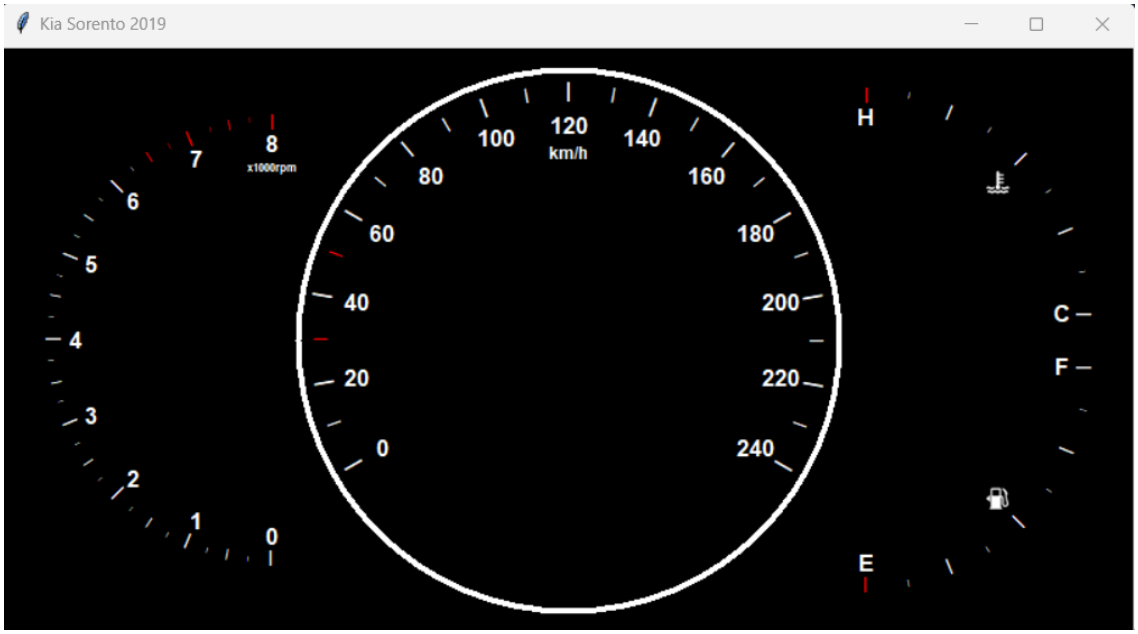


Figura 4.17: Ventana del cuadro de instrumentos

4.2.4. Generación del Instalador de la GUI

Para distribuir la interfaz basada en Tkinter como una aplicación independiente, se generan un ejecutable y un instalador. El proceso se detalla en el diagrama de flujo de la Figura 4.18, que describe los pasos para generar un ejecutable con PyInstaller, crear y compilar un script de Inno Setup, y distribuir el instalador.

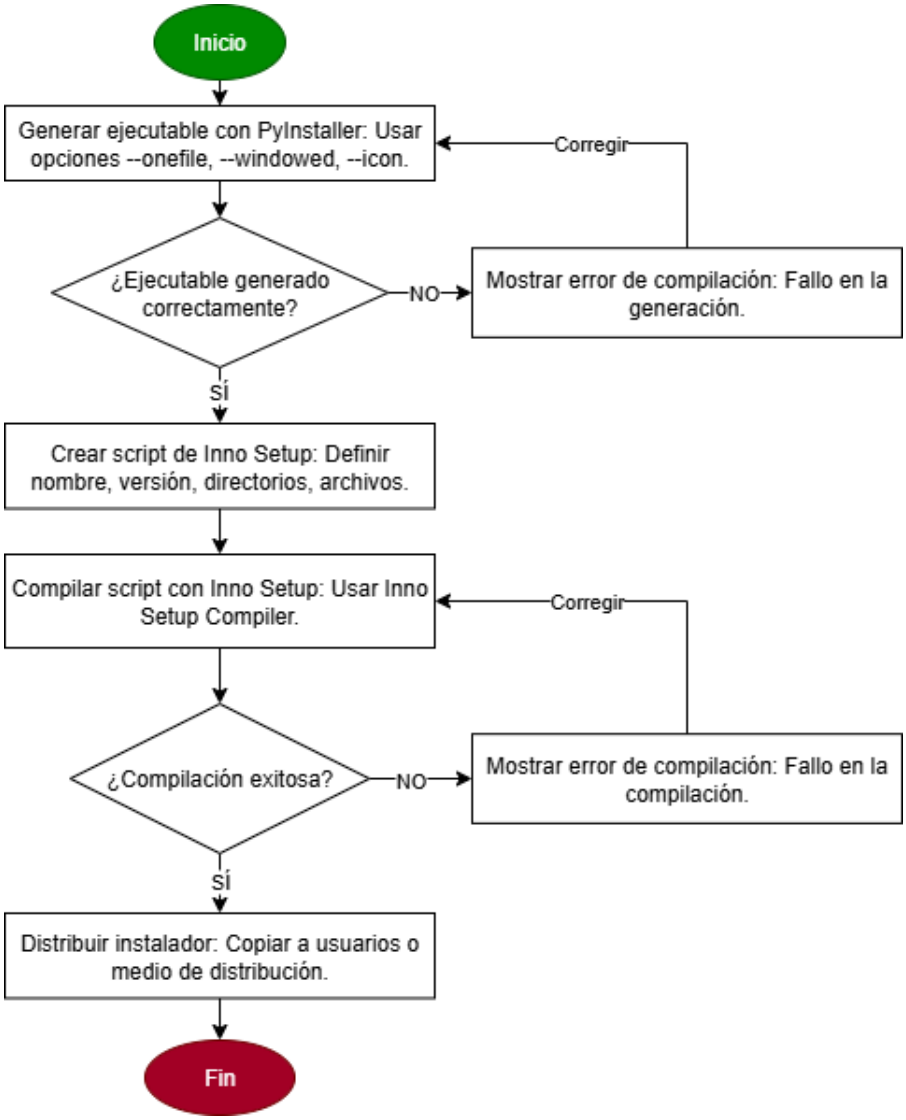


Figura 4.18: Diagrama de flujo para la generación del instalador de la GUI

Este enfoque asegura que la aplicación sea accesible en sistemas Windows sin necesidad de instalar Python, siendo ideal para fines educativos o demostrativos.

4.3. Soluciones CANBUS para hardware de código abierto

Esta sección evalúa las opciones de hardware para implementar una red CAN bus, enfocándose en soluciones de código abierto compatibles con los requisitos del proyecto.

4.3.1. Tarjetas Arduino

Las tarjetas Arduino (por ejemplo, Uno, Mega, Nano) son simples y bien documentadas, pero están limitadas en potencia de procesamiento y memoria [52]. - **Procesador**: ATmega328P o superior. - **Velocidad de reloj**: 16 MHz. - **Compatibilidad**: Módulo MCP2515 mediante SPI.

4.3.2. ESP32

El ESP32 ofrece procesamiento de doble núcleo, Wi-Fi, Bluetooth y soporte para SPI, lo que lo hace ideal para aplicaciones en tiempo real [53]. - **Procesador**: Tensilica Xtensa LX6 de doble núcleo a 240 MHz. - **Memoria**: 520 KB SRAM, soporte para memoria externa. - **Comunicaciones**: SPI, I2C, UART, Wi-Fi, Bluetooth.

4.3.3. Shields CAN Bus de SparkFun/Seeedstudio

Estos shields integran el controlador MCP2515 y el transceptor MCP2551, simplificando la implementación de CAN [54, 55]. - **Interfaz**: Compatible con Arduino Uno/Mega. - **Controlador**: MCP2515, hasta 1 Mbps. - **Transceptor**: MCP2551.

4.3.4. Módulo MCP2515

El módulo MCP2515 es económico y ampliamente utilizado, comunicándose mediante SPI [56]. - **Velocidad máxima**: 1 Mbps. - **Protocolo**: SPI. - **Voltaje de operación**: 5 V (requiere adaptadores para microcontroladores de 3.3 V).

4.3.5. Transceptor SN65HVD230

El transceptor SN65HVD230 de Texas Instruments soporta comunicación CAN de alta velocidad con protección robusta [57]. - **Velocidad máxima**: 1 Mbps. - **Protección ESD**: ±16 kV. - **Voltaje de operación**: 3.3 V o 5 V.

4.3.6. Selección del Hardware

Para la implementación de la red CAN, se realizó una evaluación exhaustiva de los componentes de hardware, considerando factores como rendimiento, costo, compatibilidad y soporte comunitario. Inicialmente, se seleccionaron el microcontrolador ESP32 y el controlador MCP2515 debido a su bajo costo (ESP32: $100-130 MXN), alta capacidad

de procesamiento (240 MHz frente a 16 MHz del Arduino UNO) y amplio soporte comunitario. El ESP32 destaca por su memoria (4-16 MB frente a 32 KB del Arduino UNO) y soporte para interfaces como SPI, lo que permite su integración con módulos como el MCP2515.

Sin embargo, para optimizar la compatibilidad y fiabilidad de la red CAN, se decidió implementar el ESP32 con el transceptor SN65HVD230. Este transceptor ofrece una latencia significativamente menor (5 μs frente a 50-200 μs del MCP2515) y es ideal para modelos de ESP32 con soporte CAN nativo, como se muestra en los esquemas de conexión (por ejemplo, ESP-WROOM-32 con SN65HVD230). En contraste, el MCP2515, aunque efectivo y compatible con Arduino y ESP32, requiere una interfaz SPI que incrementa la latencia y complejidad del diseño. La combinación de ESP32 y SN65HVD230 proporciona una solución robusta, eficiente y escalable, como se detalla en la Sección 4.4.

4.4. Red de 4 nodos con sensores y actuadores automotrices

El prototipo consta de cuatro nodos interconectados mediante un bus CAN, cada uno equipado con una unidad de control electrónico (ECU) diseñada para este proyecto. Un nodo actúa como *gateway*, mostrando datos en la GUI, mientras que los otros monitorean el nivel de combustible, la temperatura y el tren motriz (velocidad y revoluciones por minuto del motor mediante un sensor de efecto Hall). La Figura 4.19 muestra el diagrama de conexión del prototipo.

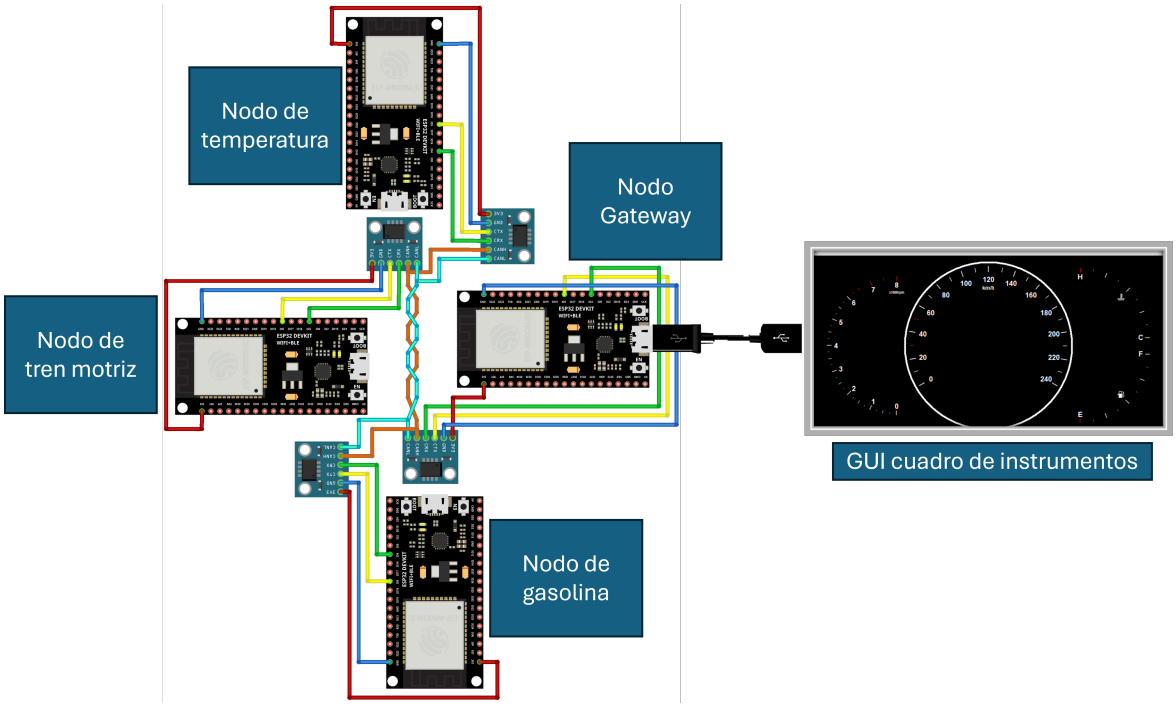


Figura 4.19: Esquema del prototipo educativo.

4.4.1. Proceso de Desarrollo

El desarrollo comenzó con la creación de la GUI en App Designer y Tkinter, seguido de pruebas de comunicación serial con el ESP32. Las pruebas iniciales utilizaron tres potenciómetros para simular datos de sensores, programados mediante Arduino IDE (Figura 4.20). La configuración se muestra en las Figuras 4.21a y 4.21b.

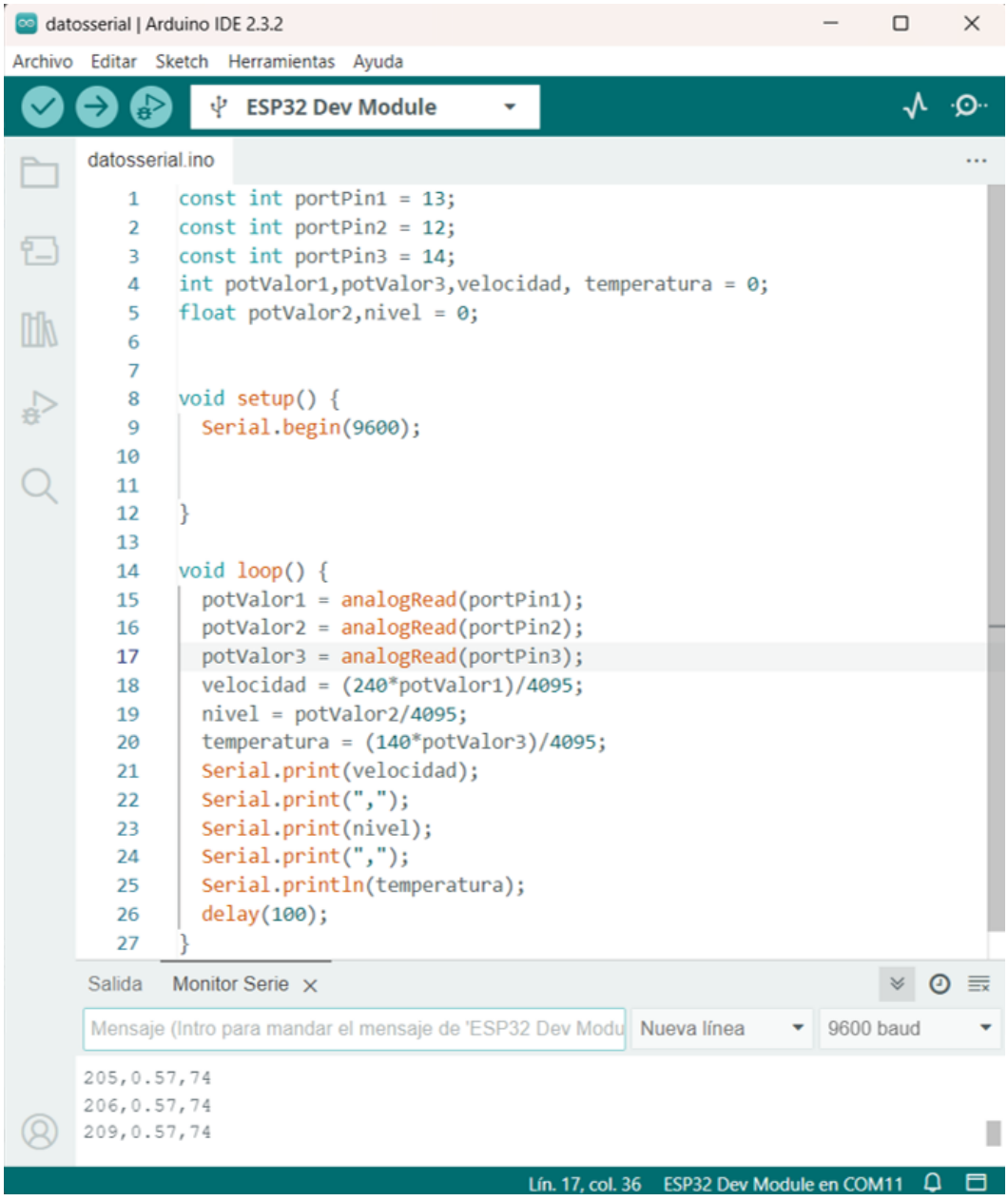
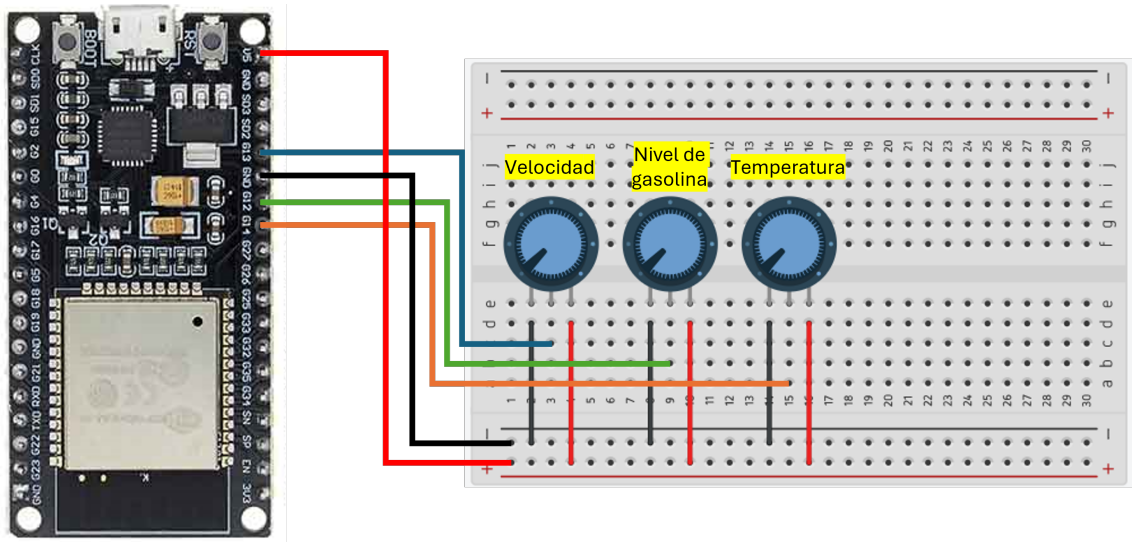
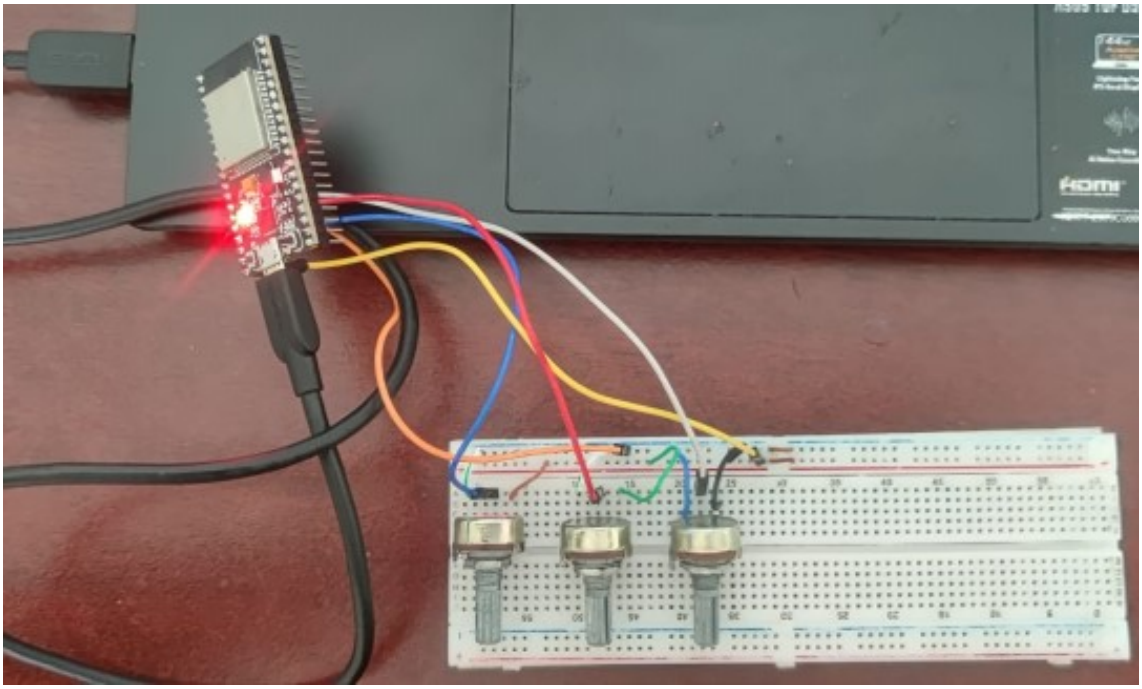


Figura 4.20: Programa de 3 potenciómetros en arduino IDE.



(a) Diagrama de conexión.



(b) Implementación física.

Figura 4.21: Prueba de comunicación serial con 3 potenciómetros.

La interacción de la interfaz Tkinter con el ESP32 se muestra en la Figura 4.22.

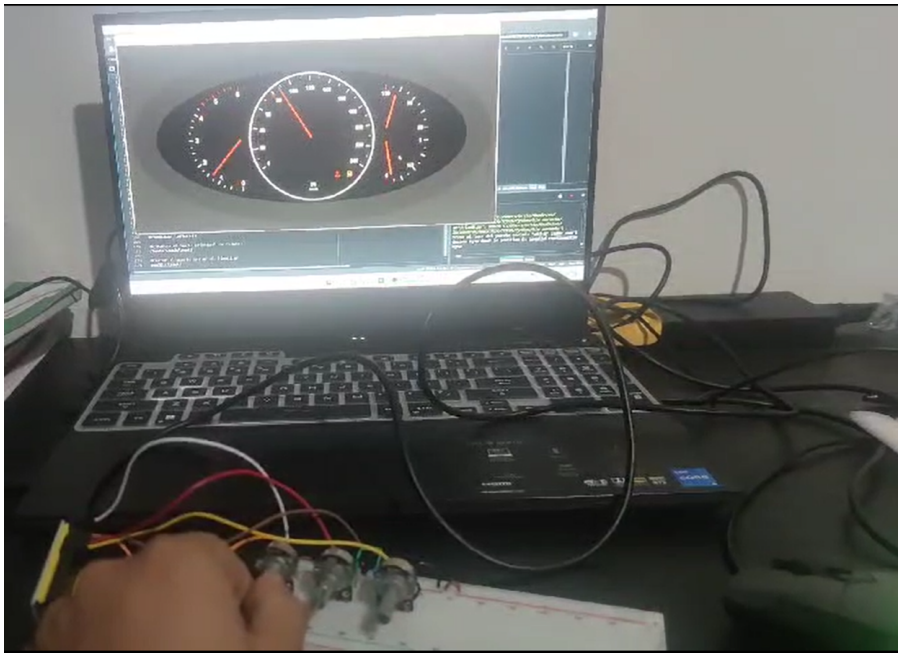


Figura 4.22: Interacción de la interfaz con la esp32

4.4.2. Nodo de Nivel de Combustible

El nodo de nivel de combustible emplea un sensor de nivel universal (Figura 4.23) con una resistencia variable de 38 Ω (tanque lleno) a 230 Ω (tanque vacío). Este sensor se conecta a un divisor de voltaje con una resistencia fija de 120 Ω, que transforma la resistencia variable en un voltaje interpretable por el pin analógico del ESP32.

Figura 4.23: Medidor de nivel de combustible universal

El sensor de nivel de combustible presenta las siguientes características:

- Resistencia de **38** Ω cuando el tanque está lleno.
- Resistencia de **230** Ω cuando el tanque está vacío.

El divisor de voltaje, compuesto por el sensor y una resistencia fija de 120 Ω, genera un voltaje de salida que varía según la resistencia del sensor. Este voltaje es leído por el ESP32 como una señal analógica, permitiendo determinar el nivel de combustible.

Cálculo del Voltaje de Salida en el Divisor de Voltaje

El divisor de voltaje se configura con los siguientes parámetros:

- $V_{in} = 5\,\text{V}$
- $R_{\text{fija}} = 120\,\Omega$
- R_{sensor} varía entre 38 Ω (tanque lleno) y 230 Ω (tanque vacío).

El voltaje de salida se calcula mediante la fórmula presentada en (4.5):

$$V_{\text{out}} = V_{\text{in}} \times \frac{R_{\text{fija}}}{R_{\text{sensor}} + R_{\text{fija}}} \tag{4.5}$$

Los valores extremos de R_{sensor} producen los siguientes resultados:

- Para $R_{\text{sensor}} = 38\,\Omega$ (tanque lleno), según (4.6):

$$V_{\text{out}} \approx 5 \times \frac{120}{38 + 120} \approx 3.89\,\text{V} \tag{4.6}$$

- Para $R_{\text{sensor}} = 230\,\Omega$ (tanque vacío), según (4.7):

$$V_{\text{out}} \approx 5 \times \frac{120}{230 + 120} \approx 1.71\,\text{V} \tag{4.7}$$

Esquema de Conexión

El sensor de nivel de combustible se conecta a un pin analógico del ESP32, con un extremo del sensor unido a la tensión de alimentación (VCC). El divisor de voltaje se forma con una resistencia fija de 120 Ω en serie con el sensor, como se muestra en la Figura 4.24.

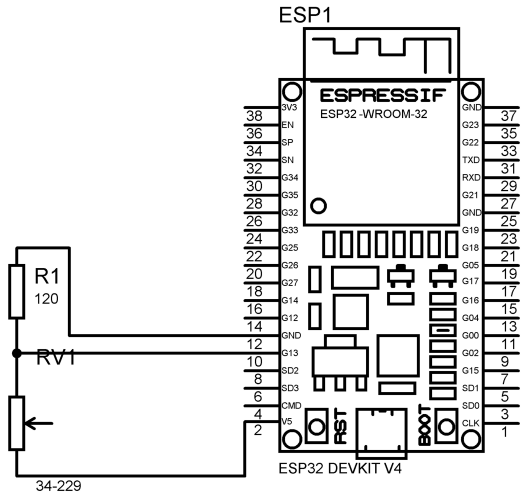
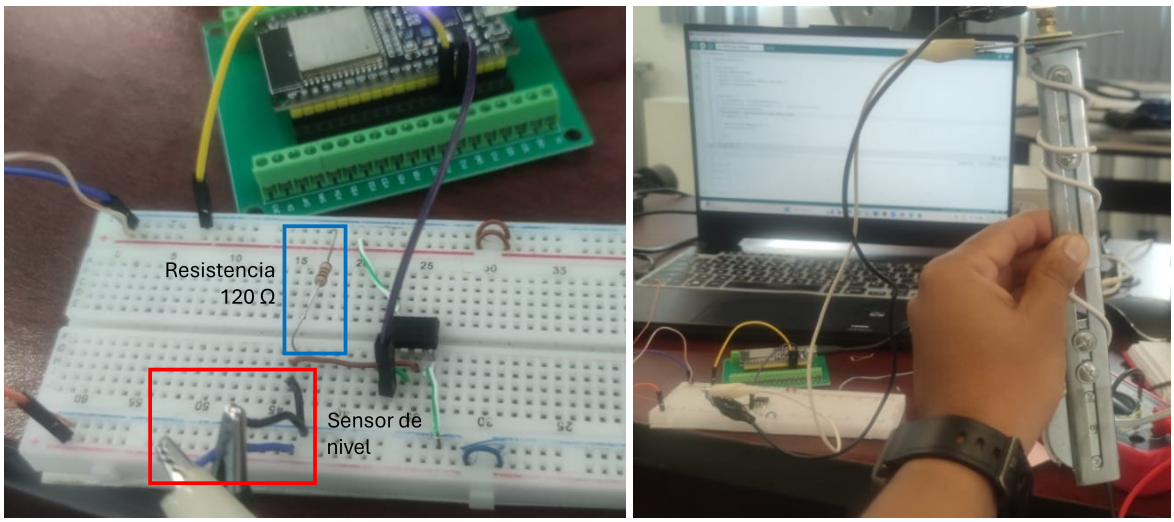


Figura 4.24: Diagrama de conexión del nodo de nivel de combustible

La conexión se detalla a continuación:

- El pin analógico del ESP32 se conecta al punto de unión entre el sensor y la resistencia fija.
- Un extremo del sensor se conecta a VCC.
- El extremo libre de la resistencia fija se conecta a tierra.

Este esquema se implementó en una protoboard, como se observa en la Figura 4.25.



(a) Montaje en protoboard (b) Implementación física

Figura 4.25: Nodo de nivel de combustible

Este montaje permite medir el diferencial de potencial para determinar el nivel de combustible en el tanque.

Implementación en el Sistema

El funcionamiento del nodo de nivel de combustible se representa en el diagrama de flujo de la Figura 4.26. Este diagrama describe el proceso de lectura del voltaje analógico, su conversión a un porcentaje de nivel de combustible y el envío de la información al puerto serial o mediante el protocolo CAN.

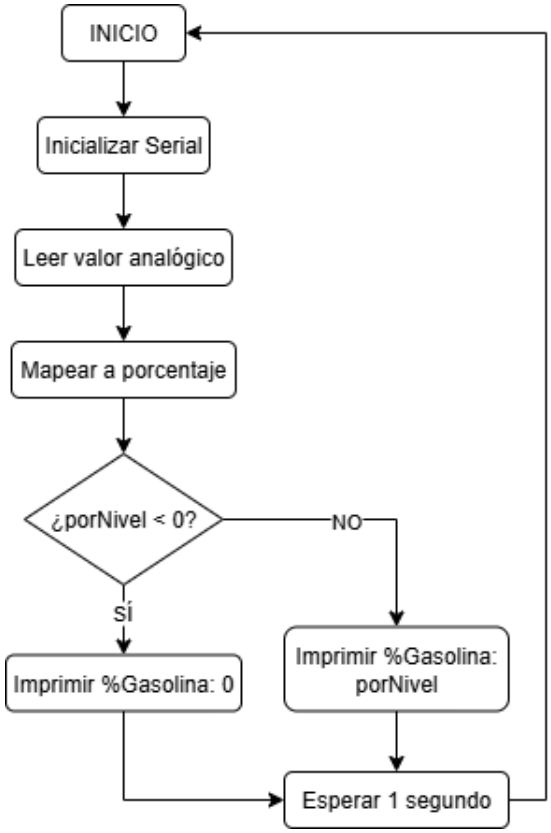


Figura 4.26: Diagrama de flujo del nodo de nivel de combustible

El sistema realiza las siguientes operaciones:

- Lee el valor analógico del divisor de voltaje a través del pin analógico del ESP32.
- Convierte el valor analógico a un porcentaje de nivel de combustible (0 % para tanque vacío, 100 % para tanque lleno), considerando los valores extremos de la lectura analógica (1950 para tanque vacío y 4095 para tanque lleno).
- Transmite el porcentaje calculado al puerto serial o mediante el protocolo CAN.
- Repite el proceso cada segundo para una monitorización continua.

Este diseño permite un monitoreo continuo y preciso del nivel de combustible, con la posibilidad de integrarlo en un sistema de comunicación más amplio mediante el protocolo CAN.

4.4.3. Nodo de Temperatura

El nodo de temperatura utiliza un termistor NTC (Coeficiente Negativo de Temperatura) de 1 kΩ nominal a temperatura ambiente, descrito en la sección 3.3. Como se explicó en el marco teórico, este sensor exhibe una relación no lineal entre su resistencia eléctrica y la temperatura, modelada con alta precisión mediante la ecuación de Steinhart-Hart. Para integrar este sensor en el sistema, se calibró para determinar los coeficientes A, B y C, que permiten convertir la resistencia medida en valores de temperatura.

Descripción del Experimento

El experimento de calibración tuvo como objetivo obtener datos confiables de temperatura y resistencia para ajustar la ecuación de Steinhart-Hart. Se emplearon los siguientes instrumentos:

- Un multímetro digital con termopar tipo K para medir la temperatura del agua.
- Un segundo multímetro digital para medir la resistencia del termistor NTC.
- Un termistor NTC de 1 kΩ nominal a 25°C.

La configuración experimental se muestra en la Figura 4.27, donde se observa el montaje para medir simultáneamente la temperatura y la resistencia.

Figura 4.27: Configuración experimental para la calibración del nodo de temperatura con termistor NTC, multímetro y termopar.

El procedimiento fue el siguiente:

1. Se sumergieron el termopar y el termistor en una olla con agua y hielo, asegurando contacto térmico directo con el líquido para establecer un punto de baja temperatura.

2. Se calentó el agua gradualmente con una estufa eléctrica, registrando mediciones simultáneas de temperatura (en °C) y resistencia (en Ω) cada vez que las lecturas se estabilizaban.

3. Se obtuvieron datos desde 6°C hasta 100°C. A partir de 98°C, la resistencia se estabilizó en 120 Ω, indicando una posible saturación del sensor o limitación del multímetro, por lo que se excluyeron valores superiores a 98°C.

4. Para garantizar una calibración precisa de la ecuación de Steinhart-Hart, se seleccionaron cuidadosamente tres puntos representativos que abarcan el rango térmico de interés:

   - **6°C: 2170** Ω, representando el extremo inferior del rango, donde la resistencia del termistor es alta debido a la baja temperatura.
   - **52°C: 410** Ω, correspondiente al rango medio, que refleja un punto típico de operación en condiciones ambientales moderadas.
   - **98°C: 120** Ω, en el extremo superior, cercano al límite de operación del sensor, donde la resistencia disminuye significativamente.

   Estos puntos fueron elegidos estratégicamente para capturar la variación no lineal de la resistencia del termistor a lo largo del rango de temperaturas, asegurando una representación adecuada de su comportamiento térmico.

Cálculo de los Coeficientes

Para usar la ecuación de Steinhart-Hart, se convirtieron las temperaturas a Kelvin, como se muestra en la ecuación (4.8):

$$T_1 = 6 + 273.15 = 279.15\,\text{K}, \quad T_2 = 52 + 273.15 = 325.15\,\text{K}, \quad T_3 = 98 + 273.15 = 371.15\,\text{K}. \tag{4.8}$$

Se calcularon los logaritmos naturales de las resistencias y sus cubos, según la ecuación (4.9):

$$\begin{aligned} L_1 &= \ln(2170) \approx 7.6825, & L_1^3 &\approx 453.4242, \\ L_2 &= \ln(410) \approx 6.0162, & L_2^3 &\approx 217.7497, \\ L_3 &= \ln(120) \approx 4.7875, & L_3^3 &\approx 109.7297. \end{aligned} \tag{4.9}$$

Se obtuvieron los inversos de las temperaturas, como se indica en la ecuación (4.10):

$$Y_1 = \frac{1}{279.15} \approx 0.0035823, \quad Y_2 = \frac{1}{325.15} \approx 0.0030755, \quad Y_3 = \frac{1}{371.15} \approx 0.0026943. \tag{4.10}$$

Se calcularon las diferencias, presentadas en la ecuación (4.11):

$$\begin{aligned} \Delta Y_{21} &= Y_2 - Y_1 \approx -0.0005068, & \Delta Y_{31} &= Y_3 - Y_1 \approx -0.0008880, \\ \Delta L_{21} &= L_2 - L_1 \approx -1.6663, & \Delta L_{31} &= L_3 - L_1 \approx -2.8950, \\ \Delta L_{21}^3 &= L_2^3 - L_1^3 \approx -235.6746, & \Delta L_{31}^3 &= L_3^3 - L_1^3 \approx -343.6946. \end{aligned} \tag{4.11}$$

Los coeficientes se calcularon con las ecuaciones (4.12), (4.13) y (4.14):

$$C = \frac{\Delta Y_{31} \cdot \Delta L_{21} - \Delta Y_{21} \cdot \Delta L_{31}}{\Delta L_{31}^3 \cdot \Delta L_{21} - \Delta L_{21}^3 \cdot \Delta L_{31}} \approx -1.1385 \times 10^{-7}, \tag{4.12}$$

$$B = \frac{\Delta Y_{21} - C \cdot \Delta L_{21}^3}{\Delta L_{21}} \approx 3.2024 \times 10^{-4}, \tag{4.13}$$

$$A = Y_1 - B \cdot L_1 - C \cdot L_1^3 \approx 1.1737 \times 10^{-3}. \tag{4.14}$$

Los coeficientes resultantes fueron $A = 1.1737 \times 10^{-3}$, $B = 3.2024 \times 10^{-4}$, y $C = -1.1385 \times 10^{-7}$, utilizados en la ecuación de Steinhart-Hart para modelar la relación resistencia-temperatura.

Estos valores coincidieron con los obtenidos mediante la calculadora de Steinhart-Hart de Stanford Research Systems Inc. (Figura 4.28), validando la precisión del procedimiento.

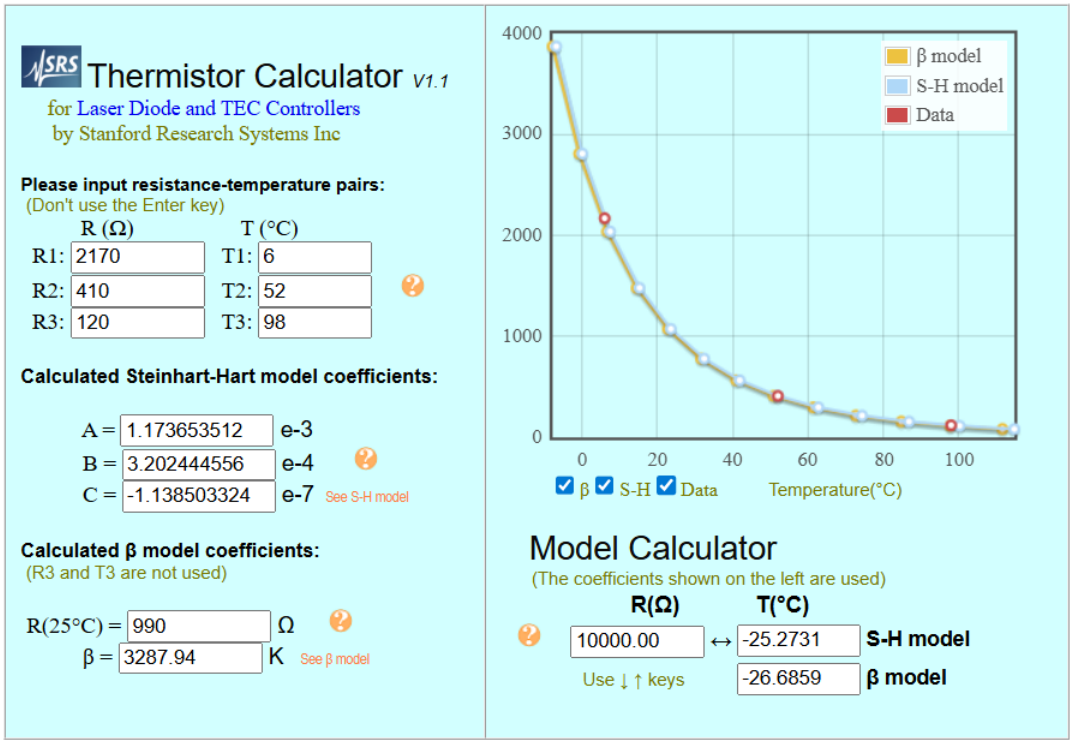


Figura 4.28: Calculadora de coeficientes Steinhart-Hart. Fuente: Stanford Research Systems Inc.

Esquema de Conexión

El termistor NTC se conecta en un divisor de voltaje con una resistencia fija de 1 kΩ, alimentado por una fuente de 3.26 V (VCC). El esquema se muestra en la Figura 4.29, donde:

- El termistor se conecta entre VCC y el punto de medición (unión con el resistor fijo).
- El resistor fijo de 1 kΩ se conecta entre el punto de medición y tierra.
- El punto de medición se conecta al pin analógico 13 del ESP32 para leer el voltaje.

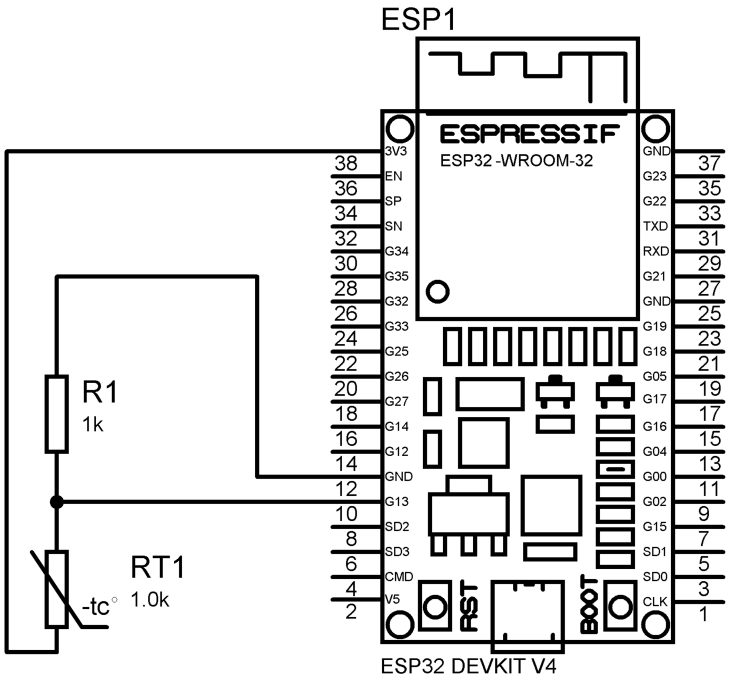


Figura 4.29: Esquema de conexión del nodo de temperatura con termistor NTC y resistor fijo.

El circuito se implementó en una protoboard, como se observa en la Figura 4.30, asegurando conexiones estables para lecturas precisas del voltaje y posterior cálculo de la temperatura.

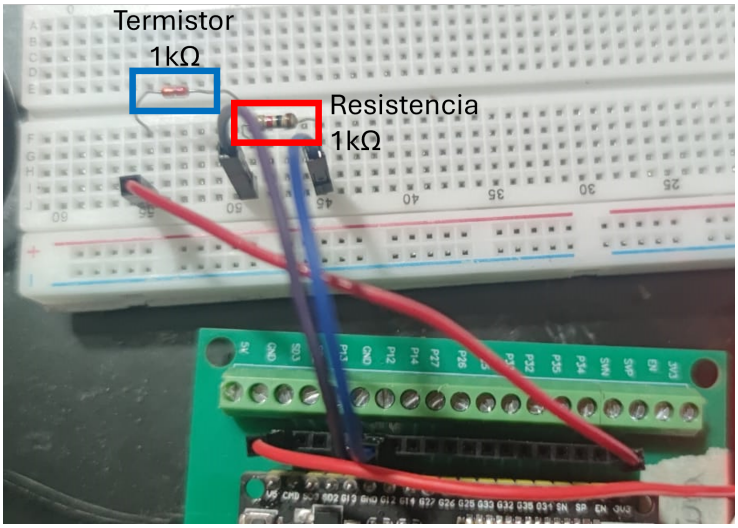


Figura 4.30: Montaje físico del nodo de temperatura en protoboard.

Funcionamiento del Nodo de Temperatura

El funcionamiento del nodo de temperatura se ilustra en el diagrama de flujo de la Figura 4.31. El proceso consta de los siguientes pasos:

- **Inicialización**: Se configura el microcontrolador ESP32 para inicializar el bus CAN y el pin analógico 13 para la lectura del voltaje.
- **Lectura analógica**: Se mide el voltaje en el punto de unión del divisor de voltaje (entre el termistor y el resistor fijo) utilizando el ADC de 12 bits del ESP32, que convierte la señal analógica en un valor digital (0–4095).
- **Cálculo de resistencia**: A partir del voltaje medido, se calcula la resistencia del termistor usando la ecuación del divisor de voltaje, mostrada en la ecuación (4.15):

$$R = \frac{(3.26 - V_o) \cdot 1000}{V_o}, \tag{4.15}$$

  donde V_o es el voltaje medido en voltios y R es la resistencia en ohmios.

- **Cálculo de temperatura**: Se aplica la ecuación de Steinhart-Hart con los coeficientes $A = 1.1737 \times 10^{-3}$, $B = 3.2024 \times 10^{-4}$, $C = -1.1385 \times 10^{-7}$ para obtener la temperatura en Kelvin, como se muestra en la ecuación (4.16):

$$\frac{1}{T_K} = A + B \cdot \ln(R) + C \cdot (\ln(R))^3, \tag{4.16}$$

  y se convierte a Celsius: $T_C = T_K - 273.15$.

- **Transmisión de datos**: La temperatura se trunca a un valor entero y se envía a través del bus CAN con el identificador 0x102. Además, se muestra en el monitor serial para verificación en tiempo real.

- **Control de frecuencia**: Se ajusta el intervalo de muestreo (500 ms para temperaturas < 100°C, 200 ms para ≥ 100°C) para optimizar la transmisión.

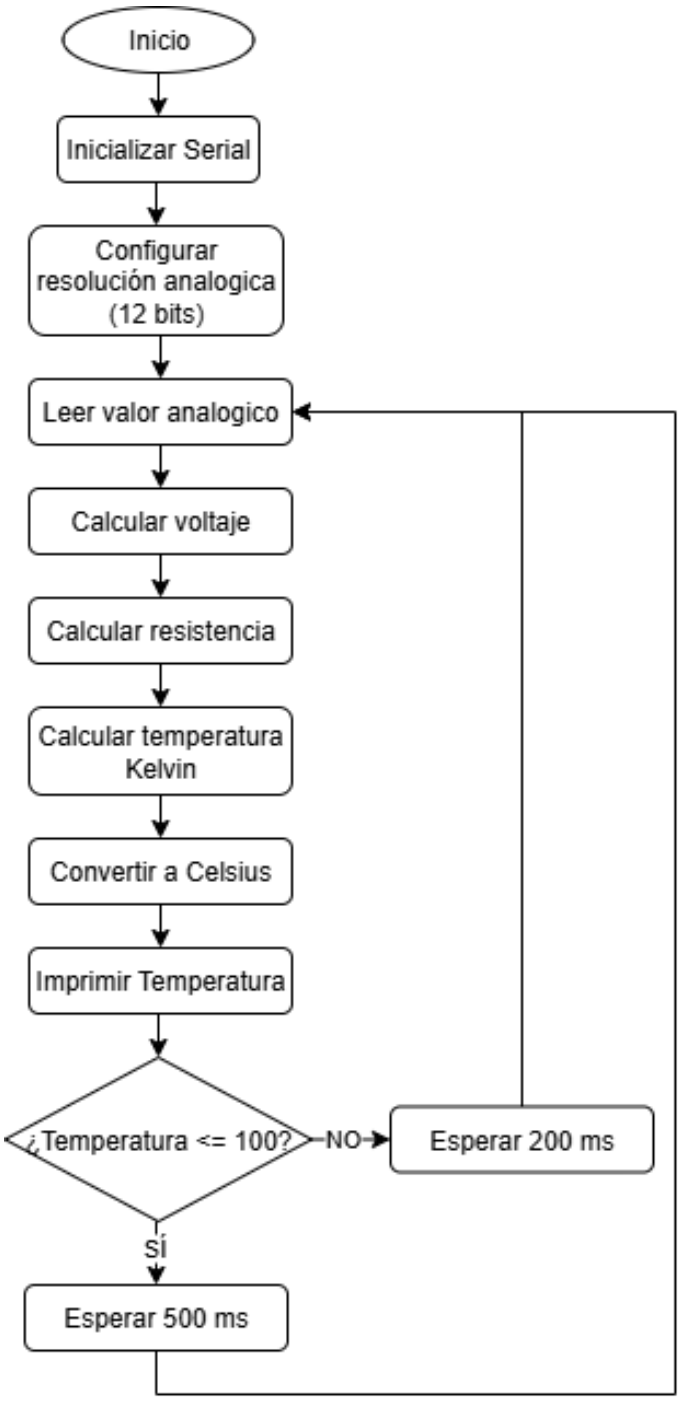


Figura 4.31: Diagrama de flujo del funcionamiento del nodo de temperatura.

Resolución del Termistor

La resolución del sensor, definida como el cambio mínimo de temperatura detectable, depende de la sensibilidad del termistor (en mV/°C) y la resolución del ADC de 12 bits del ESP32 (4096 niveles, 0.796 mV por nivel). La sensibilidad se calcula como el cambio en el voltaje del termistor (V_o) por cada grado Celsius, dado por la ecuación (4.17):

$$\frac{dV_o}{dT_C} = \left(3.26 \cdot \frac{1000}{(R+1000)^2}\right) \cdot \left(-\frac{1}{T_K^2 \cdot \left(\frac{B}{R} + 3C \cdot \frac{(\ln(R))^2}{R}\right)}\right) \cdot 1000\,\text{(mV/°C)}, \tag{4.17}$$

donde R es la resistencia del termistor en ohmios, T_K es la temperatura en Kelvin, y A, B, C son los coeficientes de Steinhart-Hart.

Usando los datos de caracterización (6°C a 100°C), la sensibilidad varía de:

- **26.12 mV/°C a 6°C**, donde la resistencia cambia rápidamente (de 2.17 kΩ a 2.14 kΩ), permitiendo detectar cambios de 0.031°C (0.796 mV / 26.12 mV/°C).
- **0.98 mV/°C a 100°C**, donde la resistencia cambia lentamente (0.12 kΩ), resultando en una resolución de 0.81°C (0.796 mV / 0.98 mV/°C).

La alta sensibilidad a bajas temperaturas (6–20°C) indica que el termistor es ideal para aplicaciones que requieren precisión en este rango. Sin embargo, la truncación de la temperatura a un entero limita la resolución práctica a 1°C. Para aprovechar la resolución teórica (0.031°C a 6°C), se recomienda transmitir la temperatura como un valor de punto flotante utilizando más bytes en el bus CAN.

Esta configuración, ilustrada en las Figuras 4.29, 4.30 y 4.31, permite un monitoreo preciso de la temperatura, especialmente en el rango de 6–20°C, adecuado para aplicaciones como el control térmico en vehículos.

4.4.4. Nodo del Tren Motriz

El nodo del tren motriz está diseñado para emular el comportamiento de un sistema de propulsión automotriz, integrando componentes electrónicos y mecánicos que permiten controlar y monitorear un motor DC de manera precisa. Este nodo está conformado por un motor DC RS-550, equipado con una rueda plástica con 8 imanes, un sensor de efecto Hall A44E para medir la velocidad de giro (RPM), un potenciómetro que simula el pedal de acelerador, un microcontrolador ESP32 para procesamiento y comunicación CAN, y un módulo L298N como controlador del motor. La configuración experimental incluyó:

- **Componentes**:
  - Motor DC RS-550 (12 V, hasta 14,000 RPM).
  - Sensor de efecto Hall A44E (resolución de detección: 1 pulso por imán, con 8 imanes por revolución, permitiendo una medición precisa de RPM).

- Potenciómetro (10 kΩ) para emular el pedal de acelerador.
- ESP32 para procesamiento y comunicación CAN.
- Módulo L298N como controlador del motor.

- **Funcionamiento del Nodo**: El nodo del tren motriz simula un sistema de propulsión automotriz mediante la integración de componentes electrónicos y mecánicos que interactúan de forma coordinada. El motor DC RS-550, controlado por el módulo L298N, recibe señales PWM generadas por el ESP32, cuya frecuencia y ciclo de trabajo son ajustados según la entrada del potenciómetro, que emula el pedal de acelerador. El sensor de efecto Hall A44E detecta los pulsos generados por los 8 imanes montados en la rueda plástica, permitiendo calcular las RPM del motor con alta precisión. Estos datos se procesan en el ESP32, que implementa una lógica de control para emular una caja de cambios automática con cinco marchas, ajustando las RPM y la salida PWM según los umbrales definidos para cambios ascendentes y descendentes. La comunicación CAN permite transmitir las RPM y la velocidad calculada a otros nodos del sistema, replicando la funcionalidad de un vehículo real. Este diseño no solo permite un control dinámico del motor, sino que también proporciona una plataforma para estudiar la respuesta del sistema ante diferentes condiciones de operación.

- **Caracterización del Motor y Selección de Frecuencia PWM**: Para optimizar el control del motor RS-550, se evaluó su respuesta a la modulación por ancho de pulso (PWM) en un rango de frecuencias de 100 Hz a 19 kHz, utilizando una resolución de 12 bits (4096 niveles). El objetivo fue determinar la frecuencia óptima que garantice un control preciso, estabilidad y compatibilidad con aplicaciones automotrices, minimizando vibraciones audibles y maximizando la eficiencia del controlador.

  La caracterización midió la velocidad del motor (RPM) en función del ciclo de trabajo del PWM (0–100 %), correlacionada con el voltaje del potenciómetro (0–5 V), mapeado al 0–100 % de entrada del pedal. Las pruebas evaluaron la linealidad, precisión y consistencia de la respuesta del motor, considerando los requisitos de un sistema de tren motriz automotriz [58]. Los análisis realizados son los siguientes:

  - **Gráficas de Dispersión y Regresión Lineal**: Se generaron gráficas de dispersión para cada frecuencia, mostrando RPM medidos frente a RPM esperados, calculados proporcionalmente al ciclo de trabajo, asumiendo una relación lineal dada por la ecuación (4.18):

$$\text{RPM}_{\text{esperado}} = 14000 \cdot \frac{\text{Ciclo de Trabajo}}{100} \tag{4.18}$$

    donde $\text{RPM}_{\text{esperado}}$ es la velocidad esperada del motor en revoluciones por minuto, 14000 es la velocidad máxima del motor RS-550 en RPM, y Ciclo de Trabajo es el porcentaje de activación de la señal PWM. Se aplicó regresión lineal para evaluar la linealidad de la respuesta del motor, calculando el coeficiente de

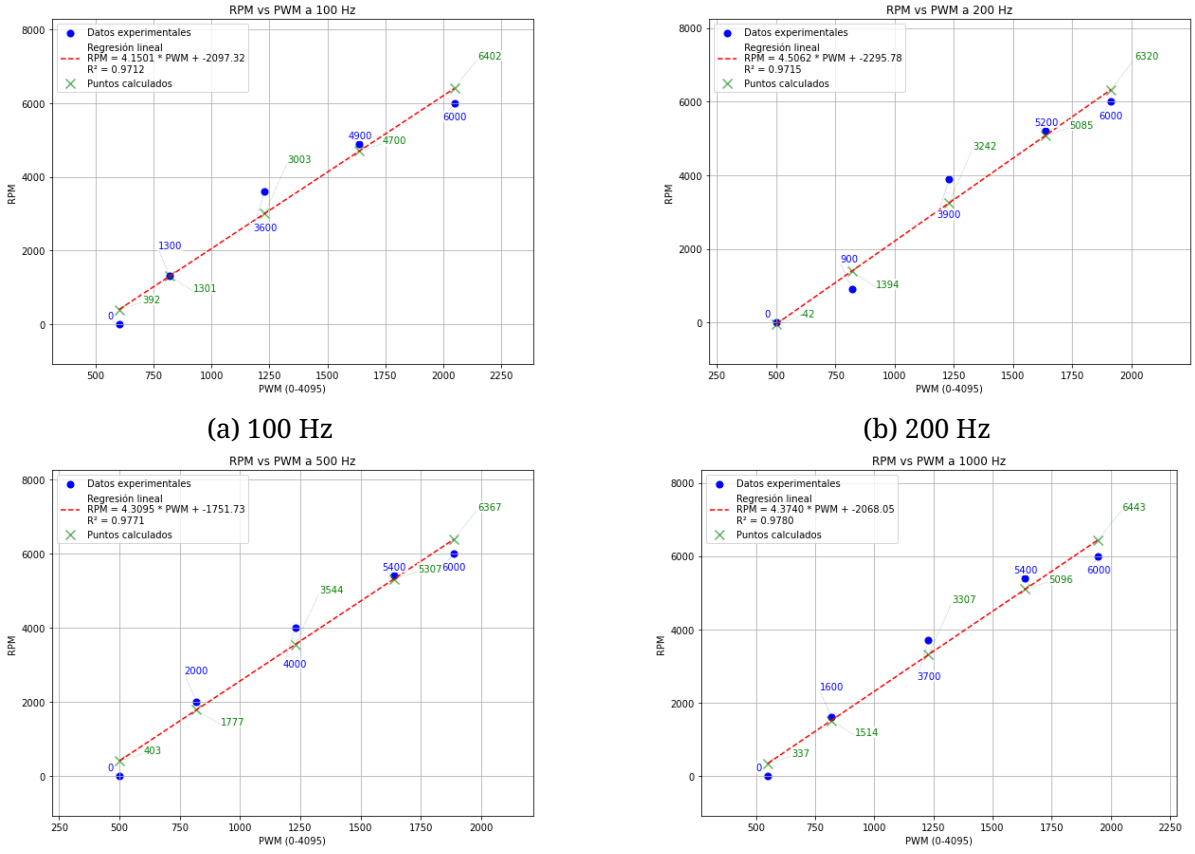
correlación (R^2). Las Figuras 4.32 y 4.33 muestran las gráficas de dispersión para todas las frecuencias probadas, permitiendo comparar visualmente las diferencias en la respuesta del motor.

- **Error Porcentual Absoluto**: Se calculó el error porcentual absoluto mediante la ecuación (4.19):

$$\text{EPA} = \frac{|\text{RPM}_{\text{medido}} - \text{RPM}_{\text{esperado}}|}{\text{RPM}_{\text{esperado}}} \times 100 \tag{4.19}$$

donde EPA es el error porcentual absoluto (%), $\text{RPM}_{\text{medido}}$ es la velocidad medida del motor en RPM, y $\text{RPM}_{\text{esperado}}$ es la velocidad calculada según la ecuación (4.18). Este cálculo evalúa la precisión del control. La media y desviación estándar del EPA se presentan en la Tabla 4.1.

- **Boxplot de Errores**: Se generó un boxplot (Figura 4.34) para comparar el EPA de todas las frecuencias, mostrando la distribución de errores y resaltando valores atípicos.



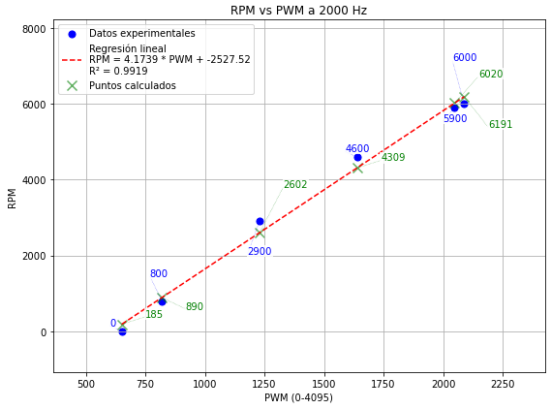
(a) 100 Hz

(b) 200 Hz

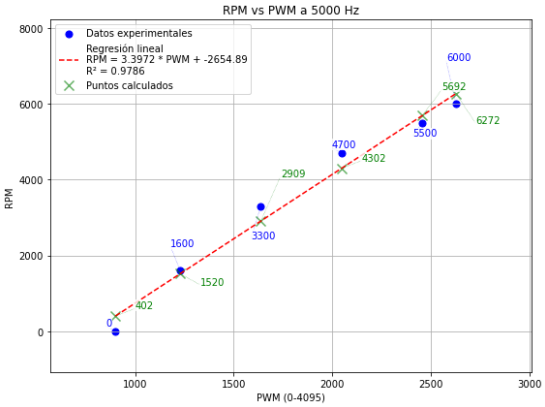
(c) 500 Hz

(d) 1 kHz

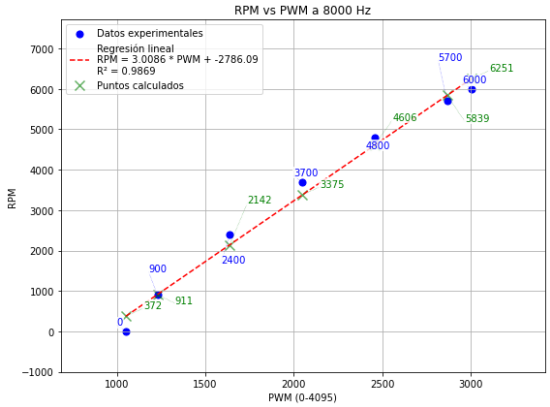
Figura 4.32: Gráficas de dispersión y regresión lineal para frecuencias PWM (100 Hz a 1 kHz)



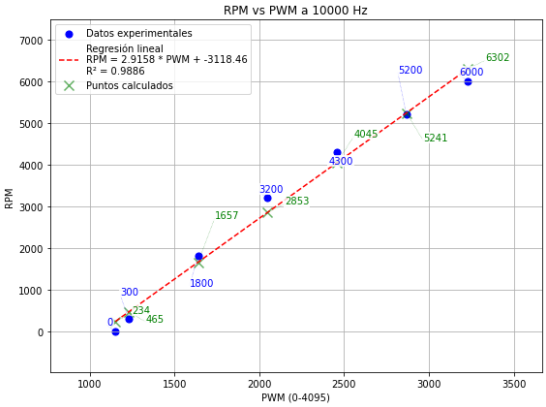
(a) 2 kHz



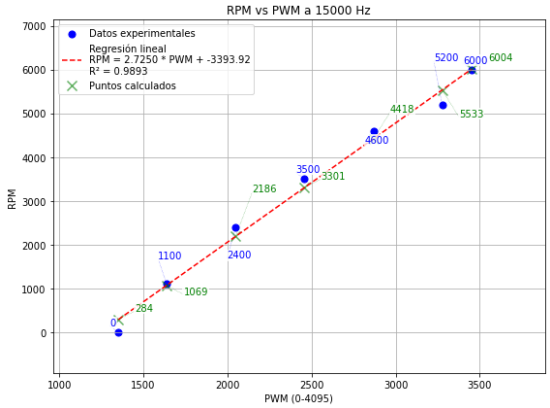
(b) 5 kHz



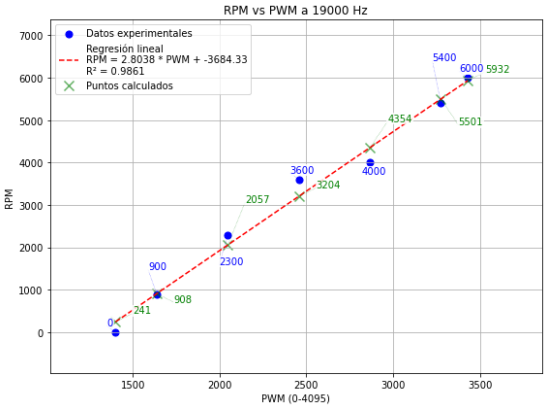
(c) 8 kHz



(d) 10 kHz



(e) 15 kHz



(f) 19 kHz

Figura 4.33: Gráficas de dispersión y regresión lineal para frecuencias PWM (2 kHz a 19 kHz)

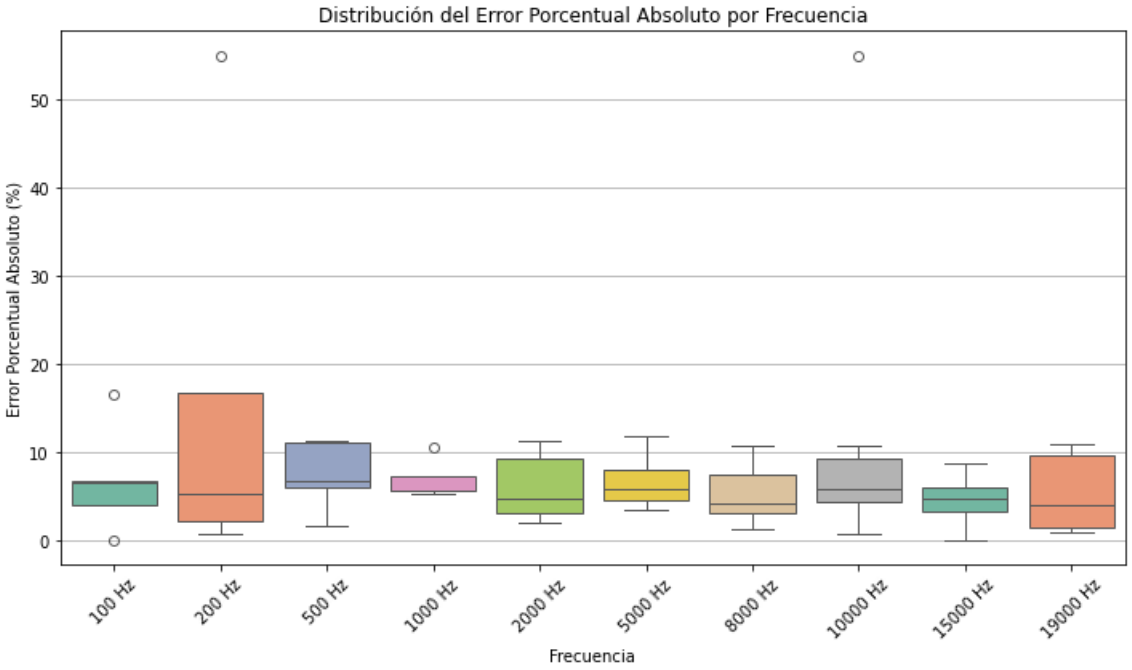


Figura 4.34: Boxplot del error porcentual absoluto para todas las frecuencias PWM

Frecuencia (Hz)	R^2	**Media EPA (%)**	**Desviación Estándar EPA (%)**
100	0.9712	6.80	5.44
200	0.9715	16.02	20.29
500	0.9771	7.41	3.58
1000	0.9780	6.92	1.98
2000	0.9919	6.05	3.63
5000	0.9786	6.67	2.81
8000	0.9869	5.39	3.15
10000	0.9886	12.78	17.49
15000	0.9893	4.64	2.59
19000	0.9861	5.47	4.18

Cuadro 4.1: Resumen de la caracterización del PWM para el motor RS-550

Análisis de las Frecuencias y Selección de 15 kHz

La caracterización del motor RS-550 a través de las frecuencias PWM (100 Hz, 200 Hz, 500 Hz, 1 kHz, 2 kHz, 5 kHz, 8 kHz, 10 kHz, 15 kHz, 19 kHz) reveló diferencias significativas en el rendimiento del control. A 100 Hz, se obtuvo una linealidad moderada ($R^2 = 0.9712$) con un error medio del 6.80 % y una desviación estándar elevada (5.44 %), sugiriendo vibraciones audibles que afectan la estabilidad del sensor. A 200 Hz, el desempeño fue el peor, con un $R^2 = 0.9715$, un error medio de 16.02 % y una desviación estándar de 20.29 %, indicando ruido significativo y baja consistencia. A 500 Hz y 1 kHz,

la linealidad mejoró ($R^2 = 0.9771$ y 0.9780) con errores medios de 7.41 % y 6.92 %, respectivamente, y desviaciones estándar más bajas (3.58 % y 1.98 %), reflejando una mayor estabilidad pero aún con errores notables.

A 2 kHz, se alcanzó una alta linealidad ($R^2 = 0.9919$) con un error medio de 6.05 % y una desviación de 3.63 %, mostrando potencial pero con margen de mejora. A 5 kHz y 8 kHz, los valores de R^2 fueron 0.9786 y 0.9869, con errores medios de 6.67 % y 5.39 %, y desviaciones de 2.81 % y 3.15 %, indicando un rendimiento sólido pero no óptimo. A 10 kHz, a pesar de una buena linealidad ($R^2 = 0.9886$), el error medio aumentó a 12.78 % con una desviación de 17.49 %, posiblemente debido a pérdidas de conmutación o ruido del sensor. A 15 kHz, se obtuvo el mejor equilibrio con un $R^2 = 0.9893$, el menor error medio (4.64 %) y la menor desviación estándar (2.59 %), reflejando una respuesta precisa y consistente, confirmada por una gráfica de dispersión con puntos bien alineados y un boxplot con distribución de errores compacta.

A 19 kHz, el R^2 fue 0.9861, con un error medio de 5.47 % y una desviación de 4.18 %, mostrando un rendimiento inferior a 15 kHz debido a pérdidas de conmutación en el módulo L298N. La operación a 15 kHz es inaudible, mejorando la comodidad del conductor comparada con frecuencias más bajas (100 Hz–2 kHz). Por lo tanto, 15 kHz fue seleccionada como la frecuencia ideal, optimizando linealidad, precisión, estabilidad y compatibilidad con el hardware.

Cálculo de la Velocidad

La velocidad del vehículo se calcula para su visualización en la interfaz gráfica utilizando la ecuación (4.20):

$$\text{Velocidad (km/h)} = \frac{\text{RPM} \times \text{Circunferencia de la rueda} \times 60}{\text{Relación de transmisión} \times 1000} \tag{4.20}$$

donde:

- Velocidad: Velocidad simulada del vehículo, en kilómetros por hora (km/h).
- RPM: Revoluciones por minuto del motor, en revoluciones por minuto (rpm), medidas por el sensor de efecto Hall.
- Circunferencia de la rueda: Circunferencia de la rueda, en metros (m), con un valor de 0.634 m.
- Relación de transmisión: Relación de engranajes para la marcha actual, sin unidades (adimensional), por ejemplo, 3.5 para la 1ª marcha.
- 60: Factor de conversión, en minutos por hora (min/h), para convertir revoluciones por minuto a revoluciones por hora.
- 1000: Factor de conversión, en metros por kilómetro (m/km), para convertir metros a kilómetros.

Para demostrar que la ecuación (4.20) produce unidades en km/h, consideremos el análisis dimensional. Las RPM están en revoluciones por minuto (rev/min), la circunferencia en metros por revolución (m/rev), por lo que RPM $\times$ Circunferencia de la rueda da metros por minuto (m/min). Multiplicando por 60 se obtiene metros por hora (m/h). Dividiendo por la relación de transmisión (adimensional) y por 1000 (para convertir metros a kilómetros) resulta en kilómetros por hora (km/h).

A continuación, se presenta la Tabla 4.2 con ejemplos de velocidades calculadas para diferentes marchas y valores de RPM dentro del rango de operación de 1500 a 2500 RPM para las marchas 1 a 4, utilizando la ecuación (4.20), incluyendo solo aquellos casos donde la velocidad es menor o igual a 220 km/h. Las relaciones de transmisión para las cinco marchas son 3.5, 2.0, 1.4, 1.0, y 0.8, respectivamente, y la circunferencia de la rueda es 0.634 m.

Marcha	**Relación de transmisión**	**RPM**	**Velocidad (km/h)**
1	3.5	1500	16.30
1	3.5	2500	27.17
2	2.0	1500	28.53
2	2.0	2500	47.55
3	1.4	1500	40.76
3	1.4	2500	67.93
4	1.0	1500	57.06
4	1.0	2500	95.10
5	0.8	1500	71.32
5	0.8	2500	118.87
5	0.8	4500	213.97

Cuadro 4.2: Velocidades calculadas para diferentes marchas en el rango de operación (1500–2500 RPM para marchas 1–4), con velocidades hasta 220 km/h

Implementación de la Caja de Cambios Automática

Para emular una transmisión automática, se implementó un sistema que simula el cambio de marchas basado en la velocidad del motor (RPM) y la entrada del pedal. En un vehículo con transmisión automática, la caja de cambios ajusta las marchas automáticamente según la velocidad del vehículo, la carga del motor y la posición del pedal de acelerador, utilizando un convertidor de par o un embrague automático para transmitir la potencia sin intervención manual. En esta emulación, el potenciómetro simula el pedal, y el sensor de efecto Hall A44E mide las RPM con una resolución de 1 pulso por imán, permitiendo detectar 8 pulsos por revolución. El sistema simula cinco marchas, con umbrales de RPM definidos para cambios ascendentes (por ejemplo, de 1ª a 2ª a 2500 RPM) y descendentes (por ejemplo, de 2ª a 1ª a 1500 RPM), considerando la entrada del pedal para una respuesta realista.

La velocidad del vehículo se calcula usando la ecuación (4.20). Las relaciones de transmisión para las cinco marchas son 3.5, 2.0, 1.4, 1.0 y 0.8, respectivamente. El ESP32 ajusta la salida PWM al cambiar de marcha, aplicando un factor de reducción (0.6) para

simular la disminución de torque en marchas más altas, y un factor de suavizado (0.1) para evitar cambios bruscos en la aceleración, emulando la respuesta de un acelerador real. Además, se implementó un impulso inicial (PWM $= 2800$, duración de 500 ms) para superar la inercia estática del motor, asegurando un arranque suave.

Contribución del Nodo al Estudio de Motores de Combustión Interna

El nodo del tren motriz permite estudiar la medición de RPM en motores de combustión interna (MCI). El sensor de efecto Hall A44E (8 pulsos por revolución) emula sensores de cigüeñal, capturando variaciones rápidas de RPM. La relación entre el potenciómetro (acelerador) y las RPM refleja la respuesta de un MCI. La caja de cambios automática simula puntos de cambio basados en RPM, mostrando cómo una ECU optimiza el rendimiento. La comunicación CAN replica la transmisión de datos en vehículos reales, facilitando el análisis de integración de sensores en entornos automotrices.

El diagrama de flujo del sistema se presenta en la Figura 4.35, mientras que el esquema de conexiones se detalla en la Figura 4.36.

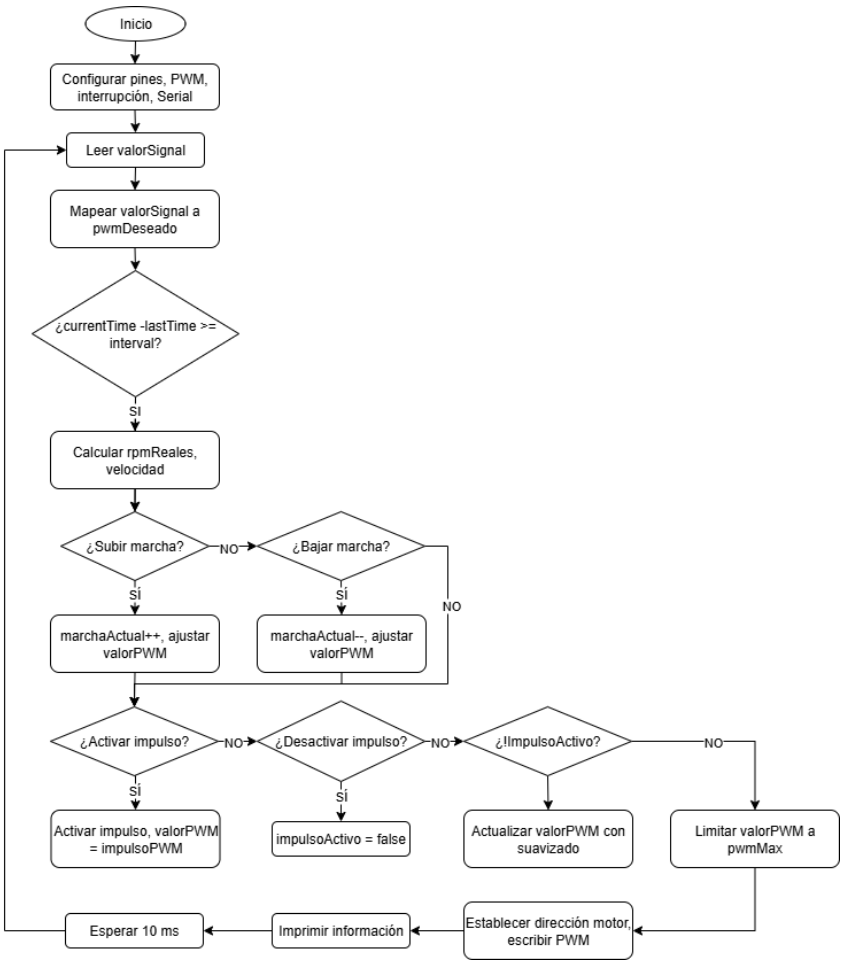


Figura 4.35: Diagrama de flujo del nodo del tren motriz

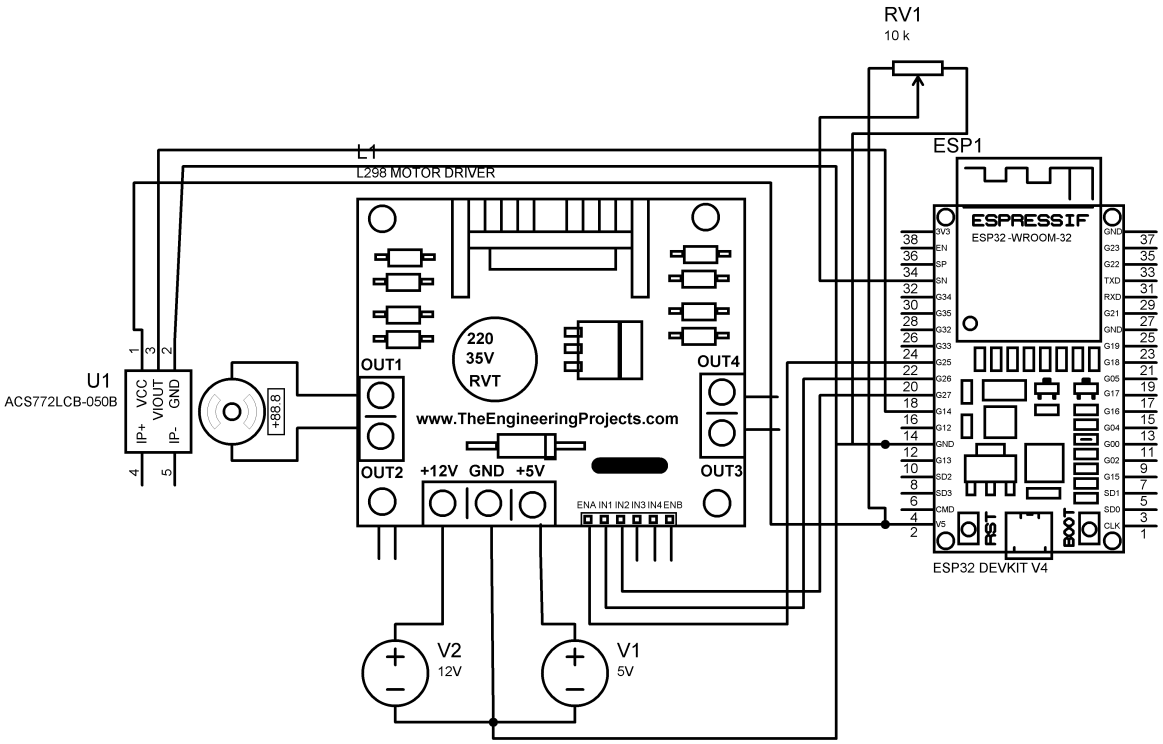


Figura 4.36: Diagrama de conexión del nodo del tren motriz

- 5 V y 12 V de la fuente de alimentación conectados a los respectivos voltajes del módulo L298N.
- ENABLE del L298N conectado al GPIO 25 del ESP32.
- IN1 e IN2 conectados a los GPIO 26 y 27, respectivamente.
- Señal del sensor de efecto Hall A44E conectada al GPIO 14.
- Señal del potenciómetro conectada al GPIO 34.

Siguiendo las instrucciones de conexión, se montó la parte física del nodo del tren motriz, como se muestra en la Figura 4.37.

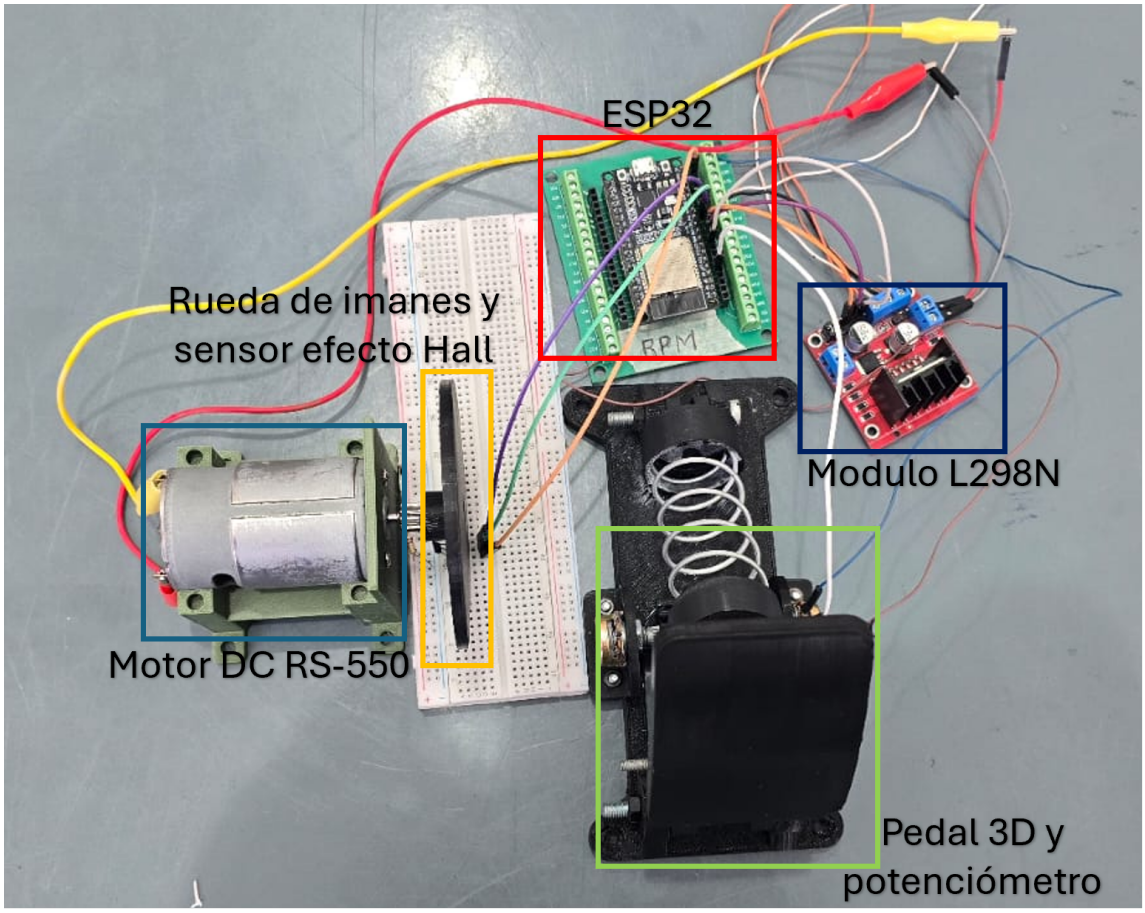


Figura 4.37: Implementación física del nodo del tren motriz

4.4.5. Implementación de la Red CAN

Se desarrollaron dos configuraciones para la red CAN, cada una diseñada para satisfacer diferentes necesidades en términos de hardware, eficiencia energética y simplicidad de integración. A continuación, se describen ambas implementaciones, incluyendo diagramas de conexión, tablas de pines y diagramas de flujo que ilustran los procesos de transmisión de datos.

1. **CAN Autónomo con ESP32 y MCP2515**:
    - **Configuración**: Cada nodo de la red utiliza un microcontrolador ESP32 conectado a un módulo MCP2515 mediante la interfaz SPI. Este módulo actúa como un controlador CAN externo, permitiendo al ESP32 enviar y recibir tramas CAN. Las conexiones entre el ESP32 y el MCP2515 se detallan en la Figura 4.40 (diagrama físico) y la Tabla 4.3 (asignación de pines).
    - **Operación**: El MCP2515 convierte las lecturas de sensores (como valores analógicos de temperatura o nivel de combustible) en tramas CAN, que son transmi-

tidas a una velocidad de 500 kbps. El nodo *gateway*, un nodo central en la red, recibe estas tramas, las procesa y, si es necesario, las reenvía a una interfaz externa, como una computadora.

- **Flujo de operación**:El proceso de transmisión de datos desde un nodo sensor se ilustra en el diagrama de flujo de la Figura 4.38, que muestra los pasos para inicializar la comunicación, leer datos del sensor y enviar tramas CAN.

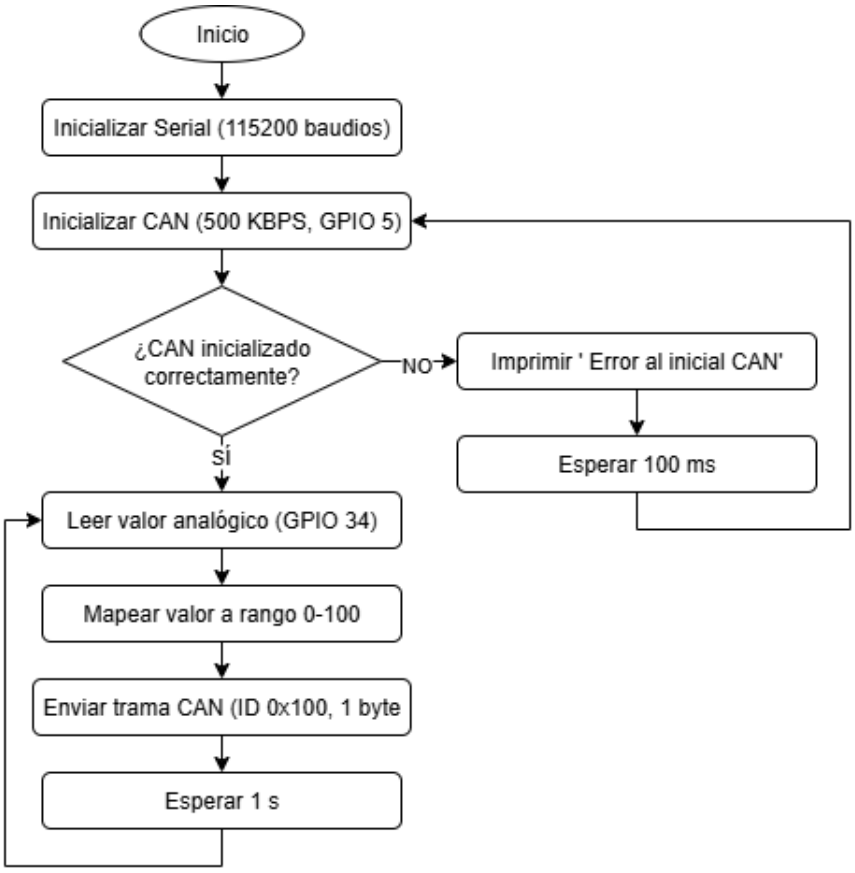


Figura 4.38: Diagrama de flujo para el transmisor CAN con ESP32 y MCP2515. Este diagrama describe el proceso de inicialización y transmisión de datos, desde la configuración del módulo CAN hasta el envío periódico de tramas.

2. **Implementación Final con ESP32 y SN65HVD230**:
   - **Configuración**: Esta configuración utiliza el controlador CAN integrado del ESP32, junto con el transceptor SN65HVD230, eliminando la necesidad de un controlador externo como el MCP2515. El transceptor convierte las señales digitales del ESP32 en señales diferenciales compatibles con el bus CAN. Las conexiones se muestran en la Figura 4.41 y la Tabla 4.4.
   - **Ventajas**: Esta implementación reduce el número de componentes, disminuye el consumo energético y es compatible con el nivel de voltaje de 3.3 V del

ESP32, lo que elimina la necesidad de circuitos adicionales de adaptación de voltaje.

- **Flujo de Operación**: El proceso de transmisión de datos se detalla en el diagrama de flujo de la Figura 4.39, que ilustra la inicialización del controlador CAN nativo y el envío de mensajes de prueba.

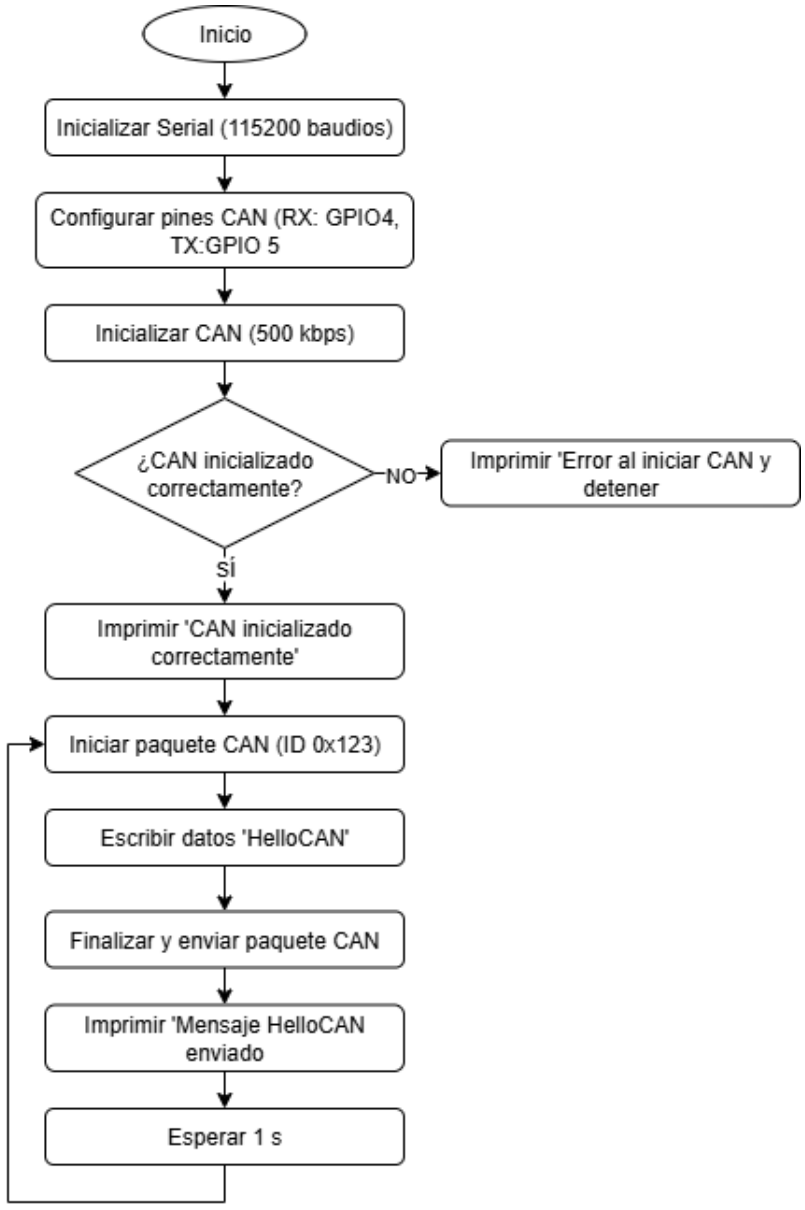


Figura 4.39: Diagrama de flujo para el transmisor CAN con ESP32 y SN65HVD230. Este diagrama muestra cómo se configura el controlador CAN nativo y se envían mensajes de prueba a través del bus CAN.

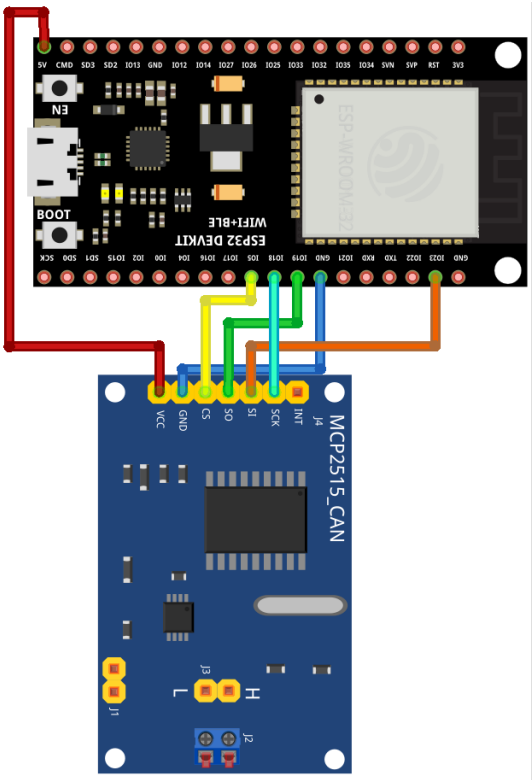


Figura 4.40: Diagrama de conexión entre el ESP32 y el MCP2515. Este esquema muestra cómo los pines del ESP32 se conectan al módulo MCP2515 para habilitar la comunicación SPI y la transmisión de datos CAN.

ESP32 (38 pines)	**MCP2515**	**Función**
5V	VCC	Alimentación
GND	GND	Tierra
GPIO 18 (SCK)	SCK	Reloj serial (SPI Clock)
GPIO 23 (MOSI)	SI	MOSI (Master Out Slave In)
GPIO 19 (MISO)	SO	MISO (Master In Slave Out)
GPIO 5 (CS)	CS	Selección de chip (Chip Select)

Cuadro 4.3: Conexiones entre el ESP32 (38 pines) y el MCP2515. Esta tabla detalla las conexiones físicas necesarias para la comunicación SPI entre ambos dispositivos.

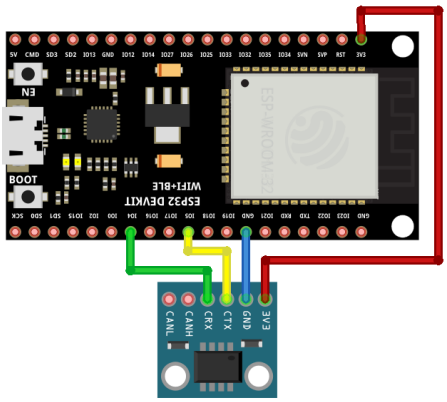


Figura 4.41: Diagrama de conexión entre el ESP32 y el SN65HVD230. Este esquema muestra las conexiones simplificadas para usar el controlador CAN nativo del ESP32 con el transceptor SN65HVD230.

ESP32 (38 pines)	SN65HVD230	Función
3.3 V	VCC	Alimentación
GND	GND	Tierra
GPIO 4	RX	RX SN65HVD230
GPIO 5	TX	TX SN65HVD230

Cuadro 4.4: Conexiones entre el ESP32 (38 pines) y el SN65HVD230. Esta tabla describe las conexiones necesarias para integrar el transceptor con el controlador CAN del ESP32.

Configuración de Software e Inicialización de CAN

La red CAN de cuatro nodos (combustible, temperatura, tren motriz y *gateway*) se configuró utilizando el entorno de desarrollo integrado (IDE) de Arduino y la biblioteca `CAN.h`, diseñada específicamente para interactuar con el controlador CAN nativo del ESP32. Esta biblioteca simplifica la configuración y gestión de la comunicación CAN. La inicialización estableció una velocidad de transmisión de 500 kbps, asignó los pines GPIO 5 (transmisión, TX) y GPIO 4 (recepción, RX), y configuró interrupciones para procesar las tramas de manera eficiente, permitiendo una comunicación fluida entre los nodos.

- **Instalación**: La biblioteca `CAN.h` se instaló mediante el Administrador de Bibliotecas del IDE de Arduino, proporcionando soporte nativo para el controlador CAN integrado del ESP32. Esta biblioteca es esencial para configurar y operar la red CAN sin hardware adicional.
- **Configuración Detallada**:La inicialización definió los parámetros de velocidad (500 kbps) y asignó los pines GPIO, con un mecanismo de reintento para garanti-

zar un arranque robusto en caso de fallos iniciales. Este proceso se ilustra en el diagrama de flujo de la Figura 4.42.

- **Consideraciones**: La integración nativa con `CAN.h` elimina la latencia asociada con la interfaz SPI (usada en el MCP2515), reduciendo el tiempo de respuesta a aproximadamente 5 microsegundos por trama. Las interrupciones se configuraron para procesar tramas entrantes de manera asíncrona, optimizando la eficiencia de la red de cuatro nodos.

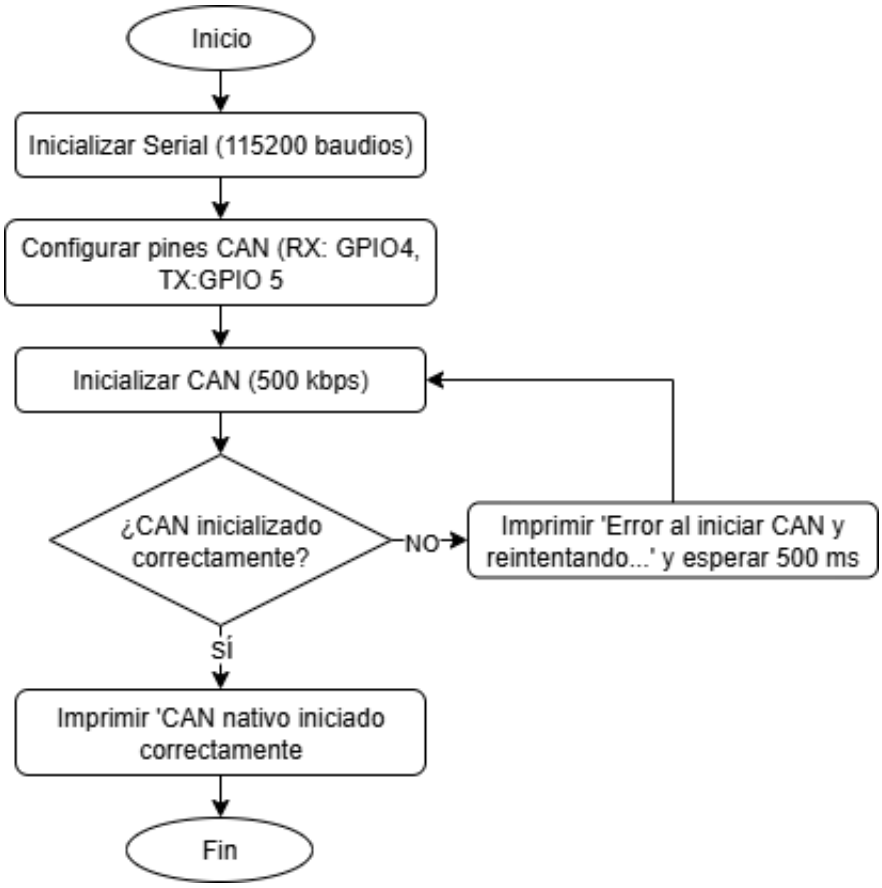


Figura 4.42: Diagrama de flujo para la inicialización del controlador CAN. Este diagrama muestra los pasos para configurar el controlador CAN nativo del ESP32, incluyendo la asignación de pines y la verificación de la inicialización.

- **Gestión de Interrupciones**: Las interrupciones se implementaron en memoria RAM para minimizar retrasos, permitiendo al ESP32 procesar tramas CAN entrantes en tiempo real. Este proceso se detalla en el diagrama de flujo de la Figura 4.43, que muestra cómo se manejan los datos recibidos.

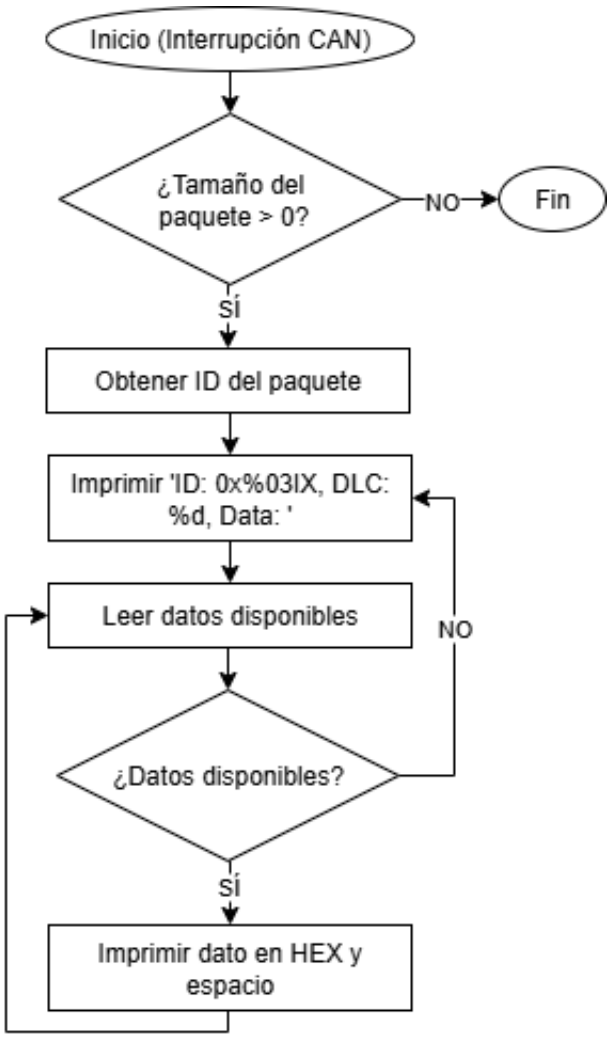


Figura 4.43: Diagrama de flujo para la gestión de interrupciones CAN. Este diagrama ilustra cómo el ESP32 procesa tramas CAN entrantes, imprimiendo su ID y datos en formato legible.

Enmarcado de Datos y Transmisión

El enmarcado de datos para la red CAN de cuatro nodos con ESP32 y SN65HVD230 utilizó tramas CAN estándar de 11 bits, diseñadas para ser compactas y eficientes. El campo de longitud de datos (DLC) varía entre 1 y 8 bytes, adaptándose a los requisitos de cada nodo (combustible, temperatura, tren motriz y *gateway*). La estructura de las tramas incluye identificadores únicos, campos de datos escalados, verificación de redundancia cíclica (CRC) generada por hardware y acuse de recibo (ACK) implícito, asegurando una comunicación confiable.

- **Razón para Usar Identificadores de 11 Bits (CAN Clásico)**:La configuración de identificadores de 11 bits, correspondiente al protocolo CAN clásico (CAN 2.0A),

se seleccionó para esta red por varias razones. Primero, los identificadores de 11 bits permiten hasta 2048 IDs únicos, más que suficientes para una red de cuatro nodos, simplificando la gestión de mensajes. Segundo, tanto el controlador CAN nativo del ESP32 como el MCP2515 son totalmente compatibles con este formato, garantizando una integración sin problemas. Tercero, los identificadores de 11 bits reducen la sobrecarga en cada trama CAN, lo que mejora la velocidad de transmisión y reduce la carga del bus, aspectos críticos para aplicaciones en tiempo real como el monitoreo de vehículos. Finalmente, el formato de 11 bits es un estándar ampliamente utilizado en la industria automotriz e industrial, lo que asegura compatibilidad con herramientas y protocolos existentes, facilitando el desarrollo y la depuración.

- **Identificadores (ID)**: Cada nodo fue asignado un ID específico: 0x02F para el nodo de combustible, 0x102 para el nodo de temperatura, 0x123 para la velocidad del tren motriz en el primer byte y las RPM del tren motriz en los dos últimos bytes.
- **Campo de Datos**:Los datos se escalaron dinámicamente para adaptarse a los rangos de los sensores: 0–100 para el porcentaje de combustible, 0–130 para temperatura (°C) y velocidad (km/h), y 0–8000 para RPM (dividido en dos bytes para mayor precisión).
- **CRC y ACK**: El hardware del ESP32 genera automáticamente el CRC para garantizar la integridad de las tramas, mientras que el ACK es confirmado por la presencia de al menos un receptor activo en la red, asegurando una comunicación robusta.
- **Flujo de Transmisión**: El proceso de transmisión del nodo del tren motriz, que incluye el cálculo de RPM y velocidad, se muestra en el diagrama de flujo de la Figura 4.44. Este nodo envía datos al *gateway*, que los reenvía a una interfaz gráfica (GUI) mediante comunicación serial.

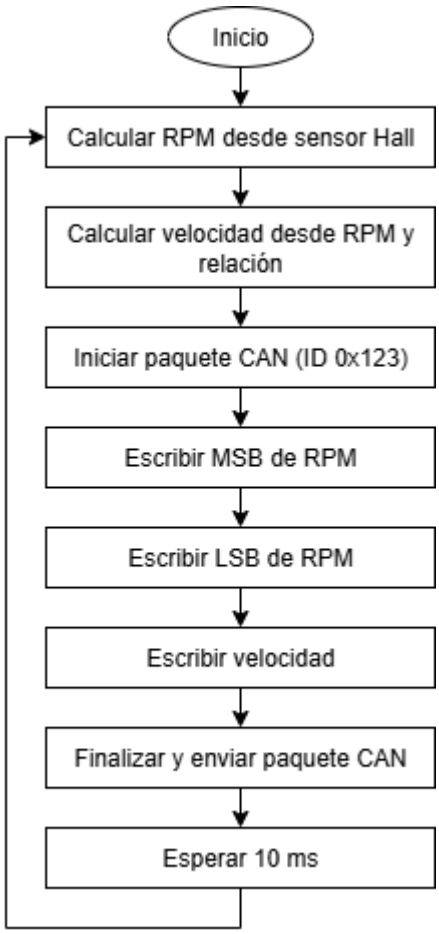


Figura 4.44: Diagrama de flujo para la transmisión de datos del nodo del tren motriz. Este diagrama muestra cómo se calculan y envían los datos de velocidad y RPM a través del bus CAN.

4.4.6. Montaje de la Red de Cuatro Nodos

El montaje de la red CAN de cuatro nodos con ESP32 y SN65HVD230, integra los nodos de combustible, temperatura, tren motriz y *gateway*; el cual se conecto con la GUI de cuadro de instrumentos. Las figuras 4.45 y 4.46 ilustran el montaje, mostrando los cuatro ESP32, los transceptores y el cableado trenzado.Posteriormente en las figuras 4.47a, 4.47b, 4.47c, 4.47d se muestran la implementación final de cada nodo; cabe mencionar que se utilizaron Conectores de Bloque de Terminal de Cable para interconectar las tierras y el voltaje, ademas de conectores rápidos para interconectar la red can; por ultimo, en la figura 4.48 se ilustra la conexión por par trenzado del bus can entre el gateway y el nodo del tren motriz.

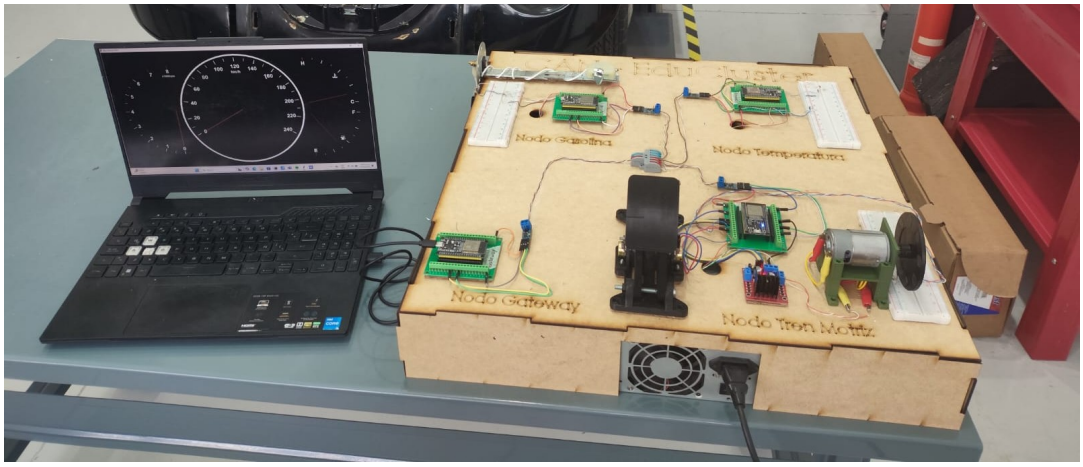


Figura 4.45: Montaje físico de la red CAN de cuatro nodos (vista general)

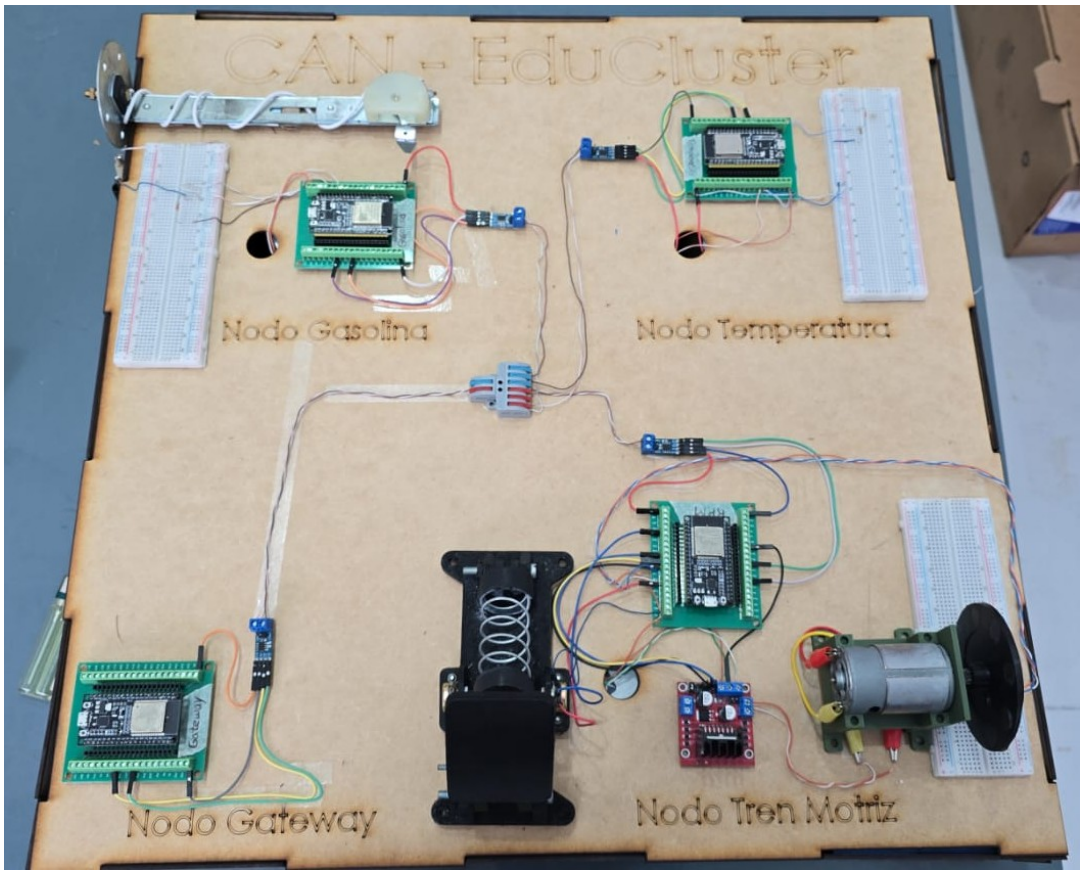
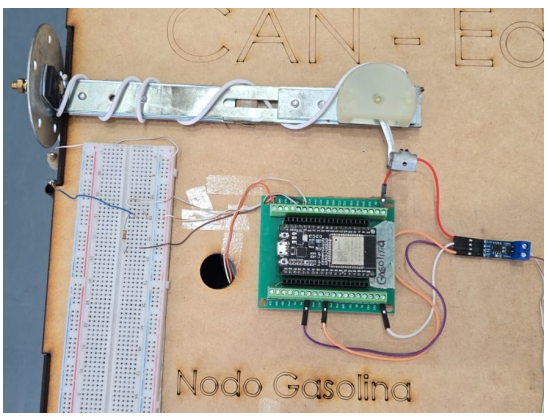
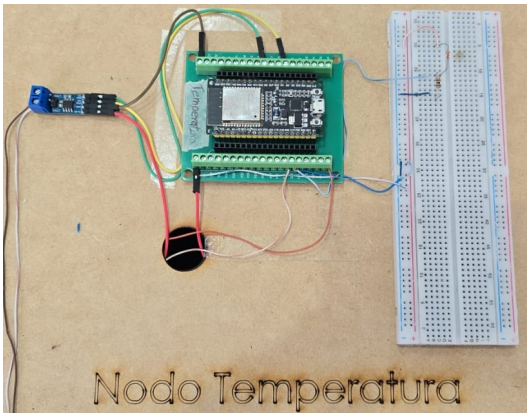


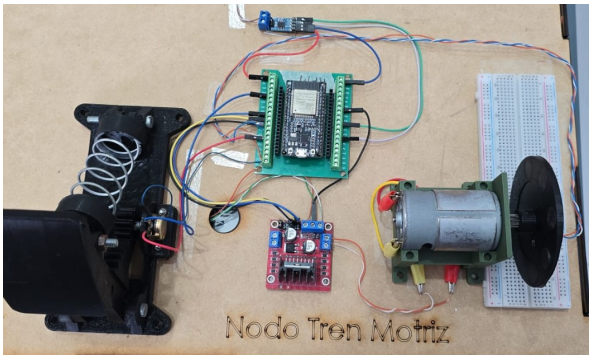
Figura 4.46: Montaje físico de la red CAN de cuatro nodos (detalle de la red)



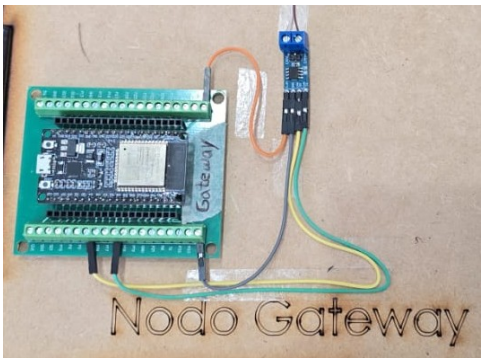
(a) Nodo Gasolina



(b) Nodo de temperatura



(c) Nodo del tren motriz



(d) Nodo gateway

Figura 4.47: Red can por nodos.

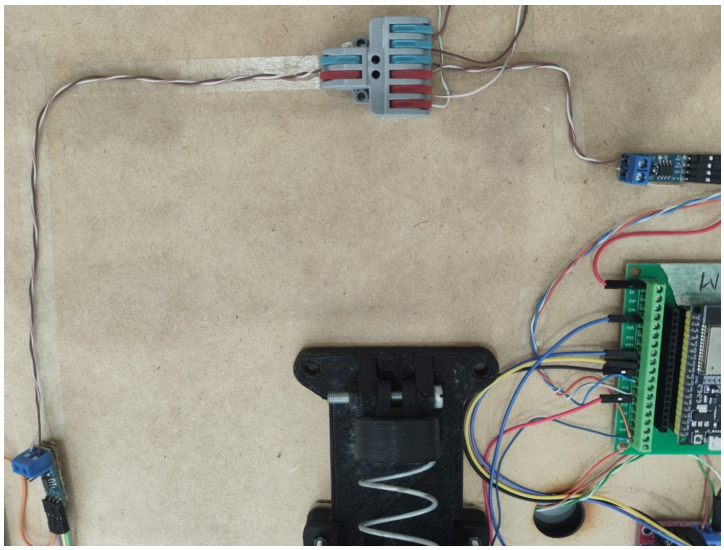


Figura 4.48: Conexión por medio de bus can entre transceptor de nodo tren motriz y gateway

Ya con la red armada y los identificadores asignados, procedimos a realizar pruebas con la interfaz conectada a la red ya armada, para poder visualizar la respuesta de la interfaz a los estímulos de cada nodo, como se observa en la figura 4.49, en donde se esta accionando el pedal y cambia el valor de rpm y velocidad dentro de la interfaz.

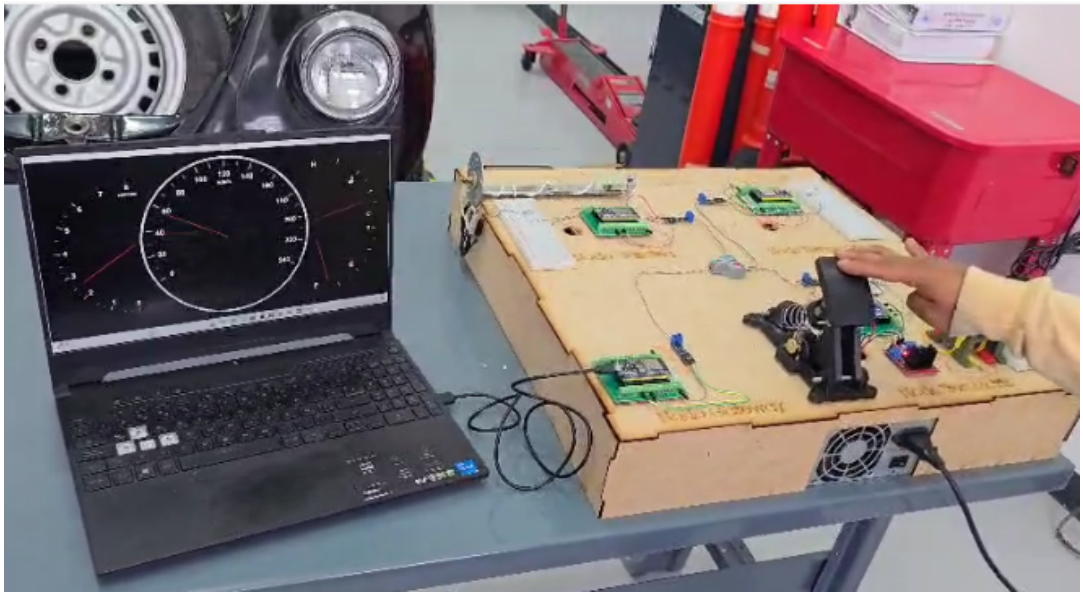


Figura 4.49: Accionamiento de pedal en el nodo del tren motriz y su interacción con la interfaz.

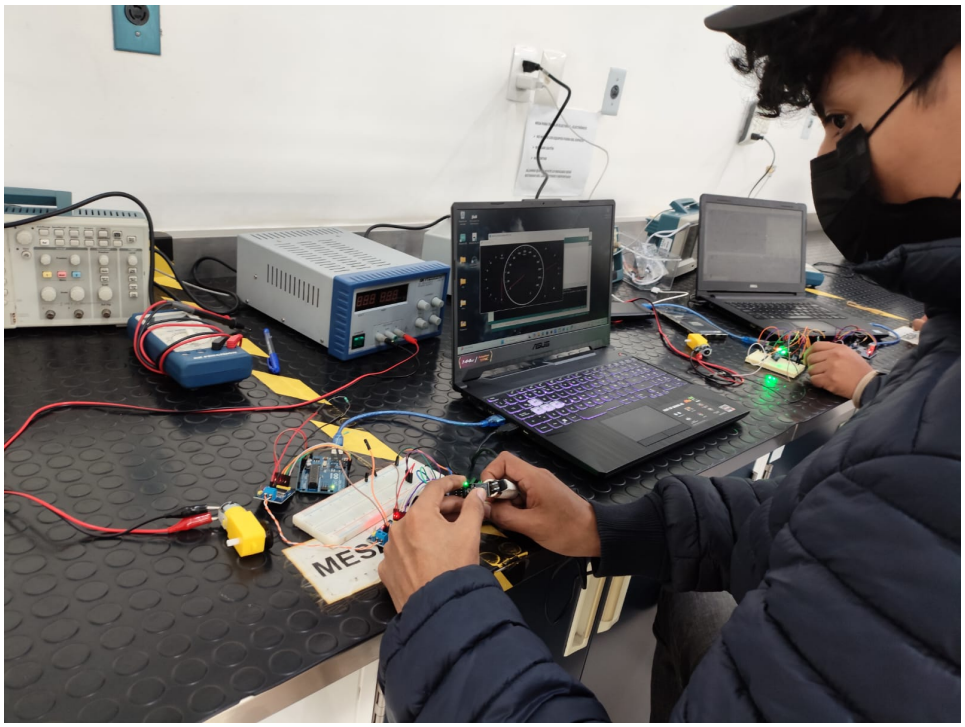
De igual manera, se realizaron pruebas con un grupo de alumnos de la carrera automotriz, los cuales probaron la red can implementada, junto con la GUI de cuadro de instrumentos. Esta interacción se puede revisar en un video de youtube.(url: https://youtu.be/jJNJJ9_5TIg).

Ademas de esa prueba, la Interfaz fue utilizada en la materia de Interfaces y Redes Vehiculares correspondiente al marco curricular de la carrera de Ingeniería en Sistemas Automotrices, en donde, los estudiantes conectaron sus redes CAN a la interfaz gráfica diseñada, utilizando los siguientes identificadores:

- **101**: Temperatura (°C, 0-130).
- **102**: Nivel de combustible (%, 0-100).
- **103**: Velocidad (km/h, 0-240).
- **104**: RPM (0-8000).

Toda la información necesaria se encuentra dentro del manual de usuario que se genero para la Interfaz Gráfica, en las figuras 4.50a y 4.50b se puede observar a los estudiantes utilizando la interfaz y conectándola a sus redes CAN.

(a) Estudiante A utilizando interfaz gráfica de cuadro de instrumentos.



(b) Estudiante B utilizando interfaz gráfica de cuadro de instrumentos.

Figura 4.50: Estudiantes usando GUI de cuadro de instrumentos con su propia red CAN.

Después de su uso solicitamos que contestaran un cuestionario, para conocer su opinión con base a la interfaz y el manual de usuario. En general, los estudiantes mostraron una alta satisfacción tanto con el manual como con la interfaz gráfica del cuadro de instrumentos. El manual destaca por su lenguaje claro, diagramas útiles y sección de solución de problemas, aunque la mayoría necesitó buscar información adicional para la configuración inicial, lo cual sugiere un área de mejora.

Respecto a la interfaz gráfica, ésta resultó intuitiva, con indicadores y advertencias claras, y un diseño atractivo y profesional. La mayoría de los usuarios completó la instalación y comenzó a utilizar el programa en menos de 10 minutos, y la totalidad de los encuestados recomendaría la solución a otros. Esto respalda que el conjunto manual + GUI cumple efectivamente con los objetivos de usabilidad y diseño para entornos académicos.

También se desarrollo un manual de practicas para poner a prueba posteriormente con grupos de la materia de interfaces y redes vehiculares.

4.4.7. Análisis de trama de datos CAN

Para validar la correcta transmisión de los datos dentro de nuestra red CAN, se analizaron las tramas de datos CAN capturadas durante un período de 41 segundos utilizando un analizador lógico de 8 canales y el software Logic 2.El objetivo fue monitorear y verificar el comportamiento de tres nodos en la red CAN: el nodo de temperatura (identificador 0x102), el nodo de gasolina (identificador 0x02F) y el nodo de tren motriz (identificador 0x123). Cada nodo tiene condiciones específicas para el envío de mensajes, las cuales se detallan a continuación. Se incluyen imágenes que muestran los 41 segundos de mensajes enviados y la composición desglosada de las tramas de datos de cada identificador, además de un diagrama de conexión del analizador lógico.

Configuración del Analizador Lógico

Para la captura de las tramas CAN, se utilizó un analizador lógico de 8 canales conectado a la red CAN. Los canales se configuraron para capturar la señal CAN_H y CAN_L, permitiendo la decodificación de los mensajes transmitidos en el bus. El software Logic 2 se empleó para visualizar y analizar los datos capturados, utilizando su funcionalidad de analizador de protocolo CAN para decodificar automáticamente las tramas en sus componentes: identificador, campo de control, datos, CRC y ACK.

En Logic 2, se asignaron dos canales específicos para CAN_H y CAN_L (por ejemplo, canal 0 y canal 1), y se seleccionó el baud rate correspondiente a la red CAN (típicamente 500 kbps, ajustado según el sistema). La configuración incluyó la activación del protocolo CAN en el software, especificando los canales asignados y habilitando la decodificación automática(figura 4.51). Una vez configurado, se inició la captura durante 41 segundos, generando un registro detallado exportado como CSV, que incluye los tiempos de inicio, duración y contenido de cada campo de las tramas.

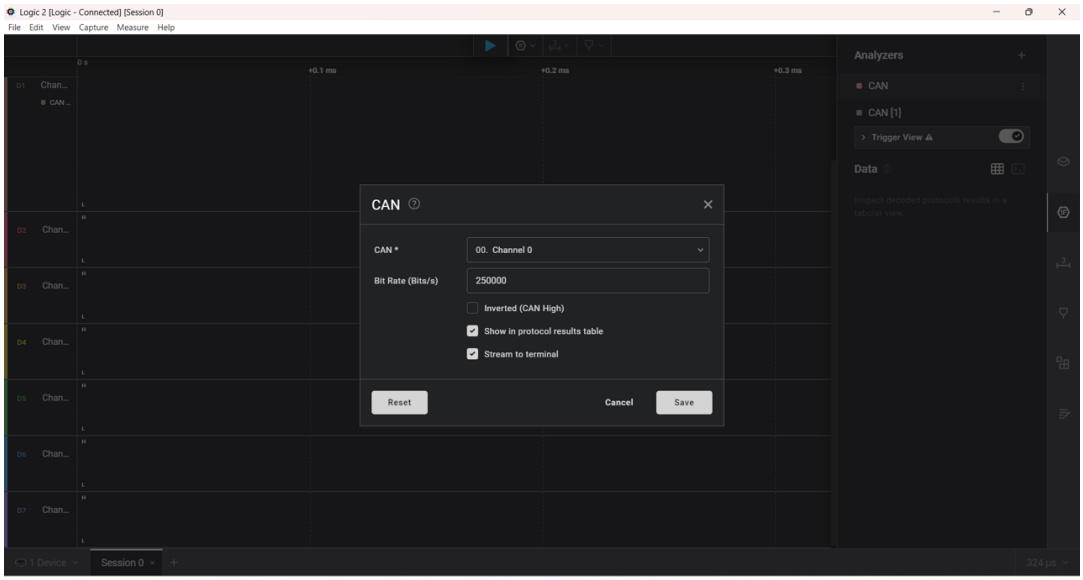


Figura 4.51: Configuración del analizador lógico conectado a la red CAN.

Nodo de Temperatura (0x102)

Especificación: Envía mensajes cada 500 ms cuando la temperatura es inferior a 100 y cada 200 ms cuando es superior a 100.

Observación: En el registro, los mensajes se enviaron cada aproximadamente 500 ms (por ejemplo, 0.14578125 s, 0.6458985 s, 1.146017125 s, con diferencias de $\sim$0.500 s). No se observaron intervalos de 200 ms, lo que indica que la temperatura permaneció por debajo de 100 durante todo el período.

Datos: Los valores de datos fueron principalmente 0x15 (21 en decimal) y ocasionalmente 0x14 (20 en decimal), consistentes con temperaturas bajas.

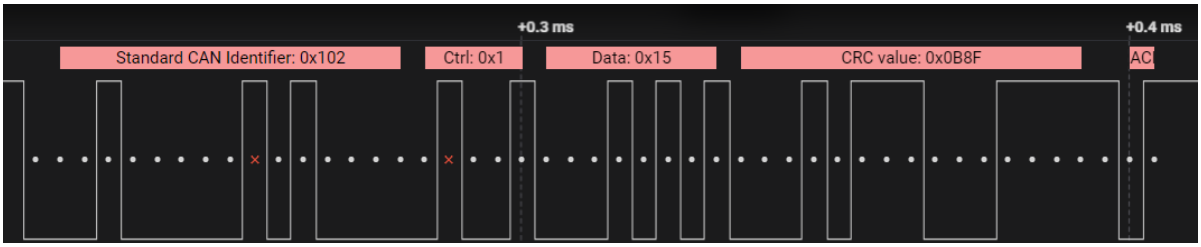


Figura 4.52: Trama de datos del nodo de temperatura con desglose de campos.

Nodo de Gasolina (0x02F)

Especificación: Envía mensajes cada 1 segundo cuando el nivel de gasolina es mayor al 10 % y cada 500 ms cuando es menor o igual al 10 %.

Observación: Los mensajes se enviaron cada aproximadamente 1 segundo (por ejemplo, 0.19135275 s, 1.191604625 s, 2.1918525 s, con diferencias de ∼1.000 s), sugiriendo que el nivel de gasolina estuvo por encima del 10 % durante todo el período.

Datos: Los valores de datos, como 0x24 (36 %), 0x25 (37 %), y 0x26 (38 %), confirman niveles superiores al 10 %.

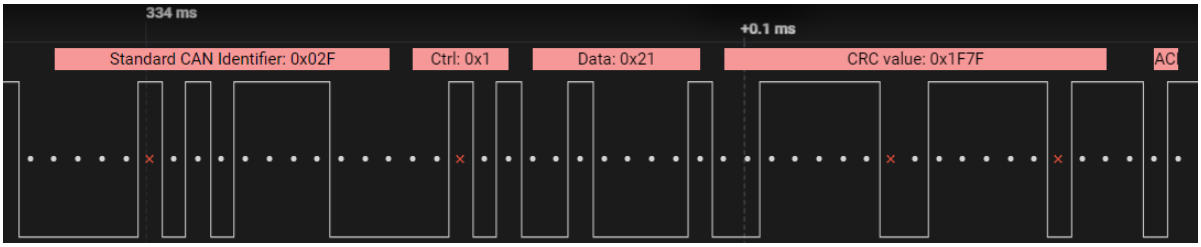


Figura 4.53: Trama de datos del nodo de gasolina con desglose de campos.

Nodo de Tren Motriz (0x123)

Especificación: Envía mensajes cada 500 ms, con el primer byte representando la velocidad y los siguientes dos bytes las RPM.

Observación: Los mensajes se enviaron cada aproximadamente 500 ms (por ejemplo, 0.43863775 s, 0.939776375 s, 1.440917875 s, con diferencias de ∼0.500 s), coincidiendo con la especificación proporcionada.

Datos: Los datos son coherentes con la estructura esperada. Por ejemplo, a los 3.946614 s, los bytes son 0x19, 0xC8, 0x06 (velocidad = 25, RPM = 0x06C8 = 1736).

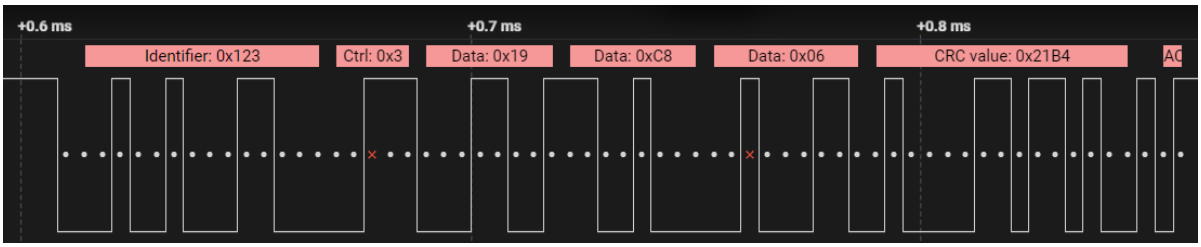


Figura 4.54: Trama de datos del nodo de tren motriz con desglose de campos.

Vista General de Mensajes

La siguiente figura muestra una vista general de todos los mensajes CAN capturados durante los 41 segundos de registro. Se pueden observar los diferentes identificadores y la frecuencia de sus mensajes a lo largo del tiempo.

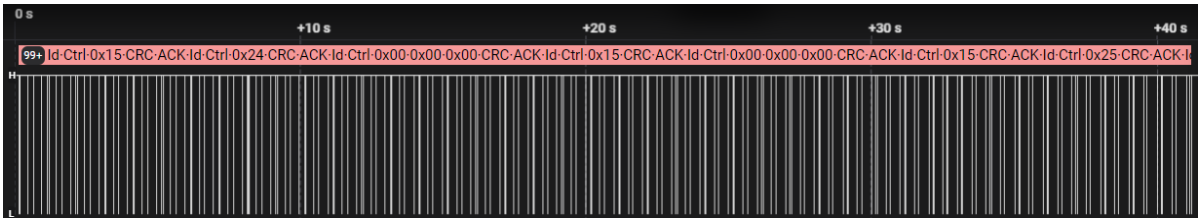


Figura 4.55: Vista general de los mensajes CAN en el tiempo.

Análisis de Resultados

El uso del analizador lógico de 8 canales junto con Logic 2 permitió una captura y visualización precisa de las tramas CAN, facilitando la verificación del comportamiento de los nodos. La decodificación automática y el registro detallado fueron esenciales para analizar los tiempos de envío y los datos transmitidos. Las imágenes complementarias, incluyendo los 41 segundos de mensajes y la composición desglosada de las tramas, reforzaron la comprensión del sistema.

En el nodo de temperatura, la consistencia de los intervalos de 500 ms y los valores de datos bajos confirman que la temperatura se mantuvo por debajo de 100 durante todo el período. Para el nodo de gasolina, los intervalos de 1 segundo y los valores de datos superiores al 10 % indican un nivel de combustible adecuado. En el nodo de tren motriz, los mensajes se enviaron cada 500 ms, tal como se especifica, y los datos de velocidad y RPM son coherentes con la estructura esperada.

Este análisis permitió confirmar que todos los nodos funcionaron según sus especificaciones durante el período de captura, proporcionando una visión clara de la dinámica de la red CAN y la integridad de los datos transmitidos.

4.4.8. Comparación de Costos de Soluciones

Por ultimo, se presenta una comparación de costos entre la solución propuesta basada en ESP32 y CAN, y soluciones comerciales y DIY que pueden configurarse con 4 nodos (3 transmisores y 1 gateway) y mostrar información mediante puerto serial o una interfaz gráfica de usuario (GUI). Los precios están expresados en pesos mexicanos (MXN), utilizando una tasa de cambio aproximada de 20 MXN por 1 USD. La tabla también evalúa la facilidad de replicación de cada solución y proporciona enlaces a las fuentes para mayor información.

Solución	Costo (MXN)	Componentes Incluidos	Facilidad de Replicación	Enlace
Solución Propuesta	1200–2000	4x ESP32, 4x SN65HVD230, sensores (nivel, termistor, Hall), motor RS-550, GUI en Python	Alta	EduCluster
NI USB-8502	20000	Interfaz CAN USB, soporte CAN FD, software NI-XNET/LabVIEW, salida serial/GUI	Baja	NI
Vector VN1610	30000	Interfaz CAN/CAN FD, 2 canales, software CANoe/CANalyzer, salida serial/GUI	Baja	Vector
ESP32-CAN-X2	4000	4x ESP32-S3-WROOM-1, controlador CAN dual, salida serial/GUI personalizable	Media	Autosport Labs

Cuadro 4.5: Comparación de soluciones para cuadro de instrumentos

Analizando el cuadro 4.5, la solución propuesta, basada en módulos ESP32 y comunicación CAN, se destaca como una opción óptima debido a sus múltiples ventajas. Con un costo significativamente menor (1200–2000 MXN) frente a soluciones comerciales como NI USB-8502 (20000 MXN), Vector VN1610 (30000 MXN) y ESP32-CAN-X2 (4,000 MXN), ofrece una alternativa accesible para entusiastas y desarrolladores. Su facilidad de replicación, gracias al uso de componentes ampliamente disponibles y software de código abierto, supera a las soluciones propietario que requieren hardware especializado y licencias costosas. Además, la solución incluye una interfaz gráfica intuitiva lista para usar, a diferencia de alternativas que demandan desarrollo adicional. La flexibilidad para integrar sensores personalizados y lógica avanzada, como la caja de cambios automática, asegura una funcionalidad completa y adaptable, posicionándola como una opción ideal para aplicaciones automotrices de bajo costo y alto rendimiento.

Capítulo 5

Conclusiones

La tesis culmina con la demostración exitosa de la integración de componentes de software de código abierto, específicamente Python con la librería Tkinter, y hardware de bajo costo, utilizando microcontroladores ESP32 y transceptores CAN, para simular de manera efectiva una red vehicular y un cuadro de instrumentos automotriz virtual. Se logró diseñar e implementar un prototipo funcional robusto y confiable que consta de cuatro nodos interconectados a través de un bus CAN. En esta arquitectura, uno de estos nodos opera estratégicamente como un gateway, facilitando la transmisión bidireccional y eficiente de datos desde la red CAN hacia la interfaz gráfica de usuario (GUI) desarrollada.

La implementación práctica de este sistema evidenció de manera contundente la viabilidad y robustez de utilizar microcontroladores ESP32 junto con transceptores CAN (SN65HVD230) para establecer una red CAN confiable y eficiente. Cada nodo de la red fue meticulosamente configurado para simular la captura de datos específicos y críticos para un vehículo, como el nivel de combustible, la temperatura del motor y diversos parámetros del tren motriz. Estos datos eran encapsulados y enviados a través del bus CAN de manera sincrónica. El nodo gateway fue el encargado crucial de recibir estos flujos de datos y transmitirlos serialmente a la computadora donde se ejecutaba la GUI, asegurando una comunicación fluida entre el hardware de bajo costo y el software de visualización.

Las pruebas exhaustivas realizadas con la participación activa de estudiantes de la carrera de sistemas automotrices fueron fundamentales y decisivas para validar la funcionalidad integral de la red CAN y la interacción fluida con la interfaz gráfica de usuario. Estas pruebas no solo confirmaron que los datos simulados eran recibidos y representados con alta precisión en el cuadro de instrumentos virtual, sino que también permitieron a los usuarios visualizar cambios en tiempo real y comprender de forma intuitiva la dinámica y el comportamiento de una red vehicular. Un aspecto crucial y diferenciador de estas pruebas fue la comparación meticulosa de los costos de esta solución de bajo presupuesto con los de laboratorios comerciales equivalentes. Esta comparación reveló un ahorro económico sustancial y significativo, lo que posiciona a esta propuesta como una alternativa viable, atractiva y democrática para instituciones educativas y centros de investigación con recursos financieros limitados. En síntesis, la tesis demuestra la construcción exitosa de un entorno de aprendizaje práctico, inmersivo y

accesible para la ingeniería automotriz, al tiempo que valida la eficacia y el potencial de la tecnología de código abierto y hardware de bajo costo en la resolución de desafíos complejos en aplicaciones como las redes vehiculares.

5.1. Contribuciones Originales

Las contribuciones originales de esta tesis incluyen:

- **Desarrollo de un Cuadro de Instrumentos Automotriz Virtual Accesible**: La creación de una interfaz gráfica de usuario (GUI) en Python (Tkinter) que simula un cuadro de instrumentos automotriz, la cual se puede ejecutar en un sistema operativo Windows, es una herramienta educativa de bajo costo. Este software se puede instalar y usar sin depender de un entorno de desarrollo específico.
- **Implementación de una Red CAN de Bajo Costo y Cuatro Nodos**: Se diseñó y montó una red CAN con cuatro nodos (combustible, temperatura, tren motriz y gateway) utilizando hardware de código abierto como el ESP32 y transceptores SN65HVD230. Esta implementación ofrece una alternativa económica para el estudio y la experimentación con redes vehiculares.
- **Integración y Comunicación Serial Robusta**: Se estableció una comunicación serial efectiva entre el gateway (ESP32) de la red CAN y la GUI de Tkinter, permitiendo la visualización en tiempo real de los datos de los sensores y actuadores. La optimización del manejo de interrupciones y el enmarcado de datos CAN contribuyeron a una transmisión eficiente.
- **Metodología de Bajo Costo para Laboratorios Educativos**: La tesis presenta una metodología viable para implementar laboratorios de interfaces y redes vehiculares con una inversión significativamente menor en comparación con las soluciones comerciales existentes. Esto democratiza el acceso a la formación práctica en sistemas automotrices para instituciones con presupuestos limitados.

5.2. Trabajo a futuro

El trabajo a futuro de este proyecto contempla las siguientes líneas:

- **Expansión de Nodos y Sensores/Actuadores**: Aumentar el número de nodos en la red CAN y diversificar los tipos de sensores y actuadores simulados para replicar escenarios vehiculares más complejos. Esto incluiría la integración de elementos como sistemas de frenos ABS, bolsas de aire o sistemas de dirección asistida.
- **Implementación de Otros Protocolos de Comunicación**: Explorar la integración de otros protocolos de comunicación vehicular como LIN, MOST o FlexRay, para ofrecer una plataforma más completa para el estudio de redes automotrices.

- **Mejoras en la Interfaz Gráfica**: Desarrollar funcionalidades adicionales para la GUI, como la capacidad de configuración dinámica de los instrumentos, la inclusión de gráficos históricos de datos, o la simulación de fallas y diagnósticos más avanzados.
- **Validación con Hardware Real de Vehículos**: Realizar pruebas de integración con sistemas OBD-II y CAN reales de vehículos para validar la precisión y compatibilidad de la simulación con entornos automotrices auténticos.
- **Desarrollo de Módulos de Entrenamiento Interactivos**: Crear módulos de entrenamiento específicos dentro de la GUI que permitan a los usuarios interactuar con la simulación para comprender mejor el funcionamiento de los sistemas automotrices y el diagnóstico de fallas.
- **Portabilidad a Otras Plataformas**: Adaptar la GUI y la lógica de control para que puedan ejecutarse en otras plataformas o dispositivos, como tabletas o sistemas embebidos, para aumentar la flexibilidad y accesibilidad del sistema.
- **Análisis de Seguridad en la Red CAN**: Investigar la seguridad de la red CAN implementada y explorar posibles vulnerabilidades, así como proponer mecanismos de seguridad para proteger la integridad de los datos en entornos vehiculares.

Apendice A: Recursos del Proyecto

Todos los recursos desarrollados como parte de este proyecto de tesis, incluyendo el código fuente de los nodos del bus CAN, el ejecutable de la interfaz gráfica de usuario (GUI), el manual de prácticas para estudiantes, el manual de usuario, los archivos STL para la impresión 3D de componentes específicos y las hojas de datos (datasheets) de los sensores utilizados, se encuentran disponibles públicamente para su acceso y descarga.

Estos materiales han sido depositados en el siguiente repositorio de GitHub, facilitando su replicación, estudio y mejora por parte de la comunidad académica y de investigación:

URL del Repositorio de GitHub:https://github.com/SergioOriz/CAN-EduCluster

Este repositorio está estructurado para proporcionar un acceso intuitivo a los siguientes componentes:

- **Firmware de los Nodos**: Código fuente para la programación de los microcontroladores ESP32 que actúan como nodos en la red CAN.
- **Ejecutable de la Interfaz Gráfica**: Archivo *.exe* de la GUI desarrollada en Python (Tkinter), listo para su instalación y ejecución en sistemas operativos Windows.
- **Manual de Prácticas**: Guía didáctica diseñada para la realización de actividades prácticas relacionadas con la red CAN y la interfaz.
- **Manual de Usuario**: Documentación detallada sobre el uso y las funcionalidades de la interfaz gráfica de usuario.
- **Archivos STL**: Modelos 3D en formato STL para la impresión de carcasas o soportes de los componentes electrónicos, garantizando un montaje ordenado y profesional.
- **Datasheets de Componentes**: Hoja de datos técnicos de los sensores y demás componentes electrónicos fundamentales utilizados en el proyecto.

Se alienta a los interesados a explorar el repositorio para una comprensión más profunda del proyecto y a utilizar los recursos para fines educativos o de desarrollo.

Apendice B: Manual de Usuario

5.3. Introducción

GUI Cuadro de Instrumentos es una aplicación diseñada para simular el tablero de instrumentos de un Kia Sorento 2019. La interfaz gráfica muestra en tiempo real la velocidad (0-240 km/h), RPM (0-8000), nivel de combustible (0-100 %) y temperatura (0-130°C), obteniendo los datos desde un nodo (gateway) diseñado por los alumnos, conectado a través de un puerto serial. La aplicación es ideal para proyectos educativos y demostraciones de redes vehiculares.

Este manual proporciona instrucciones detalladas para instalar, configurar y utilizar el programa, así como para conectar el nodo que envía los datos.

5.3.1. Descripción del Programa

El programa muestra:

- **Velocímetro**: Velocidad en un rango de 0 a 240 km/h.
- **Tacómetro**: Revoluciones por minuto (RPM) en un rango de 0 a 8000.
- **Indicador de combustible**: Nivel de combustible en un rango de 0 a 100 %.
- **Indicador de temperatura**: Temperatura del motor en un rango de 0 a 130°C.
- **Testigos de advertencia**:
  - Ícono de combustible en el velocímetro si el nivel es menor a 10 %.
  - Ícono de temperatura en el velocímetro si la temperatura excede los 110°C.

Los datos son recibidos desde un nodo (gateway) a través de un puerto serial, utilizando los identificadores:

- **101**: Temperatura (°C, 0-130).
- **102**: Nivel de combustible (%, 0-100).
- **103**: Velocidad (km/h, 0-240).
- **104**: RPM (0-8000).

5.4. Requisitos del Sistema

- **Sistema operativo**: Windows 10 o 11 (64 bits recomendado).
- **Procesador**: 1 GHz o superior.
- **Memoria RAM**: Mínimo 512 MB (1 GB recomendado).
- **Espacio en disco**: Aproximadamente 20 MB.
- **Puerto serial**: Puerto COM para conectar el nodo (gateway).
- **Controladores**: Controladores del dispositivo serial instalados (por ejemplo, para Arduino o ESP32).
- **Nodo (gateway)**: Microcontrolador configurado para enviar datos seriales.
- **Cable USB**: Para conectar el nodo al computador.

5.5. Instalación del Programa

1. **Obtener el instalador**:
   - Descargue el archivo `Setup_GUI_Cuadro_Instrumentos.exe` proporcionado por el instructor.
2. **Ejecutar el instalador**:
   - Haga doble clic en `Setup_GUI_Cuadro_Instrumentos.exe`.
   - Acepte la licencia mostrada.
   - Seleccione la carpeta de instalación (por defecto: `C:\Program Files\GUI Cuadro de Instrumentos`).
   - Elija crear accesos directos en el escritorio y/o menú de inicio.
   - Haga clic en “Instalar”.
3. **Verificar la instalación**:
   - Confirme que la carpeta contiene `cuadro_instrumentos_kia_sorento_3.0_COM.exe`, la carpeta `imagenes`, `license.txt` y `README.txt`.
   - Un acceso directo estará disponible en el escritorio o menú de inicio.

Figura 5.1: Ventana inicial para seleccionar el puerto serial.

5.6. Configuración del Nodo (Gateway)

El nodo (gateway) debe enviar datos al programa a través del puerto serial.

5.6.1. Requisitos del Nodo

- Microcontrolador: Arduino, ESP32 o similar con comunicación serial.
- Formato de datos: `101:valor,102:valor,103:valor,104:valor` (por ejemplo, `101:45.5,102:20.0,103:80.0,104:2500`)
- Configuración serial:
  - Baudrate: 115200.
  - Sin paridad, 8 bits de datos, 1 bit de parada.

5.6.2. Programar el Nodo

Programe el nodo para enviar datos en el formato correcto. Ejemplo para Arduino:

```
const int portPin1 = 13;
const int portPin2 = 12;
const int portPin3 = 14;
int potValor1,potValor2,potValor3=0;
```

```
5
6  void setup() -
7       Serial.begin(115200); // Inicia la comunicacion serial a 115200 baudios
8       delay(1000); // Espera para evitar logs de arranque
9     ˝
10
11    void loop() -
12      // Simulacion de datos (puedes reemplazar con sensores reales)
13      int temperature = 0; // Temperatura en C
14      int fuelLevel = 0;    // Nivel de gasolina en porcentaje
15      int speed = 0;       // Velocidad en km/h
16      int rpm = 4000;        // RPM (revoluciones por minuto)
17      potValor1 = analogRead(portPin1);
18      speed = (240*potValor1)/4095;
19      potValor2 = analogRead(portPin2);
20      temperature = (130*potValor2)/4095;
21      potValor3 = analogRead(portPin3);
22      fuelLevel = (100*potValor3)/4095;
23      // Enviar datos con identificadores en formato 101:valor,102:valor,103:valor
      ,104:valor
24      Serial.print(”101:”);
25      Serial.print(temperature);
26      Serial.print(”,102:”);
27      Serial.print(fuelLevel);
28      Serial.print(”,103:”);
29      Serial.print(speed);
30      Serial.print(”,104:”);
31      Serial.print(rpm);
32      Serial.println(); // Nueva linea para marcar fin de mensaje
33
34      delay(1000); // Enviar datos cada segundo
35    ˝
```

5.6.3. Conectar el Nodo

1. Conecte el nodo (Arduino o ESP32) al computador mediante un cable USB.
2. Instale los controladores:
   - Arduino: Descargue desde https://www.arduino.cc/en/software.
   - ESP32: Descargue desde https://www.silabs.com/developer-tools/usb-to-uart-bridge-vcp-drivers?tab.
3. Verifique el puerto COM en el “Administrador de dispositivos” de Windows (por ejemplo, COM3).

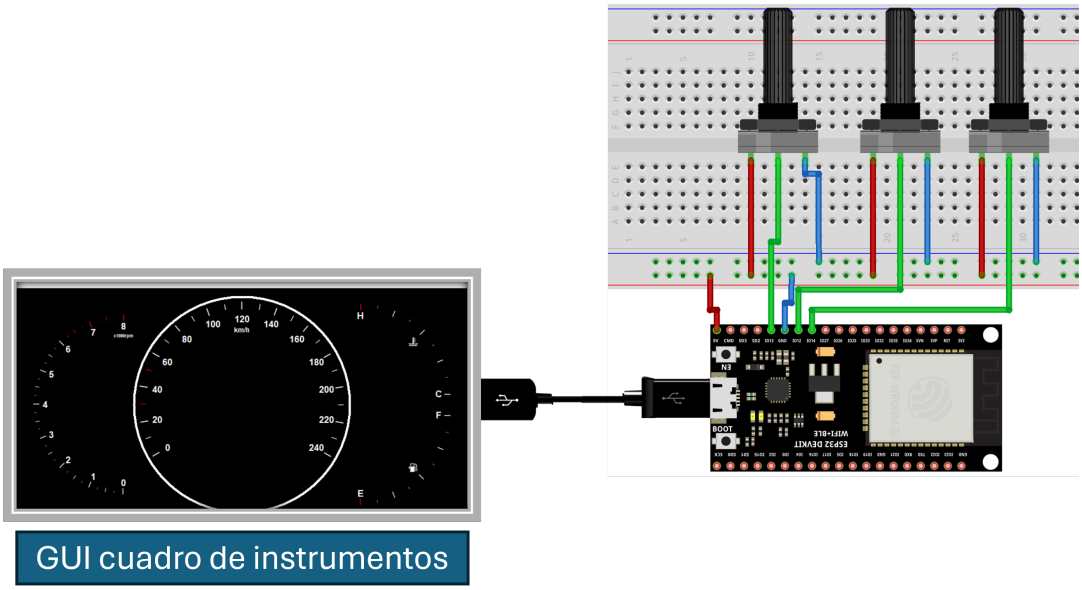


Figura 5.2: Diagrama de conexión entre la computadora y el nodo.

5.7. Uso del Programa

1. **Iniciar el programa**:
   - Haga doble clic en el acceso directo “GUI Cuadro de Instrumentos”.
2. **Seleccionar el puerto serial**:
   - En la ventana inicial, seleccione el puerto COM (por ejemplo, COM3).
   - Haga clic en “Conectar”.
3. **Interfaz principal**:
   - **Velocímetro** (centro): Muestra la velocidad.
   - **Tacómetro** (izquierda): Muestra los RPM.
   - **Indicador de temperatura** (derecha, arriba): Muestra la temperatura.
   - **Indicador de combustible** (derecha, abajo): Muestra el nivel de combustible.
   - **Testigos**: Aparecen en el velocímetro si el combustible es menor de 10 % o la temperatura mayor a 110°C.
4. **Cerrar el programa**:
   - Haga clic en la “X” de la ventana principal.

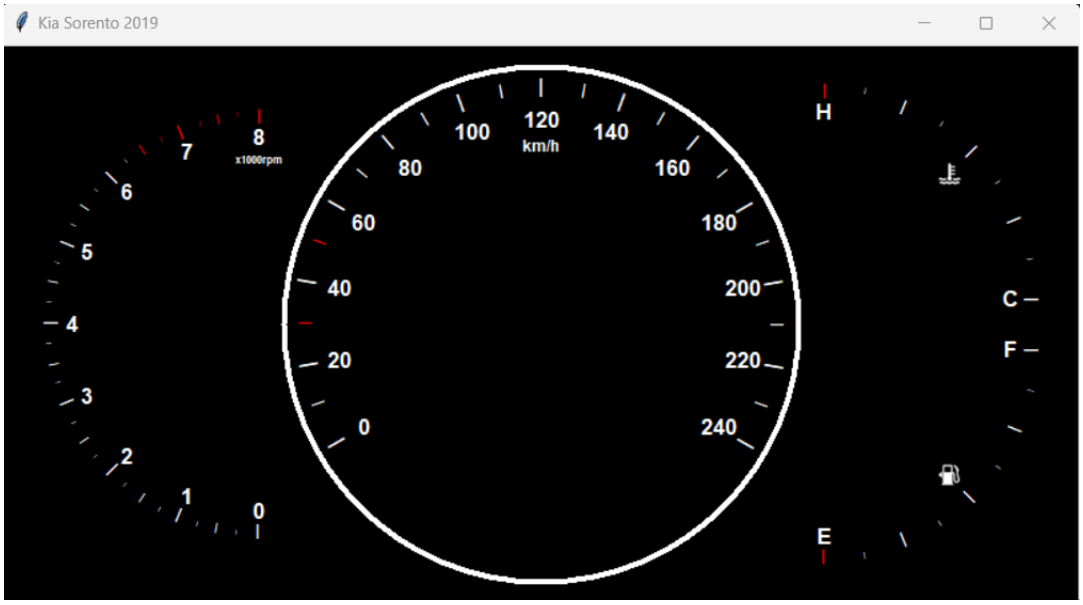


Figura 5.3: Interfaz principal del cuadro de instrumentos.

5.8. Solución de Problemas

- **El programa no detecta el puerto COM**:
  - Verifique la conexión del nodo y los controladores.
  - Confirme el puerto en el “Administrador de dispositivos”.
  - Reinicie el programa.
- **Las imágenes no se cargan**:
  - Asegúrese de que la carpeta `imagenes` esté en `C:\Program Files \GUI Cuadro de Instrumentos`.
  - Reinstale el programa.
- **Los indicadores no se actualizan**:
  - Verifique el formato de datos (`101:valor,102:valor,103:valor,104:valor`).
  - Use un monitor serial para depurar.
  - Confirme el baudrate (115200).
- **Error al conectar al puerto serial**:
  - Cierre otras aplicaciones que usen el puerto.
  - Reinicie el computador.

5.9. Licencia

El programa está licenciado bajo la Licencia MIT. Consulte el archivo `license.txt` en la carpeta de instalación.

5.10. Contacto

Para soporte o reportar problemas, contacte a:

- **Nombre**: Ing. Sergio Josué Ortiz Hernández
- **Correo**: ortizjosue95@gmail.com

5.11. Créditos

- **Desarrollador**: Ing. Sergio Josué Ortiz Hernández
- **Bibliotecas**: Pillow, pyserial, tkinter.
- **Nodo**: Diseñado por los alumnos.

Bibliografía

[1] M. Daddario, "Push to start: A brief history of car dashboards."

[2] M. J. Llanos López, *Circuitos eléctricos auxiliares del vehículo*. Paraninfo, 2ª edición actualizada; 2ª edición, 2017 ed., 2017. OCLC: 1117401775.

[3] L. Hernández del Arco, "Los tableros de los autos dejarán de equipar pantallas digitales."

[4] T. Denton, *Diagnóstico avanzado de fallas automotrices: tecnología automotriz : mantenimiento y reparación de vehículos*. Alfaomega, tercera edición ed., 2016. OCLC: 1023864189.

[5] W. Lawrenz, ed., *CAN System Engineering: From Theory to Practical Applications*. Springer London, 2013.

[6] W. Stallings, *Comunicaciones y redes de computadores*. Pearson Prentice Hall, 6ª ed ed., 2003. OCLC: 1026179292.

[7] S. Tuohy, M. Glavin, C. Hughes, E. Jones, M. Trivedi, and L. Kilmartin, "Intra-vehicle networks: A review," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 2, pp. 534–545, 2015.

[8] X. Tang and Y. Xi, "Application of hardware-in-loop in teaching power electronic course based on a low-cost platform," *Computer Applications in Engineering Education*, vol. 28, no. 4, pp. 965–978, 2020.

[9] H. Thi Nguyen, G. Yang, A. Hejde Nielsen, P. Højgaard Jensen, and C. F. Coimbra, "Control parameterisation for POD via software-in-the-loop simulation," *The Journal of Engineering*, vol. 2019, no. 18, pp. 4864–4868, 2019.

[10] E. Dávila Delgado, J. J. Raygoza Panduro, E. C. Becerra Álvarez, F. J. Espinoza Jurado, and E. F. Gutiérrez Frías, "Low cost DSP-based educational embedded platform for real-time simulation and fast implementation of complex systems in simulink," *Computer Applications in Engineering Education*, vol. 27, no. 4, pp. 955–970, 2019.

[11] V. Tuominen, H. Reponen, A. Kulmala, S. Lu, and S. Repo, "Real-time hardware- and software-in-the-loop simulation of decentralised distribution network control architecture," *IET Generation, Transmission & Distribution*, vol. 11, no. 12, pp. 3057–3064, 2017.

[12] M. Firth, “Introduction to automotive augmented reality head-up displays using dlp® technology,” May 2019. Consultado el 22 de noviembre de 2024.

[13] X. Zhang and C. Mi, *Vehicle Power Management: Modeling, Control and Optimization*. Power Systems, Springer London, 2011.

[14] A. Mareška, T. Kordová, and M. Havlík Míka, “Production of head-up display windshield and its relation with the image quality,” *Transactions on Transport Sciences*, vol. 13, no. 2, pp. 63–70, 2022.

[15] W. B. Ribbens, *Understanding Automotive Electronics: An Engineering Perspective*. Elsevier, 2012.

[16] Hella GmbH, “Electronic control units in modern vehicles,” tech. rep., Hella Automotive, 2018.

[17] W. Zeng, M. Khalid, and S. Chowdhury, “Automotive ethernet and can-fd: Next-generation in-vehicle networking,” *IEEE Transactions on Vehicular Technology*, vol. 65, no. 6, pp. 3704–3716, 2016.

[18] J. D. Turner, ed., *Automotive sensors*. Sensors technology series, Momentum Press, 1st ed (online-ausg.) ed., 2009.

[19] Environmental Protection Agency, “On-board diagnostics (obd-ii) program overview,” tech. rep., U.S. EPA, 2005.

[20] Autel Intelligent Technology Corp, “Advanced automotive diagnostics and telematics,” tech. rep., Autel, 2021.

[21] N. Navet and F. Simonot-Lion, eds., *Automotive Embedded Systems Handbook*. CRC Press, 1 ed., 2017.

[22] J. Ma and W. Zhang, “Autonomous driving: Trends and challenges in ecu design,” *IEEE Access*, vol. 8, pp. 123456–123467, 2020.

[23] M. Ring and D. Dürrwang, “Cybersecurity for automotive embedded systems,” *IEEE Security & Privacy*, vol. 17, no. 4, pp. 32–40, 2019.

[24] Y. Liu and L. Zhang, “Electrification and automation in automotive systems,” *Automotive Innovation*, vol. 4, pp. 123–135, 2021.

[25] Robert Bosch GmbH, *Bosch Automotive Handbook*. Wiley, 2018.

[26] T. Litman, “Autonomous vehicle safety technologies: Policy implications,” *Transportation Research Procedia*, vol. 55, pp. 1–8, 2021.

[27] I. y mecanica automotriz, “¿qué son las bolsas de aire (airbags srs) y cómo funcionan?,” November 2024. Consultado el 22 de noviembre de 2024.

[28] J. Kang, S. Lee, and S. Mun, “Optimizing lane departure warning system,” *Sensors*, vol. 24, no. 8, p. 2505, 2024.

[29] J.-H. Kim and S.-H. Lee, "Advanced hvac systems for vehicle comfort," *Journal of Automotive Engineering*, vol. 12, no. 3, pp. 45–53, 2020.

[30] M.-J. Lee and S.-C. Park, "Ergonomic design of automotive seating systems," *Applied Ergonomics*, vol. 98, p. 103567, 2022.

[31] C. de Fallas, "Partes de un aire acondicionado automotriz." https://codigosdefallas.com/partes-de-un-aire-acondicionado-automotriz/, 2024. Accedido el 24 de noviembre de 2024.

[32] D. Frej, "The effect of changing the angle of the passenger car seat backrest on the head trajectories of the 50th percentile male dummy," *Sensors*, vol. 24, no. 12, p. 3868, 2024.

[33] W. Chen and X. Li, "Optimizing powertrain efficiency with sensor integration," *Energy Reports*, vol. 9, pp. 2345–2356, 2023.

[34] M. Auto, "Sistema de inyección," 2024. Accedido el 24 de noviembre de 2024.

[35] YouTube, "Common rail injection system explanation." https://www.youtube.com/watch?v=IWGrSfhrYB4, 2024. Accedido el 24 de noviembre de 2024.

[36] Solectroshop, "¿qué es un termistor?." https://solectroshop.com/es/blog/que-es-termistor-n48, 2024. Accedido el 24 de noviembre de 2024.

[37] I. y Mecánica Automotriz, "¿qué es el sensor de posición de pedal de acelerador app y cómo funciona?." https://www.ingenieriaymecanicaautomotriz.com, 2024. Accedido el 24 de noviembre de 2024.

[38] Autoavance, "Sensor de pedal acelerador," November 2024. Consultado el 22 de noviembre de 2024.

[39] C. DTC, "Sensor de posición del pedal de acelerador (app) — qué es, funcionamiento y fallas." https://codigosdtc.com, 2024. Accedido el 24 de noviembre de 2024.

[40] E. Gary, "¿cómo funcionan los sensores de velocidad del cigüeñal?." https://www.puromotores.com, 2017. Accedido el 24 de noviembre de 2024.

[41] Electricity and Magnetism, "¿cómo funcionan los sensores de efecto hall en un circuito?." https://www.electricity-magnetism.org, 2024. Accedido el 24 de noviembre de 2024.

[42] S. Automotriz, "Descubre cómo funciona el sensor ckp de efecto hall en tu vehículo." https://sensorautomotriz.es, 2024. Accedido el 24 de noviembre de 2024.

[43] HELLA, "Sensor del cigüeñal: Revisión y averías." https://www.hella.com, 2024. Accedido el 24 de noviembre de 2024.

[44] Widetech, "Todo lo que necesitas saber sobre el sensor de nivel de combustible." https://widetech.co, 2024. Accedido el 24 de noviembre de 2024.

[45] Navixy, “Sensores de nivel de combustible — ¿cómo elegir el adecuado?.” https://www.navixy.com, 2024. Accedido el 24 de noviembre de 2024.

[46] M. del Motor, “Sensor de nivel de combustible: Fls - que es, ubicación, fallas.” https://www.mundodelmotor.net, 2024. Accedido el 24 de noviembre de 2024.

[47] J. Doe, *Automotive Actuators: Principles and Applications*. New York: Automotive Engineering Press, 1st ed., 2021.

[48] M. R. T. Specifications, “Rs550 12v dc motor specifications and applications,” *Motor Technology Review*, vol. 15, pp. 210–214, 2023.

[49] J. Smith, “Fundamentals of dc motors in automotive applications,” *Journal of Electric Motors*, vol. 10, pp. 50–60, 2020.

[50] T. Domínguez Mínguez, *Desarrollo de interfaces gráficas en Python 3 con Tkinter*. Marcombo, 1ª ed ed., 2022. OCLC: 1319084714.

[51] M. Fitzpatrick, *Create GUI Applications with Python and Qt5: The Hands-On Guide to Making Apps with Python*. Independently Published, 2020.

[52] Arduino, “Official arduino documentation,” 2024. Último acceso: 24 de noviembre de 2024.

[53] E. Systems, *ESP32 Series Datasheet*, 2024. Último acceso: 24 de noviembre de 2024.

[54] S. Electronics, “Can-bus shield,” 2024. Último acceso: 24 de noviembre de 2024.

[55] SeeedStudio, “Can-bus shield v2.0,” 2024. Último acceso: 24 de noviembre de 2024.

[56] M. T. Inc., *MCP2515 Stand-Alone CAN Controller with SPI Interface*, 2024. Último acceso: 24 de noviembre de 2024.

[57] T. Instruments, *SN65HVD23x CAN Transceivers*, 2024. Último acceso: 24 de noviembre de 2024.

[58] M. Liserre, F. Blaabjerg, and S. Hansen, “Design and control of an lcl-filter-based three-phase active rectifier for pwm applications,” *IEEE Transactions on Industry Applications*, vol. 41, no. 5, pp. 1281–1291, 2005.