COMUNICACIONES
SECRETARÍA DE INFRAESTRUCTURA, COMUNICACIONES Y TRANSPORTES
INSTITUTO MEXICANO DEL TRANSPORTE

Desarrollo de lector CAN para vehículos de carretera

Mauricio Eliseo Cruz Acevedo
Marco Antonio Hernández Nochebuena
David Vazquez Vega
Óscar Flores Centeno
José Ricardo Hernández Jiménez

Publicación Técnica No. 860
Querétaro, México
2024

ISSN 0188-7297

Esta investigación fue realizada en la Coordinación de Ingeniería Vehicular e Integridad Estructural (CIVIE) del Instituto Mexicano del Transporte (IMT), por MTA. Mauricio Eliseo Cruz Acevedo, MC. David Vázquez Vega, MC. José Ricardo Hernández Jiménez, el MC. Marco Antonio Hernández Nochebuena y al MC. Óscar Flores Centeno.

Esta investigación es el producto final del proyecto de investigación interna EI 03/24 “Desarrollo de sistema para la adquisición de variables dinámicas de vehículos de carretera a través del protocolo CAN”.

Se agradece la colaboración del Dr. Francisco Javier Carrión Viramontes Coordinador de la CIVIE en la revisión de la presente publicación, así como la colaboración del estudiante Jesús Eduardo Gómez Zea, de la Universidad Politécnica de Querétaro.

Tabla de Contenido

Sinopsis

Se describe el desarrollo de un sistema lector de señales CAN para vehículos para integrar y construir una herramienta electrónica capaz de adquirir y almacenar información proveniente de diversos sensores. Este sistema permite analizar las magnitudes físicas relacionadas con el comportamiento dinámico de los vehículos, enfocándose especialmente en unidades de autotransporte de carga y pasaje.

El lector CAN se diseñó utilizando una placa Arduino UNO, un módulo MCP2515 y una interfaz gráfica en LabVIEW© para almacenar y procesar los datos. Las pruebas experimentales se realizaron en tres vehículos; dos vehículos ligeros -- uno a gasolina y otro eléctrico -- y un tractocamión Kenworth. En todos los casos, el sistema demostró ser funcional y la información se pudo almacenar en una computadora portátil. Para los vehículos ligeros, se registró la información sin haberla correlacionado con los sensores o sistemas de la computadora de los vehículos, ya que no se cuenta con el DBC. En el caso del tractocamión, el análisis se realizó siguiendo la norma SAE J1939/71_201205.

El lector desarrollado permite obtener información relevante de los sensores instalados en vehículos con protocolo CAN, constituyendo una herramienta valiosa para complementar estudios que se realizan de seguridad y dinámica vehicular.

Abstract

The development of a CAN signal reader system for vehicles is described to integrate and build an electronic tool capable of acquiring and storing information from various sensors. This system allows the analysis of physical magnitudes related to the dynamic behavior of vehicles, focusing especially on cargo and passenger transport units.

The CAN reader was designed using an Arduino UNO board, an MCP2515 module and a graphical interface in LabVIEW© to store and process the data. The experimental tests were carried out on three vehicles; two light vehicles -- one gasoline and one electric -- and a Kenworth tractor-trailer. In all cases, the system proved to be functional and the information could be stored on a laptop. For light vehicles, the information was recorded without having been correlated with the sensors or computer systems of the vehicles, since the DBC is not available. In the case of the tractor-trailer, the analysis was carried out following the SAE J1939/71_201205 standard.

The developed reader allows obtaining relevant information from sensors installed in vehicles with CAN protocol, constituting a valuable tool to complement studies carried out on vehicle safety and dynamics.

Introducción

La implementación de un prototipo de sistema para la adquisición y monitoreo de variables provenientes del BUS CAN (*Controller Area Network*) en vehículos de autotransporte se presenta como una iniciativa estratégica y esencial en el panorama actual del sector automotriz. La creciente complejidad de los sistemas electrónicos en los vehículos modernos, junto con la necesidad de garantizar la seguridad y la eficiencia operativa, ha impulsado el desarrollo de herramientas capaces de captar y analizar información en tiempo real desde los múltiples sensores y actuadores que interactúan en el funcionamiento de un vehículo.

Este trabajo propone el diseño y construcción de un lector de señales CAN orientado a la recolección y análisis de datos de vehículos de carretera. El principal objetivo es desarrollar un sistema económico, accesible y eficiente que permita adquirir y almacenar la mayor cantidad posible de información proveniente de los sistemas y sensores integrados en este tipo de vehículos. Al extraer y analizar datos clave como la velocidad, las condiciones del motor o el comportamiento de los sistemas de frenado, se puede obtener información valiosa para mejorar los estándares de seguridad vehicular, optimizar el mantenimiento preventivo e incluso predictivo y contribuir al desarrollo de tecnologías de conducción autónoma y conectada.

El prototipo desarrollado utiliza una placa Arduino UNO junto con un módulo MCP2515 para la recepción de datos CAN, complementado por una interfaz gráfica en LabVIEW© que permite almacenar y procesar los mensajes obtenidos. Las pruebas experimentales se llevaron a cabo en tres tipos de vehículos: un sedán Honda Accord, un vehículo eléctrico y un tractocamión Kenworth. En cada caso, el sistema demostró ser capaz de reconocer y registrar correctamente los identificadores y mensajes CAN, utilizando una computadora portátil para el almacenamiento de los datos.

En los vehículos Honda y eléctrico, debido a la ausencia de diccionarios para decodificar las variables CAN, se llevó a cabo un análisis a partir de un posprocesamiento de datos para identificar el parámetro de la velocidad de giro del motor expresada en revoluciones por minutos e identificada como RPM. Por otro lado, en el tractocamión Kenworth, el análisis se realizó siguiendo los lineamientos de la norma SAE J1939/71_201205, ampliamente utilizada en vehículos pesados.

Este sistema prototipo representa un avance significativo en la adquisición de datos vehiculares, aportando una herramienta funcional y versátil que puede ser utilizada en diversos escenarios de análisis dinámicos y de seguridad vehicular. Su implementación y los resultados obtenidos refuerzan la importancia de contar con tecnologías accesibles para la interpretación de datos CAN en un mercado automotriz cada vez más interconectado y tecnológicamente avanzado.

Para fundamentar y comprender mejor el aporte de este trabajo, en el capítulo 1 se describe el protocolo de comunicación CAN, así como los principios del protocolo de comunicación CAN en el capítulo 2 se describe la parte técnica de los componentes principales utilizados para la construcción del lector de mensajes CAN. Posteriormente, en el capítulo 3 se describe la metodología utilizada, así como el protocolo de pruebas sugerido para su evaluación; en el capítulo 4 se presentan los resultados de la evaluación del lector. Finalmente, se exponen las conclusiones y se sugieren propuestas de mejora para el desarrollo tecnológico.

1. Marco Teórico

Para el desarrollo de este proyecto se requieren abordar conceptos relacionados a los protocolos de comunicación en vehículos. El protocolo de comunicación en el que se centra este proyecto es el CAN, ya que a través de este protocolo se puede obtener información de los diferentes sensores instalados en los vehículos. De igual forma se abordará los conceptos claves para la decodificación de mensajes y la importancia de la norma SAE J1939 en la combinación CAN para vehículos pesados. A continuación, se presenta una breve descripción de los conceptos utilizados para llevar a cabo este proyecto.

1.1 Protocolo de comunicación CAN

El protocolo de comunicación CAN es un estándar de red diseñado originalmente por Bosch en la década de 1980 para facilitar la comunicación entre distintos componentes electrónicos dentro de vehículos. Su propósito principal es permitir el intercambio de datos en tiempo real entre controladores, sensores y actuadores en sistemas embebidos sin necesidad de un controlador centralizado [1].

La estructura del protocolo CAN se basa en una topología de bus que permite la transmisión de mensajes a través de un solo par de cables, lo que reduce considerablemente el cableado necesario. Cada nodo en la red tiene la capacidad de enviar y recibir mensajes, y todos los nodos pueden acceder a la información transmitida, lo cual aumenta la eficiencia del sistema. Además, el protocolo utiliza un método de arbitraje para resolver conflictos cuando varios nodos intentan transmitir datos al mismo tiempo, asegurando que el mensaje con mayor prioridad tenga preferencia [1],[2].

En la industria automotriz, el protocolo CAN se ha convertido en un estándar debido a su robustez, confiabilidad y capacidad para operar en entornos adversos, como temperaturas extremas y presencia de interferencias electromagnéticas. Su importancia radica en que permite una integración eficiente de sistemas críticos en el vehículo, tales como el control del motor, el sistema de frenos, el sistema de transmisión y diversos sistemas de seguridad activa y pasiva, mejorando así la coordinación y el desempeño general del vehículo. Asimismo, el protocolo CAN facilita la implementación de sistemas avanzados de asistencia al conductor (ADAS, por sus siglas en inglés) y otras tecnologías de

automatización, permitiendo la comunicación rápida y confiable entre los diversos sensores, subsistemas, cámaras y unidades de control [3].

1.1.1 Características principales del BUS CAN

El protocolo de comunicación CAN posee varias características que lo hacen ideal para su uso en vehículos. Estas son algunas de las principales:

- Topología en Bus: CAN utiliza una configuración de bus, lo que significa que múltiples dispositivos (unidades de control electrónico o ECUs) están conectados a una única línea de comunicación. Esta estructura permite simplificar el cableado, reducir el peso y los costos, y facilita la comunicación entre sistemas sin la necesidad de tener un controlador central.
- Comunicación Multidifusión: Los mensajes en una red CAN se envían en forma de multidifusión, de modo que cualquier nodo en el bus puede recibir el mensaje. Esto permite que varios sistemas compartan la información sin duplicación de mensajes, lo cual es fundamental para la coordinación entre los sistemas de frenado, de la dirección, del motor, etc.
- Prioridad de Mensajes: Cada mensaje en la red CAN tiene un identificador único que define su prioridad. Los mensajes con identificadores de menor valor (mayor prioridad) tienen preferencia en la red, permitiendo que datos críticos (como el control de motor o frenos) se transmitan antes que otros mensajes menos urgentes.
- Detección y Corrección de Errores: CAN es un protocolo robusto con varios mecanismos de detección de errores, incluyendo chequeo de redundancia cíclica (CRC), comprobación de bits y detección de errores de formato. Estos mecanismos ayudan a garantizar la integridad de los datos transmitidos y permiten detectar fallos en la comunicación, lo cual es crucial en aplicaciones de seguridad en vehículos.
- Velocidades de Transmisión Ajustables: La red CAN permite la comunicación en diferentes velocidades, comúnmente de 125 kbps a 1 Mbps, dependiendo de los requisitos de la aplicación y la longitud del bus. En automóviles, por ejemplo, la red CAN de alta velocidad (hasta 1 Mbps) se usa para sistemas críticos, mientras que la CAN de baja velocidad se destina a otros sistemas menos urgentes.
- Formato de Mensajes Compacto: Los mensajes en CAN son pequeños, con un tamaño de 8 bytes de datos como máximo, lo que permite una transmisión rápida y asegura que la red no se sobrecargue. A su vez, cada mensaje incluye identificadores y bits de control que ayudan a gestionar su transmisión y asegurar la compatibilidad en toda la red.

- Bajo Consumo de Energía: CAN es un protocolo eficiente en términos de consumo energético, lo cual es importante en vehículos, ya que ayuda a reducir la carga en el sistema eléctrico y contribuye a la eficiencia general del automóvil.
- Compatibilidad con Redes Distribuidas: Debido a su capacidad de multidifusión y a su arquitectura de bus, el protocolo CAN permite que múltiples sistemas distribuidos funcionen juntos sin problemas. Esto es esencial para los vehículos modernos, que tienen múltiples ECU que deben cooperar en tiempo real [2].

Estas características convierten al protocolo CAN en un pilar fundamental para los sistemas electrónicos en vehículos modernos, permitiendo la comunicación y coordinación eficiente entre distintos sistemas, así como un control seguro y robusto de las funciones críticas del vehículo.

1.1.2 Formato estándar y extendido

El protocolo CAN utiliza dos formatos principales para los mensajes: el Formato Estándar (CAN 2.0A) y el Formato Extendido (CAN 2.0B). El Formato Estándar es el más común y se caracteriza por un identificador de 11 bits. En la Figura 1.1 se puede observar el orden del Formato Estándar de un mensaje CAN.

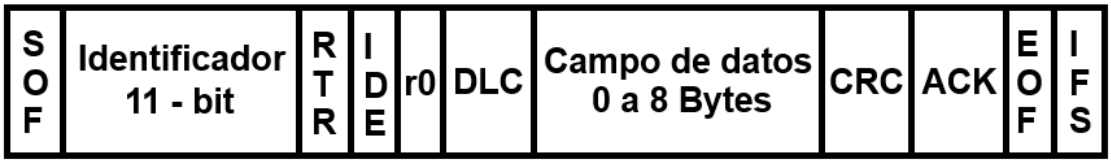

Figura 1.1 Formato Estándar para mensajes CAN

A continuación, se describen las principales características del Formato Estándar de un mensaje CAN:

1.1.2.1 Estructura del Mensaje CAN (Formato Estándar)

Campo de Inicio de Trama (SOF - *Start of Frame*): Es un único bit dominante que indica el inicio del mensaje. Todos los nodos sincronizan su reloj con este bit para iniciar la recepción de la trama [4].

Campo de Identificador (11 bits): Este campo contiene un identificador único que define el tipo de mensaje y su prioridad. Los identificadores más bajos tienen mayor prioridad y se transmiten primero cuando hay conflicto de acceso al bus [4].

Campo de Control:

Incluye bits adicionales para definir la longitud del mensaje y asegurar la integridad de la transmisión.

- RTR (*Remote Transmission Request*): Un bit que indica si el mensaje es de datos o una solicitud de datos. En un mensaje de datos, este bit es dominante.
- Bit reservado (r0): Debe ser dominante (0), pero aceptado tanto dominante como recesivo
- IDE (*Identifier Extension*): Indica si el mensaje utiliza el Formato Estándar (11 bits) o extendido (29 bits). En el Formato Estándar, este bit es dominante.
- Campo de longitud de datos (DLC - *Data Length Code*): Es un campo de 4 bits que indica la cantidad de bytes en el campo de datos (puede ser entre 0 y 8 bytes) [4].

Campo de Datos: Este campo contiene los datos reales que se transmiten en el mensaje. Puede tener entre 0 y 8 bytes, lo cual permite una transmisión de datos rápida y eficiente, evitando la sobrecarga en la red [4].

Campo de revisión de Redundancia Cíclica (CRC - *Cyclic Redundancy Check*):

- Es un campo de 15 bits que proporciona un método de detección de errores en los datos. Este chequeo asegura que los datos recibidos coincidan con los datos enviados, mejorando la confiabilidad de la transmisión [4].

Campo de Reconocimiento (ACK - *Acknowledge*):

- Este campo permite que el nodo receptor confirme la recepción del mensaje sin errores. Consiste en un bit de espacio y un bit dominante que es enviado por cualquier nodo que haya recibido el mensaje correctamente [4].

Campo de Fin de Trama (EOF - *End of Frame*):

- Es un conjunto de 7 bits recesivos que indican el final de la trama de datos. Asegura que todos los nodos en la red reconozcan que el mensaje ha concluido [4].

Espacio de Inter-Trama (IFS - *Intermission Frame Space*):

- Este campo consta de 3 bits recesivos que crean una pausa antes de que se envíe otro mensaje en el bus. Sirve para garantizar un tiempo de inactividad mínimo entre mensajes [4].

1.1.2.2 Características Clave del Formato Estándar del CAN

- **Identificación y Prioridad**: Los 11 bits del identificador permiten asignar prioridad a los mensajes, lo que es esencial para gestionar el tráfico en el bus y asegurar que los mensajes críticos se transmitan primero [5].

- **Eficiencia de Datos**: La capacidad de hasta 8 bytes de datos permite transmitir la información de manera compacta, evitando sobrecargar la red y optimizando la velocidad de transmisión [5].

- **Detección de Errores**: El uso de un campo CRC y un campo de ACK proporciona una verificación constante de la integridad de los datos, minimizando errores de transmisión y mejorando la confiabilidad del sistema [5].

- **Sincronización y Robustez**: Los bits SOF y EOF aseguran la correcta sincronización de todos los nodos y permiten que los mensajes se envíen y reciban con precisión, incluso en entornos electromagnéticamente ruidosos como los vehículos [5].

Esta estructura permite que el protocolo CAN en su Formato Estándar sea confiable, eficiente y capaz de operar en entornos exigentes, lo cual es ideal para aplicaciones en la industria automotriz y otros sectores donde la seguridad y la integridad de los datos son críticas.

1.1.2.3 Estructura del Mensaje CAN (Formato Extendido)

El Formato Extendido del protocolo CAN (también conocido como CAN 2.0B) amplía el Formato Estándar para admitir un identificador de 29 bits, en lugar de los 11 bits de identificador que usa el Formato Estándar. Esta ampliación permite una mayor cantidad de mensajes únicos en la red, lo cual es útil en sistemas complejos con una gran cantidad de módulos y dispositivos [6]. En la Figura 1.2 se puede observar un ejemplo del orden de mensaje CAN en formato Extendido.

Figura 1.2 Formato Extendido para mensajes CAN

A continuación, se detallan las principales características y la estructura del Formato Extendido de CAN:

Campo de Inicio de Trama (SOF - *Start of Frame*):

- Similar al Formato Estándar, es un único bit dominante que marca el inicio de la transmisión del mensaje y sincroniza los nodos para el inicio de la trama [7].

Campo de Identificador (29 bits):

- El identificador en el Formato Extendido consta de 29 bits en lugar de 11, permitiendo una mayor cantidad de identificadores únicos (2^{29} posibles = 536’870,912). Esto es ideal para redes complejas, como las que se encuentran en vehículos de gran tamaño o sistemas con múltiples ECUs. EL identificador está formado por 11 bits iniciales y un complemento de 18 bits que en total dan el identificador de 29 bits [7].

Campo de Control:

- En el Formato Extendido, el campo de control incluye varios bits importantes:

- **SRR (*Substitute Remote Request*)**: Un bit que permite compatibilidad con el Formato Estándar y actúa como un indicador de prioridad.

- **IDE (*Identifier Extension*)**: Este bit es recesivo en el Formato Extendido e indica que se están utilizando 29 bits para el identificador, diferenciándolo del Formato Estándar de 11 bits.

- **Campo de longitud de datos (DLC - *Data Length Code*)**: Es un campo de 4 bits que indica la cantidad de bytes de datos en el mensaje (entre 0 y 8 bytes) [7].

Campo de Datos:

- Al igual que en el Formato Estándar, este campo puede contener entre 0 y 8 bytes de datos, permitiendo la transmisión de información precisa y optimizada [7].

Campo de Revisión de Redundancia Cíclica (CRC - *Cyclic Redundancy Check*):

- En el Formato Extendido, el campo CRC sigue siendo de 15 bits y proporciona verificación de errores para garantizar la integridad de los datos [7].

Campo de Reconocimiento (ACK - *Acknowledge*):

- Este campo sigue funcionando de la misma manera que en el Formato Estándar, permitiendo que los nodos confirmen la recepción de un mensaje sin errores [7].

Campo de Fin de Trama (EOF - *End of Frame*):

- Se utilizan 7 bits recesivos para marcar el final de la transmisión del mensaje y asegurar que todos los nodos en la red sepan que el mensaje ha concluido [7].

Espacio de Inter-Trama (IFS - *Intermission Frame Space*):

- Consiste en un espacio de 3 bits recesivos, igual que en el Formato Estándar, que proporciona un tiempo de inactividad antes de que se transmita otro mensaje en el bus [7].

1.1.2.4 Características Clave del Formato Extendido de CAN

1. **Mayor Capacidad de Identificación**: Con 29 bits de identificador, el Formato Extendido permite 2^{29} combinaciones (más de 500 millones de identificadores), lo que lo hace adecuado para redes con gran cantidad de dispositivos y donde se requiere una mayor diferenciación entre mensajes [8].

2. **Compatibilidad con el Formato Estándar**: El campo SRR y el bit IDE permiten que el Formato Extendido sea compatible con el Estándar. Esto es útil en redes donde se requieren ambos formatos, permitiendo que los dispositivos con diferentes necesidades de identificador coexistan en la misma red [8].

3. **Priorización de Mensajes Complejos**: A pesar de que el Formato Extendido es menos eficiente en términos de tiempo de transmisión

(dado que el mensaje es más largo), el uso de un identificador más amplio permite una mayor flexibilidad para priorizar mensajes de sistemas complejos, tales como redes de comunicaciones en vehículos industriales o vehículos de pasajeros con múltiples subsistemas [8].

4. **Confiabilidad en Transmisión**: Al igual que el Formato Estándar, el Formato Extendido incluye campos CRC y ACK, proporcionando detección de errores y verificación de recepción, asegurando que los datos sean confiables incluso en condiciones de redes complejas [7].

1.1.2.5 Consideraciones del Formato Extendido

- **Menor Eficiencia en la Transmisión**: Debido al tamaño adicional del identificador (29 bits en lugar de 11), el Formato Extendido es más lento en comparación con el Formato Estándar. Esto puede ser un inconveniente en redes donde se requiere una transmisión rápida y frecuente de mensajes.

- **Uso en Redes de Alta Densidad**: Este formato es ideal para aplicaciones en redes que requieren más espacio de identificación, como sistemas de vehículos con numerosos módulos de control o en aplicaciones industriales y de transporte donde se manejan numerosos datos.

El Formato Extendido permite, en general, que el protocolo CAN sea más adaptable y escalable, cumpliendo con las necesidades de sistemas complejos que requieren identificar y gestionar una gran cantidad de mensajes en la red.

1.1.3 Decodificación de mensajes

Decodificar mensajes CAN implica interpretar los bits transmitidos en cada mensaje para entender el valor y el significado de los datos enviados. Cada mensaje CAN incluye varios campos, como el identificador, la longitud de datos y el propio campo de datos, que contienen la información relevante. El proceso de decodificación requiere conocer el formato de estos mensajes y, generalmente, una Base de Datos "DBC" que explica cómo interpretar los valores en el contexto de una aplicación específica. A continuación, se explica cómo decodificar un mensaje CAN paso a paso:

Identificar los Campos en el Mensaje CAN

Primero, se deben separar y analizar los campos básicos del mensaje CAN:

1. Identificador (ID):

- Este campo (11 bits en Formato Estándar o 29 bits en Formato Extendido) identifica el mensaje y generalmente indica el tipo de datos que contiene (por ejemplo, velocidad, posición del acelerador, etc.).
- Cada identificador corresponde a un conjunto de datos específicos según el sistema del vehículo o la configuración de red.

2. Campo de Longitud de Datos (DLC - *Data Length Code*):

- Este campo de 4 bits indica el número de bytes en el campo de datos (de 0 a 8 bytes).

3. Campo de Datos:

- Este campo contiene los valores reales que el mensaje está transmitiendo. La longitud de este campo se define por el DLC y puede ser de hasta 8 bytes.
- La interpretación de estos bytes depende del identificador y la configuración de la red CAN, y cada bit o conjunto de bits tiene un significado específico que es necesario descifrar [8].

Usar una Base de Datos DBC para la Decodificación

Las redes CAN en aplicaciones específicas suelen utilizar archivos DBC (*CAN Database File*) para organizar y definir cómo se deben interpretar los datos de cada identificador. Los archivos DBC contienen:

- Identificadores: Mapean cada identificador a un mensaje específico.
- Nombres de Señal: Explican qué representa cada segmento de datos (por ejemplo, velocidad del vehículo, temperatura del motor).
- Ubicación de Bits: Definen qué bits del campo de datos corresponden a cada señal.
- Factores de Escala y Desplazamiento: Indican cómo convertir el valor bruto a su unidad correspondiente (por ejemplo, transformar el valor en bits a kilómetros por hora).

Si se tiene un archivo DBC, el software de análisis de CAN (como *Vector CANalyzer*, *PCAN-Explorer* o *Python* con librerías como *python-can*) puede interpretar automáticamente los mensajes.

Para decodificar un mensaje CAN sin un archivo DBC, se deben conocer los siguientes elementos sobre la estructura de cada mensaje. En la Figura 1.3 se pueden observar algunos elementos necesarios para decodificar mensajes can sin un DBC.

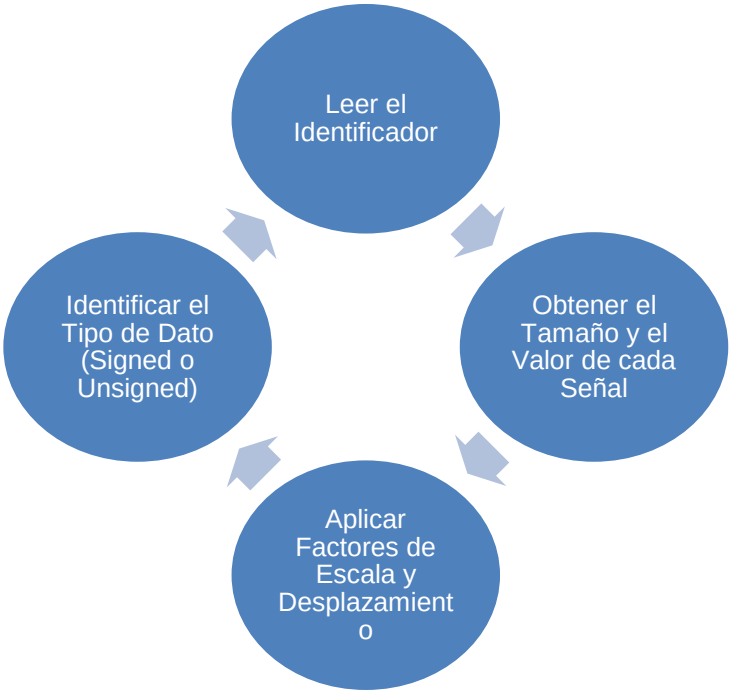

Figura 1.3 Proceso para decodificar mensajes CAN sin un DBC

1. Leer el Identificador:
    - Identifique el propósito del mensaje mediante su ID. Por ejemplo, si el ID 0x100 está asignado a la velocidad del motor, entonces sabemos que el mensaje con ese ID contiene datos relacionados con esta señal.
2. Obtener el Tamaño y el Valor de cada Señal:

- Divida el campo de datos según la información específica para cada mensaje. Supongamos que los primeros 2 bytes (16 bits) representan la velocidad en RPM:
- Si el campo de datos es '0x0FA0', los 2 bytes (0F y A0) deben interpretarse en el sistema numérico de base hexadecimal (16^n) como un número entero (0FA0). En este caso, '0x0FA0' equivale a '4000' en el sistema numérico de base decimal (correspondiente al que usamos cotidianamente).
- Esto pudiera significar 4000 RPM si no hay factor de escala o si se dispone de un factor de escala, este debe aplicarse para obtener el valor real.

3. Aplicar Factores de Escala y Desplazamiento:
    - Algunos datos requieren tratamientos adicionales para tener sentido. Si el valor se requiere en una unidad específica, se necesita aplicar un factor de escala conocido. Por ejemplo, si el factor de escala conocido es de 0.1 (decimal) para obtener la temperatura en grados Celsius (°C), y se identifica que el valor hexadecimal de temperatura es '0x0190' (hexadecimal), primero se requiere convertir el valor '0190' hexadecimal a decimal lo cual correspondería a '400' decimal y aplicando el factor 0.1 a esta magnitud se obtendría que el valor de la temperatura es de 40°C (400x0.1). Por lo tanto, un valor hexadecimal de '0x0190' corresponde a una temperatura de 40°C, expresado en notación decimal.
4. Identificar el Tipo de Dato (*Signed o Unsigned*):
    - En ocasiones, el dato obtenido en la trama CAN puede ser un número con signo (*signed*) o un número sin signo (*unsigned*). Esto es importante en casos de valores negativos, como por ejemplo la aceleración (aumento de velocidad, magnitud positiva) o la desaceleración (reducción de la velocidad o frenado, magnitud negativa).

1.1.3.1 Ejemplo de Decodificación de un Mensaje CAN

Si se asume que se tiene un mensaje CAN con los siguientes datos:

- Identificador: 0x200
- DLC: 8 (indica 8 bytes de datos)
- Campo de Datos: 0x0A 0xF0 0x03 0xD9 0x64 0x0C 0x1C 0x00

Adicionalmente se dispone de información (DBC) sobre la manera de interpretar el mensaje. La Tabla 1.1 muestra las características para decodificar el mensaje 0x200 con base en el DBC o la especificación correspondiente:

Tabla 1.1 Ejemplo de caracteristicas del identificador 0x200

Byte(s)	Contenido del byte	Señal	Unidades	Factor de escala	Desplazamiento
0 1	0A F0	Velocidad del motor	RPM	1	0
2 3	03 D9	Temperatura de motor	°C	0.1	0
4	64	Nivel de combustible	%	1	0
5 6	0C 1C	Presión de aceite	kPa	0.1	0

Procedimiento para la decodificación:

1. Velocidad de Motor:

- Bytes 0 y 1. **0x0A** y **0xF0** → **0x0AF0** en hexadecimal
- Conversión hexadecimal a decimal: **0x0AF0** hexadecimal equivale a **2800**, decimal
- Factor de escala 1, por lo tanto 2800 * 1 =2800
- Velocidad del motor es de **2800 RPM**

2. Temperatura de Motor:

- Bytes 2 y 3. **0x03** y **0xD9** → **0x03D9** en hexadecimal
- Conversión hexadecimal a decimal: **0x03D9** hexadecimal equivale a **985**, decimal
- Factor de escala 0.1, por lo tanto 985 * 0.1 = 98.5
- Temperatura del motor es de **98.5°C**

3. Nivel de Combustible:

- Byte 4. **0x64** en hexadecimal
- Conversión hexadecimal a decimal: **0x64** hexadecimal equivale a **100**, decimal
- Factor de escala 1, por lo tanto 100 * 1 = 100
- Nivel del combustible es de **100 %**

4. Presión de Aceite:

- Bytes 5 y 6. **0x0C** y **0x1C** → **0x0C1C** en hexadecimal

- Conversión hexadecimal a decimal: **0x0C1C** hexadecimal equivale a **3100**, decimal
- Factor de escala 0.1, por lo tanto 3100 * 0.1 = 310
- Presión de aceite es de **310 kPa**

Este es el proceso para decodificar los mensajes CAN y obtener los valores reales de cada señal en el campo de datos. En una implementación práctica, un software o una herramienta de decodificación con el archivo DBC simplifica este proceso.

1.2 Norma SAE J1939

La norma SAE J1939 es un estándar de comunicación desarrollado por la Sociedad de Ingenieros de Automoción (SAE, *Society of Automotive Engineers*) que define un protocolo para la transmisión de datos en redes CAN utilizadas en vehículos comerciales y aplicaciones fuera de carretera. Este estándar es ampliamente adoptado en sectores como el transporte pesado, maquinaria agrícola, equipos de construcción y embarcaciones, donde múltiples sistemas electrónicos necesitan intercambiar información de manera eficiente y confiable [9].

SAE J1939 se basa en el protocolo CAN 2.0B y extiende sus capacidades para soportar redes más complejas mediante un identificador de 29 bits, que proporciona mayor flexibilidad en la organización de los mensajes. Este estándar organiza la información en un formato jerárquico que facilita la interoperabilidad entre dispositivos de diferentes fabricantes, permitiendo una integración más sencilla y reduciendo los costos de desarrollo [9].

Además, SAE J1939 define mensajes específicos para diversas aplicaciones, como la gestión de motores, sistemas de transmisión, frenos, diagnósticos y telemática. También introduce conceptos clave como los PGNs (*Parameter Group Numbers*), que estructuran la información transmitida, y los SPNs (*Suspect Parameter Numbers*), que detallan las señales específicas dentro de los mensajes. Estas características hacen de SAE J1939 una herramienta esencial para la conectividad y el control en vehículos industriales modernos, proveyendo de una operación segura, eficiente y compatible con estándares internacionales [9].

El apartado SAE J1939/71_201205 es una de las secciones más importantes del estándar SAE J1939, ya que define el formato y contenido de los mensajes que se transmiten en una red J1939. Este documento detalla los *Parameter Group Numbers* (PGNs) y los *Suspect Parameter Numbers* (SPNs) que se utilizan para estructurar y describir los datos enviados entre los nodos de la red [9].

1.2.1 Principales Características del SAE J1939/71_201205

Definición de PGNs (*Parameter Group Numbers*):

- Los PGNs son números únicos que identifican grupos de parámetros relacionados. Cada PGN define un conjunto de datos específicos que deben ser interpretados de manera uniforme por todos los dispositivos en la red.
- Cada PGN está asociado a un rango de identificadores CAN de 29 bits, garantizando la compatibilidad con el Formato Extendido del protocolo CAN.
- Ejemplo: El PGN 61444 puede contener datos relacionados con el "Torque del Motor y RPM" [10].

Descripción de SPNs (*Suspect Parameter Numbers*):

- Los SPNs son identificadores individuales para cada señal o parámetro dentro de un PGN. Por ejemplo, en un PGN que describe el estado del motor, los SPNs pueden representar la temperatura del refrigerante, la presión de aceite, o la velocidad del motor.
- Cada SPN incluye información sobre su rango, resolución y unidad de medida.
- Ejemplo: El SPN 190 puede representar la "Temperatura del Refrigerante del Motor".

Estructura de los Mensajes J1939:

- J1939-71 especifica cómo deben estructurarse los mensajes, incluyendo los campos de datos, la longitud de estos (que puede ser de hasta 8 bytes en un solo mensaje), y el significado de cada bit o conjunto de bits.
- Define cómo los datos deben ser interpretados, por ejemplo, si son enteros, números con signo o valores escalados [10].

Definición de Unidades y Escalado:

- Cada SPN incluye información detallada sobre la escala y la unidad en que debe interpretarse el valor, asegurando la consistencia en la interpretación de los datos [10].

Mensajes de Diagnóstico y Control:

- J1939-71 también cubre mensajes específicos para diagnósticos, incluidos los códigos de falla que los sistemas de control pueden emitir para informar problemas o condiciones fuera de rango [10].

Compatibilidad con Sistemas Multidominio:

- Los mensajes definidos en J1939-71 se utilizan en múltiples dominios, como motores, transmisiones, frenos y otros sistemas de vehículos, asegurando que las comunicaciones sean eficientes y universales [10].

Supongamos que se transmite un mensaje con el PGN 65263 (Temperatura del motor):

- Identificador CAN de 29 bits: Contiene el PGN y la dirección del origen y destino.
- Datos (Campo de 8 bytes):
    - Byte 1: Temperatura del refrigerante (SPN 190).
    - Byte 2: Presión del aceite del motor (SPN 100).
    - Byte 3: Temperatura del aire de admisión (SPN 171).
    - Etc.

Cada byte o conjunto de bits debe ser interpretado de acuerdo con la definición de los SPNs en el estándar.

Este apartado es fundamental para garantizar que los dispositivos en una red J1939 puedan comunicarse de manera eficiente y consistente. Al estandarizar los mensajes y parámetros, J1939-71 asegura que los componentes de diferentes fabricantes sean interoperables y que los datos sean interpretados de manera uniforme en toda la red. Esto es esencial para aplicaciones como el monitoreo del estado del motor, la gestión de flotas y los diagnósticos remotos.

2. Materiales y equipo utilizado

El presente trabajo describe el desarrollo de un sistema prototipo para la adquisición de datos provenientes del BUS CAN en vehículos de autotransporte, integrando herramientas electrónicas y software especializado. El diseño del sistema incluyó el uso de materiales de bajo costo y fácil acceso, permitiendo la creación de una solución funcional para la captura, almacenamiento y análisis de variables críticas relacionadas con el comportamiento de los vehículos.

Entre los principales materiales utilizados se encuentran una placa Arduino UNO, empleada como unidad central de procesamiento, y un módulo MCP2515, que actúa como interfaz para la comunicación con el BUS CAN. Adicionalmente, se utilizaron conectores OBD2 y J1939 para garantizar la compatibilidad del sistema con diferentes categorías de vehículos, incluyendo automóviles ligeros, vehículos eléctricos y vehículos pesados. Una computadora portátil fue integrada como herramienta de almacenamiento y procesamiento de datos, sirviendo como plataforma para ejecutar una interfaz desarrollada en LabVIEW©.

A continuación, se describen los principales circuitos electrónicos utilizados.

2.1 Arduino UNO

El Arduino UNO (ver Figura 2.1) es una placa de desarrollo basada en el microcontrolador ATmega328P, ampliamente utilizada en proyectos de electrónica y programación debido a su versatilidad, facilidad de uso y bajo costo. Diseñada específicamente para aplicaciones educativas, de prototipado y desarrollo de hardware, esta placa se ha convertido en una de las herramientas más populares en el ámbito de la electrónica y el Internet de las Cosas (*IoT*) [11]. En la Tabla 2.1 se pueden observar algunas características del Arduino UNO.

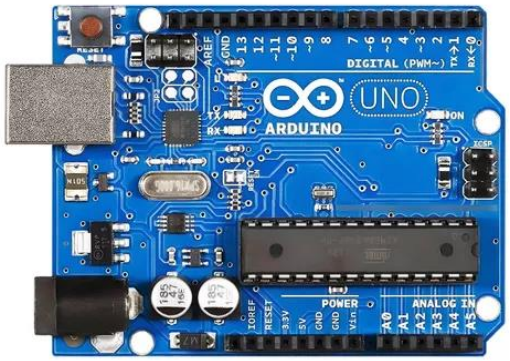

Figura 2.1 Arduino UNO

Fuente: Arduino (2024).

Tabla 2.1 caracteristicas del Arduino UNO

Característica	Descripción
Microcontrolador	ATmega328P de 8 bits, con una frecuencia de reloj de 16 MHz.
Memoria	Memoria Flash de 32 KB (con 0.5 KB utilizados por el gestor de arranque) 2 KB de SRAM y 1 KB de EEPROM para el almacenamiento de datos.
Entradas y salidas	14 pines digitales, de los cuales 6 pueden funcionar como salidas PWM. 6 entradas analógicas con resolución de 10 bits.
Conectividad	Interfaz USB para comunicación con la computadora y alimentación.
Interfaz de comunicación	Protocolos I2C, SPI y UART integrados.
Compatibilidad	Ecosistema de bibliotecas y tarjetas que extienden su funcionalidad.

En el contexto del desarrollo de un lector para señales CAN, el Arduino UNO desempeña un papel central como unidad de procesamiento y control del sistema. Su relevancia radica en los siguientes aspectos:

- El Arduino UNO se utiliza para gestionar la comunicación SPI necesaria para interactuar con el módulo MCP2515, encargado de la recepción y transmisión de datos desde el BUS CAN.
- La amplia disponibilidad de bibliotecas y recursos para el Arduino facilita la implementación del código necesario para capturar, procesar y enviar los datos CAN a la computadora.
- Su capacidad para trabajar con diferentes voltajes y protocolos de comunicación lo hace compatible con otros componentes del sistema, como los conectores OBD2 y J1939.
- Al tratarse de un dispositivo de bajo costo y ampliamente distribuido, el Arduino UNO permite desarrollar soluciones económicas sin comprometer la funcionalidad del sistema.
- Su tamaño compacto y consumo energético reducido facilitan su integración en entornos vehiculares, haciendo posible realizar pruebas experimentales en movimiento.

En este proyecto, el Arduino UNO fue fundamental para capturar los datos CAN, procesarlos y enviarlos a una computadora portátil para su almacenamiento y análisis mediante una interfaz en LabVIEW©. Su capacidad para gestionar múltiples protocolos de comunicación y su facilidad de programación contribuyeron significativamente al éxito en la implementación del sistema.

2.1.1 Librería *Seeed_Arduino_CAN-master*

La librería *Seeed_Arduino_CAN-master* es una herramienta de software diseñada para facilitar la interacción con módulos compatibles con el protocolo CAN en plataformas Arduino. Desarrollada por *Seeed Studio*, esta librería está optimizada para trabajar con módulos basados en controladores CAN como el MCP2515 y transceptores específicos, incluyendo los módulos *CAN Shield* V1.2 de *Seeed* [12].

De las características relevantes de esta librería se puede resaltar lo siguiente:

- Diseñada principalmente para módulos MCP2515 y MCP2551.
- Compatible con diversas placas Arduino, como el Arduino UNO, Mega y Leonardo.
- Compatible con velocidades de comunicación estándar (por ejemplo, 125 kbps, 250 kbps, 500 kbps).
- Admite configuraciones personalizadas para ajustar el *baud rate* según los requisitos del sistema.
- Envío y recepción de mensajes CAN.
- Filtrado y enmascarado de mensajes para capturar solo los datos relevantes.

- Soporte para mensajes estándar (11 bits) y extendidos (29 bits).
- Basada en la comunicación SPI para interactuar con el módulo MCP2515.
- Fácil de integrar mediante comandos intuitivos que simplifican el envío y recepción de datos.

Su diseño modular y bien documentado facilita a los desarrolladores implementar proyectos basados en CAN sin necesidad de profundizar en los detalles de bajo nivel del protocolo. Esta librería es de fácil Integración con otros módulos de *Seeed Studio*, maximizando la compatibilidad y funcionalidad. Además, permite adaptarse a diferentes necesidades mediante ajustes en los parámetros de la librería.

En el marco del desarrollo de sistemas para adquisición de datos CAN, como el presentado en este proyecto, la librería *Seeed_Arduino_CAN-master* resulta especialmente útil por su simplicidad y eficiencia. Al ser compatible con el módulo MCP2515, la librería permite configurar y operar el sistema de forma rápida, enfocándose en tareas como:

- Capturar datos del BUS CAN de diferentes vehículos.
- Filtrar mensajes para aislar variables específicas, como velocidad o parámetros del motor.
- Almacenar los datos capturados para su posterior análisis.

La librería se puede descargar directamente en https://github.com/Seeed-Studio/Seeed_Arduino_CAN.

2.2 Módulo MCP2515

El módulo MCP2515 es un controlador de comunicación CAN que opera a través de la interfaz SPI. Diseñado para facilitar la integración del protocolo CAN en sistemas embebidos, este módulo es ampliamente utilizado en aplicaciones automotrices, industriales y de monitoreo, gracias a su capacidad para gestionar la comunicación CAN de manera eficiente y su compatibilidad con microcontroladores de bajo costo, como el Arduino UNO.

El MCP2515 es un circuito integrado diseñado por Microchip que actúa como intermediario entre un microcontrolador y el BUS CAN. Este chip implementa las funciones del protocolo CAN 2.0A/B, lo que incluye soporte para mensajes estándar (11 bits) y extendidos (29 bits) [13].

El módulo MCP2515 es una placa que integra este chip junto con un transceptor CAN, como el MCP2551, para convertir las señales digitales en

niveles eléctricos compatibles con el BUS CAN. Esto permite que el sistema interactúe físicamente con la red vehicular [13].

Algunas características principales del módulo MCP2515 son las siguientes:

- Protocolo CAN: Compatible con el estándar CAN 2.0A/B.
- Velocidades soportadas: Hasta 1 Mbps.
- Interfaz SPI: Comunicación eficiente con microcontroladores a través de SPI, con una velocidad de reloj de hasta 10 MHz.
- Filtros y máscaras: Integrados para filtrar mensajes no deseados y reducir la carga de procesamiento del microcontrolador.
- Búferes de mensajes: Dos búferes para transmisión y recepción de datos.
- Voltaje de operación: Funciona a 5V.
- Cristal externo: Requiere un cristal para definir su frecuencia de operación, comúnmente de 8 MHz o 16 MHz.
- Componentes del módulo MCP2515 [13]

A continuación, se describen los pines para la conexión con el microcontrolador mediante la interfaz SPI, incluyendo los pines de alimentación y comunicación CAN:

- CS (*Chip Select*): Selección del dispositivo.
- SI (*Serial Input*): Entrada de datos al MCP2515.
- SO (*Serial Output*): Salida de datos del MCP2515.
- SCK (*Serial Clock*): Señal de reloj.
- CANH (*CAN High*) y CANL (CAN Low): Conexiones físicas al BUS CAN.
- VCC: Alimentación del módulo (5V).
- GND: Tierra del sistema.
- Pin de interrupción (INT): Genera una señal al microcontrolador cuando hay datos nuevos disponibles o un evento que requiere atención, como errores en el BUS CAN.
- LED indicador: Indica actividad en la comunicación CAN o en la conexión SPI [13].

En la Figura 2.2 se puede observar un esquema del módulo MCP2515 y sus partes.

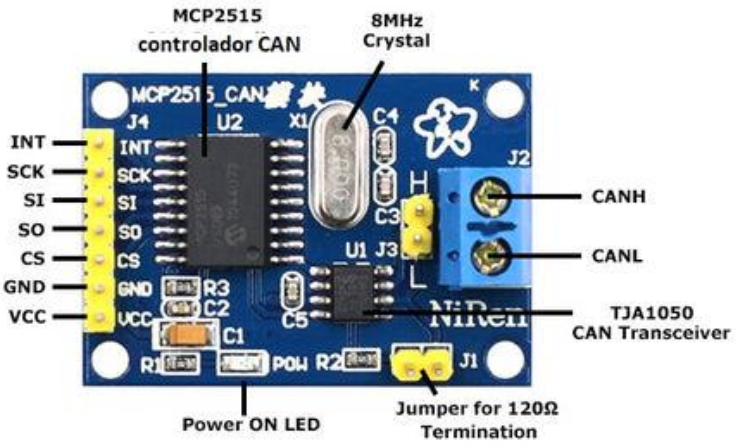

Figura 2.2 Modulo MCP2515

Fuente: Talos Electronics (2021).

En este proyecto, el módulo MCP2515 es crucial, ya que permite al microcontrolador (Arduino UNO) interactuar directamente con el BUS CAN. Sus filtros y máscaras optimizan la captura de mensajes relevantes, mientras que el transceptor CAN asegura la compatibilidad eléctrica con la red vehicular. Al integrarse con un cristal de 8 MHz, el módulo fue configurado para operar a una velocidad de 500 kbps, garantizando una comunicación estable y eficiente durante las pruebas en distintos vehículos.

Este módulo, en combinación con un microcontrolador y herramientas de análisis, ofrece una solución accesible para implementar sistemas de monitoreo y diagnóstico basados en el protocolo CAN.

2.3 Conector OBD2 y J1939

El OBD2 (*On-Board Diagnostics 2*) es un estándar de diagnóstico vehicular utilizado en automóviles ligeros y algunos vehículos comerciales para monitorear y diagnosticar sistemas relacionados con las emisiones, el motor y otros parámetros. El conector OBD2 es obligatorio en todos los vehículos fabricados después de 1996 en muchos países, como parte de las normativas ambientales [14]. En la Figura 2.3 se pueden observar las terminales principales del conector OBD2.

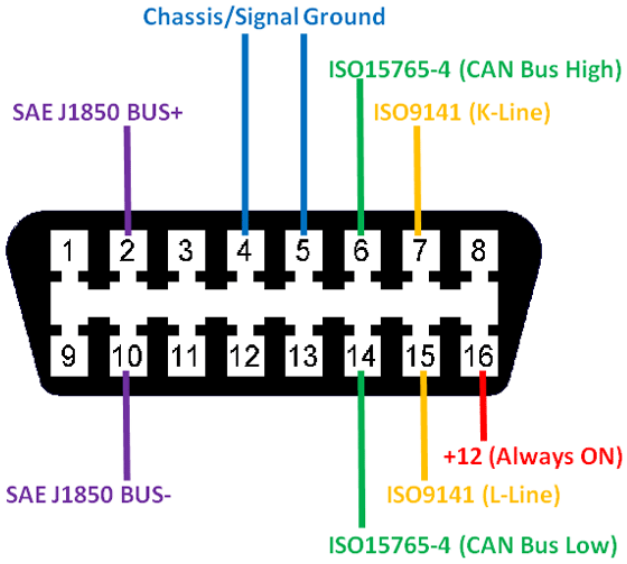

Figura 2.3 Conector OBD2

Fuente: OBD2 AUSTRALIA (2020).

Características principales

- Formato: Conector trapezoidal de 16 pines.
- Ubicación típica: Debajo del tablero, cerca del conductor.
- Protocolos soportados: SAE J1850, ISO 9141-2, ISO 14230-4 (KWP2000), ISO 15765-4 (CAN).
- Compatibilidad CAN: Desde 2008, el protocolo CAN es obligatorio en todos los vehículos OBD2.

Para la comunicación CAN, el conector OBD2 utiliza los siguientes pines:

- Pin 6 (*CAN High*): Línea alta del BUS CAN.
- Pin 14 (*CAN Low*): Línea baja del BUS CAN.
- Pin 4 (*Chassis Ground*): Tierra del chasis.
- Pin 5 (*Signal Ground*): Tierra de la señal.
- Pin 16 (*Battery Positive*): Alimentación directa de 12V del vehículo.

Estos pines permiten que dispositivos externos, como lectores CAN o herramientas de diagnóstico, accedan a los datos del BUS CAN del vehículo para monitorear y analizar variables dinámicas.

2.3.1 Conector J1939: Descripción y Conexiones para CAN

El J1939 es un estándar de comunicación utilizado principalmente en vehículos comerciales y maquinaria pesada. Este protocolo está basado en el BUS CAN y se enfoca en la interoperabilidad y el intercambio de datos entre unidades electrónicas de control (ECUs) en vehículos pesados, como camiones, autobuses y tractores [15]. En la Figura 2.4 se puede observar un esquema del conector utilizado para vehículos pesados.

Pin	Value
A	Ground
B	+12V
C	CAN1/J1939 Hi
D	CAN1/J1939 Lo
E	CAN1/J1939 Shield
F	J1708/J1587 Hi
G	J1708/J1587 Lo
H	OEM Specific
J	ISO9141 K-Line

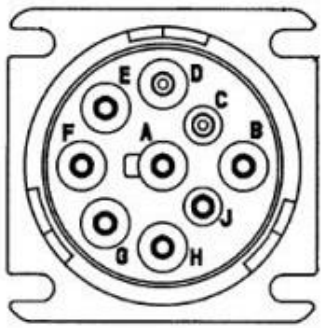

Figura 2.4 Conector J1939

Fuente: ELECTRICAL ENGINEERING (2019).

Características principales

- Formato: Conector redondo estandarizado, conocido como *Deutsch* 9 pines o conector J1939-9.
- Ubicación típica: En el área del tablero o la cabina del vehículo pesado.
- Protocolo base: SAE J1939, que utiliza CAN 2.0B con mensajes extendidos de 29 bits.
- En el conector J1939, las señales CAN están asignadas de la siguiente manera:
- Pin C (CAN High): Línea alta del BUS CAN.
- Pin D (CAN Low): Línea baja del BUS CAN.
- Pin A (*Ground*): Tierra del sistema.
- Pin B (*Battery Positive*): Alimentación del sistema (12V o 24V, dependiendo del vehículo).

Opcionalmente, el conector puede incluir:

- Pin E: CAN secundario (*CAN Shielded*).
- Pin F: Línea baja del CAN secundario.

En la Tabla 2.2 se puede observar las diferencias entre el conector OBD2 y el J1939.

Tabla 2.2 Comparación entre conector OBD2 y J1939

Característica	OBD2	J1939
Uso principal	Automóviles ligeros	Vehículos pesados y maquinaria
Protocolo base	CAN 2.0A/B (11/29 bits)	CAN 2.0B (29 bits)
Alimentación	12V	12V o 24V
Número de pines	16	9
Ubicación típica	Debajo del tablero	Tablero o cabina

En este proyecto, tanto el conector OBD2 como el J1939 son esenciales para la captura de datos CAN. El conector OBD2 permitió realizar pruebas en vehículos ligeros, como el Honda Accord y el vehículo eléctrico, mientras que el conector J1939 fue clave para analizar las variables CAN en el tractocamión Kenworth. Ambos conectores aseguraron una conexión física confiable entre el módulo MCP2515 y el BUS CAN de los diferentes tipos de vehículos, habilitando el acceso a los datos críticos para el análisis y posprocesamiento.

2.4 LabVIEW©

LabVIEW© (*Laboratory Virtual Instrument Engineering Workbench*) es una herramienta gráfica de programación desarrollada por National Instruments, ampliamente utilizada en aplicaciones de instrumentación, monitoreo y control. Su capacidad para interactuar con dispositivos externos y protocolos de comunicación como CAN lo convierte en una opción poderosa para desarrollar interfaces personalizadas que permitan visualizar, procesar y analizar datos del BUS CAN. LabVIEW© soporta una amplia gama de dispositivos de hardware para la comunicación CAN, como Interfaces CAN de *National Instruments* (por ejemplo, NI USB-8473, PCI-CAN) [16], módulos de terceros compatibles con los controladores NI-XNET o con bibliotecas externas. LabVIEW© puede interactuar con hardware MCP2515 o similar mediante bibliotecas externas, para este proyecto únicamente se utilizó la comunicación serial para leer los datos CAN usando el Arduino UNO.

LabVIEW© utiliza bloques gráficos llamados VIs (Virtual Instruments), que permiten crear interfaces de usuario personalizadas para visualizar datos del BUS CAN de forma dinámica. Estas interfaces incluyen:

- Gráficos en tiempo real: Para visualizar variables como velocidad, temperatura o presión.

- Paneles de control: Con botones, indicadores y sliders para interactuar con el sistema.
- Tablas y registros: Para almacenar y mostrar mensajes CAN y sus identificadores (ID) [16].

LabVIEW© facilita el procesamiento de los datos recibidos del BUS CAN mediante funciones predefinidas, tales como:

- Filtrado de mensajes específicos mediante sus identificadores (ID).
- Decodificación de mensajes según estándares como SAE J1939 u OBD2.
- Conversión de datos crudos en magnitudes físicas legibles, como velocidad (km/h) o temperatura (°C) [16].

Ejemplo de flujo de trabajo en LabVIEW© para la interfaz de visualización de datos CAN:

a) Seleccionar la interfaz CAN y definir parámetros como la velocidad de transmisión (*baud rate*).
b) Usar bloques gráficos para capturar mensajes CAN del hardware.
c) Aplicar filtros y máscaras para aislar los datos relevantes.
d) Mostrar los datos en gráficos, indicadores y tablas en el panel frontal del VI.
e) Guardar los datos para análisis posterior en archivos como CSV o bases de datos [16].

Ventajas de usar LabVIEW© para el BUS CAN

- Programación gráfica: Facilita el desarrollo rápido de aplicaciones sin necesidad de escribir código extenso.
- Flexibilidad: Permite diseñar interfaces personalizadas adaptadas a necesidades específicas.
- Multiplataforma: Compatible con Windows y sistemas embebidos para aplicaciones de monitoreo en tiempo real.
- Extensibilidad: Integra funciones adicionales mediante módulos o drivers específicos [16].
- Compatibilidad industrial: Ideal para entornos de prueba y monitoreo en tiempo real en el sector automotriz e industrial.

Con su capacidad para trabajar con hardware CAN y su enfoque en la visualización y el análisis, LabVIEW© es una herramienta versátil y poderosa para proyectos relacionados con el monitoreo y diagnóstico de datos vehiculares.

3. Metodología

El desarrollo de un lector CAN eficiente y accesible representa un desafío clave en proyectos de adquisición de datos vehiculares. Este trabajo presenta la metodología empleada para la construcción de un sistema lector basado en un Arduino UNO y un módulo MCP2515, complementado por una interfaz gráfica desarrollada en LabVIEW© que permite visualizar y almacenar información en una computadora portátil. La integración de estos elementos proporciona una solución económica y funcional para la adquisición de datos del BUS CAN, ampliamente utilizado en vehículos modernos.

Para validar el correcto funcionamiento del lector CAN, inicialmente se diseñó un generador de mensajes CAN utilizando un segundo Arduino UNO. Este generador simula el comportamiento de la computadora de un vehículo, transmitiendo mensajes CAN aleatorios al lector CAN. Mediante este esquema, se verificó que el sistema era capaz de recibir y procesar correctamente los mensajes CAN, asegurando su fiabilidad en un entorno controlado.

Posteriormente, se realizaron pruebas con vehículos reales, incluyendo dos vehículos ligeros y uno pesado, en la pista de pruebas del Instituto Mexicano del Transporte. Estas pruebas permitieron evaluar la capacidad del lector para interactuar con sistemas CAN reales, extrayendo y almacenando datos provenientes de los sensores y unidades de control vehicular.

La combinación de hardware accesible, un entorno de programación gráfico como LabVIEW© y un enfoque sistemático de validación garantiza que el dispositivo no solo sea funcional, sino también adaptable a diferentes escenarios y necesidades en el ámbito de la adquisición de datos vehiculares. La metodología propuesta establece un marco sólido para futuros desarrollos en esta área.

3.1 Construcción del lector CAN

La construcción del lector CAN comenzó con la integración de los componentes principales: el Arduino UNO y el módulo MCP2515, los cuales forman la base del sistema para adquirir mensajes del BUS CAN. Este proceso incluyó la conexión física entre ambos dispositivos, la configuración inicial, y la programación utilizando la biblioteca *Seeed_Arduino_CAN-master* en el entorno de desarrollo Arduino IDE.

El módulo MCP2515 se conecta al Arduino UNO mediante la interfaz SPI, que permite una comunicación rápida y eficiente. Las conexiones específicas entre ambos dispositivos son las siguientes:

- VCC (MCP2515) ↔ 5V (Arduino UNO): Alimenta el módulo MCP2515.
- GND (MCP2515) ↔ GND (Arduino UNO): Proporciona tierra común para ambos dispositivos.
- CS (*Chip Select*, MCP2515) ↔ Pin 10 (Arduino UNO): Señal utilizada para seleccionar el módulo MCP2515 en la comunicación SPI.
- SI (MOSI, MCP2515) ↔ Pin 11 (Arduino UNO): Canal para enviar datos desde el Arduino hacia el MCP2515.
- SO (MISO, MCP2515) ↔ Pin 12 (Arduino UNO): Canal para recibir datos desde el MCP2515 hacia el Arduino.
- SCK (*Serial Clock*, MCP2515) ↔ Pin 13 (Arduino UNO): Línea de reloj que sincroniza la comunicación SPI.
- INT (Interrupción, MCP2515) ↔ Pin 2 (Arduino UNO): Línea de interrupción para notificar al Arduino cuando hay nuevos datos disponibles.

En la Figura 3.1 se puede observar un diagrama simple de la conexión física entre el Arduino UNO y El MCP2515.

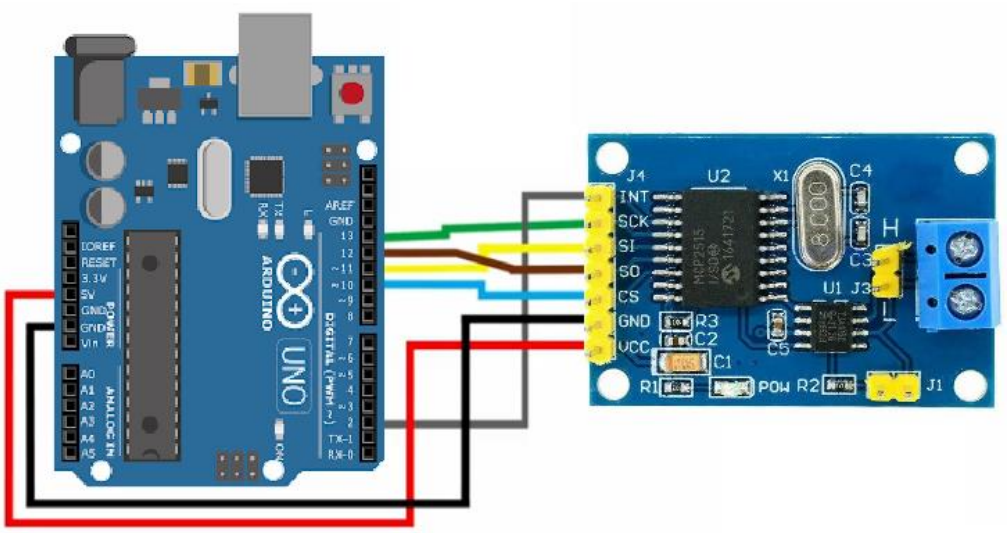

Figura 3.1 Conexión entre Arduino UNO y MCP2515

Una vez realizada la conexión física, el lector CAN fue programado utilizando el entorno Arduino IDE y la biblioteca Seeed_Arduino_CAN-master, que facilita la comunicación con el módulo MCP2515 y la configuración de los parámetros del BUS CAN. Los pasos principales en la programación fueron:

- Inicialización del módulo MCP2515: Configuración de la velocidad de transmisión (500 kbps) y los filtros necesarios para capturar mensajes específicos del BUS CAN.
- Recepción de mensajes CAN: Uso de las funciones proporcionadas por la biblioteca para leer mensajes CAN, incluyendo su identificador (ID) y datos asociados.
- Envío de mensajes de prueba (opcional): Implementación de una función para transmitir mensajes CAN, útil para pruebas en sistemas controlados.
- Integración con LabVIEW©: Los datos recibidos se enviaron a la computadora a través del puerto serial, donde se procesaron y visualizaron en tiempo real utilizando una interfaz gráfica desarrollada en LabVIEW©.

A continuación, se explica a detalle el funcionamiento del código utilizado para el lector CAN:

En la Figura 3.2 se observa la inclusión de bibliotecas y configuración de Pines.

```
#include <SPI.h>
#include "mcp2515_can.h"

const int SPI_CS_PIN = 10;  // Pin CS del MCP2515
const int CAN_INT_PIN = 2;  // Pin de interrupción del MCP2515

mcp2515_can CAN(SPI_CS_PIN);  // Inicializa el MCP2515 en el pin CS
```

Figura 3.2 Bibliotecas y configuración de pines

Bibliotecas utilizadas:

- SPI.h: Permite la comunicación SPI entre el Arduino UNO y el MCP2515.
- mcp2515_can.h: Controla las funciones del módulo MCP2515 para el manejo del protocolo CAN.

Definición de pines:

- SPI_CS_PIN (10): Selección de chip (CS) para la comunicación SPI con el MCP2515.
- CAN_INT_PIN (2): Pin de interrupción para notificaciones del MCP2515 cuando se recibe un mensaje.

Creación del objeto CAN: Configura el MCP2515 en el pin CS especificado.

En la Figura 3.3 se observa la configuración inicial para el módulo MCP2515.

```
void setup() {
    Serial.begin(115200);
    while (!Serial) {}

    // Inicializa el MCP2515 a 500 kbps
    while (CAN_OK != CAN.begin(CAN_500KBPS)) {
        Serial.println("Error al iniciar MCP2515, intentando de nuevo...");
        delay(100);
    }
    Serial.println("MCP2515 inicializado correctamente");
}
```

Figura 3.3 Configuración inicial

Serial.begin(115200): Configura la comunicación serial con la computadora a 115200 baudios para depuración.

Inicialización del MCP2515:

- El módulo MCP2515 se configura para operar a 500 kbps (velocidad típica en vehículos).
- Si la inicialización falla, el programa lo intenta repetidamente hasta que sea exitoso.

Mensaje de éxito: Se imprime un mensaje en el monitor serial al completar la configuración.

En la Figura 3.4 se muestra el bucle principal donde se reciben los mensajes CAN.

```
void loop() {
    // Comprueba si hay mensajes CAN recibidos
    if (CAN_MSGAVAIL == CAN.checkReceive()) {
        // ID del mensaje CAN
        unsigned long canId;

        // Buffer para almacenar los datos recibidos
        byte len = 0;
        byte buf[8];

        // Lee el mensaje CAN
        CAN.readMsgBuf(&len, buf);
        canId = CAN.getCanId();

        // Imprime los detalles del mensaje recibido
        Serial.print("Mensaje recibido con ID: ");
        Serial.print(canId, HEX);
        Serial.print(" Datos: ");

        for (int i = 0; i < len; i++) {
            Serial.print(buf[i], HEX);
            Serial.print(" ");
        }
        Serial.println();
    }
}
```

Figura 3.4 Bucle principal

Comprobación de Mensajes Disponibles:

- CAN.checkReceive() verifica si el MCP2515 tiene un mensaje pendiente.
- Si hay un mensaje disponible (CAN_MSGAVAIL), el programa procede a leerlo.

Lectura del Mensaje CAN:

- CAN.readMsgBuf(&len, buf); obtiene el contenido del mensaje:
  - len: Número de bytes en el mensaje.
  - buf: Array donde se almacenan los datos recibidos (hasta 8 bytes).
- CAN.getCanId(); obtiene el identificador único del mensaje CAN.

Impresión del Mensaje Recibido:

- Imprime en el monitor serial:
  - El ID del mensaje en formato hexadecimal.
  - Los datos del mensaje, byte por byte, también en formato hexadecimal.

En la Figura 3.5 se puede observar el flujo de ejecución del lector CAN.

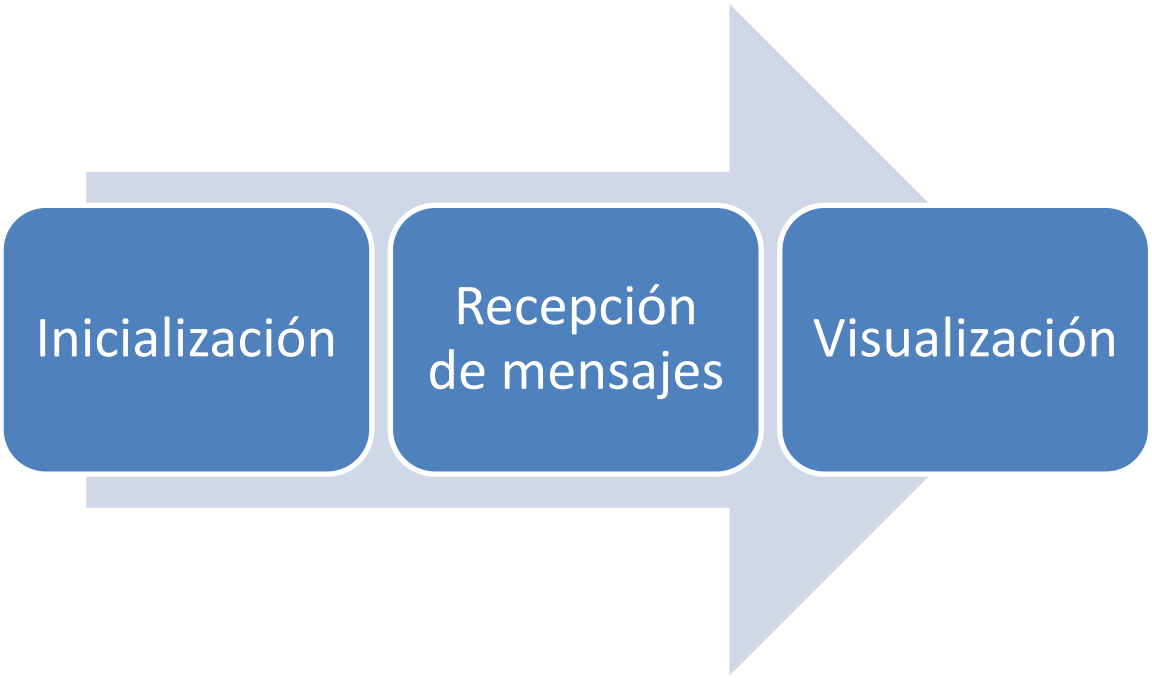

Figura 3.5 Flujo de ejecución del lector CAN

Inicialización:

- Configura el MCP2515 y establece la velocidad del BUS CAN a 500 kbps.
- Se asegura de que la comunicación esté lista antes de continuar.

Recepción de Mensajes:

- El lector monitorea continuamente el BUS CAN.

- Cuando detecta un mensaje, lo lee y obtiene su identificador (ID) y datos.

Visualización:

- Imprime los detalles del mensaje en el monitor serial, lo que permite validar la recepción de datos.

3.2 Desarrollo de interfaz

La interfaz gráfica fue desarrollada utilizando la plataforma de programación gráfica LabVIEW©, ampliamente reconocida por su versatilidad para la integración de hardware externo y la creación de aplicaciones personalizadas. En este caso, LabVIEW© permitió una integración eficiente entre el hardware de adquisición basado en Arduino UNO y MCP2515 y el software encargado de recibir, procesar, visualizar y almacenar datos provenientes del BUS CAN. Gracias a la comunicación serial y las capacidades flexibles de la plataforma, fue posible diseñar una solución robusta y amigable para el monitoreo vehicular. Para el desarrollo de la interfaz se tomó como base una interfaz ya antes desarrollada en el IMT para un lector CAN comercial [17]. En la Figura 3.6 se puede observar la ventana principal de la interfaz.

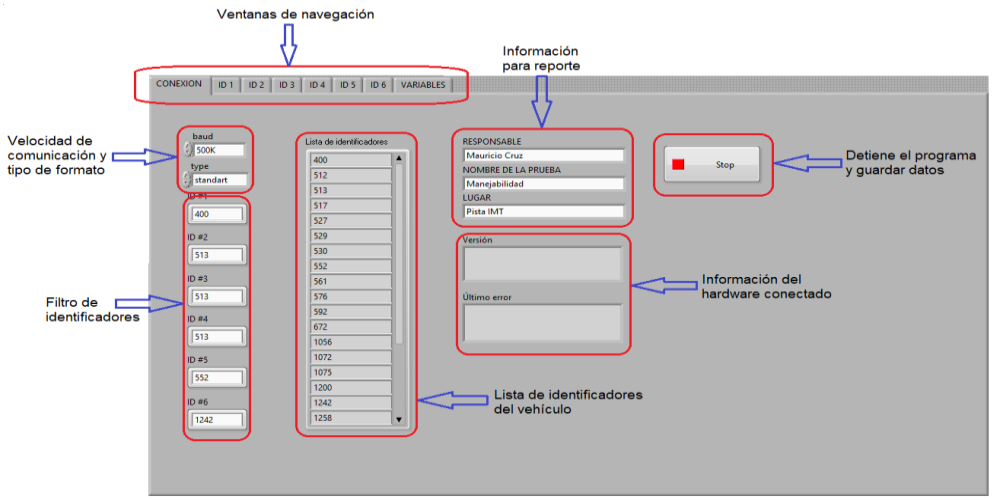

Figura 3.6 Ventana principal de la interfaz

A continuación, se describe el proceso para tratar los datos CAN que entran a través de la comunicación serial a la computadora. La interfaz cuenta con una pantalla inicial que organiza la funcionalidad del sistema en siete apartados principales, cada uno enfocado en tareas específicas:

Ventanas de Navegación: Este apartado facilita la interacción del usuario con diferentes vistas de la interfaz:

- Ventana de Conexión: Configuración inicial del hardware y establecimiento de la comunicación.
- Ventana de Información Filtrada: Visualización de datos de hasta seis identificadores específicos seleccionados por el usuario.
- Ventana de Variables: Presenta información de todos los identificadores procesados, incluyendo ajustes de escala y offset aplicados a las variables cuando es necesario. En caso contrario,

muestra el valor bruto de los bytes o la combinación de dos bytes seleccionados.

Velocidad de Comunicación y Tipo de Formato: Este apartado permite configurar dos aspectos críticos del sistema:

- Velocidad de Comunicación: Sincronización entre el BUS CAN y el hardware de adquisición. La velocidad es ajustable para adaptarse a las especificaciones del vehículo, que dependen de la cantidad de sensores y actuadores, así como de la longitud del cableado del BUS CAN.
- Tipo de Formato: Selección entre Formato Estándar (identificadores de 11 bits) y Formato Extendido (identificadores de 29 bits). Este último es común en vehículos pesados o bajo la norma SAE J1939.

Información para el Reporte: Este apartado permite ingresar datos administrativos y operativos, tales como:

- Responsable de la prueba.
- Tipo de prueba o actividad realizada.
- Ubicación de las pruebas.

Además, se muestra información del hardware conectado, como modelo y versión, y el estado de la conexión. En caso de error, este se registra en el campo "Último Error".

Botón de Detención (*Stop*): Al presionar este botón, el programa detiene la adquisición de datos y despliega una ventana emergente con tres opciones:

- Guardar solo los identificadores y variables filtrados.
- Guardar todos los datos, incluyendo identificadores no filtrados, en un archivo Excel con un desglose detallado.
- Salir sin guardar.

Gráficos de Bytes: En esta sección, se presentan gráficos que visualizan los valores de amplitud de los bytes correspondientes a cada identificador. Por limitaciones de espacio, se muestran gráficos para cuatro bytes simultáneamente. Un selector permite alternar entre el grupo de bytes 1-4 y el grupo 5-8.

Operación con Bytes: Aquí se agrupan herramientas para realizar operaciones avanzadas con los datos CAN:

- Unión de Bytes: Combina dos bytes seleccionados en un solo valor, útil para representar variables multibyte.
- Escala y Offset: Permite aplicar ajustes para convertir valores crudos en magnitudes físicas.
- Nombre de Variable: Se puede asignar un nombre a cada variable para facilitar su identificación.

En la Figura 3.7 se puede observar la ventana de operaciones con bytes.

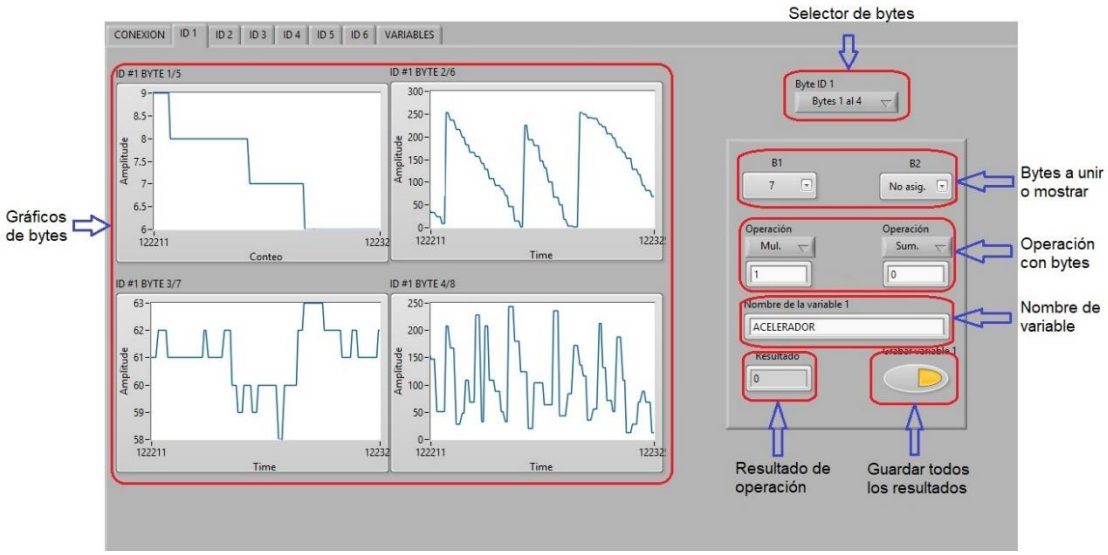

Figura 3.7 Ventana de operaciones con bytes

Resultados y Almacenamiento

- Resultado: Muestra el valor final después de realizar las operaciones de unión, escala y offset.
- Guardar Todos los Resultados: Al activar esta opción, todos los resultados calculados se almacenan automáticamente en el buffer del programa para su posterior exportación.

Para describir de forma general el flujo de trabajo de la interfaz, en la Figura 3.8 se presenta la secuencia lógica de como funciona la interfaz desde que recibe los datos hasta que los almacena.

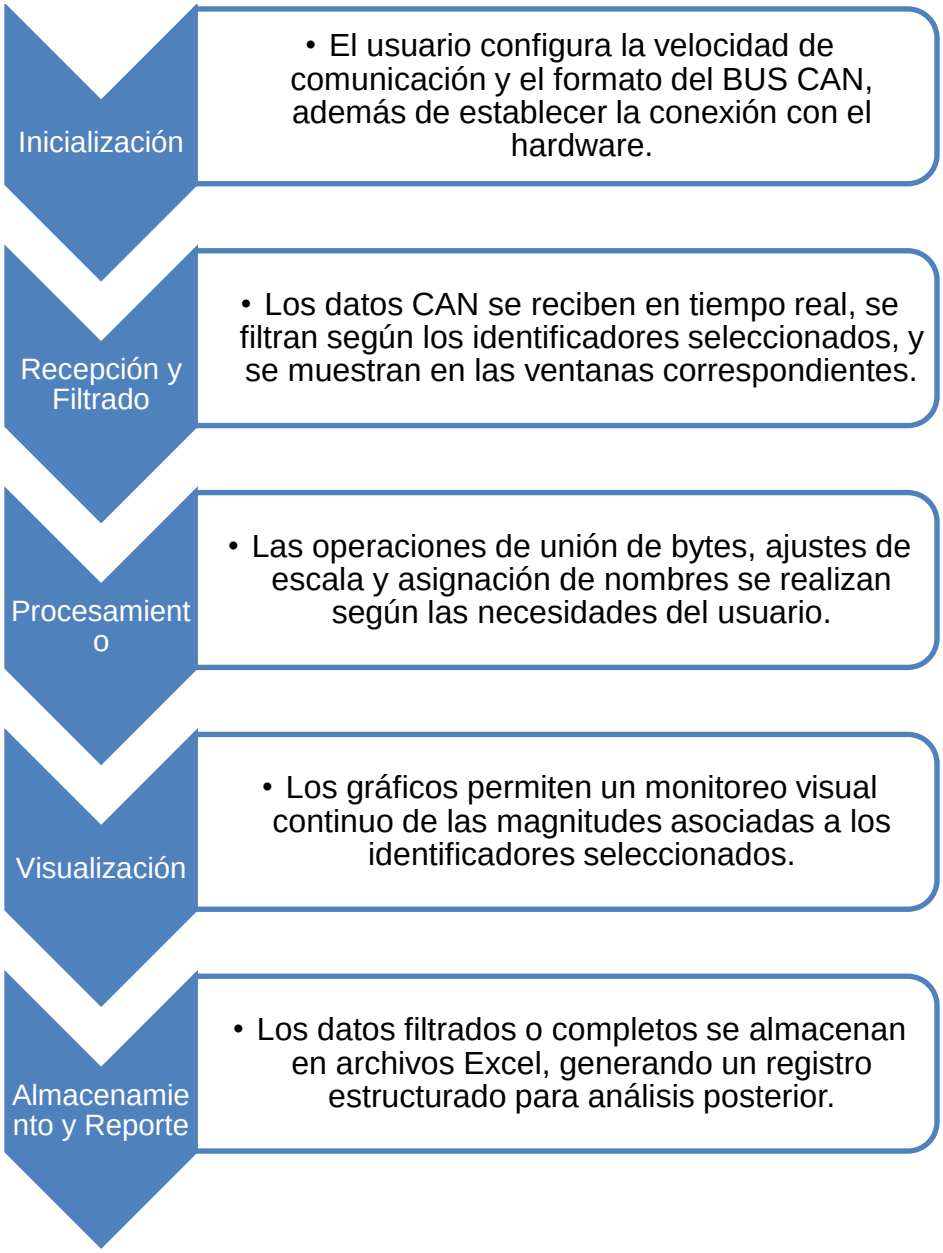

Figura 3.8 Flujo de trabajo de la interfaz

Con esta interfaz, el sistema proporciona una solución robusta y eficiente para la visualización y almacenamiento de datos CAN, satisfaciendo las necesidades de pruebas y monitoreo vehicular en escenarios reales y simulados.

3.3 Validación de dispositivo

Para validar el dispositivo se construyó un simulador de mensajes CAN con otro Arduino UNO y otro modulo MCP2515. Este dispositivo genera mensajes CAN aleatorios, simulando la transmisión de datos que podrían provenir de la computadora de un vehículo. En la Figura 3.9 se puede observar los módulos MCP2515 conectados entre sí a través de las terminales CAN-H y CAN-L, la conexión entre Arduino UNO y modulo MCP2515 es la misma presentada en la sección 3.1 de este documento.

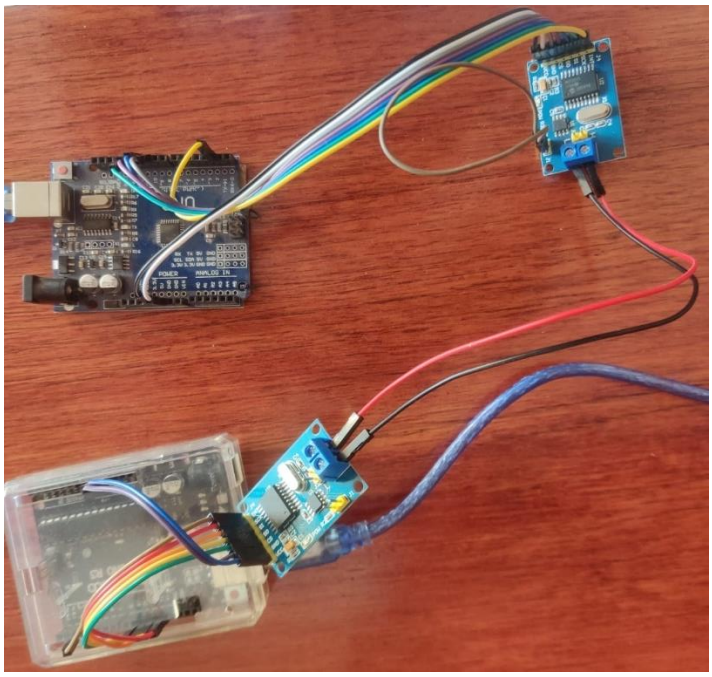

Figura 3.8 Conexión de dos módulos MCP2515 para pruebas de validación del lector CAN

El programa utiliza un Arduino UNO y un módulo MCP2515 configurado para operar a 500 kbps. El siguiente desglose explica cada sección del código:

En la Figura 3.9 se presenta la primera parte del código, esta incluye las bibliotecas y definición de pines.

CAN-ArduinoUNO-sender.ino

```
1   #include <SPI.h>
2   #include "mcp2515_can.h"
3
4   const int SPI_CS_PIN = 10;  // Pin CS del MCP2515
5   const int CAN_INT_PIN = 2;  // Pin de interrupción del MCP2515
6
7   mcp2515_can CAN(SPI_CS_PIN);  // Inicializa el MCP2515 en el pin CS
8   #define MAX_DATA_SIZE 8
9
```

Figura 3.9 Inclusión de bibliotecas y definición de pines

Se incluyen las bibliotecas necesarias:

- SPI.h para la comunicación SPI con el MCP2515.
- mcp2515_can.h para manejar las funciones del módulo MCP2515.

Se definen los pines de conexión:

- SPI_CS_PIN (10): Selección de chip para el módulo MCP2515.
- CAN_INT_PIN (2): Pin de interrupción para notificaciones del MCP2515.

En la Figura 3.10 se observa cómo se inicializa el objeto CAN para manejar la comunicación CAN, esta parte corresponde a la configuración inicial.

```
void setup() {
    Serial.begin(115200);
    while (!Serial) {}

    // Inicializa el MCP2515 a 500 kbps
    while (CAN_OK != CAN.begin(CAN_500KBPS)) {
        Serial.println("Error al iniciar MCP2515, intentando de nuevo...");
        delay(100);
    }
    Serial.println("MCP2515 inicializado correctamente");

    randomSeed(millis());  // Inicializa semilla aleatoria para generar datos aleatorios
}
```

Figura 3.10 configuración inicial

- Serial.begin(115200): Configura la comunicación serial para depuración.
- CAN.begin(CAN_500KBPS): Inicializa el MCP2515 para operar a 500 kbps. Si falla, se intenta nuevamente hasta que sea exitoso.
- randomSeed(millis()): Configura una semilla aleatoria basada en el tiempo de ejecución para generar valores aleatorios.

En la Figura 3.10 se presenta la parte del código que se utiliza para la generación y envío de mensajes CAN.

```
void loop() {
    type = random(4);

    // Genera un ID aleatorio para el mensaje CAN (estándar o extendido según tipo)
    if (type & 0x1) {
        id = random(0x1U << 14);
        id |= (uint32_t)random(0x1U << 15) << 14;
    } else {
        id = random(0x1U << 11);
    }

    // Genera datos aleatorios para el mensaje CAN
    len = random(0, MAX_DATA_SIZE + 1);
    for (int i = 0; i < len; i++) {
        cdata[i] = random(0x100);
    }

    // Envía el mensaje CAN con el ID generado
    CAN.sendMsgBuf(id, bool(type & 0x1), bool(type & 0x2), len, cdata);

    // Imprime el mensaje CAN enviado
    char prbuf[32 + MAX_DATA_SIZE * 3];
    int n = sprintf(prbuf, "TX: [%08lX](%02X) ", (unsigned long)id, (type & 0x1) ? 0x02 : 0x00);

    for (int i = 0; i < len; i++) {
        n += sprintf(prbuf + n, "%02X ", cdata[i]);
    }
    Serial.println(prbuf);

    // Espera antes de enviar el siguiente mensaje
    delay(random(30));
}
```

Figura 3.10 configuración para generar y enviar mensajes

Generación del Tipo de Mensaje:

- type: Define si el mensaje es estándar (11 bits) o extendido (29 bits), y si es un mensaje de datos o solicitud.

Generación de un ID Aleatorio:

- Para un mensaje extendido (tipo 1), se generan hasta 29 bits.
- Para un mensaje estándar (tipo 0), se generan hasta 11 bits.

Generación de Datos Aleatorios:

- Se crean hasta 8 bytes de datos (máximo permitido por el protocolo CAN).

Envío del Mensaje:

- CAN.sendMsgBuf(id, ext, rtr, len, cdata); envía el mensaje CAN con:
  - id: Identificador del mensaje.
  - ext: Formato estándar (0) o extendido (1).
  - rtr: Tipo de trama, datos (0) o solicitud (1).
  - len: Longitud del mensaje.
  - cdata: Datos del mensaje.

Impresión del Mensaje:

- Se muestra en el monitor serial el ID, tipo y datos enviados.

Retraso Aleatorio:

- Introduce un intervalo variable entre mensajes para simular un BUS CAN dinámico.

Este generador se utilizó para validar el lector CAN construido. Al enviar mensajes con identificadores y datos aleatorios, se pudo confirmar que el lector era capaz de:

1. Recibir mensajes correctamente.
2. Identificar y procesar datos según el tipo de formato (estándar o extendido).
3. Manejar mensajes con diferentes longitudes y contenidos.

Esta estrategia de prueba garantizó la funcionalidad del lector CAN antes de su implementación en pruebas con vehículos reales. El código completo se puede encontrar en el anexo 1.

3.4 Pista de pruebas

Las pruebas se ejecutaron en la pista de pruebas del Centro Experimental Nacional de Innovación Tecnológica (CENIT) para la Seguridad Vehicular (SV) del Instituto Mexicano del Transporte (IMT). Esta pista tiene un trazo principal de forma semi-ovalada con una longitud aproximada de 2 km; cuenta en sus extremos con curvas semi-circulares de 150 m de radio, una de ellas posee el 10 % de sobreelevación, mientras que la otra tiene una pendiente transversal del 2 % para el desalojo del agua pluvial (bombeo). El circuito que conforma esta pista tiene un ancho de calzada de 14 m. La pendiente longitudinal de los tramos rectos de la pista es de alrededor del 1.2 % y poseen una pendiente transversal del 2 % para el bombeo [18]. La Figura 3.11 muestra una vista aérea de la pista de pruebas del CENIT-SV.

Figura 3.11 Pista de pruebas del CENIT-SV del IMT

Las pruebas consistieron en un recorrido por toda la longitud de la pista, donde a través del pedal del acelerador se incrementaban las RPM conforme se avanzaba, a su vez se iba tomando nota de las RPM que llevaba en cada tramo de la pista el vehículo de pruebas, esto con el fin de facilitar la identificación de la variable de RPM en los vehículos probados. En la Figura 3.12 se puede observar los cables con los conectores para vehículos utilizados en las pruebas en pista, el conector J1939 con cables verde y amarillo es el que se conecta al lector CAN y el otro es un cable que tiene un conector OBD2 en un extremo y en el otro extremo un conector J1939, el cual sirve para conectarse a vehículos ligeros con conector OBD2.

Figura 3.12 Conectores utilizados para pruebas experimentales en vehículos

4. Resultados

Para evaluar el correcto funcionamiento del dispositivo se instaló en 3 vehículos diferentes, un vehículo tipo sedan del año 2011 marca Honda modelo Accord, un vehículo eléctrico del año 2024 y un tractocamión Kenworth modelo 2023. Para todos los vehículos se realizaron recorridos en pista a velocidad variable y constante, con el fin de probar el correcto funcionamiento del lector CAN tanto para vehículos ligeros como pesados y a su vez poder decodificar algunos identificadores para los vehículos ligeros. Para el vehículo pesado se recurrió a un extracto de la norma SAE J1939 para identificar la variable de RPM del motor. A continuación, se presentan los resultados de los daros adquiridos a través del lector CAN.

4.1 Vehículos ligeros

4.1.1 Honda Accord

Al procesar los archivos obtenidos de las pruebas en pista, se obtuvo una lista de 29 identificadores para el vehículo Honda, al no contar con la base de datos CAN del vehículo fue complicado identificar los sensores dentro de cada mensaje, los más fácil de identificar fueron los de velocidad y RPM, ya que se puede corroborar la información con el panel de instrumentos del vehículo. En la Tabla 4.1 se pueden ver los identificadores encontrados en formato hexadecimal.

Tabla 4.1 Identificadores de vehiculo Honda Accord

Identificadores encontrados (Hex)		
0x39	0x1AA	0x377
0x95	0x1B0	0x378
0x136	0x1D0	0x3D7
0x13A	0x1DC	0x405
0x13F	0x294	0x40C
0x158	0x305	0x428
0x164	0x320	0x454
0x17C	0x324	0x465
0x188	0x333	0x6C1
0x1A4	0x374	

En el identificador 0x158 se encontró la variable de velocidad de avance en los bytes 1 y 2 , tomando en cuenta que el mensaje es de 8 bytes y la cuenta para el procesamiento se toma del byte 1 al byte 8. El resultado obtenido de la unión de estos 2 bytes se encontraba en una escala incorrecta, por lo cual para que coincidiera con lo observado en el cuadro de instrumentos al resultado se le aplico un factor de escala, multiplicando el resultado por 0.01 a criterio propio, el resultado se considera que es la velocidad medida en km/h. En la Figura 4.1 se puede observar la gráfica de velocidad obtenida.

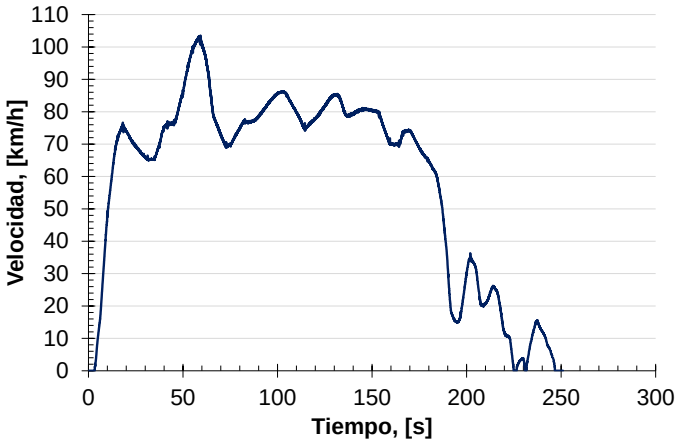

Figura 4.1 Gráfica de velocidad

Otro identificador encontrado fue el 0x17C donde se observó que los bytes 5 y 6 presentan un comportamiento similar a las RPM del motor, para este caso no se aplicó ningún factor de escala o de desfase. En la Figura 4.2 se puede observar la gráfica del comportamiento de esta señal.

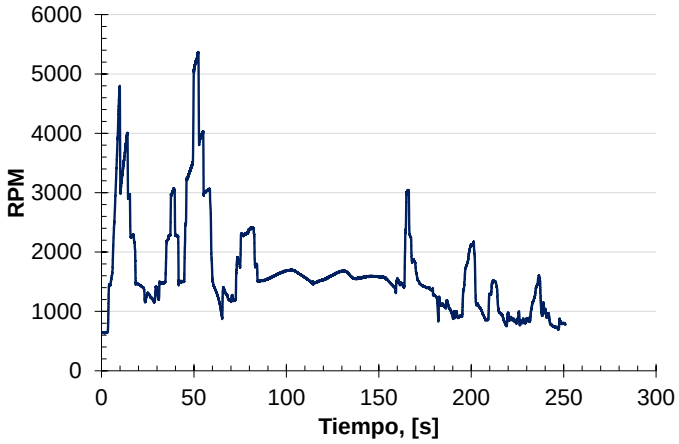

Figura 4.2 Gráfica de RPM

4.1.2 Vehículo eléctrico

Para el vehículo eléctrico modelo 2024 utilizado, se encontraron 33 identificadores. Como se puede observar en la Tabla 4.2 existen Identificadores en Formato Estándar y Extendido, para que el lector pudiera captar ambos formatos se tuvieron que realizar dos pruebas, una solo adquiriendo el Formato Estándar y otro para adquirir el Formato Extendido.

Tabla 4.2 Identificadores de vehiculo eléctrico

Identificadores encontrados (Hex)		
0x18	0x283	0x701
0x28	0x297	0xAF10101
0x80	0x303	0xAF10201
0x125	0x376	0xAF10301
0x180	0x380	0xAF10401
0x181	0x381	0xAF10501
0x182	0x383	0xAF10601
0x183	0x406	0xAF10701
0x280	0x480	0xAF10801
0x281	0x481	0xAF10901
0x282	0x482	0xAF10A08

Debido a que no se contaba con la base de datos CAN para decodificar los identificadores, se realizaron pruebas para identificar las RPM del motor eléctrico. Después de procesar la información adquirida se encontró que el identificador 0x297 contenía esta información en los bytes del 5 al 8, para este caso no se requirió ningún factor de escala y desfase. En la Figura 4.3 se puede observar un fragmento de los datos adquiridos de una prueba a velocidad controlada.

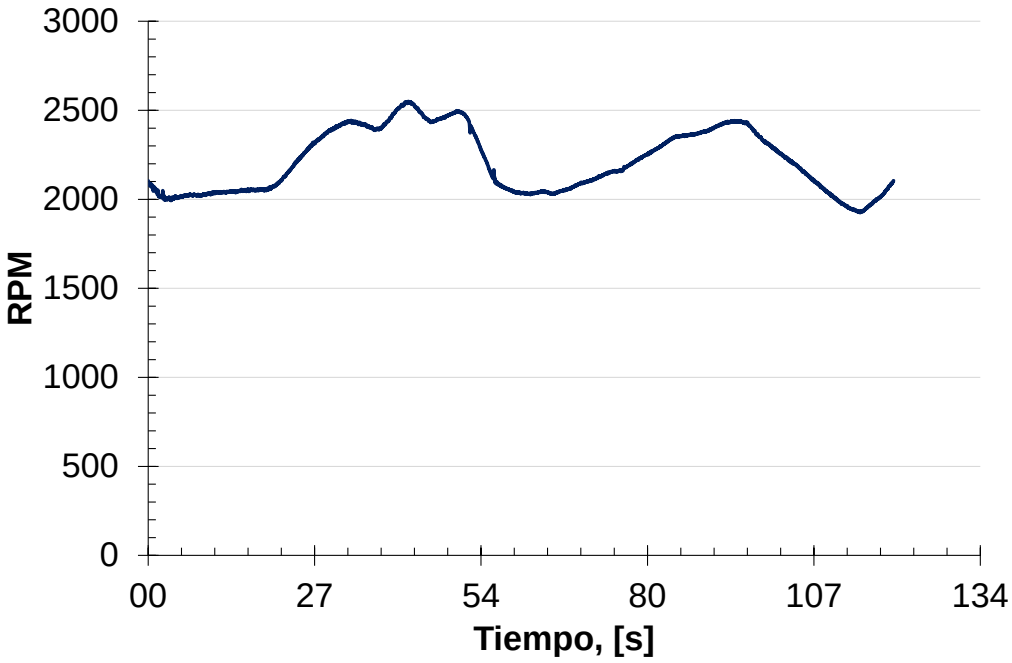

Figura 4.3 Gráfica de RPM motor eléctrico

4.2 Vehículo Kenworth

Para el caso del vehículo pesado, el cual fue un tractocamión de la marca Kenworth modelo 2023, se configuró el lector para adquirir datos en Formato Extendido a una velocidad de 500 kb/s. Después de procesar la información obtenida se encontraron 133 Identificadores, lo cual en comparación a la cantidad de identificadores encontrados en vehículos ligeros es muy superior. En la Tabla 4.3 se pueden observar todos los identificadores encontrados.

Tabla 4.3 Identificadores de vehiculo Kenworth

Identificadores del vehículo Kenworth						
6FDD427	18870027	18F00100	18FD9400	18FECA31	18FEF031	18FF9247
8FE6E0B	18D0FF31	18F0010B	18FD9B00	18FECA33	18FEF100	1CEBFF00
C00000B	18D0FF4D	18F00127	18FDA427	18FECAE8	18FEF127	1CEBFF0B
C00030B	18D83133	18F00131	18FDB200	18FEDA27	18FEF131	1CEBFF0F
C000F0B	18D93331	18F0018B	18FDC40B	18FEDB00	18FEF200	1CEBFF33
C00100B	18E0FF00	18F00500	18FDCD31	18FEDC00	18FEF231	1CEBFFE8
C00290B	18E0FF31	18F0090B	18FE4F0B	18FEDD00	18FEF500	1CECFF00
C01030B	18E8FF33	18F00927	18FE5600	18FEDF00	18FEF600	1CECFF0B
CF00200	18EA00FB	18F00E00	18FE5BE8	18FEE000	18FEF700	1CECFF0F
CF00300	18EA0B31	18F00F00	18FE700B	18FEE400	18FEF731	1CECFF33
CF00327	18EAFF0B	18F0ED31	18FE9900	18FEE500	18FEF831	1CECFFE8
CF00331	18EAFF27	18FC2B27	18FE9A00	18FEE6FB	18FEFA31	1CFEB300
CF00400	18EAFF31	18FC9600	18FEA400	18FEE700	18FEFC31	1CFEC327
CF00A00	18EAFFFB	18FCC500	18FEAE31	18FEE900	18FEFF00	10FDA300
CFD9200	18EBFF8B	18FCE700	18FEBD00	18FEED00	18FF0733	18F00027
CFD9227	18ECFF8B	18FCF200	18FEBF0B	18FEED27	18FF4327	18FD8C00
CFDCC27	18EF0027	18FD0700	18FEC100	18FEEE00	18FF4627	18FECA0B
CFE4127	18EF2700	18FD2000	18FEC131	18FEEF00	18FF5231	18FEF027
10FCFD00	18F0000F	18FD3E00	18FECA00	18FEF000	18FF5727	18FF5827

Debido a que solo se cuenta con un extracto de la norma SAE J1939 solo se pudieron decodificar algunos identificadores, por ejemplo el identificador 0xCF00400 el cual corresponde la unidad de control del motor, asociado al PGN 61444, donde en los bytes 4 y 5 se tiene la información correspondiente a las RPM del motor, la misma norma menciona que para esta variable existe un factor de escala de 0.125, por lo cual el resultado de la unión de los bytes 4 y 5 se tiene que multiplicar por este factor para obtener las RPM. En la Figura 4.4 se puede observar la gráfica correspondiente a las RPM del motor.

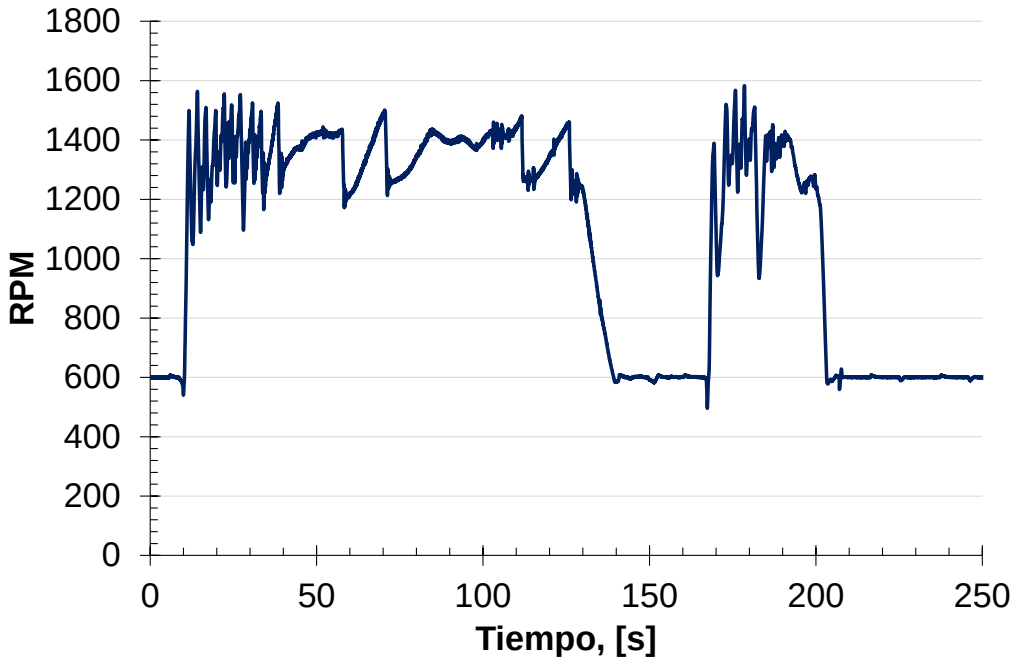

Figura 4.4 Gráfica de RPM de motor diésel

Como se observó en este y en los casos anteriores, el lector pudo adquirir información importante del protocolo de comunicación CAN. Para el caso de vehículos ligeros el obstáculo más grande al procesar la información obtenida es no contar con la base de datos CAN correspondiente y para determinar la información contenida por cada identificador es necesario realizar pruebas específicas para resaltar la variable buscada. En el caso de vehículos pesados si se cuenta con la norma SAE J1939/71_201205 es posible encontrar fácilmente las variables contenidas en cada identificador.

Conclusiones

El desarrollo del lector CAN permitió demostrar la viabilidad de construir un sistema de adquisición de datos funcional, eficiente y de bajo costo para monitorear y analizar el comportamiento del BUS CAN en vehículos. A continuación, se presentan las conclusiones principales del proyecto:

1. **Integración exitosa de hardware y software:**
    - La combinación de un Arduino UNO, un módulo MCP2515 y una computadora portátil permitió crear un dispositivo confiable para la captura de mensajes CAN.
    - El uso de la librería *Seeed_Arduino_CAN* simplificó la programación del módulo MCP2515, mientras que la interfaz desarrollada en LabVIEW© proporcionó una herramienta intuitiva para visualizar y almacenar los datos adquiridos.
2. **Validación del lector CAN:**
    - La creación de un generador de mensajes CAN mediante otro sistema Arduino UNO fue esencial para probar el correcto funcionamiento del lector antes de aplicarlo en condiciones reales.
    - En las pruebas con vehículos reales, el lector demostró ser capaz de captar y procesar los mensajes CAN generados por los sistemas electrónicos de vehículos ligeros y pesados, lo que valida su aplicación en diversos escenarios.
3. **Flexibilidad en la adquisición de datos:**
    - El lector fue capaz de operar tanto con identificadores estándar (11 bits) como extendidos (29 bits), cubriendo así una amplia gama de aplicaciones, incluyendo vehículos ligeros y vehículos que cumplan con la norma SAE J1939, por ejemplo, vehículos pesados.

4. **Almacenamiento y procesamiento de datos:**
    - La integración con LabVIEW© permitió no solo la visualización en tiempo real de los datos, sino también su almacenamiento y posterior análisis, lo cual resulta clave para tareas como la identificación de variables específicas (por ejemplo, la velocidad) y el cumplimiento de normativas técnicas.

En conclusión, el proyecto demostró que es posible implementar un lector CAN eficiente y versátil utilizando recursos de bajo costo y herramientas de software ampliamente disponibles. Este desarrollo abre la puerta a futuros proyectos de monitoreo vehicular y aplicaciones avanzadas en el sector automotriz.

Trabajos Futuros

El desarrollo del lector CAN plantea múltiples posibilidades de expansión e integración con proyectos existentes en el Instituto Mexicano del Transporte (IMT). Una de las direcciones más prometedoras es la vinculación de este dispositivo con sistemas avanzados de monitoreo remoto de vehículos. Algunas propuestas de trabajos futuros incluyen:

1. **Integración con plataformas de monitoreo remoto:**
    - Incorporar el lector CAN a sistemas de monitoreo remoto ya desarrollados en el IMT, permitiendo la transmisión en tiempo real de datos provenientes del BUS CAN hacia plataformas en la nube.
    - Utilizar tecnologías como redes 4G/5G o sistemas de comunicación satelital para garantizar la cobertura en rutas remotas o áreas de difícil acceso.
2. **Optimización de protocolos de transmisión:**
    - Adaptar el lector para que funcione con protocolos de comunicación como MQTT o HTTP, facilitando su integración con sistemas de gestión vehicular centralizados.
    - Implementar compresión de datos y priorización de mensajes para garantizar una transmisión eficiente, especialmente en vehículos con un alto volumen de sensores.

3. **Aplicación en programas de mantenimiento predictivo:**
    - Utilizar los datos adquiridos para alimentar algoritmos de análisis predictivo, lo que permitiría identificar fallas incipientes en los sistemas del vehículo, optimizando los planes de mantenimiento y reduciendo tiempos de inactividad.
4. **Expansión hacia la telemetría multimodal:**
    - Adaptar el lector para trabajar con diferentes modos de transporte, incluyendo camiones de carga, autobuses y trenes, extendiendo su alcance y utilidad en proyectos de monitoreo multimodal.
5. **Integración con sistemas de gestión de flotas:**
    - Incorporar el lector en plataformas de gestión de flotas para obtener datos críticos sobre el desempeño del vehículo, como consumo de combustible, velocidad promedio, tiempo en ralentí, entre otros.
6. **Mejora de la interfaz y usabilidad:**
    - Desarrollar aplicaciones móviles o web que permitan la visualización y control del lector de manera remota, facilitando su operación y ampliando su accesibilidad a usuarios externos.
7. **Validación en escenarios reales:**
    - Realizar pruebas en flotas comerciales en operación para evaluar el impacto de este lector en términos de ahorro operativo, seguridad y eficiencia logística.

En resumen, la integración del lector CAN con otros sistemas desarrollados en el IMT representa una oportunidad para potenciar su impacto en el monitoreo vehicular y el análisis de datos en tiempo real. Este enfoque permitirá contribuir significativamente a la innovación en la gestión del transporte y el desarrollo de tecnologías que favorezcan la sostenibilidad, seguridad y eficiencia en el sector automotriz.

Bibliografía

[1]. CAN in Automation. (2021). Introduction to CANopen. [Consulta en línea] https://www.can-cia.org

[2]. CAN in Automation. (2024). Introduction to CAN lower and higher-layer protocols. [Consulta en línea] https://www.can-cia.org

[3]. Srinivasan, C. (2023). Advanced Driver Assistance System (ADAS) in Autonomous Vehicles: A Complete Analysis. [Consulta en línea] https://ieeexplore.ieee.org/document/10192617

[4]. Szydlowski, C. (1992). CAN Specification 2.0: Protocol and Implementations [Consulta en línea]. https://www.sae.org/publications/technical-papers/content/921603/

[5]. Dewesoft. (2024). ¿Qué es Can Bus (red de área de controlador)? [Consulta en línea] https://dewesoft.com/es/blog/que-es-el-bus-can

[6]. Blackman, J. (2024). Overview of 3.3V CAN (Controller Area Network). [Archivo PDF] https://www.ti.com/lit/an/slla337/slla337.pdf

[7]. National Instruments. (2024). Controller Area Network Protocol Overview. [Consulta en línea] https://www.ni.com/en/shop/seamlessly-connect-to-third-party-devices-and-supervisory-system/controller-area-network--can--overview.html

[8]. Molano, M. (2015). Revisión documental del protocolo CAN como herramienta de comunicación y aplicación en vehículos. [Publicación técnica No. 474]. México: Instituto Mexicano del Transporte. [Archivo PDF] https://imt.mx/archivos/Publicaciones/PublicacionTecnica/pt474.pdf

[9]. Society of Automotive Engineers. (2018). Serial Control and Communications Heavy Duty Vehicle Network [Consulta en línea] https://www.sae.org/standards/content/j1939_201808/

[10]. Society of Automotive Engineers. (2015). Vehicle Application Layer. [Consulta en línea] https://www.sae.org/standards/content/j1939/71_202208/

[11]. Suarez, R. (2024). ARDUINO. [Consulta en línea] https://www3.gobiernodecanarias.org/medusa/ecoblog/rsuagued/arduino/

[12]. Jeffson, M. (2024). Seeed-Studio/Seeed_Arduino_CAN: Seeed Arduino CAN-BUS library. [Consulta en línea] https://github.com/Seeed-Studio/Seeed_Arduino_CAN

[13]. Unit Electronics. (2024). Modulo MCP2515. [Consulta en línea] https://uelectronics.com/producto/mcp2515-can-bus-spi-tja1050/

[14]. Weis, O. (2024). Conector OBD2 Explicado. [Consulta en línea] https://www.flexihub.com/es/oobd2-pinout/#:~:text=Tipos%20de%20Conexiones%20de%20Puerto%20OBD2,-Existen%20dos%20tipos&text=Tienen%20una%20tensi%C3%B3n%20de%20salida,velocidad%20m%C3%A1xima%20de%20250.000%20baudios.

[15]. Au grupo. (2023). Manual de usuario del cable J1939. [Consulta en línea] https://manuals.plus/es/au-group-electronics/j1939-cable-with-db9-female-connector-manual#references

[16]. National Instruments. (2024). What is NI LabVIEW? Graphical Programming for Test & Measurement. [Consulta en línea] https://www.ni.com/en/shop/labview.html

[17]. M, Cruz. (2020). Programa para detección y adquisición de variables de operación de vehículos bajo el protocolo CAN. [Publicación técnica No. 661]. México: Instituto Mexicano del Transporte. [Archivo PDF] https://www.imt.mx/archivos/Publicaciones/PublicacionTecnica/pt661.pdf

[18]. Instituto Mexicano del Transporte. (2021). CeNIT para la seguridad vehicular. [Archivo PDF] https://imt.mx/images/files/GRAL/documentos/folletoCENITseguridadvehicular.pdf

Anexo 1. Códigos utilizados

Código del lector

```
#include <SPI.h>
#include "mcp2515_can.h"

const int SPI_CS_PIN = 10;  // Pin CS del MCP2515
const int CAN_INT_PIN = 2;  // Pin de interrupción del MCP2515

mcp2515_can CAN(SPI_CS_PIN);  // Inicializa el MCP2515 en el pin CS

void setup() {
    Serial.begin(115200);
    while (!Serial) {}

    // Inicializa el MCP2515 a 500 kbps
    while (CAN_OK != CAN.begin(CAN_500KBPS)) {
        Serial.println("Error al iniciar MCP2515, intentando de nuevo...");
        delay(100);
    }
    Serial.println("MCP2515 inicializado correctamente");
}
```

```
void loop() {
  // Comprueba si hay mensajes CAN recibidos
  if (CAN_MSGAVAIL == CAN.checkReceive()) {
    // ID del mensaje CAN
    unsigned long canId;

    // Buffer para almacenar los datos recibidos
    byte len = 0;
    byte buf[8];

    // Lee el mensaje CAN
    CAN.readMsgBuf(&len, buf);
    canId = CAN.getCanId();

    // Imprime los detalles del mensaje recibido
    Serial.print("Mensaje recibido con ID: ");
    Serial.print(canId, HEX);
    Serial.print(" Datos: ");

    for (int i = 0; i < len; i++) {
      Serial.print(buf[i], HEX);
      Serial.print(" ");
    }
    Serial.println();
```

```
    }
}
```

Código del generador de mensajes CAN

```
#include <SPI.h>
#include "mcp2515_can.h"

const int SPI_CS_PIN = 10;  // Pin CS del MCP2515
const int CAN_INT_PIN = 2;  // Pin de interrupción del MCP2515

mcp2515_can CAN(SPI_CS_PIN);  // Inicializa el MCP2515 en el pin CS
#define MAX_DATA_SIZE 8

void setup() {
    Serial.begin(115200);
    while (!Serial) {}

    // Inicializa el MCP2515 a 500 kbps
    while (CAN_OK != CAN.begin(CAN_500KBPS)) {
        Serial.println("Error al iniciar MCP2515, intentando de nuevo...");
        delay(100);
    }
    Serial.println("MCP2515 inicializado correctamente");
```

```
    randomSeed(millis());  // Inicializa semilla aleatoria para generar datos aleatorios
}

uint32_t id;
uint8_t  type;  // bit0: ext, bit1: rtr
unsigned len;
byte cdata[MAX_DATA_SIZE] = {0};

void loop() {
    type = random(4);

    // Genera un ID aleatorio para el mensaje CAN (estándar o extendido según tipo)
    if (type & 0x1) {
        id = random(0x1U << 14);
        id |= (uint32_t)random(0x1U << 15) << 14;
    } else {
        id = random(0x1U << 11);
    }

    // Genera datos aleatorios para el mensaje CAN
    len = random(0, MAX_DATA_SIZE + 1);
    for (int i = 0; i < len; i++) {
        cdata[i] = random(0x100);
```

```
    }


    // Envía el mensaje CAN con el ID generado
    CAN.sendMsgBuf(id, bool(type & 0x1), bool(type & 0x2), len, cdata);


    // Imprime el mensaje CAN enviado
    char prbuf[32 + MAX_DATA_SIZE * 3];
    int n = sprintf(prbuf, "TX: [%08lX](%02X) ", (unsigned long)id, (type & 0x1)
? 0x02 : 0x00);


    for (int i = 0; i < len; i++) {
        n += sprintf(prbuf + n, "%02X ", cdata[i]);
    }
    Serial.println(prbuf);


    // Espera antes de enviar el siguiente mensaje
    delay(random(30));
}
```

COMUNICACIONES

SECRETARÍA DE INFRAESTRUCTURA, COMUNICACIONES Y TRANSPORTES

Km 12+000 Carretera Estatal 431 “El Colorado-Galindo”
San Fandila, Pedro Escobedo
C.P. 76703
Querétaro, México
Tel: +52 442 216 97 77 ext. 2610

publicaciones@imt.mx

http://www.imt.mx/