UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

CARRERA DE INGENIERÍA EN MANTENIMIENTO AUTOMOTRIZ

TRABAJO DE GRADO PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN MANTENIMIENTO AUTOMOTRIZ

TEMA: Desarrollo de una interfaz electrónica para la detección de códigos de falla en vehículos a través de la red CAN.

AUTOR: AGUIRRE PÁEZ DIEGO ALEXANDER

DIRECTOR: ING. RAMIRO ANDRÉS ROSERO AÑAZCO MSc.

IBARRA, SEPTIEMBRE 2020

ACEPTACIÓN DEL DIRECTOR

En calidad de director del trabajo de grado, previo a la obtención del título de Ingeniero en Mantenimiento Automotriz, nombrado por el Honorable Consejo Directivo de la Facultad de Ingeniería en Ciencias Aplicadas.

CERTIFICO:

Que una vez analizado el plan de trabajo cuyo título es: **"Desarrollo de una interfaz electrónica para la detección de códigos de falla en vehículos a través de la red CAN"** presentado por el señor: Aguirre Páez Diego Alexander, con cédula identidad N° 172611940-5, doy fe que dicho trabajo reúne los requisitos y méritos suficientes para ser sometido a presentación pública y evaluación por parte de los señores integrantes del jurado examinador que se designe.

En la ciudad de Ibarra, a los 28 días del mes de septiembre del 2020.

Atentamente:

Ramiro Rosero A.

Ing. Ramiro Andrés Rosero Añazco MSc.
DIRECTOR DEL TRABAJO DE GRADO.

UNIVERSIDAD TÉCNICA DEL NORTE

BIBLIOTECA UNIVERSITARIA

AUTORIZACIÓN DE USO Y PUBLICACIÓN A FAVOR DE LA UNIVERSIDAD TÉCNICA DEL NORTE

1. IDENTIFICACIÓN DE LA OBRA

En cumplimiento del Art. 144 de la Ley de Educación Superior, hago la entrega del presente trabajo a la Universidad Técnica del Norte para que sea publicado en el Repositorio Digital Institucional, para lo cual pongo a disposición la siguiente información:

DATOS DE CONTACTO			
CÉDULA DE IDENTIDAD:	1726119405		
APELLIDOS Y NOMBRES:	AGUIRRE PÁEZ DIEGO ALEXANDER		
DIRECCIÓN:	CAYAMBE, calle Terán N3-98 y 24 de Mayo		
EMAIL:	daaguirrep@utn.edu.ec		
TELÉFONO FIJO:	(02)2362075	**TELÉFONO MÓVIL:**	0980262139

DATOS DE LA OBRA	
TÍTULO:	Desarrollo de una interfaz electrónica para la detección de códigos de falla en vehículos a través de la red CAN
AUTOR (ES):	AGUIRRE PÁEZ DIEGO ALEXANDER
FECHA: DD/MM/AAAA	28/09/2020
SOLO PARA TRABAJOS DE GRADO	
PROGRAMA:	■ **PREGRADO** ☐ **POSGRADO**
TITULO POR EL QUE OPTA:	INGENIERO EN MANTENIMIENTO AUTOMOTRIZ
ASESOR /DIRECTOR:	Ing. Ramiro Andrés Rosero Añazco MSc.

2. CONSTANCIAS

El autor manifiesta que la obra objeto de la presente autorización es original y se la desarrolló, sin violar derechos de autor de terceros, por lo tanto la obra es original y que es el titular de los derechos patrimoniales, por lo que asume la responsabilidad sobre el contenido de la misma y saldrá en defensa de la Universidad en caso de reclamación por parte de terceros.

Ibarra, a los 28 días del mes de septiembre del 2020.

EL AUTOR:

AGUIRRE PÁEZ DIEGO ALEXANDER

DEDICATORIA

El presente trabajo de grado es dedicado de una manera muy especial a mis padres quienes con mucho esfuerzo y dedicación me han apoyado durante toda mi vida académica. A mi novia y a mi hijo quienes son la principal razón para buscar nuevas metas gracias a su apoyo incondicional. A mis hermanos y a toda la demás familia que ha estado siempre presente en todos los momentos de mi vida.

Diego Aguirre

AGRADECIMIENTO

Agradezco a mis padres por encaminarme siempre por el buen camino y forjarme como la persona que soy. A mi novia y a mi hijo por ser esa fuente de ánimo e inspiración en el día a día. Además quiero expresar mi más sincero agradecimiento al Ing. Andrés Cevallos MSc. quien inicialmente fue director de tesis, ya que desde el primer momento tuvo la predisposición y el conocimiento para guiarme durante todo este trabajo de grado. A mi actual director Ing. Ramiro Rosero MSc. y a mis opositores Ing Ignacio Benavides MSc. e Ing. Paúl Hernández MSc. quienes con su conocimiento y experiencia han ayudado a la culminación de este trabajo. A la Universidad Técnica del Norte y los docentes de la carrera de Ingeniería en Mantenimiento Automotriz y todas las personas de las cuales he adquirido conocimiento para formarme como profesional.

Diego Aguirre

ÍNDICE DE CONTENIDOS

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

PÁGINA

ÍNDICE DE ANEXOS

PÁGINA

RESUMEN

El presente trabajo de grado plantea la elaboración de una interfaz electrónica para la detección de códigos de falla a través de la red CAN. En base a que esta es una herramienta indispensable en todos los talleres automotrices para poder realizar un diagnóstico adecuado de fallas en los vehículos. Además se pretende incentivar a la producción de este tipo de herramientas a nivel nacional con prestaciones similares a las que se encuentran en el mercado. Para esto se ha realizado la infiltración a la red CAN del vehículo usando dos diferentes dispositivos como son el Microchip CAN Bus Analyzer y el ELM327, ambos funcionan como interfaces para el envío y la adquisición de datos. Con estos dispositivos se ha logrado visualizar el tipo de escritura propia de la red CAN para poder replicarla posteriormente. Para el envío y adquisición de datos se utilizó diferentes modos de diagnóstico como son: modo 1 (lectura de datos en tiempo real), modo 3 (lectura de códigos de falla) y modo 4 (borrado de códigos de falla). Además se identificó los headers de escritura y lectura por los cuales transitan los datos que se estarán enviando y recibiendo. Obtenida dicha información se realizó el análisis de los datos obtenidos para interpretar cada uno de estos y hacer la selección de cuales datos son útiles para la presentación de los resultados y cuales no lo son. En este punto se dio paso a la programación de la interfaz en Arduino IDE, donde se realizó un código capaz de escribir directamente sobre la red CAN del vehículo y de la misma forma receptar los datos proporcionados por la ECU. Este código fue cargado a una placa Arduino que a su vez tiene montada en ella una placa CAN Bus Shield que es la que permite la comunicación entre el programa y la red CAN. Para una interacción más amigable entre la interfaz y el operador, se realizó la programación en LabVIEW con la capacidad de exportar e importar datos al monitor serial del Arduino para presentarlos de manera más organizada. Las funciones con las que cuenta la interfaz son: lectura de datos en tiempo real de: voltaje del sensor de oxígeno y ajuste de combustible a corto plazo, presión absoluta en el colector de admisión y RPM del motor. Además de lectura de códigos de falla y borrado de los mismos.

ABSTRACT

This degree work proposes the elaboration of an electronic interface for the detection of diagnostic trouble codes (or fault codes) through the CAN network. This is an indispensable tool in all automotive workshops to carry out an adequate diagnosis of vehicle failures. It is expected to encourage the production of this type of tool at a national level with important economic benefits. For this, infiltration into the vehicle's CAN network was performed using two devices such as the Microchip CAN Bus Analyzer and the ELM327, both of which function as interfaces for transmitting and acquiring data. With these devices, it has been possible to understand the type of writing typical of the CAN network and to replicate it. For data sending and acquisition, different diagnostic modes such as mode 1 (reading data in real-time), mode 3 (reading fault codes), and mode 4 (clearing fault codes) were used. Besides, the write and read headers through which the data to be sent and received are identified. Once this information was obtained, data analysis was performed to interpret it and make differentiation of useful data from that which are not. At this point, the interface programming was started in Arduino IDE, where a code capable of writing directly on the red CAN of the vehicle and in the same way of receiving the data provided by the ECU was made. This code was loaded to an Arduino board, which in turn has a CAN Bus Shield board mounted on it, which allows communication between the program and the CAN network. For a more favorable interaction between the interface and the operator, programming was done in LabVIEW with the capability to export and import data to the serial monitor of the Arduino to present it in a more standardized way. The functions with the account interface are real-time data reading of oxygen sensor voltage and short-term fuel trim, absolute pressure at the intake manifold, and engine RPM. In addition to reading fault codes and deleting them.

INTRODUCCIÓN

El protocolo de comunicación CAN fue desarrollado en 1980 por la compañía alemana Bosch con el propósito de realizar la unión de varios dispositivos con conexiones más sencillas. La principal característica de esta red es que evita las conexiones punto a punto en cada uno de los componentes del sistema disminuyendo notablemente el tamaño del cableado usado.

Por otro lado, el OBD fue implementado en los vehículos con el fin de limitar las emisiones provocadas sin que esto conlleve a la pérdida de potencia en el automotor. Este sistema presentaba fallas considerables a la vez que no estaba estandarizado por todas las marcas sino que todas disponían de este de manera distinta. A partir de las regulaciones ambientales descritas en ese entonces todos los fabricantes debían acogerse a un sistema OBD estandarizado. Luego de varias mejoras en el sistema y de lograr estandarizar tanto códigos de falla como herramientas de diagnóstico aparece el OBD-II.

En base a esto el presente proyecto tiene la finalidad de elaborar un dispositivo de diagnóstico capaz de realizar lectura de datos en tiempo real, lectura de códigos de falla y el borrado de ellos. Al inferir en la red CAN del vehículo se accede al modo de diagnóstico con la ayuda de analizadores de redes CAN. Con esto se tiene acceso a los mensajes hexadecimales que transitan en la red para determinar cuáles de ellos tienen que ver con el sistema de diagnóstico para posteriormente desarrollar una interfaz electrónica compuesta únicamente por una placa Arduino Mega 2560 y una placa CAN Bus Shield. Este dispositivo vendrá cargado con un código elaborado en el software Arduino IDE y será controlado desde una interfaz gráfica desarrollada en LabVIEW.

Este proyecto dejará una base para que futuras investigaciones continúen con el desarrollo de una herramienta de diagnóstico automotriz más compleja y con prestaciones similares a las que se tiene acceso en el mercado.

CAPÍTULO I

1. REVISIÓN BIBLIOGRÁFICA

1.1 ANTECEDENTES

La red de comunicación CAN (*Controller Area Network*) es un protocolo de comunicación desarrollado en 1980 que permite vincular varios dispositivos con conexiones más sencillas y a mayor velocidad. Este desarrollo fue posible gracias a un grupo de ingenieros de la compañía alemana Roberth Bosch, quienes estudiaron la posibilidad de aplicar sistemas de bus seriales dentro de los automóviles con la finalidad de satisfacer las demandas de la Sociedad de Ingenieros Automotrices (SAE, del inglés, *Society of Automotive Engineers*) y como resultado se obtuvo la especificación del protocolo CAN (Gutiérrez Gómez, 2017).

Las primeras investigaciones se realizaron en el año 1983, donde Bosch buscaba desarrollar una conexión entre los diferentes dispositivos del automóvil de una manera más sencilla y eficiente. Durante la especificación del sistema de bus serial se integró la empresa fabricante de automóviles Mercedez-Benz y el fabricante de semiconductores Intel Corp (Gutiérrez Gómez, 2017). Posteriormente en febrero de 1906 se dió a conocer el CAN bus en el congreso de la SAE en Estados Unidos.

A pesar de que la red de comunicación CAN bus se desarrolló para usos automotrices, su uso se extendió a otras industrias, como por ejemplo: la industria ferroviaria, que usan las redes CAN para controlar las puertas, los frenos, entre otros. Otro ejemplo claro es en la aviación donde se usa las redes CAN principalmente para los sensores de vuelo y sistemas de navegación. Por otro lado, las redes CAN también han sido utilizadas por los fabricantes de insumos médicos para el control de varios aparatos que deben estar conectados entre sí.

No obstante, en el año 1992 la red CAN fue implementada en el Mercedes-Benz clase E del mismo año. En sus inicios, este protocolo se limitaba a interconectar las unidades de control electrónico del tren de potencia (motor, caja de cambios y transmisión), sin embargo, pronto conectarían además las unidades de control necesarias para los componentes de carrocería, sistemas auxiliares, etc.

En el año 2008, en Estados Unidos, la Agencia de Protección del Medio Ambiente estableció como norma que todos los fabricantes de vehículos utilizaran la red CAN para el diagnóstico de las fallas en todo tipo de vehículo.

La implementación de este modelo de comunicaciones supuso un avance en cuanto a la cantidad de conexiones entre dispositivos que eran necesarias para mantener todos los elementos comunicados, ya que el protocolo CAN emplea un único bus de comunicaciones, compartido por todos los dispositivos y evita la necesidad de establecer una conexión punto a punto con cada uno de ellos (Martínez Requena, 2017).

Una de las principales ventajas que posee este tipo de protocolo de comunicación es que presta facilidades para conectar más módulos de control. Otra ventaja presente es la notable disminución del tamaño del cableado y por consecuencia, ofrece mayor facilidad para poder detectar fallas en el funcionamiento del sistema. Otro beneficio a considerar es que todas las unidades de control electrónico poseen una sola interfaz CAN reduciendo el número de entradas de cada unidad de control a una sola. Esto ayuda a reducir costos de materiales, reduce peso y simplifica el sistema. Se estima que puede ofrecer una reducción de 5 a 1 en los costes de cableado (Martínez Requena, 2017). Además, esta red de datos nos aporta flexibilidad en la configuración, tanto en el número de nodos, como en la disposición de los mismos, pudiendo añadirse o quitarse nodos de forma dinámica sin afectar al protocolo, pudiendo conectarse hasta 110 nodos a una red CAN (Martínez Requena, 2017).

Derivado de la importancia del protocolo CAN y su consolidación en la industria automotriz, se investiga la implementación de un módulo observador de la red de comunicación CAN que permite observar que dos entidades de un sistema de comunicación se comuniquen entre sí (tester y unidad a probar), así como validar el intercambio de mensajes generados entre el tester y la unidad de prueba (Gutiérrez Gómez, 2017).

Hoy en día el protocolo de comunicación CAN es acreditado como una red de comunicación confiable y compacta utilizada para controlar sistemas que requieran ser monitoreados en tiempo real.

Por otro lado, el sistema OBD fue implementado en los vehículos a partir de la necesidad social de limitar las emisiones de los mismos. Con esto se logró hacer que las emisiones de los vehículos se redujeran sin restar potencia, puesto a que otras opciones para la reducción

de emisiones de los motores de combustión daba como resultado una pérdida de potencia considerable.

Los sistemas OBD tuvieron muchas variaciones dependiendo de las marcas de los fabricantes de vehículos los cuales presentaban fallas y no eran normalizados para un uso en general de los demás fabricantes. Es por esto que existían falencias en el desarrollo de estos sistemas, como era el caso de que los diagnósticos no se los realizaba cuando el vehículo estaba en movimiento, sino que era necesario cuamplir con una serie de situaciones en lo que respecta al ciclo de conducción para poder realizar el diagnóstico. Esto traía problemas puesto que el operador del vehículo debía esperar al final de un ciclo de operación para poder diagnosticar algún tipo de falla en su vehículo. Otra de las principales fallas que surgió en el desarrollo de este sistema es la luz de advertencia que presentaba, puesto que era una luz testigo la cual podía significar desde una avería leve hasta una avería extremadamente grave. Pero esto llego a su fin cuando el congreso de los Estados Unidos en 1990 aprobó la ley del aire limpio. A partir de esto, todos los fabricantes debían regirse al OBD, estandarizando los códigos de falla por medio de números y letras que hicieran fácil su lectura y comprensión. Además al regular todos estos aspectos también se comenzó a regular y estandarizar las herramientas de diagnóstico por lo que surgió la necesidad de estandarizar también los conectores de acceso al OBD.

Al mismo tiempo que los cambios ocurrían para el sistema de diagnóstico, las computadoras de automóvil se hacían más poderosas, no solo verificaban los sensores, si no que controlaban todo el sistema de combustible, de aire y de sensores, dando paso a tecnologías rezagadas como Fuel Injection (Cervantes & Espinoza , 2010).

Con todas estas mejoras, el sistema denominado OBD-II, entró en marcha en 1996, aunque algunos vehículos modelo 94 y 95 ya contaban con él, pero con fines de experimentación (Cervantes & Espinoza , 2010).

Estos sistemas con el paso del tiempo han estado expuestos a una serie de actualizaciones y renovaciones que lo hacen estar al día con las nuevas innovaciones en la industria automotriz.

Todos los vehículos fabricados ya cuentan con un sistema de diagnóstico, el cual almacena los datos y proporciona toda la información cuando ha ocurrido alguna falla (Cervantes & Espinoza , 2010). Es por esto que se desarrolló herramientas de diagnóstico que nos permitan

acceder a los datos que la unidad de control del vehículo nos está dando para poder determinar por medio de estos datos las fallas en los diferentes sistemas del vehículo. Además con este tipo de herramientas se puede acceder a datos de funcionamiento y también se puede realizar algún tipo de reparaciones del sistema.

1.2 SITUACIÓN ACTUAL

Actualmente los vehículos contienen sistemas de diagnóstico y transferencia de datos para procurar el buen funcionamiento y la máxima eficiencia posible. Además dichos sistemas también van orientados al control de gases de escape que produce el motor de combustión presente en dichos vehículos.

Para poder acceder a todos los datos del vehículo es necesario la aplicación de un escáner automotriz utilizando el sistema OBD-II el cual incluye a la mayoría de las marcas de automóviles a partir de los modelos 1996 hasta modelos actuales que utilizan el protocolo CAN) implementado en todos los modelos fabricados desde el 2008 (Cervantes & Espinoza , 2010).

Es por esto que en la actualidad es indispensable que un taller automotriz cuente con un escáner automotriz con las capacidades de detectar códigos de falla y poder borrarlos una vez solucionado el problema. Una de las principales limitantes en la adquisición de este tipo de instrumentos es el costo elevado que algunos de estos pueden llegar a tener.

Algunos de los avances tecnológicos en cuanto al escáner automotriz es que son muy compactos y muchas veces no están conectados directamente por medio de cables al conector del OBD-II.

Por otro lado, la mayoría de escáneres tienen la capacidad de escanear a varias marcas de vehículos, aunque también podemos encontrar escáneres que son específicamente para una sola marca. Otro avance significativo en cuanto a escáneres es que en la actualidad tienen la capacidad de escanear dos vehículos al mismo tiempo, estos son ocupados en talleres en los cuales tienen mucha demanda de este tipo de trabajos.

1.3 PROSPECTIVA

Lo que se espera con este trabajo es elaborar un escáner automotriz con las capacidades suficientes para poder ser utilizado en talleres nuevos en el mercado. Con esto se estaría ayudando a la industria, ya que, se dispondría de un escáner con funciones básicas para un diagnóstico de códigos de falla y su respectivo tratamiento. Por lo tanto al desarrollar un escáner básico, este tendría un costo accesible para todas las personas que requieran de esta herramienta que actualmente es indispensable para el diagnóstico de fallas en el campo automotriz.

1.4 PLANTEAMIENTO DEL PROBLEMA

En la realidad automotriz de nuestro país es necesario conocer acerca del protocolo de comunicación CAN bus, puesto que actualmente la mayoría de vehículos presentes en nuestra sociedad dispone de este tipo de protocolo de comunicación. Por lo tanto, es esencial que los técnicos del área automotriz sean capaces de trabajar con esta red de comunicación para la detección de fallas dentro de los diferentes sistemas del vehículo. Es por esto que es de vital importancia para un taller automotriz, disponer de un escáner con el que se pueda realizar el diagnóstico de fallas en vehículos con OBDII. Aunque en el mercado en el que nos encontramos podemos mencionar que el costo de este tipo de herramientas de diagnóstico son relativamente alto. Además se puede decir que el costo de estos equipos va a depender de las funciones y avances tecnológicos que presente el mismo.

1.5 DELIMITACIÓN

1.5.1 Temporal

El presente trabajo investigativo empezó su desarrollo durante los primeros días del mes de diciembre del año 2018, teniendo una duración de un año aproximadamente y llegando a su culminación en el mes de noviembre del año 2019, cumpliéndose con todos los objetivos y expectativas planteadas al inicio del mismo.

1.5.2 Espacial

En su totalidad el presente proyecto investigativo fue llevado a cabo en los laboratorios automotrices de la carrera de Ingeniería en Mantenimiento Automotriz de la Universidad Técnica del Norte campus “El Olivo” el mismo que se desarrolló bajo la estricta supervisión y asesoramiento de personal docente altamente calificado en el campo de la industria automotriz y dentro del marco de las políticas y restricciones tanto de la carrera de Ingeniería en Mantenimiento Automotriz, como de la Universidad Técnica del Norte.

1.6 OBJETIVOS

1.6.1 Objetivo General

Desarrollar una interfaz electrónica para la detección de códigos de fallas en vehículos a través de la red CAN.

1.6.2 Objetivos Específicos

- Identificar en la red CAN del automóvil los mensajes hexadecimales que solicitan códigos de falla e información de diagnóstico.
- Sortear la seguridad del vehículo, para acceder a su modo de diagnóstico y controlar sus funciones básicas.
- Programar una interfaz en Arduino que permita realizar un diagnóstico básico del vehículo.

1.7 ALCANCE

Mediante el proyecto de investigación se planea realizar un diagnóstico básico a través de la red CAN en vehículos con OBD-II genérico. Para las pruebas de diagnóstico y toma de datos se utilizará un vehículo Chevrolet Sail modelo 2018. El diagnóstico básico compete la lectura de códigos de falla, el borrado de dichos códigos y la obtención de datos en tiempo real útiles para la detección de fallas electrónicas. En consecuencia este proyecto tiene como alcance los siguientes aspectos puntuales:

- Lectura de códigos de falla

- Borrado de códigos
- Lectura de datos en tiempo real de:
  - Revoluciones del motor (RPM).
  - Voltaje del sensor de oxígeno.
  - Ajuste de combustible a corto plazo.
  - Presión absoluta en el colector de admisión.

Estos puntos serán cumplidos luego de la elaboración de un dispositivo electrónico capaz de cumplir dichas funciones por medio de una interfaz electrónica que permita establecer una conexión con la computadora del vehículo.

1.8 JUSTIFICACIÓN

La información acerca de la red de comunicación CAN en nuestra realidad es algo limitada, por lo que se pretende ampliar al conocimiento de este tipo de redes de comunicación vehiculares. Es por esto que la presente investigación intenta brindar más información acerca del funcionamiento y cómo se puede indagar en los datos que están siendo transportados por la red CAN; a través de un dispositivo capaz de interferir y solicitar a la computadora del vehículo datos en tiempo real del comportamiento de los diferentes sistemas interconectados.

Por otro lado partiendo de escáneres básicos se buscará la manera de imitar su funcionamiento con el fin de que, mediante la intrusión en la red CAN se pueda determinar los códigos referentes al diagnóstico OBD-II haciendo que la interpretación de los mensajes hexadecimales que nos proporciona la computadora del vehículo sean más amigables con las personas que operen estos dispositivos y por lo tanto, sean de fácil interpretación.

Con lo mencionado anteriormente se elaborará escáneres sencillos y básicos que sean de útiles para realizar diagnósticos electrónicos. Además estos escáneres serán de un costo relativamente bajo en comparación con los escáneres que podemos encontrar en nuestro mercado. Con lo que se propone un prototipo en busca del mejoramiento de la matriz productiva del país.

Este trabajo además va encaminado con el objetivo número cinco del Plan de Desarrollo, donde se expresa:

Impulsar la productividad y competitividad para el crecimiento económico sostenible de manera redistributiva y solidaria (Senplades, Plan Nacional de Desarrollo 2017-2021-Toda una Vida, 2017).

Dentro de este objetivo el presente trabajo va direccionado con las siguientes políticas:

La política 5.2 indica que se debe promover la productividad, competitividad y calidad de los productos nacionales, como también la disponibilidad de servicios conexos y otros insumos, para generar valor agregado y procesos de industrialización en los sectores productivos con enfoque a satisfacer la demanda nacional y de exportación (Senplades, Plan Nacional de Desarrollo 2017-2021-Toda una Vida, 2017). Sustentándose en esta política, este trabajo busca servir de base para la producción de escáneres hechos nacionalmente, lo que pretende incentivar la productividad nacional, elaborando productos capaces de competir con sus similares de importación.

Po otro lado la política 5.4 pretende incrementar la productividad y generación de valor agregado creando incentivos diferenciados al sector productivo, para satisfacer la demanda interna, y diversificar la oferta exportable de manera estratégica (Senplades, Plan Nacional de Desarrollo 2017-2021-Toda una Vida, 2017). Es por esto que el presente proyecto tiene como base incentivar a la creación de escáneres automotrices, lo que significa que se pretende producir dispositivos tecnológicos los cuales podrían satisfacer las necesidades de una pequeña parte de la población que tengan talleres automotrices con poca tecnología.

Por último la política 5.6 busca promover la investigación, la formación, la capacitación, el desarrollo y la transferencia tecnológica, la innovación y el emprendimiento, la protección de la propiedad intelectual, para impulsar el cambio de la matriz productiva mediante la vinculación entre el sector público, productivo y las universidades (Senplades, Plan Nacional de Desarrollo 2017-2021-Toda una Vida, 2017). Con este lineamiento, al realizar esta investigación se proporcionará información específica acerca de los temas a tratarse, puesto que la mayoría de talleres de nuestro país no tienen en claro el funcionamiento del protocolo CAN bus y el diagnostico a través del mismo. Además se busca motivar a la producción de tecnología nacional al producir dispositivos tecnológicos capaces de inferir en la red de intercomunicación automotriz.

1.9MARCO TEÓRICO

1.9.1 Protocolo CAN bus.

El protocolo de comunicación CAN (*Controlled Area Network*), se desarrolló en los años 80 por Bosch partiendo de la necesidad de crear un protocolo de comunicación que sea más compacto y de mejor funcionamiento. Puesto que, previo al desarrollo del protocolo de comunicación CAN todos los sistemas del vehículo tenían conexiones en las que se necesitaba gran cantidad de cables lo que implicaba más costos y más peso en el vehículo. El CAN bus tuvo gran aceptación por los fabricantes de vehículos por lo que se buscó estandarizar este tipo de red de comunicación, tanto fue su aceptación que este fue utilizado en otras industrias diferentes a la automotriz.

1.9.1.1 Descripción del protocolo CAN.

Como se mencionó anteriormente, el protocolo de comunicación CAN fue ideado principalmente para reducir el tamaño y la complejidad del cableado y por ende reducir costos de fabricación de los vehículos. Por otro lado, una de las características más importantes del CAN es que está estandarizado. Se trata de un estándar definido en las normas ISO (*Internacional Organization for Standardization*), concretamente la ISO11898, que se divide a su vez en varias partes, cada una de las cuales aborda diferentes aspectos de CAN (Martínez Mas, 2012).

La estructura física del CAN también cumple con un estándar en el que se dispone de dos cables trenzados o en ocasiones se puede disponer de un solo cable. La información que circula entre las unidades a través de los dos cables (bus) son paquetes de bits (0‘s y 1‘s) con una longitud limitada y con una estructura definida de campos que conforman el mensaje (Martínez Mas, 2012). En cuanto a los nodos que se pueden llegar a tener en este tipo de protocolo de comunicación se puede determinar que se puede realizar una conexión de hasta 110 dispositivos en la misma red. Además se dispone de un sistema de desconexión automática de nodos que puedan estar fallando o dejen de funcionar correctamente, para que el resto del sistema no deje de trabajar y no se detenga el tráfico de datos que van por la red CAN. Contrario a esto también existe la posibilidad de añadir nodos sin que se deba realizar una reprogramación de todo el sistema.

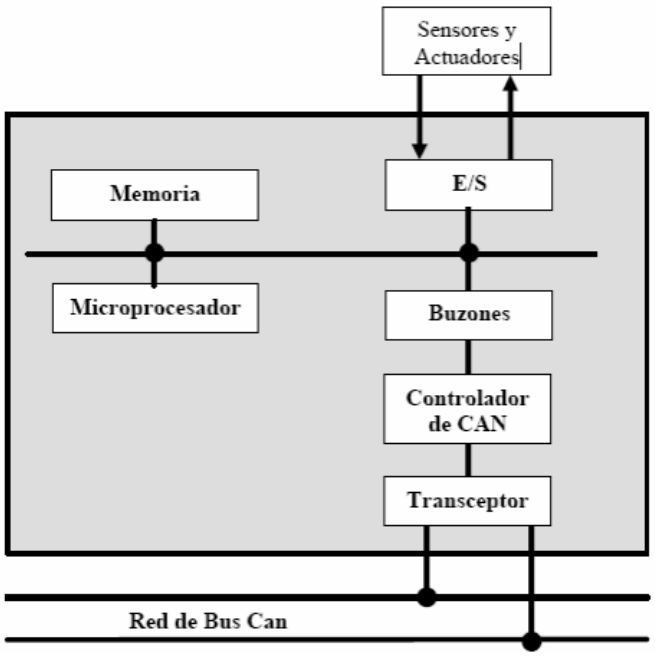


Figura 1.1 Estructura del nodo

(Martínez Mas, 2012)

ISO define dos tipos de redes CAN: una red de alta velocidad (de hasta 1 Mbps) definida por la ISO 11898-2, y una red de baja velocidad tolerante a fallos (menor o igual a 125 Kbps) definida por la ISO 11898-3 (Martínez Mas, 2012). Al punto anterior habría que añadir que la velocidad también depende de la distancia hasta un máximo de 1000 metros (aunque podemos aumentar la distancia con bridges o repetidores) (Martínez Mas, 2012).

Este protocolo está orientado a la transmisión de datos los cuales tienen un modo de identificarse e irse ordenando de acuerdo a tramas para su transmisión. Como los mensajes están identificados cada componente tiene la capacidad de seleccionar y permitir el paso únicamente de los mensajes que este solicita.

En el modo Multidifusión (*multicast*) se permite que todos los nodos puedan acceder al bus de forma simultánea con sincronización de tiempos o en el modo medio compartido (*broadcasting*) la información es enviada en la red a todos los destinos de forma simultánea (Martínez Mas, 2012). Por lo tanto los nodos deben ser capaces de aceptar o rechazar los datos que pasan por la red dependiendo del requerimiento de la información.

1.9.1.2 Ventajas.

CAN es una red duradera y económica que permite a varios dispositivos comunicarse entre sí. Un beneficio es que permite a las unidades de control electrónico (ECUs) tener una sola interfaz CAN, en lugar de diferentes entradas analógicas y digitales para cada dispositivo en el sistema (NATIONAL INSTRUMENTS , 2011).

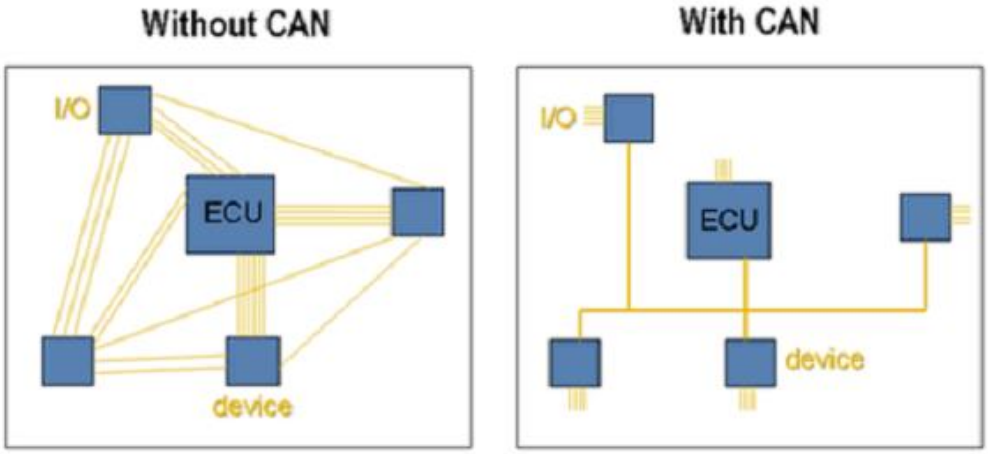


Figura 1.2 Comparación entre conexiones sin y con el protocolo CAN

(https://www.ni.com/es-cr/innovations/white-papers/06/controller-area-network--can--overview.html)

1.9.1.3 Tramas del CAN.

Trama de datos

Se la usa para transmitir información que puede ser recibida por uno o todos los nodos conectados a la red. Además es utilizada por los nodos para enviar de igual manera información para que sea determinada por la computadora. . Puede incluir entre 0 u 8 bytes de información útil (Martínez Mas, 2012).

Trama de información remota

Todos los nodos involucrados en el vehículo pueden requerir información de otros nodos, por lo que se solicita una trama de datos que hace que, el nodo que dispone de la información envíe la misma a la red CAN para que pueda ser utilizada por todos los nodos que requieran dicha información.

Trama de error

Estos pueden ser generados por cualquier nodo que perciba algún error. Consiste en dos campos: Indicador de error ("*Error Flag*") y Delimitador de error (“*Error Delimeter*”) (Martínez Mas, 2012).

Si un nodo en estado de error "Activo" detecta un error en el bus, se interrumpe la comunicación del mensaje en proceso generando un "Indicador de error activo" que consiste en una secuencia de 6 bits dominantes sucesivos (Martínez Mas, 2012). Esta secuencia interfiere con el llenado automático de bits en los mensajes, lo que trae como consecuencia que se produzcan más errores en los demás nodos. Para contrarrestar esto el mensaje de error puede llegar a contener de 6 a 12 bits dominantes con un campo de delimitación que contiene 8 bits recesivos. Con esto se logra que la comunicación se reestablezca y el nodo con falla vuelva a transmitir datos.

Si un nodo en estado de error "Pasivo" detecta un error, el nodo transmite un "Indicador de error pasivo" seguido, de nuevo, por el campo delimitador de error (Martínez Mas, 2012). Este indicador es formado por 6 bits recesivos seguidos, entonces esta trama de error es comprendida por 14 bits recesivos. Es por esto que este indicador de error pasivo no afecta a los nodos conectados, a menos que el error sea reconocido por el mismo nodo que lo genero.

El llenado automático de bits es cuando cada cinco bits iguales consecutivos en el mensaje se incluye un bit de valor contrario a los anteriormente mencionados.

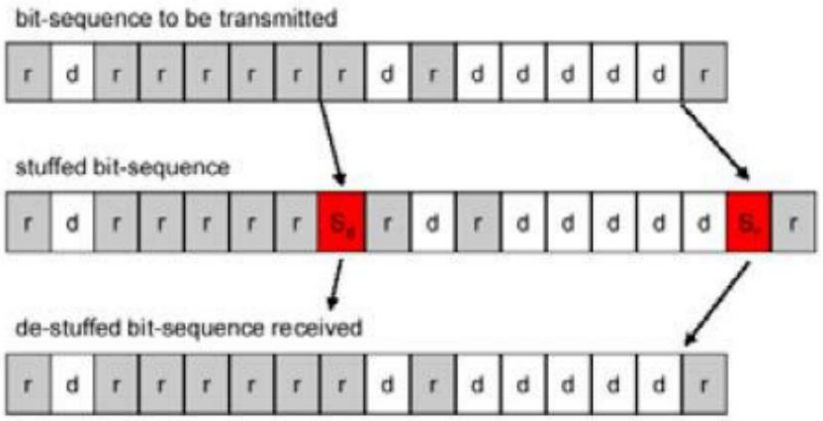


Figura 1.3 Llenado automático de bits

(Martínez Mas, 2012)

Trama de sobrecarga

Es utilizada cuando un nodo requiere más tiempo para procesar los datos que ha recibido y esta puede ser creada en el espacio entre tramas en cualquiera de los nodos conectados, haciendo que no pueda recibir otros datos hasta que termine la interpretación de los datos que se están procesando. Además un mismo nodo puede únicamente generar hasta dos tramas de sobrecarga seguidas.

Espaciado entre tramas

Este sirve para separar una trama de otra y consta de por lo menos 3 bits recesivos, lo que se denomina "*intermission*". Dado este espacio en un error pasivo se puede continuar la transmisión de datos o se puede entrar en un modo de reposo. Mientras que en un estado de error pasivo es necesario que transcurran 8 bits recesivos para poder continuar con la transmisión de datos.

Bus en reposo

Cuando el protocolo de comunicación no tiene actividad entra en un modo de reposo en el que los datos son únicamente de nivel recesivo.

1.9.1.4 Adjudicación de Mensajes en CAN

En ocasiones cuando varias unidades de control requieran enviar su protocolo de comunicación de manera simultánea, se debe determinar cuál protocolo se transmite primero, en este caso será el de prioridad superior. Cada bit que forma parte del código tiene una validación que puede ser de 0V (validación superior) o de 5V (validación inferior).

Tabla 1.1 Validación de bits

Bit con	Valor	Validación
0V	0	Superior
5V	1	Inferior

Los protocolos de datos tienen once bits con los cuales se puede determinar la prioridad del mismo. Por ejemplo, si se requiere transmitir protocolos de la unidad de control del ABS/EDB, del motor o de una transmisión automática al mismo tiempo se busca las prioridades dependiendo de los bits del protocolo de cada una de ellas como se muestra a continuación.

Tabla 1.2 Detección de prioridades en protocolos de datos

Prioridad	Protocolo de datos	Campo de estado
1	Freno	001 1010 0000
2	Motor	010 1000 0000
3	Transmisión	100 0100 0000

Como se puede observar en la tabla 1.2 el protocolo de datos del freno tiene mayor prioridad puesto que el campo de estado empieza con dos bits de validación superior. Por lo tanto, este sería el que se transmite primero en el caso de encontrarse con otros protocolos de menor prioridad.

1.9.1.5 Formas de conexión de red CAN

Configuración punto a punto

Es el tipo de conexión más sencilla porque únicamente está compuesta por dos módulos. Se puede usar uno o dos cables trenzados.

Figura 1.4 Conexión punto a punto

(www.autosoporte.com)

Configuración de anillo

Este tipo de configuración de conexión se puede encontrar en conexiones grandes donde pueden estar involucrados de 4 a 20 módulos. Esta conexión se caracteriza por la unión de

los componentes en serie, es decir, donde termina uno se conecta el otro. La principal ventaja de este es que si la conexión se rompe en algún punto la información puede tomar otro camino para llegar a los diferentes módulos. Por otro lado, presenta como desventaja que por cada módulo se requiere mínimo dos nodos de conexión lo que implica mayor número de conexiones y cableado.

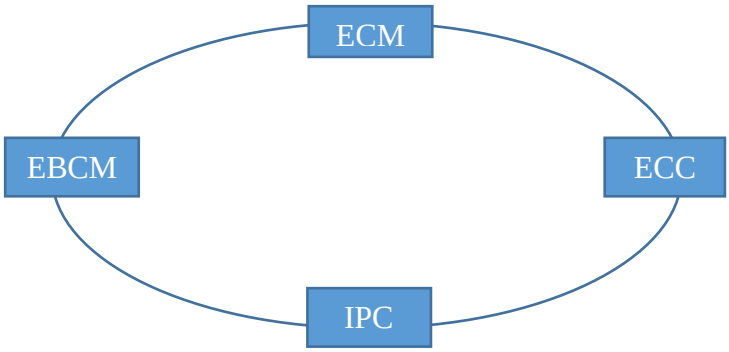


Figura 1.5 Conexión de anillo

(www.autosoporte.com)

Conexión estrella

Esta se caracteriza porque todos los componentes se conectan en un solo punto. En este tipo de conexión se tiene como ventaja que si existe algún problema con un módulo o con su conexión se suspenderá la comunicación únicamente con este permitiendo que el resto de sistema siga funcionando con normalidad. Como desventaja presenta que todos los módulos se conectan a un nodo maestro, el cual al recibir la conexión de todos los módulos presenta una gran cantidad de cableado.

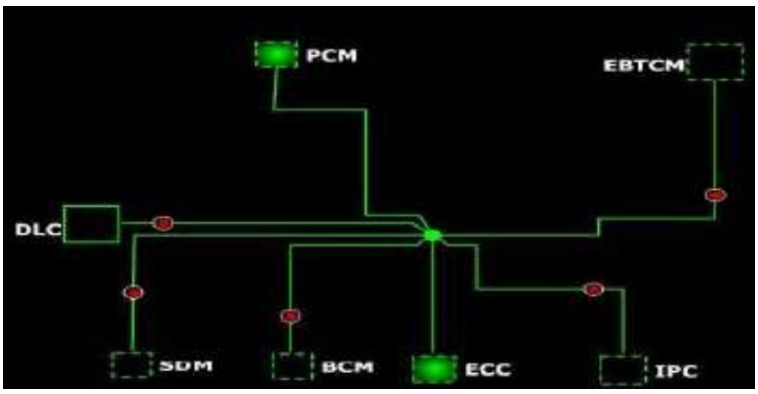


Figura 1.6 Conexión estrella

(www.autosoporte.com)

Configuración lineal

En este tipo de conexión es mínima la cantidad de cable y la ruta de alambrado dentro del vehículo es más sencilla. La desventaja de este tipo de conexión es que si se rompe la conexión quedara deshabilitado todos los módulos que están detrás del punto de ruptura.

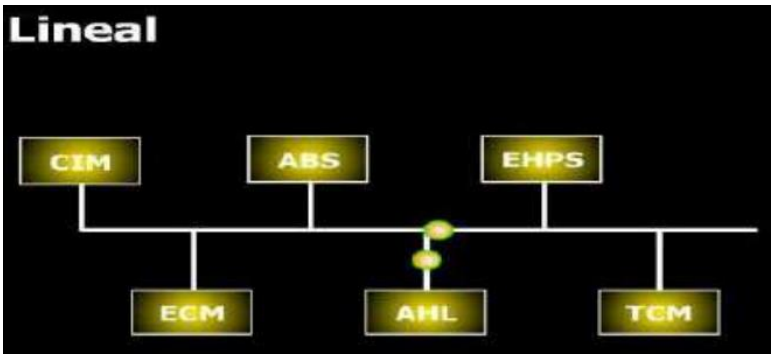


Figura 1.7 Conexión lineal

(www.autosoporte.com)

Configuración DAYSY CHAIN (dos cables).

En la actualidad es el tipo de conexión más utilizada por los fabricantes de vehículos por su estructura sencilla con una cantidad mínima de nodos y con dos cables que transmiten la misma información, hacen de este tipo de conexión uno de los más confiables. Por otro lado, al igual que en la configuración en línea si se presenta una ruptura en la conexión o en algún módulo la comunicación quedaría interrumpida desde el punto de ruptura.

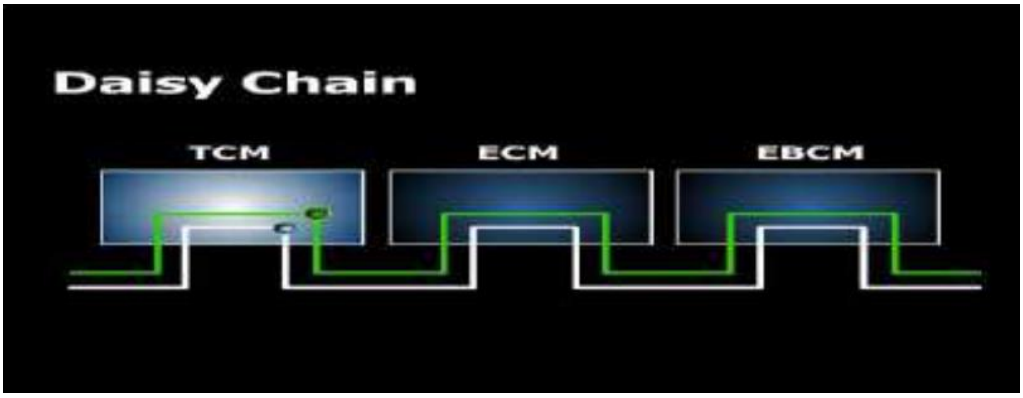


Figura 1.8 Configuración Daysy Chain

(www.autosoporte.com)

1.9.2 Sistema de diagnóstico a bordo OBD-II.

1.9.2.1 Descripción del OBD-II.

El sistema OBD *(On Board Diagnostic*) es un término que se refiere al autodiagnóstico de los diferentes sistemas que conforman el vehículo. El mismo que proporciona información técnica acerca del funcionamiento del vehículo. La información que transita por el OBD ha ido variando dependiendo de los avances tecnológicos con el pasar del tiempo. En las primeras versiones del OBD cuando se detectaba una falla únicamente se tenía una luz de advertencia la cual, era identificada por el número y la duración de las pulsaciones de la luz MIL. El problema de OBD I es que no estaba estandarizado y las marcas podían dar tantos datos como vean necesarios, incluso los DLC no estaban estandarizados. Los modernos sistemas OBD utilizan un puerto de comunicaciones digitales estandarizado para proporcionar datos en tiempo real, además de una serie estandarizada de códigos de problemas de diagnóstico (DTC), los cuales permiten que un técnico identifique y remedie las fallas del vehículo (Denton, 2016). Las últimas actualizaciones en cuanto al OBD es el OBD-II y la versión europea EOBD-II las cuales tienen grandes similitudes.

Para poder tener acceso a los códigos hay una serie de actividades que se llevan a cabo:

- La unidad de pruebas envía códigos.
- La ECU responde con otro código de respuesta.
- La unidad de prueba adopta el conjunto apropiado del código previamente recibido.
- La ECU transmite el código de falla.

Además, los códigos DTC tienen otras capacidades como son:

- Identificación de que los códigos de respuesta estén de acuerdo con los códigos de solicitud de información.
- Adquisición de datos en tiempo real con detección de códigos con valores erróneos.

1.9.2.2 Componentes del OBD-II

Los componentes del sistema OBD-II son: la ECU (*Engine Control Unit*) conocida como la computadora del automóvil, los transductores encargados de enviar los datos hacia la ECU, la luz indicadora de fallas (MIL, *Malfunction Indicator Light*) ubicado en el tablero, y el

conector de diagnóstico (DLC, *Data Link Connector*) que sirve de interfaz entre la ECU y los dispositivos de diagnóstico automotriz (Simbaña, Caiza , Chávez , & López , 2016).

ECU

La ECU (*Engine Control Unit*) es esencialmente la computadora del vehículo que se encarga de solicitar, recibir y procesar los datos de los transductores.

Transductores del vehículo

Estos son unidades que tienen la función de controlar el funcionamiento y el modo de operación del motor. Entre las medidas que pueden obtener los transductores del motor están: revoluciones por minuto del motor, temperatura del líquido refrigerante del motor, presión absoluta del colector de admisión, presión barométrica, temperatura de aire de admisión, posición del acelerador, velocidad del automóvil, entre otras (Simbaña, Caiza , Chávez , & López , 2016).

Tabla 1.3 Transductores del motor del vehículo

SENSOR	MEDIDA
Sensor CKP (*Crankshat Position*)	Revoluciones por minuto
Sensor ECT (*Engine Coolant Temperature*)	Temperatura del refrigerante del motor
Sensor MAP (*Mainfold Absolute Pressure*)	Presión absoluta en el colector de admisión
Sensor IAT (*Intake Air Temperature*)	Temperatura del aire de admisión
Sensor de presión Barométrica	Presión barométrica
Sensor TPS (*Throttle Position Sensor*)	Posición del acelerador
Sensor VVS (*Vehicle Speed Sensor*)	Velocidad del automóvil

Fuente:(Simbaña, Caiza , Chávez , & López , 2016)

Luz indicadora de fallas

La luz indicadora de fallas MIL es utilizada por el sistema OBD-II para alertar al conductor del vehículo que el sistema de diagnóstico ha detectado una falla en el funcionamiento del motor.

Conector DLC

Además sirve como interfaz de acceso y recuperación de datos desde la ECU hacia un equipo de diagnóstico (escáner automotriz) (Simbaña, Caiza , Chávez , & López , 2016). En la tabla 1.4 se detallan los pines del conector DLC.

Tabla 1.4 Pines del conector DLC

PIN	CARACTERÍSTICAS
1	Uso del fabricante
2	Bus (+) J1850 VPM y PWM
3	Uso del fabricante
4	Tierra (chasis)
5	Señal de tierra
6	Bus de datos CAN H (J-2284)
7	Línea K ISO 9141-2
8	Uso del fabricante
9	Uso del fabricante
10	Bus (-) J1850
11	Uso del fabricante
12	Uso del fabricante
13	Uso del fabricante
14	Bus de datos CAN L (J-2284)
15	Línea L ISO 9141-2
16	Voltaje de batería

Fuente: (Simbaña, Caiza , Chávez , & López , 2016)

El conector DLC es una interfaz con forma trapezoidal de 16 pines basado en el estándar SAE J1962, que se ubica bajo el tablero, generalmente en el lado del conductor.

Figura 1.9 Conector DTC

(http://www.aficionadosalamecanica.com/obd2.htm)

1.9.2.3 Protocolos de señal OBD-II

Los protocolos de señal OBD-II presentan variaciones dependiendo de los diferentes fabricantes de vehículos existentes, donde los principales cambios son los pines utilizados que están dispuestos en diferente orden o con diferentes funciones. Entre ellos podemos encontrar los siguientes:

- SAE J1850 PWM ((*Pulse Width Modulatión*) o (modulación de pulso y ancho)) usado generalmente por Ford USA.
- SAE J1850 VPW ((*Variable Pulse Width*) o (ancho de pulso variable)) usado por General Motors.
- ISO 9141-2 usados en vehículos Chrysler.
- ISO 14230 KWP2000 (Teclado de protocolo 2000) usado por Renault.
- ISO 5765 CAN: el protocolo CAN fue desarrollado por Bosch para control automotriz e industrial, desde 2008 a todos los vehículos vendidos en Estados Unidos (y muchos otros) se les exige implementar el CAN como uno de los protocolos de comunicación de señales (Denton, 2016).

Todas las clavijas de salida del OBD-II usan el mismo conector pero se utilizan diferentes clavijas, excepto la clavija 4 (tierra de la batería), la clavija 5 (tierra de señal) y la clavija 16 positivo de la batería (Denton, 2016).

1.9.2.4 Códigos PID.

Para la obtención de datos el OBD-II dispone de 10 modos de medición, donde cada uno de ellos cumple diferentes funciones en el diagnóstico. Por lo que, para solicitar información acerca del funcionamiento del vehículo por medio del OBD-II se utilizan códigos PID

(*Parameter Identification*). Cada código PID está relacionado con una medida específica de los modos 1 y 2 del sistema OBD-II (Simbaña, Caiza , Chávez , & López , 2016).

Modos de los códigos PID.

Los PID son estandarizados según el protocolo de señal OBD-II. Para la obtención de respuestas utilizando PID se siguen los siguientes pasos:

- Introducir el PID necesario
- Envío del PID por el protocolo de comunicación
- La ECU reconoce el PID y solicita la información al componente indicado
- La información regresa por el protocolo de comunicación

Tabla 1.5 Modos de medición OBD-II

MODO	CARACTERÍSTICAS
Modo 1	Obtención de datos actualizados
Modo 2	Acceso a cuadro de datos congelados
Modo 3	Obtención de códigos de falla.
Modo 4	Borrado de códigos de falla y valores almacenados.
Modo 5	Resultado de las pruebas de los transductores de oxígeno
Modo 6	Resultado de las pruebas de otros transductores
Modo 7	Muestra de códigos de falla pendientes
Modo 8	Control de funcionamiento de componentes
Modo 9	Información del automóvil

Fuente: (Simbaña, Caiza , Chávez , & López , 2016)

Modo 01 (Mostrar datos actuales)

Este modo tiene la capacidad de solicitar datos en tiempo real de los componentes conectados al protocolo de comunicación. Además vale recalcar que las respuestas que nos dan los diferentes componentes vienen dados en modo hexadecimal, los mismos que debemos transformar con una serie de ecuaciones que nos permitirá tener los datos en modo decimal.

Modo 02 (Acceso a cuadro de datos congelados)

Este modo nos da acceso a datos del funcionamiento del motor en el momento exacto donde se produjo algún tipo de falla o desperfecto. Con este modo podemos realizar un diagnóstico un poco más claro del porque se produjo las fallas. Estos datos son almacenados por la ECU del vehículo.

Modo 03 (Obtener códigos de falla)

Este modo nos permite tener los códigos de falla o DTC propios del vehículo, los mismos que vienen en paquetes de 6 bytes que darán información de la falla en sí. El primer carácter proporciona información del lugar donde se generó el código de falla.

Tabla 1.6 Interpretación del primer caracter DTC

A7	A6	Primer caracter DTC
0	0	P – Motor
0	1	C – Chasis
1	0	B – Carrocería
1	1	U – Red

Fuente: (Ortiz López, 2014)

Por otro lado el segundo carácter determina si el DTC es universal (también denominado SAE), o si es propio de un fabricante.

Tabla 1.7 Interpretación del Segundo carácter DTC

A5	A4	Segundo caracter DTC
0	0	0
0	1	1
1	0	2
1	1	3

Fuente: (Ortiz López, 2014)

El tercer caracter, en el caso de tener una falla en el motor, identifica el subsistema en el que se produjo la falla. Por último el cuarto y quinto carácter proporcionan el número de falla.

Tabla 1.8 Interpretación del tercer, cuarto y quinto caracter DTC

A3	A2	A1	A0	Tercer caracter DTC
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	A
1	0	1	1	B
1	1	0	0	C
1	1	0	1	D
1	1	1	0	E
1	1	1	1	F

Fuente: (Ortiz López, 2014)

En la figura 1.12 se puede observar la estructura de un código DTC con su respectiva interpretación.

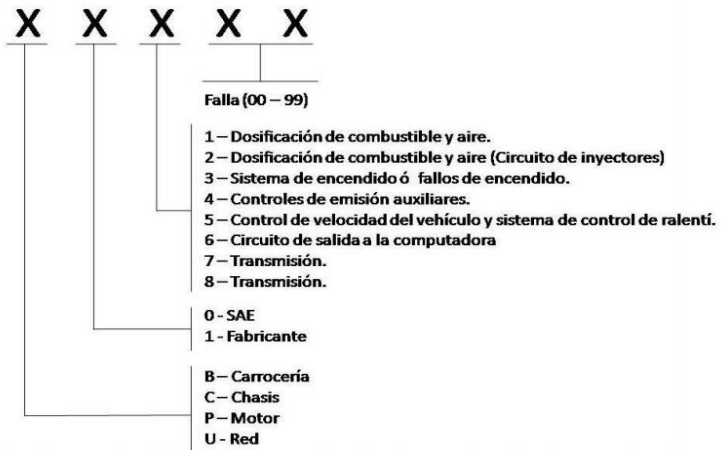


Figura 1.10 Interpretación de código DTC

(Ortiz López, 2014)

Modo 04 (Borrar códigos de falla y datos almacenados)

Este modo nos permite borrar el código de falla que está presente y hace que la luz MIL se apague.

Modo 05 (Resultados de las pruebas de los transductores de oxígeno)

Este modo muestra los resultados de las pruebas realizadas a los sensores de oxígeno para determinar el funcionamiento de los mismos así como la eficiencia del convertidor catalítico (Ortiz López, 2014). El número de sensores de oxigeno presentes en el vehículo depende de la tecnología que el mismo contenga.

Modo 06 (Resultado de pruebas de otros transductores)

Con este modo se obtiene los resultados del monitoreo de todos los componentes conectados al protocolo de comunicación, verificando que se encuentren funcionando normalmente. Es un modo muy útil que puede revelar el funcionamiento interno del OBDII y permite predecir cuándo un DTC puede aparecer, comparando a parámetros de funcionamiento provistos por el fabricante y los parámetros actuales de los sensores (Ortiz López, 2014).

Modo 07 (Muestra códigos de falla pendientes)

Muestra los códigos de falla que fueron detectados en el último o en el actual ciclo de conducción.

Modo 08 (Control de funcionamiento de componentes)

Este modo permite realizar comprobaciones en el funcionamiento de actuadores.

Modo 09 (Información del automóvil)

Este modo contiene información del número de identificación del vehículo o *Vehicle Identification Number* (VIN) y la información de la calibración de la ECU (Ortiz López,

2014). Además este modo es útil porque a partir de la información de la calibración de la ECU se puede modificar los parámetros y obtener el rendimiento que se desee.

CAPÍTULO II

2. MATERIALES Y MÉTODOS

2.1 METODOLOGÍA DE LA INVESTIGACIÓN

En el presente capítulo se detallan los procesos realizados para el cumplimiento de los objetivos planteados anteriormente. Se comienza por la infiltración a la red CAN del vehículo para obtener los códigos hexadecimales y discriminar los que sirvieron para la programación de la interfaz electrónica, para ello se proporciona una breve descripción de los equipos utilizados con su respectivo software. Además, se detalla el procedimiento a seguir para la programación tanto en Arduino IDE como en LabVIEW, que son los software con los que se realizó el programa final.

2.1.1 Propósito Investigativo.

La finalidad de la identificación de los datos hexadecimales correspondientes al envío y recepción de mensajes de información del vehículo es comprender el modo en que un escáner automotriz solicita y recepta la información en la red CAN. Esto a partir de la utilización de ciertos modos de diagnóstico que permiten tener acceso a datos en tiempo real, lectura de códigos de falla y borrado de los mismos. Se elaborará un escáner automotriz genérico con las funciones anteriormente detalladas, el mismo que puede ser usado para el diagnóstico en vehículos con OBD-II genérico.

2.1.2 Determinación de metodología de investigación.

Para el presente trabajo se optó por el método experimental, dado que a lo largo de este fue necesario experimentar con varios códigos que conllevan diversas variables que fueron cambiando para posteriormente verificar el correcto funcionamiento de la interfaz y la veracidad de los datos obtenidos. Esto con el fin de poder hacer la comparación con un escáner automotriz convencional. Una vez obtenidos los resultados deseados con cada uno de estos códigos se unen en un solo programa que cumple con las propuestas establecidas.

2.1.3 Procesos metodológicos

En el diagrama de flujo de la figura 2.1 se detalla el proceso general a seguir para la obtención de los objetivos planteados en el presente proyecto.

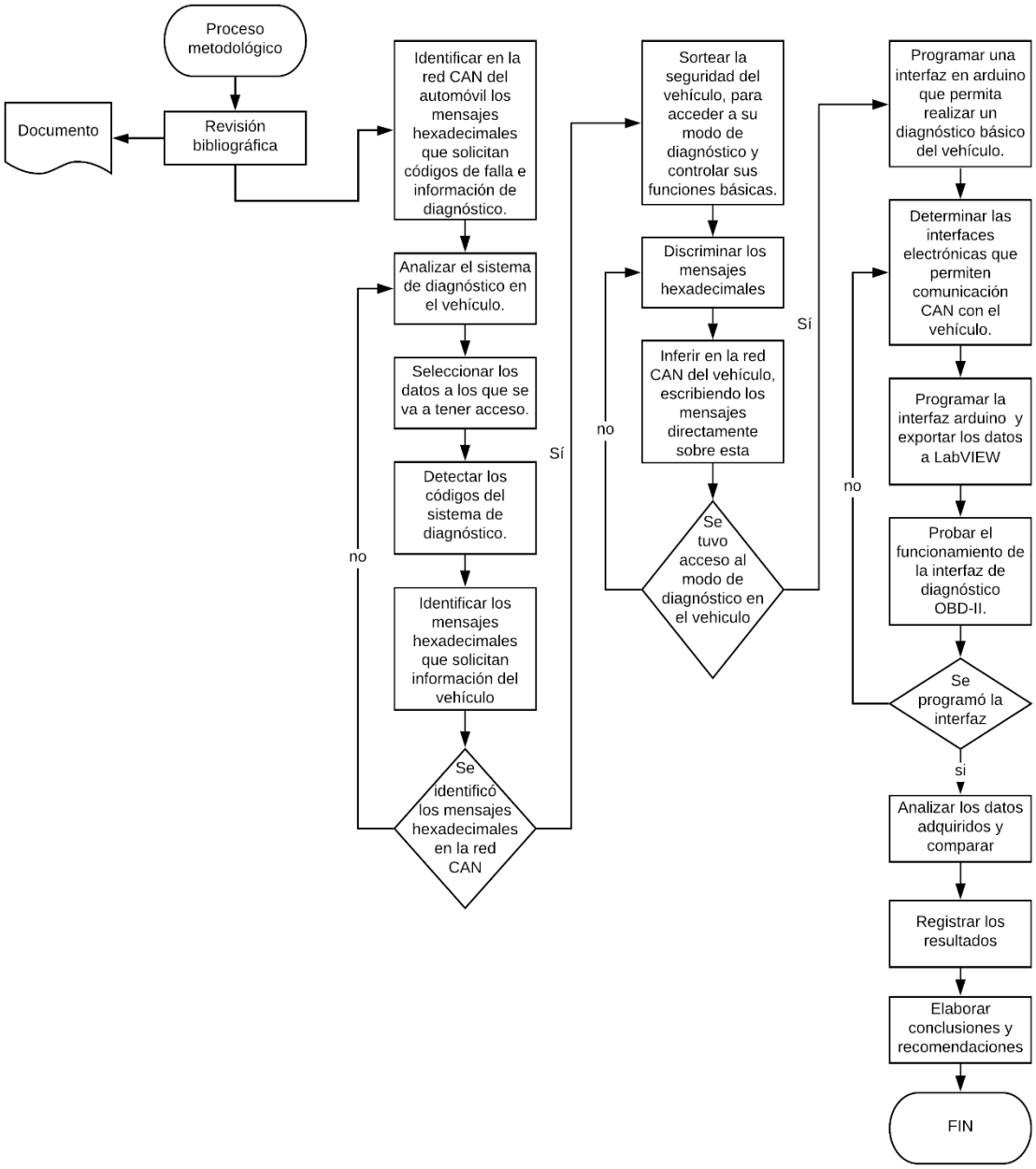


Figura 2.1 Diagrama de flujo de objetivos

2.2 MATERIALES Y EQUIPOS

Para las diferentes pruebas realizadas fue indispensable el uso de un vehículo equipado con OBD-II genérico para que responda con los demás equipos y asegurar el correcto funcionamiento de los mismos. En este caso se utilizó un Chevrolet SAIL año de fabricación 2018, en el que se hizo toda la extracción de códigos y posteriormente las pruebas de los mismos. Para ello se utilizaron los equipos que se describen a continuación.

2.2.1 Dispositivo CAN Bus Analyzer.

La herramienta CAN BUS Analyzer está diseñada para ser un monitor de bus CAN fácil de usar y de bajo costo que se puede usar para desarrollar y depurar una red CAN de alta velocidad (Microchip-Technology, 2011). Esta útil herramienta puede ser utilizada en cualquier campo en el que se utilice el protocolo de comunicación CAN.

Figura 2.2 Dispositivo CAN bus Analyzer

(Microchip Technology, 2011)

La herramienta CAN Analyzer admite CAN 2.0b e ISO 11898-2 (CAN de alta velocidad con velocidades de transmisión de hasta 1 Mbit / s) (Microchip-Technology, 2011). Este dispositivo se conecta al vehículo a través de un conector RS232C.

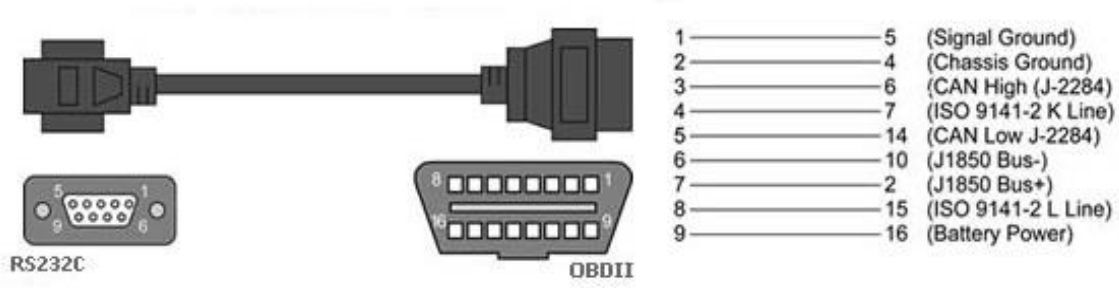


Figura 2.3 Cable de conexión entre el OBD-II y RS232C

(Microchip Technology, 2011)

Tabla 2.1 Pines del conector RS232C

Pin Number	***Signal Name***	***Signal Description***
1	*No connected*	*N/A*
2	*CAN L*	*Dominant Low*
3	*GND*	*Ground*
4	*No connected*	*N/A*
5	*No connected*	*N/A*
6	*GND*	*Ground*
7	*CAN H*	*Dominant High*
8	*No connected*	*N/A*
9	*No connected*	*N/A*

Fuente: (Microchip-Technology, 2011)

El dispositivo CAN bus Analyzer puede ser comparado a los analizadores de redes CAN de gama alta que por consecuencia tienen un costo relativamente alto en comparación con este dispositivo. Esta herramienta da la facilidad de ver y almacenar datos que transitan por el protocolo de comunicación CAN. Además se puede enviar mensajes a la red CAN.

2.2.2 ELM 327 CAN bus scanner.

El ELM 327 es utilizado para la trasmisión de datos desde el conector OBD-II del vehículo hasta el conector RC232 del analizador de red CAN. Este dispositivo es capaz de conectarse con 12 diferentes protocolos de comunicación OBD-II, identificando automáticamente el protocolo con el que se está conectando.

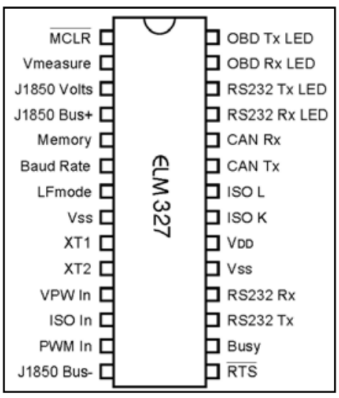


Figura 2.4 Pines del ELM 327

(Cervantes & Espinoza , 2010)

Tabla 2.2 Descripción de los pines del ELM 327

PIN	TIPO	DESCRIPCIÓN
MCLR	Entrada	Reinicia al ELM327
Vmeasure	Entrada	Entrada el convertidor A/D, el cual mide un voltaje de 0V a 5V.
J1850 volts	Salida	Control del bus J1850
J1850 Bus+	Salida	Proporciona los datos hacia la línea J1850 Bus+
Memory	Entrada	Controla las funciones de la memoria interna
Braud Rate	Entrada	Controla la velocidad del bus
LF Mode	Entrada	Controla el formato de término de los mensajes
XT1 y XT2	Cristal	Un cristal debe ser conectado a esta unidad
VPW ln	Entrada	Recibe los datos del bus J1850 VPW
ISO ln	Entrada	Recibe los datos del bus ISO 9141 e ISO 14230
PWM ln	Entrada	Recibe los datos del bus J1850 PWM
J1850 Bus-	Salida	Proporciona los datos hacia la línea J1850 Bus-
RTS	Entrada	Controla la prioridad de las peticiones de un nuevo comando
Busy	Salida	Indica si el ELM327 está listo para recibir comandos
RS232 Tx y Rx	RS232	Líneas del bus RS232
Vdd	Alimentación	Debe ser conectado a 5V
ISO-K e ISO-L	Salida	Proporciona los datos hacia los protocolos ISO9141 e ISO14230
CAN Tx y Rx	CAN	Línea de bus CAN
RS232 y OBD Tx y Rx LED	Salida	Proporciona salida de 5V y alimenta LEDs

Fuente: (Cervantes & Espinoza , 2010)

2.2.3 Interfaces de programación y comunicación CAN bus.

2.2.3.1 Placa Arduino mega 2560.

La placa Arduino mega 2560 es fabricada con un controlador Atmega 2560 el mismo que cuenta con pines de entrada y salida ya sean análogas o digitales. Esta placa está diseñada para objetos interactivos teniendo la posibilidad de conectarse a una computadora por el

puerto USB. El Arduino Mega tiene 54 pines de entradas/salidas digitales (14 de las cuales pueden ser utilizadas como salidas PWM), 16 entradas análogas, 4 UARTs (puertos serial por hardware), cristal oscilador de 16MHz, conexión USB, jack de alimentación, conector ICSP y botón de *reset* (Olimex). Esta placa puede trabajar tanto conectada a la computadora o conectada a una fuente de alimentación de 9 a 12 V de corriente continua. Además esta placa es compatible con casi todos los *Shields* disponibles para Arduino.

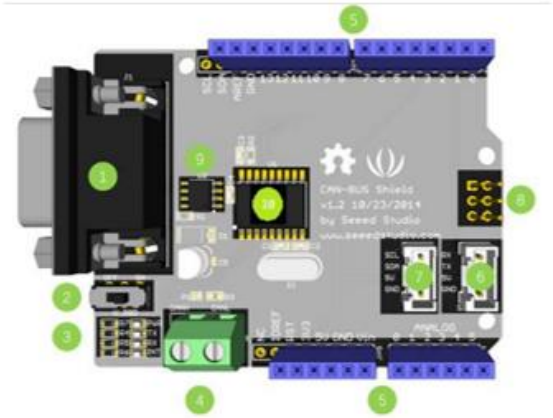
Figura 2.5 Características de la placa Arduino mega 2560

(http://arduino.cl/arduino-mega-2560)

2.2.3.2 CAN bus shield Sunflower.

Las *shield* son placas que sirven para complementar el funcionamiento de las placas Arduino, en este caso de la placa Arduino mega 2560. Estas placas se comunican con el Arduino por medio de los pines digitales y análogos, y además la alimentación de estas viene dada por la placa Arduino. Esta funciona como una interfaz para la comunicación entre el Arduino IDE y la red CAN.

Otras características que posee la placa utilizada son: Alcanza velocidades de hasta 1 Mbps en la implementación CAN 2.0B, soporta tramas de 11 y 29 bits, tiene una interfaz SPI de hasta 10MHz, posee indicadores LED y tiene dos buffers de recepción con almacenamiento de mensajes con prioridad (Sánchez Carrizo , 2017).



N°	Componente
1	Interfaz DB9
2	V-OBD
3	Indicador LED
4	CAN high y CAN low
5	Pin de salida Arduino
6	Conector serial Grove
7	Conector I2C Grove
8	Conectores del ICSP
9	IC-MCP2551
10	IC-MCP2515

Figura 2.6 Componentes del CAN bus shield sunflower

(Sánchez Carrizo , 2017)

La figura 2.7 muestra el modo de conexión entre el puerto OBD-II del vehículo y conector DB9 de la placa CAN Bus Shield. En este caso se puede optar por usar únicamente los pines correspondientes a CAN_H y CAN_L, esto debido a que la alimentación y tierra vendrán desde la placa Arduino.

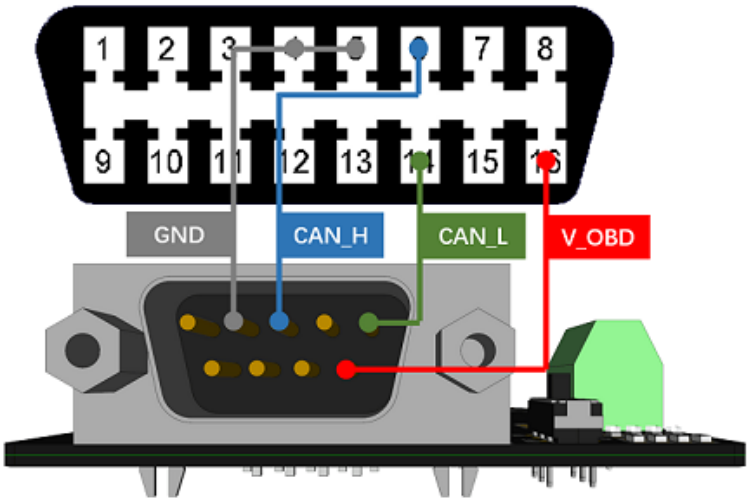


Figura 2.7 Conexión entre el OBD-II y conector DB9

(http://wiki.seeedstudio.com/CAN-BUS_Shield_V1.2/)

IC-MCP2551

Este tipo de dispositivo MCP2551 proporciona una capacidad de transmisión y recepción diferencial para el controlador de protocolo CAN y es totalmente compatible con la norma ISO-11898, incluidos los requisitos de 24V, y funciona a velocidades de hasta 1Mb/s (Sánchez Carrizo , 2017). Este componente tiene la capacidad de tolerar fallas en el sistema y actúa como una interfaz entre el CAN y un bus físico.

IC-MCP2515

El MCP2515 cuenta con dos máscaras de aceptación y seis filtros de aceptación que se utilizan para descartar mensajes no deseados, lo que reduce la sobrecarga del micro controlador anfitrión (Sánchez Carrizo , 2017). Este es el responsable de la recepción y transmisión de tramas de datos en el sistema.

2.3 ANÁLISIS DEL SISTEMA DE DIAGNÓSTICO EN EL VEHÍCULO.

Para el análisis del sistema de diagnóstico del vehículo Chevrolet Sail modelo 2018 se realizaron varias pruebas para verificar que en este se puedan realizar los ensayos posteriores. Para empezar se realiza una prueba sencilla de comunicación entre la ECU del vehículo con el dispositivo ELM327. Si se obtiene comunicación se puede decir que el vehículo cuenta con OBD-II genérico, dado que si no fuera de este tipo no se tendría una respuesta adecuada. Además se observó el puerto OBD-II donde se compara los pines existentes con los pines descritos en la tabla 1.4 y a su vez con los diagramas de conexión del Can Bus Shield y del Mycrochip. De todo lo mencionado anteriormente se debe hacer mayor énfasis en los pines correspondientes a Can High, Can Low, y tierra de señal. Todos los ensayos que se realizaran posteriormente estarán estrictamente relacionados con dichos pines.

2.4 DETECCIÓN DE CÓDIGOS DE SISTEMA DE DIAGNÓSTICO.

En la figura 2.8 se muestra un diagrama de flujo del proceso de infiltración a la red CAN del vehículo, de donde se obtuvieron los datos para proceder con la programación de la interfaz.

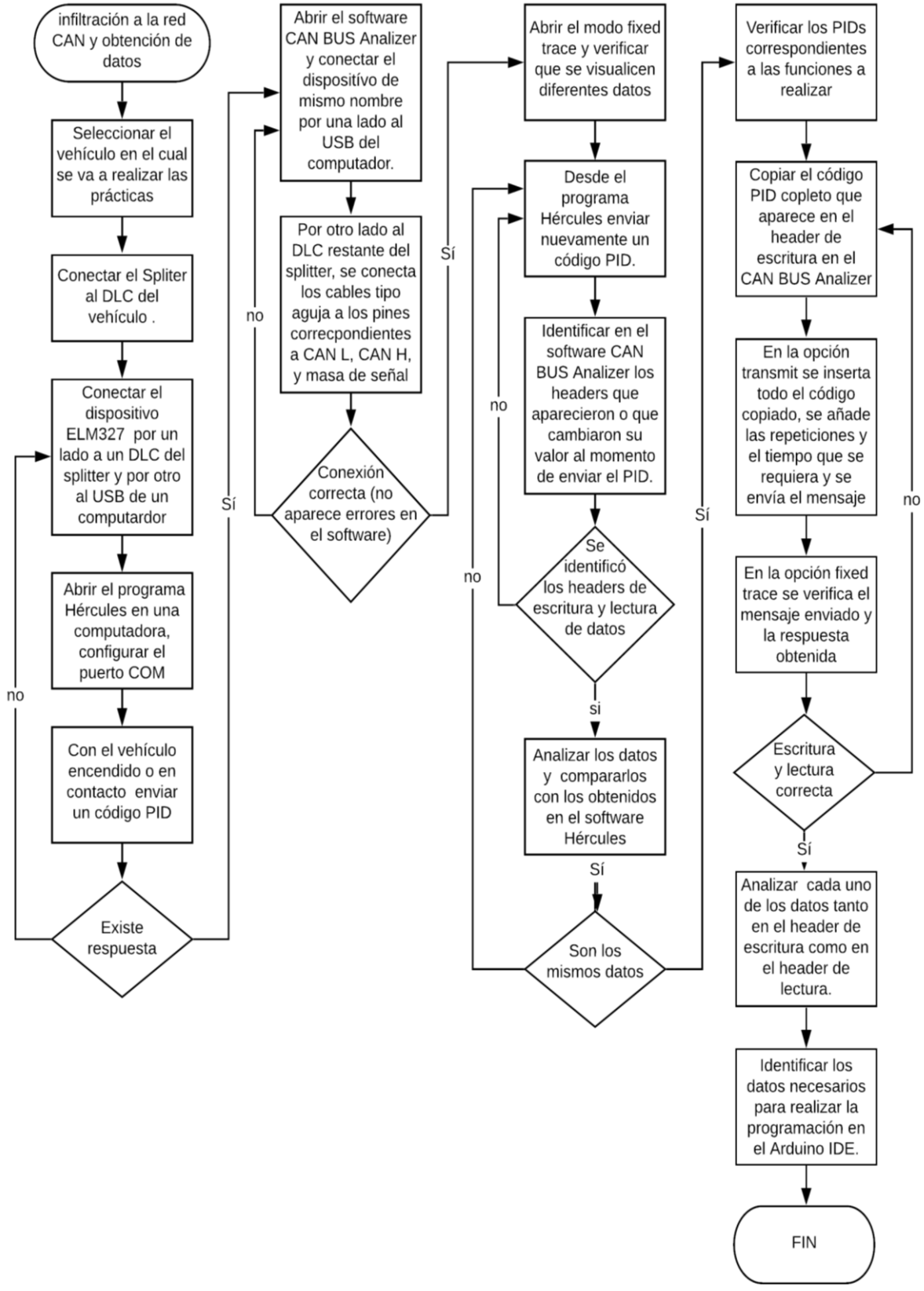


Figura 2.8 Diagrama de flujo de la infiltración a la red CAN.

Para la detección de los códigos se realizó la conexión de los dispositivos ELM327 y el CAN bus Analyzer, estos pueden estar conectados por un lado individualmente a una computadora cada uno o a una sola computadora a la vez. Por el otro lado deben estar conectados al puerto OBD-II del vehículo. Para esto se utiliza una extensión del conector OBD-II (*splitter*) que proporciona dos salidas del puerto para conectar ambos dispositivos individualmente. El *splitter* cuenta con un conector macho y dos hembras, donde se tiene conexión en los mismos pines que el puerto OBD-II del vehículo a ser utilizado.

Figura 2.9 Splitter de conexión OBD-II

Una vez conectados los dispositivos es necesario configurar los programas con los que funcionaran los mismos. El dispositivo CAN Bus Analyzer trabaja con su propio software, mientras que el ELM327 trabajará con el *hyperterminal* Hércules. Estos dos programas deben estar configurados en cuanto a puertos COM y velocidades de comunicación.

Cabe recalcar que para que los dos dispositivos funcionen y se pueda obtener los datos en sus respectivos software es indispensable que el vehículo al que estén conectados debe estar con la llave en posición de contacto o encendido, caso contrario no se visualizará ningún dato.

2.5 PRUEBAS DE FUNCIONAMIENTO DE CÓDIGOS ADQUIRIDOS.

Para verificar que los *headers* y todos los datos obtenidos sean los correctos se lo hace utilizando el programa CAN Bus Analyzer. Para esto es necesario abrir la pestaña "*transmit*" que se encuentra en la barra de tareas opción "*tools*". Esta opción permite realizar la escritura directamente en la red CAN del vehículo, donde es necesario escribir absolutamente todos

los datos que se requiere para tener una respuesta adecuada. A diferencia de la escritura en el *hyperterminal* Hércules se deben escribir los datos que sirven de relleno para completar el mensaje de 8 bytes. Además de es posible escribir el código en reiteradas repeticiones en un lapso de tiempo establecido. Si al enviar el mensaje en la ventana "*fixed trace*" se obtienen las mismas líneas de escritura y lectura que cuando se envió desde el software Hércules, quiere decir que el código escrito es el correcto y puede ser usado para la programación en el Arduino IDE.

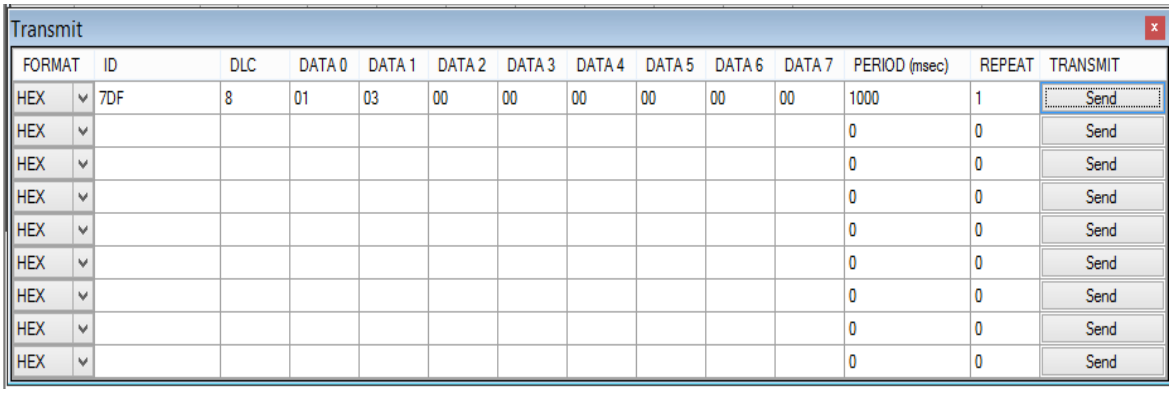


Figura 2.10 Opción Transmit en el programa CAN BUS Analyzer

2.6 DESARROLLO DE INTERFAZ PARA COMUNICACIÓN POR OBD-II.

El desarrollo de la interfaz se la realiza utilizando por un lado el software Arduino IDE para la escritura y lectura de los datos en la red CAN y por otro lado el software LabVIEW, el cual permite realizar una aplicación más amigable y organizada, además de lograr exportar e importar datos del monitor serial del Arduino para una mejor presentación de los datos obtenidos.

2.6.1 Software Arduino IDE

La figura 2.12 muestra un diagrama de flujo para la elaboración del código en el software Arduino IDE que consta con las características que se indicó anteriormente.

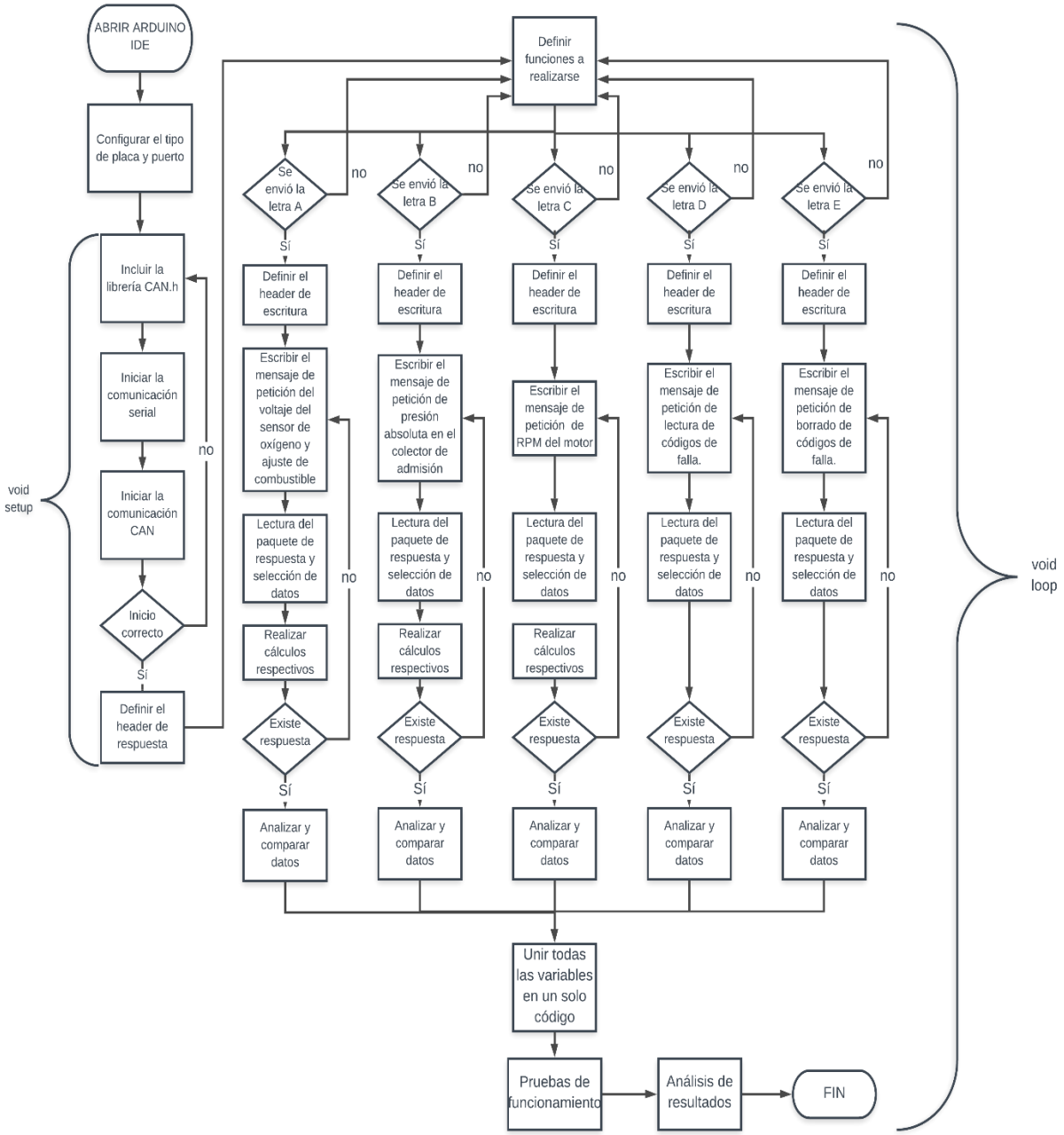


Figura 2.11 Diagrama de flujo para la elaboración del software en Arduino IDE

Antes de empezar la programación es necesario buscar una librería la cual fuera compatible con el dispositivo CAN bus shield que es el encargado de la comunicación del Arduino con la red CAN del vehículo. Una vez determinada la librería a utilizar, se hizo una revisión los códigos con los que esta trabaja y se identifica los que son útiles para el desarrollo de la interfaz. Por otro lado también se analizan los programas de ejemplo ya establecidos que vienen dentro de la librería para que de esta manera se tenga una mejor idea del formato y escritura que se utiliza para el programa en cuestión. Además es indispensable realizar las

configuraciones de acuerdo a la placa que se está usando y al puerto COM con el que se va a trabajar.

Una vez realizadas las configuraciones correspondientes se puede iniciar con la escritura del código de programación siguiendo el siguiente orden:

2.6.1.1 Declaración de variables y librerías.

- Incluir la librería previamente añadida al software.
- Precisar que las direcciones (*headers*) van a ser constantes en todo el programa.

2.6.1.2 Void setup

- Iniciar el monitor serial a la velocidad requerida, en este caso 9600 braudios.
- Incluir un mensaje de inicio como muestra de que el monitor serial se inició correctamente.
- Correr el CAN bus a una velocidad de 500 kb por segundo.
- En caso de fallas en la iniciación del CAN bus añadir un mensaje de inicio fallido.
- Definir las direcciones o *headers* en los cuales se van a tener los datos de respuesta a los códigos PIDs que serán enviados.

2.6.1.3 Void loop

- Especificar los *headers* en los cuales se va a realizar la escritura de los códigos PIDs.
- Enviar el paquete de dados correspondiente el PID requerido, para esto se basa en el formato de escritura que se identificó al enviar el mensaje en el programa CAN BUS Analyzer en la opción *transmit*. Cada dato debe ser enviado individualmente, es decir, en una línea cada uno. No es necesario enviar los datos que únicamente sirven de relleno para completar el mensaje de 8 bytes. Por último, se debe indicar que el paquete de datos a enviar está completo para finalizar la escritura.

- Se analiza el paquete de respuesta indicando que los valores deben ser diferentes de 0. Los valores correspondientes a la longitud del mensaje, a la respuesta al modo y al PID deben ser leídos, pero a la vez discriminados, ya que estos no interfieren en los datos que se están solicitando.

- Los datos que sí serán usados para el cálculo del resultado al PID, deben ser leídos e incluidos en la formula propia del PID en cuestión.

- Por último se imprime el valor resultante de la fórmula utilizada anteriormente, por motivo de prueba se añade un tiempo de repetición del ciclo.

Este proceso se repite para los PIDs del modo 1 de diagnóstico, haciendo programas individuales para comprobar su funcionamiento de una manera más fácil. Mientras que para el modo 3 y 4 que también serán utilizados, se hace una variación únicamente en el *void loop*, como se detalla a continuación:

- Especificar los *headers* en los cuales se va a realizar la escritura de los códigos PIDs.

- Enviar el paquete de datos correspondiente al modo, sea este 3 o 4, para esto se basa en el formato de escritura que se identificó al enviar el mensaje en el programa CAN BUS Analyzer en la opción *transmit*. Cada dato debe ser enviado individualmente, es decir, en una línea cada uno. En esta sección se envía un dato menos puesto que solo se envía el modo, más no un PID. No es necesario enviar los datos que únicamente sirven de relleno. Por último se debe indicar que el paquete de datos a enviar está completo para finalizar la escritura.

- Se analiza el paquete de respuesta indicando que se los valores deben ser diferentes de 0. Los valores correspondientes a la longitud del mensaje, a la respuesta al modo deben ser leídos, pero a la vez discriminados ya que estos no interfieren en los valores que se están solicitando.

- En el caso del modo 3 o lectura de códigos de falla es necesario que los valores que por defecto fueron transformados a modo decimal sean regresados a su modo original en hexadecimal. Para finalizar se imprimen los valores y se realizan las pruebas respectivas.

- Por otro lado para el modo 4 o borrado de códigos de falla, basta solo con enviar el paquete de datos ya que no se obtendrá una respuesta la cual sea necesario presentar,

sino que, es suficiente con un mensaje de confirmación que indique que el código fue ejecutado.

2.6.1.4 Unión de todos los códigos en uno solo

Para realizar un código general que contenga a todos los descritos anteriormente, se opta por hacerlo creando variables como letras para utilizarlas como condición para que cada código sea ejecutado dependiendo de las constantes definidas anteriormente en la delimitación del proyecto.

- La parte de librería, inicio del monitor serial y arranque del CAN bus se mantienen iguales, los cambios son únicamente en la sección de *Void loop* que es la parte que se requiere una repetición cíclica.
- Se crean caracteres tantos como acciones a ejecutar. Estos deben estar descritos como datos de lectura dentro del monitor serial. Esto quiere decir que el programa debe esperar a que se escriba un dato para ejecutar alguna acción, de caso contrario solo se iniciará el programa.
- Para limitar la acción se crea una condición donde cada caracter creado anteriormente corresponda a una letra, y se añade la parte del *void loop* de uno de los códigos inicialmente elaborados.
- El proceso anterior se repite para todas las acciones requeridas, únicamente modificando la letra que va a ser la encargada de activar la misma.
- Cabe recalcar que en este punto ya no es necesario incluir un tiempo para la repetición del ciclo, porque únicamente dependerá de las veces que se escriba la variable de cada acción para su ejecución. Es decir, si se escribe 10 veces la letra definida el programa proporcionará 10 respuestas.

2.6.2 Software LabVIEW

La figura 2.13 detalla brevemente el proceso para la elaboración de la interfaz en LabVIEW, la misma que debe tener las capacidades de enlazarse con el programa realizado previamente en Arduino IDE.

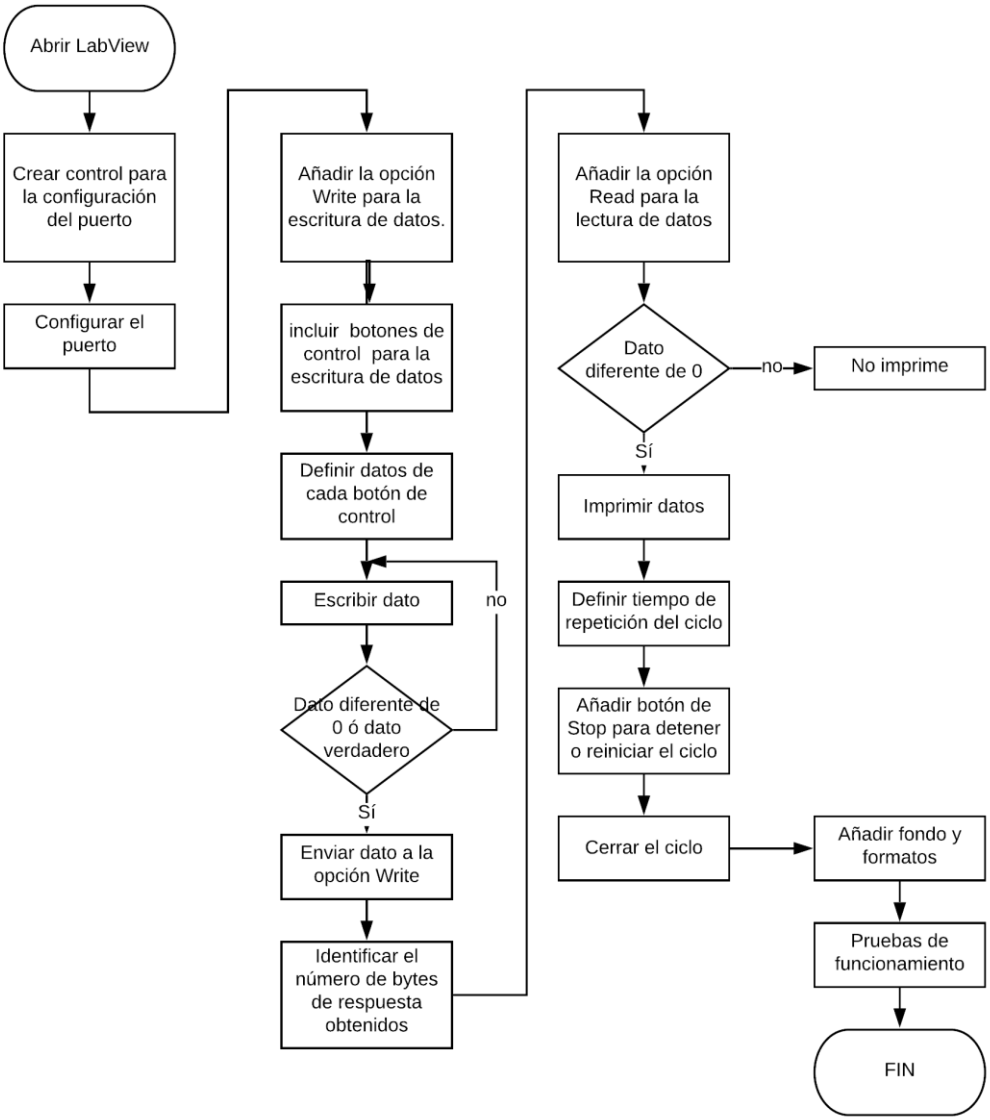


Figura 2.12 Diagrama de flujo de interfaz en LabVIEW

2.6.2.1 Diagrama de bloques

Al ser necesario la comunicación con el Arduino es indispensable realizar la programación utilizando los elementos de la opción Serial.

- Para empezar se da clic derecho en la ventana de diagrama de bloques y se ancla la paleta de funciones, seguidamente en la opción "*Instrument I/O*" se selecciona el recuadro que dice "Serial" y también se lo ancla a ventana principal para tener los elementos disponibles cuando sean requeridos.

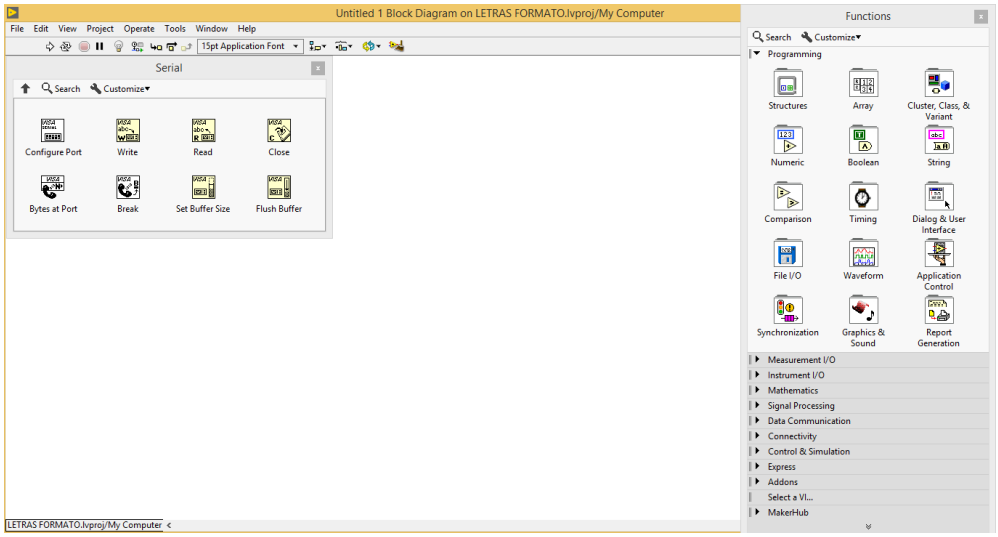


Figura 2.13 Paneles de función y serial anclados

Configuración del puerto

- Se arrastra el icono de configuración del puerto y se crea un botón de control en la entrada llamada "*VISA resourse name*" para poder realizar la configuración desde el panel frontal.

Después de la configuración se crea una estructura "*while loop*" que va a ser la encargada de realizar las acciones de una manera cíclica dependiendo de un tiempo que se establecerá más adelante. Para esto se da clic en el icono "*structures*" donde se escoge "*while loop*". Dentro de esta estructura debe ir todo lo que se requiere repetir cíclicamente, es decir, fuera de esta deben encontrarse únicamente la configuración del puerto y el cierre de la comunicación serial.

Escritura de constantes

- Añadir la opción "*Write*" la cual permite hacer la escritura en el monitor serial de Arduino para controlar las funciones preestablecidas anteriormente.
- Para evitar la escritura de manera manual se añade botones para crear constantes que simulen la escritura de datos. Para esto en la ventana de panel frontal se añade botones tipo "ON/OFF". En la paleta de control, dentro de la opción "*Boolean*" se selecciona "*Ok button*" y se añaden tantos como acciones a realizarse.

Simultáneamente aparecerán los mismos botones en el diagrama de bloques para su configuración.

- Insertar un "*Array*" para la configuración de cada botón de control. En la paleta de funciones en "*Array*" se selecciona "*Build Array*". Una vez añadido se lo extiende pasta tener uno por cada botón de control.
- Al ser necesario convertir el "*Array*" a un número se añade un transformador. En la paleta de funciones en "*Boolean*" se selecciona "*Array to number*".
- Se inserta una estructura de caso, la cual determina si los datos de escritura son correctos o no. En la paleta de funciones, opción "*structures*" se elige "case *structure*". Dentro de este se añade una constante de tipo "*String*" que serán las variables que se establecieron en la programación del Arduino.
- En la parte superior del "case estructure" se define cada uno de los casos, es decir, cada caso (cada botón) envíe una variable al bucle de escritura de manera automática.

Número de bytes entrantes

- Seguido del buffer de escritura, es importante reconocer cuantos bytes se están recibiendo, para esto en la paleta de comunicación serial se selecciona la opción "*bytes at port*" y se la conecta. A su vez esta herramienta sirve para detener la lectura en el caso que no se reciban datos. Por este motivo se añade un código de comparación que se encuentra en la paleta de funciones dentro de "*comparison*" la opción "*Not Equal to* 0?".

Lectura

- Se añade otro "case estructure" quien determinará si se realiza o no la lectura, y por ende la impresión de los datos.
- Seguido de "*Bytes at Port*" se añade la opción de lectura "*Write*" que se encuentra dentro de la paleta de comunicación serial. A esta es indispensable crear un indicador

el cual va a imprimir los datos en un recuadro ubicado en el panel frontal. Este indicador debe ir dentro del último "*case structure*".

Tiempo

- Se incluye el tiempo en el que se repetirá el ciclo en la opción "*Timing*" adjuntamos un "*Wait*" el mismo que permite ingresar el tiempo en milisegundos.

Botón de reinicio

- Por otro lado también se debe adjuntar un botón de "*Stop*" que servirá para parar o reiniciar el programa en caso de algún error en el transcurso del mismo.

Cierre

- Por último, fuera del ciclo *While* se cierra la comunicación serial con la opción "*Close*".

Hecho esto se puede realizar las pruebas respectivas.

2.6.2.2 Panel frontal

- Es necesario organizar todos los controles, indicadores y botones que se encuentran en esta sección.
- En cuanto a los botones, se requiere realizar la configuración de su acción mecánica dependiendo del modo de funcionamiento que se requiera.
- Se añade el fondo y los formatos que se desee para la presentación final.

Para terminar se exporta el programa en una aplicación .exe para poder ejecutarla sin necesidad de tener instalado el software LabVIEW.

2.7 PRUEBAS DE FUNCIONAMIENTO DE LA INTERFAZ.

Al haber elaborado códigos individuales para cada modo y para cada PID es necesario realizar pruebas de funcionamiento a cada uno de ellos por separado y únicamente dentro del monitor serial del Arduino IDE. Por otro lado al tener ya un código general, de la misma manera se realizó las pruebas correspondientes, esta vez enviando las diferentes variables que ejecuten las acciones programadas.

Al verificar el correcto funcionamiento en el software Arduino IDE, se puede pasar a realizar las pruebas enlazando el LabVIEW con el monitor serial del Arduino. Para esto es necesario cerrar completamente el Arduino IDE, ya que, no es posible tener abierto un puerto COM en dos programas a la vez.

CAPÍTULO III

3. RESULTADOS Y DISCUSIONES

El presente capítulo detalla todos los códigos hexadecimales extraídos en los procesos descritos anteriormente correspondientes a los datos que transitan por la red CAN del vehículo. Además, se muestran los *headers* tanto de lectura como de escritura que son utilizados para la comunicación con la ECU. Por último se describe la interfaz elaborada en cada uno de los programas utilizados, para posteriormente mostrar el resultado final de dicha programación.

1.1 CONEXIÓN DE LOS DISPOSITIVOS

1.1.1 Conexión y configuración del dispositivo ELM327

- Se conecta el socket macho del *splitter* al OBD-II del vehículo.
- Por un lado se conecta el socket macho del ELM327 al uno de los sockets hembra del *splitter*, mientras que por el otro extremo se conecta el cable USB al puerto de la computadora.
- Para la configuración y el uso de este dispositivo se lo hizo con el *hyperterminal* Hércules, ya que nos ofrece la facilidad de enviar y recibir datos de una manera simple y rápida.
- Se abre la aplicación y se muestra la página principal como muestra la figura 3.1.

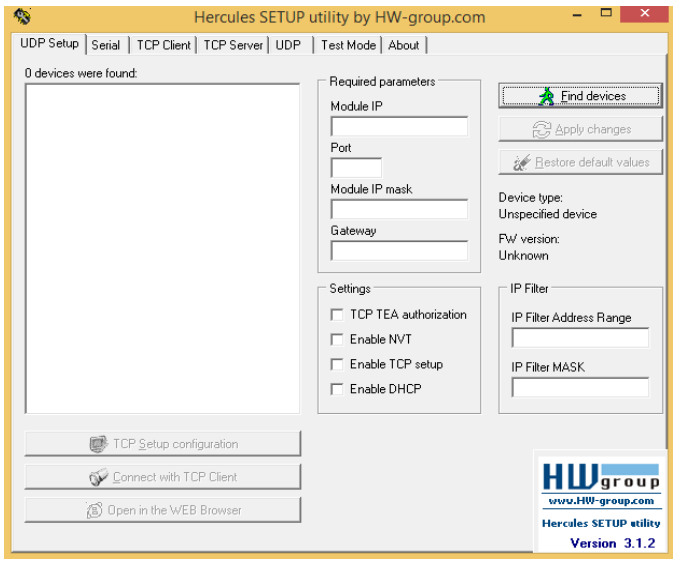


Figura 3.1 Página principal del *hyperterminal* Hércules.

En la barra de tareas se selecciona la opción serial donde en el lado derecho de la ventana se tiene las opciones de configuración en las que se modifica tanto el puerto que está usando el ELM327 y la velocidad a la que se quiera trabajar como muestra la figura 3.2. Las demás opciones se las puede mantener en los valores predeterminados a menos que se requiera algún cambio.

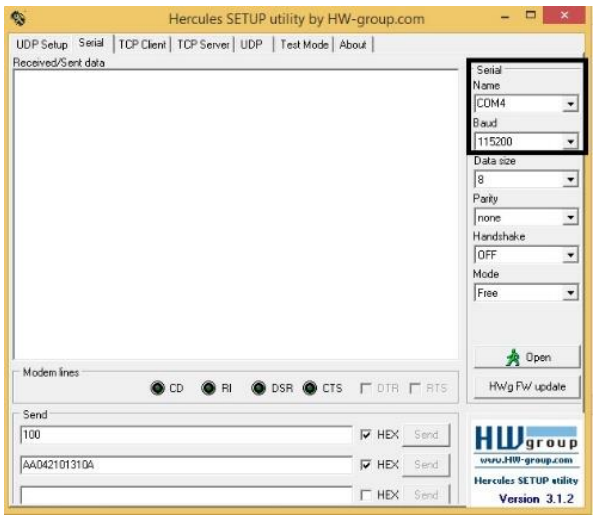


Figura 3.2 Configuración del puerto y velocidad del ELM327

- Por último se selecciona en el botón de open y si toda la conexión y configuración está correcta saldrá un mensaje en la pantalla de color verde indicando que el puerto seleccionado está abierto y está listo para usar, caso contrario saldrá un mensaje en color rojo diciendo "*Serial port COM4 openning error*" que indica un error al abrir el puerto serial seleccionado.

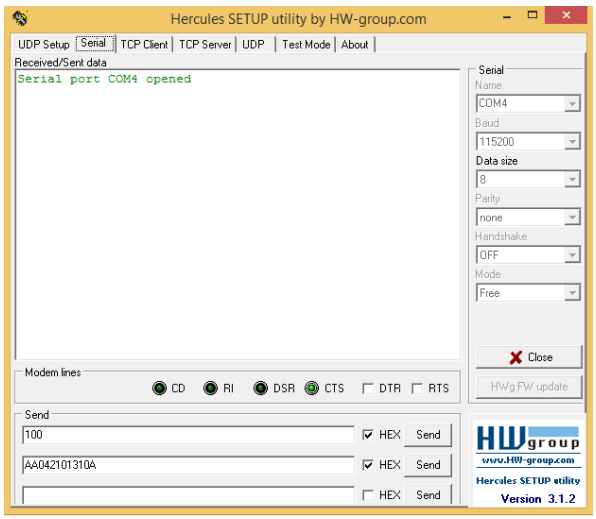


Figura 3.3 Hyperterminal Hércules y dispositivo ELM327 configurados.

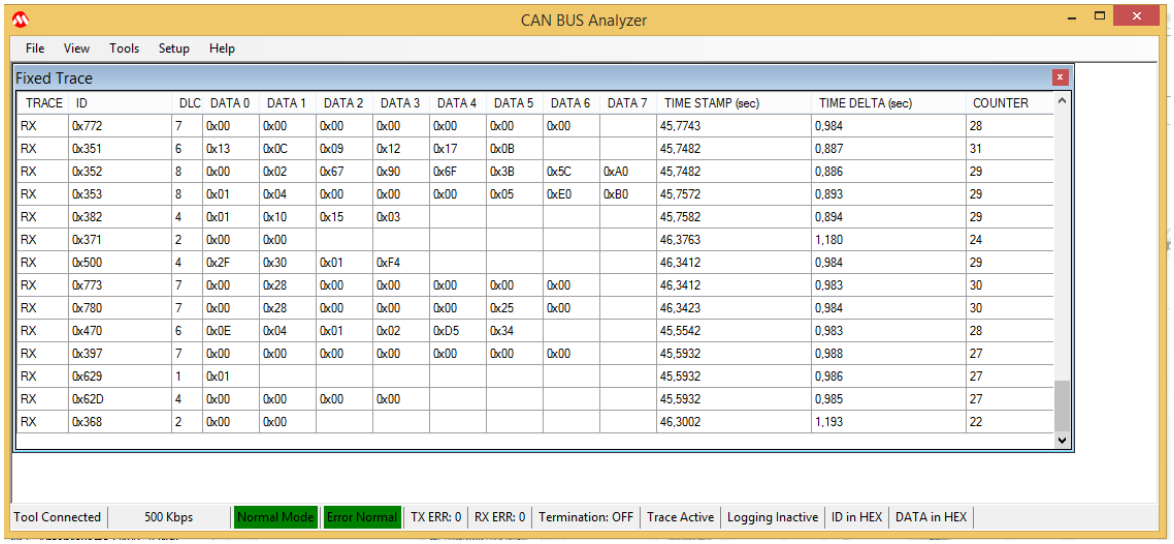
1.1.2 Conexión y configuración del dispositivo CAN bus Analyzer.

- En base a la figura 2.3 se identifica el modo de conexión entre el conector DB9 del microchip y el conector OBD-II. Según el diagrama que se presenta es necesario realizar un cable para conectar los pines correspondientes a la tierra de señal, CAN High y CAN Low. Una vez hecho el cable conector se procede a conectar tanto el DB9 al microchip como el socket OBD-II macho al hembra restante del *splitter*.

- Por otro lado se conecta por medio del cable USB la computadora con el microchip.

- Para el uso de este dispositivo se usa su propio programa que lleva el mismo nombre, este software nos permite visualizar los datos que se están transmitiendo por la red CAN del vehículo.

- Al abrir el programa primero se debe verificar que en la parte inferior de la ventana principal aparezcan dos cuadros de color verde indicando que la conexión es correcta. Seguidamente en la barra de tareas se da clic en la opción *tools* y se selecciona la opción *fixed trace*. En este punto se deben visualizar los datos de la red CAN que están transitando en ese momento.



TRACE	ID	DLC	DATA 0	DATA 1	DATA 2	DATA 3	DATA 4	DATA 5	DATA 6	DATA 7	TIME STAMP (sec)	TIME DELTA (sec)	COUNTER
RX	0x772	7	0x00	0x00	0x00	0x00	0x00	0x00	0x00		45,7743	0,984	28
RX	0x351	6	0x13	0x0C	0x09	0x12	0x17	0x0B			45,7482	0,887	31
RX	0x352	8	0x00	0x02	0x67	0x90	0x6F	0x3B	0x5C	0xA0	45,7482	0,886	29
RX	0x353	8	0x01	0x04	0x00	0x00	0x00	0x05	0xE0	0xB0	45,7572	0,893	29
RX	0x382	4	0x01	0x10	0x15	0x03					45,7582	0,894	29
RX	0x371	2	0x00	0x00							46,3763	1,180	24
RX	0x500	4	0x2F	0x30	0x01	0xF4					46,3412	0,984	29
RX	0x773	7	0x00	0x28	0x00	0x00	0x00	0x00	0x00		46,3412	0,983	30
RX	0x780	7	0x00	0x28	0x00	0x00	0x00	0x25	0x00		46,3423	0,984	30
RX	0x470	6	0x0E	0x04	0x01	0x02	0xD5	0x34			45,5542	0,983	28
RX	0x397	7	0x00	0x00	0x00	0x00	0x00	0x00	0x00		45,5932	0,988	27
RX	0x629	1	0x01								45,5932	0,986	27
RX	0x62D	4	0x00	0x00	0x00	0x00					45,5932	0,985	27
RX	0x368	2	0x00	0x00							46,3002	1,193	22

Figura 3.4 Dispositivo y Software CAN Bus Analyzer configurados.

Una vez realizadas las conexiones como muestra la figura 3.5 y los programas estén funcionando sin errores se puede empezar la detección de los ID's o *headers* en los cuales se realiza la escritura de los datos de petición de información y en los que se leen las

respuestas proporcionadas de la ECU del vehículo. Para la detección de los *headers* se lo hace en base a la tabla 1.5 correspondiente a los modos de diagnóstico, y a los PID's propios de las variables descritas en el alcance del proyecto.

Figura 3.5 Conexión del ELM 327 y el CAN BUS Analyzer

1.2 RESULTADOS DE PRUEBAS DE CÓDIGOS ADQUIRIDOS

En esta sección se analiza los códigos obtenidos después de realizar la infiltración a la red CAN, presentando los datos obtenidos y haciendo énfasis únicamente en los datos que sirven para la programación de la interfaz y el muestreo de datos. Además se realiza un análisis de cada uno de los datos para conocer su significado

1.2.1 Códigos de datos en tiempo real (Modo 1 de diagnóstico).

Para el Modo 1 de diagnóstico se utiliza códigos PID's, los cuales son los encargados de solicitar información en tiempo real a la ECU de los componentes conectados a la red de comunicación.

La tabla 3.1 muestra todos los datos acerca de los PID's utilizados como: el número de bytes de respuesta, los valores máximos y mínimos que se deben obtener, las fórmulas que se

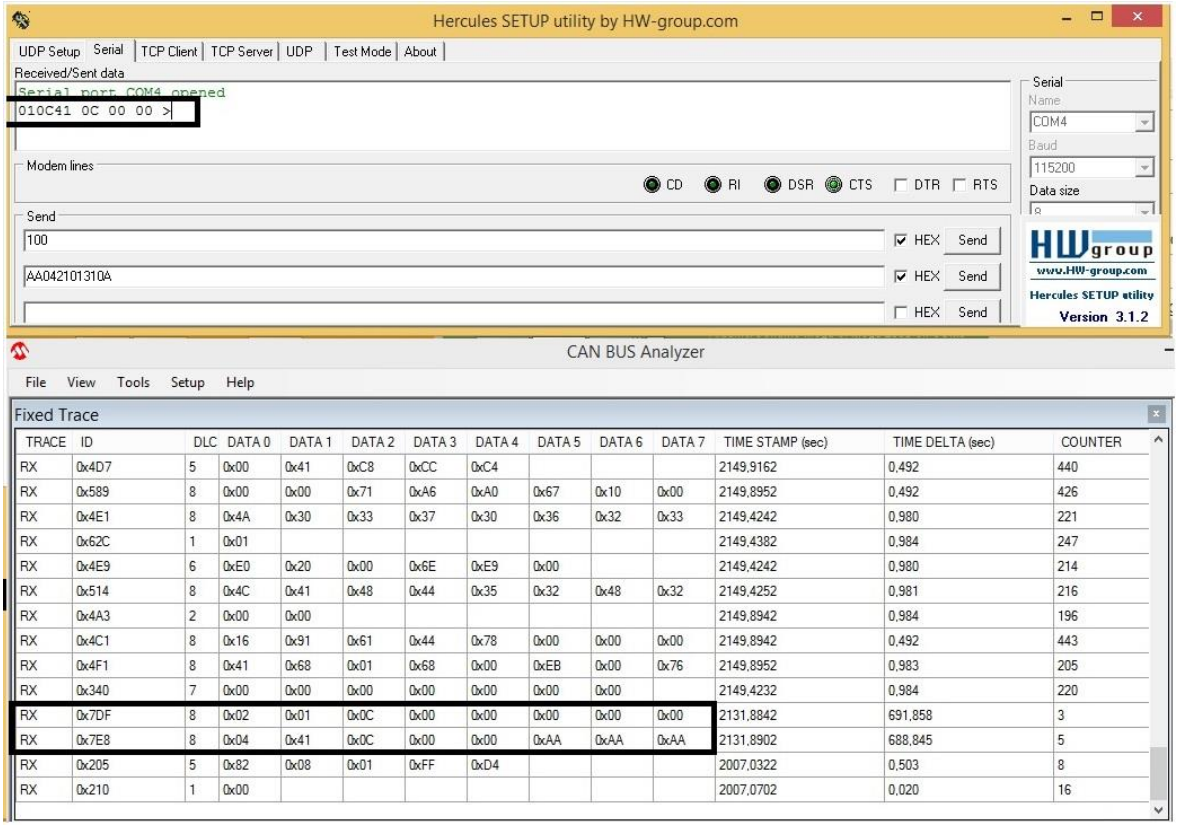
deben emplear para obtener los datos netos para poder mostrarlos de una manera entendible con su respectiva unidad de medida. Cabe recalcar que los datos obtenidos serán de modo hexadecimal los cuales se deben transformar a modo decimal para poder insertarlos en las fórmulas y realizar los respectivos cálculos. Para identificar cual es el dato A y cual el B basta con identificar el orden en el mensaje, es decir el dato A es el primero que se obtiene de respuesta y el B es el segundo. En este caso se utilizaron los siguientes PID's:

Tabla 3.1 PID's utilizados

PID's	**Bytes de respuesta**	**Descripción**	**Valor mínimo**	**Valor máximo**	**Unidad**	**Fórmula**
OC	2	RPM del motor	0	16383.75	RPM's	(256A+B)/4
14	2	Voltaje del sensor de oxígeno y	0	1275	Volts	A/200
		Ajuste de combustible a corto plazo	-100	99.2	%	B/1.28-100
0B	1	Presión absoluta en el colector de admisión	0	255	kPa	A

Para la obtención de datos por medio del software Hércules se procede a escribir los códigos correspondientes, donde primero se escribe el modo en el cual se quiere trabajar y seguidamente se escribe el PID que se quiera analizar. Instantáneamente se obtendrá una respuesta que incluirá datos hexadecimales que son propiamente los datos proporcionados por la ECU al PID que se envió. La figura 3.6 muestra un ejemplo de la escritura de datos en el *hyperterminal* Hércules, el modo utilizado es el 01 mientras que el PID correspondiente a RPM es el 0C, por consiguiente, el mensaje escrito para esta función es (010C). Hecho esto, se analizó en el software CAN BUS Analyzer si aparecieron nuevos *headers* que dentro de ellos contengan la misma información que se envió previamente desde el otro programa. Identificados los *headers* se puede reconocer cual corresponde a la escritura y cual a la lectura de datos, además se puede observar el mensaje completo que es necesario enviar en el momento que se realice la programación para obtener una respuesta adecuada. Así mismo,

se analiza el mensaje de respuesta para escoger únicamente los datos necesarios para obtener los valores netos que se solicitaron previamente con el PID.



TRACE	ID	DLC	DATA 0	DATA 1	DATA 2	DATA 3	DATA 4	DATA 5	DATA 6	DATA 7	TIME STAMP (sec)	TIME DELTA (sec)	COUNTER
RX	0x4D7	5	0x00	0x41	0xC8	0xCC	0xC4				2149,9162	0,492	440
RX	0x589	8	0x00	0x00	0x71	0xA6	0xA0	0x67	0x10	0x00	2149,8952	0,492	426
RX	0x4E1	8	0x4A	0x30	0x33	0x37	0x30	0x36	0x32	0x33	2149,4242	0,980	221
RX	0x62C	1	0x01								2149,4382	0,984	247
RX	0x4E9	6	0xE0	0x20	0x00	0x6E	0xE9	0x00			2149,4242	0,980	214
RX	0x514	8	0x4C	0x41	0x48	0x44	0x35	0x32	0x48	0x32	2149,4252	0,981	216
RX	0x4A3	2	0x00	0x00							2149,8942	0,984	196
RX	0x4C1	8	0x16	0x91	0x61	0x44	0x78	0x00	0x00	0x00	2149,8942	0,492	443
RX	0x4F1	8	0x41	0x68	0x01	0x68	0x00	0xEB	0x00	0x76	2149,8952	0,983	205
RX	0x340	7	0x00	0x00	0x00	0x00	0x00	0x00	0x00		2149,4232	0,984	220
RX	0x7DF	8	0x02	0x01	0x0C	0x00	0x00	0x00	0x00	0x00	2131,8842	691,858	3
RX	0x7E8	8	0x04	0x41	0x0C	0x00	0x00	0xAA	0xAA	0xAA	2131,8902	688,845	5
RX	0x205	5	0x82	0x08	0x01	0xFF	0xD4				2007,0322	0,503	8
RX	0x210	1	0x00								2007,0702	0,020	16

Figura 3.6 Comparación de datos entre Software Hércules y CAN Bus Analyzer

Como se puede observar en la ventana del CAN Bus Analyzer se tienen dos líneas que contienen los mismos datos que fueron enviados y recibidos desde el Hércules. En base a esto se puede determinar los *headers* o ID's de escritura y lectura. Para la escritura se puede identificar que el ID usado es el 0x7DF mientras que para la lectura o respuesta se utiliza el ID 0x7E8.

1.2.1.1 Análisis de los códigos adquiridos

La tabla 3.2 presenta el análisis de cada uno de los datos que están contenidos dentro de los mensajes de escritura y de lectura obtenidos.

Tabla 3.2 Análisis de mensajes de escritura y lectura en la red CAN

Datos de escritura			Datos de lectura		
ID	0x7DF	*Header* de escritura de datos	ID	0x7E8	*Header* de lectura de datos
DLC	8	Longitud total del mensaje	DLC	8	Longitud total del mensaje
DATA 0	02	Número adicional de bytes utilizados	DATA 0	04	Número adicional de bytes utilizados
DATA 1	01	Modo utilizado	DATA 1	41	Respuesta al modo utilizado
DATA 2	0C	PID	DATA 2	0C	Respuesta al PID
			DATA 3	00	Mensajes hexadecimales de respuesta.
			DATA 4	00	

Los mensajes tanto de escritura como de lectura tienen una longitud total de 8 bytes como se indica en la figura 3.6, pero únicamente los descritos en la tabla 3.2 son los útiles ya que se puede decir que los demás funcionan como relleno para completar el mensaje.

Para este modo de diagnóstico los mensajes son similares únicamente cambia el PID y la longitud del mensaje de respuesta dependiendo de los bytes adicionales.

PID 0C (RPM)

Como se explicó anteriormente al estar trabajando con el modo 01 de diagnóstico es necesario añadir el PID, en este caso 0C. Este PID proporciona dos bytes adicionales de respuesta.

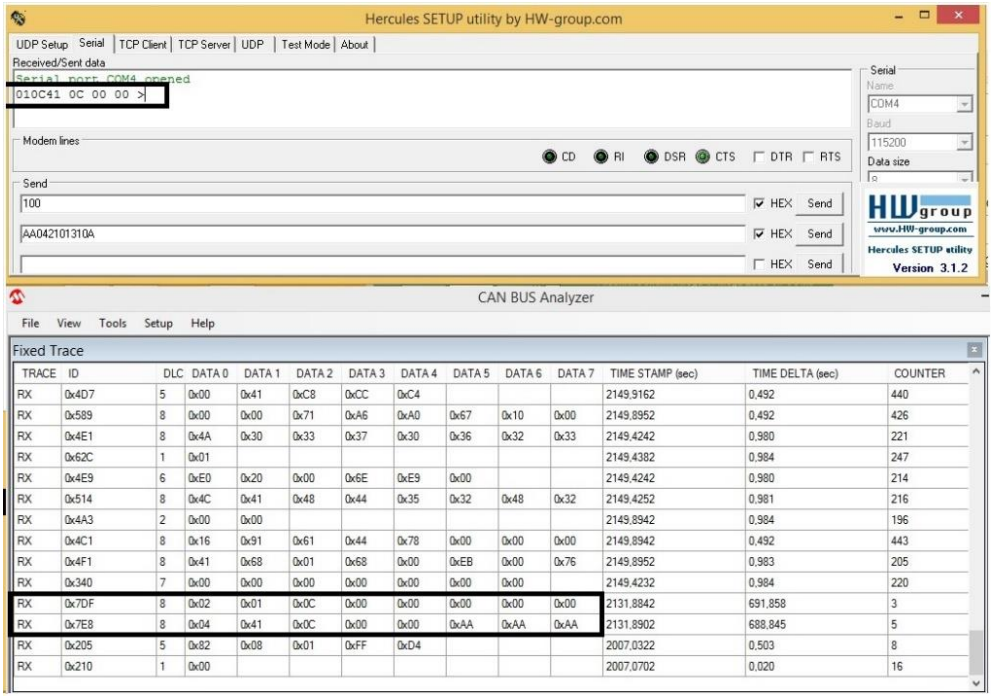


Figura 3. 7 Datos obtenidos del PID 0C.

PID 14 (Voltaje del sensor de oxígeno y ajuste de combustible a corto plazo).

Para este PID el código enviado es (0114), este también proporciona dos bytes adicionales de respuesta, de los cuales uno será el dato para el voltaje del sensor de oxígeno y el otro para el ajuste de combustible a corto plazo.

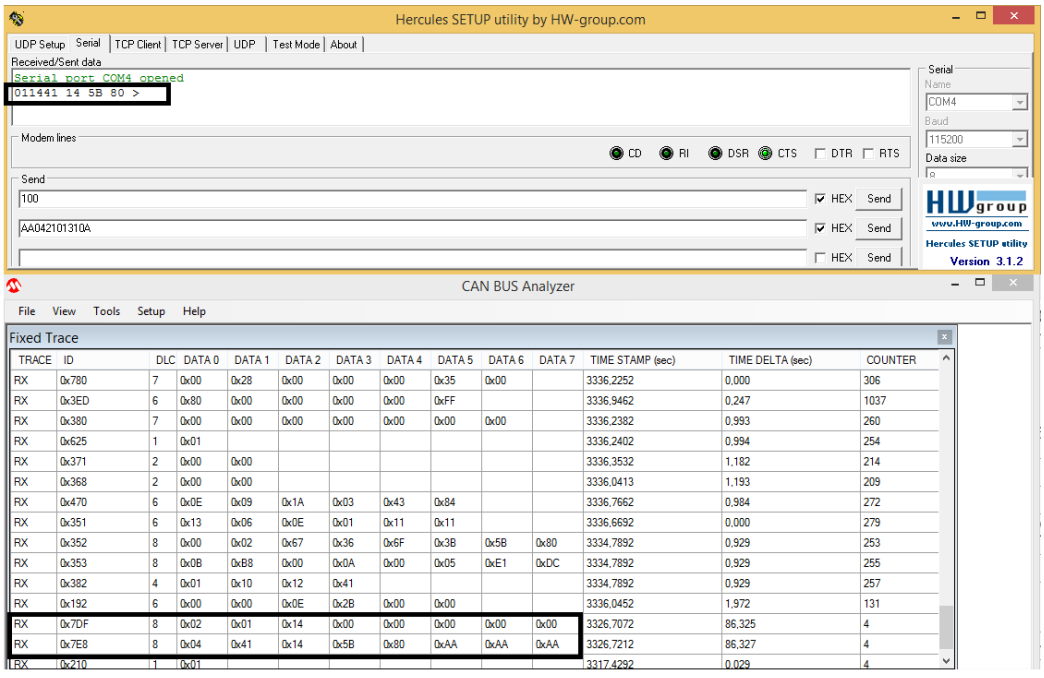
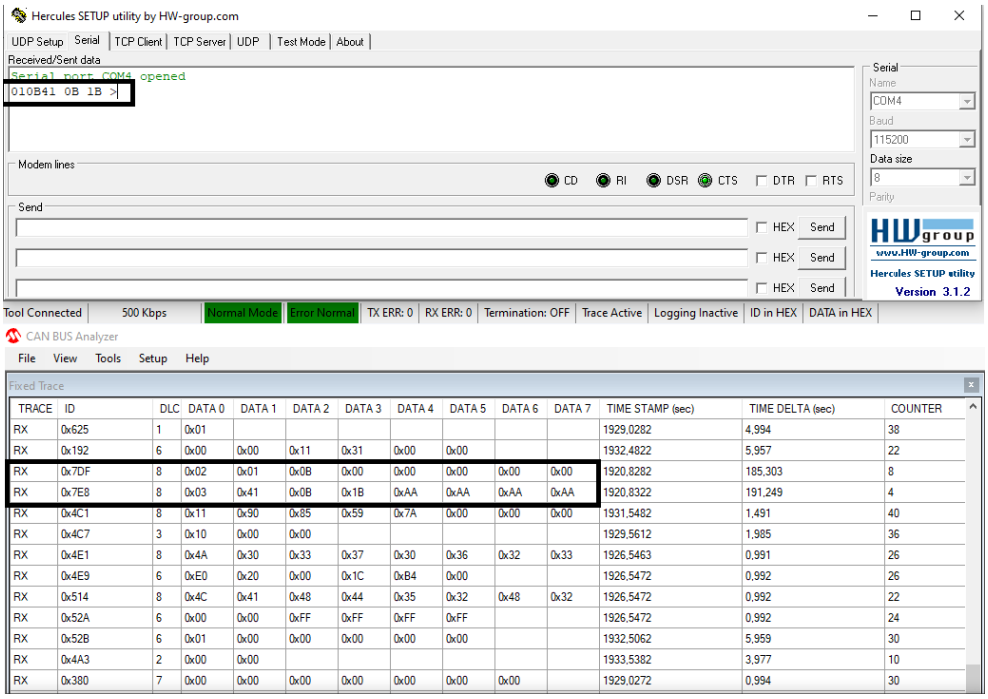


Figura 3.8 Datos obtenidos del PID 14.

PID 0B (Presión absoluta en el colector de admisión)

Para este PID se envía el código (010B), a diferencia de los anteriores PID's este proporciona un solo byte adicional de respuesta, es por esto que la longitud del mensaje también es menor a los anteriores.



TRACE	ID	DLC	DATA 0	DATA 1	DATA 2	DATA 3	DATA 4	DATA 5	DATA 6	DATA 7	TIME STAMP (sec)	TIME DELTA (sec)	COUNTER
RX	0x625	1	0x01								1929,0282	4,994	38
RX	0x192	6	0x00	0x00	0x11	0x31	0x00	0x00			1932,4822	5,957	22
RX	0x7DF	8	0x02	0x01	0x0B	0x00	0x00	0x00	0x00	0x00	1920,8282	185,303	8
RX	0x7E8	8	0x03	0x41	0x0B	0x1B	0xAA	0xAA	0xAA	0xAA	1920,8322	191,249	4
RX	0x4C1	8	0x11	0x90	0x85	0x59	0x7A	0x00	0x00	0x00	1931,5482	1,491	40
RX	0x4C7	3	0x10	0x00	0x00						1929,5612	1,985	36
RX	0x4E1	8	0x4A	0x30	0x33	0x37	0x30	0x36	0x32	0x33	1926,5463	0,991	26
RX	0x4E9	6	0xE0	0x20	0x00	0x1C	0xB4	0x00			1926,5472	0,992	26
RX	0x514	8	0x4C	0x41	0x48	0x44	0x35	0x32	0x48	0x32	1926,5472	0,992	22
RX	0x52A	6	0x00	0x00	0xFF	0xFF	0xFF	0xFF			1926,5472	0,992	24
RX	0x52B	6	0x01	0x00	0x00	0x00	0x00	0x00			1932,5062	5,959	30
RX	0x4A3	2	0x00	0x00							1933,5382	3,977	10
RX	0x380	7	0x00	0x00	0x00	0x00	0x00	0x00	0x00		1929,0272	0,994	30

Figura 3.9 Datos obtenidos del PID 0B.

La tabla 3.3 muestra los datos correspondiente a cada PID, determinando el significado de cada uno de ellos.

Tabla 3.3 Extracción de datos del modo 1 de diagnóstico

PID	ID	DLC	DATA0	DATA1	DATA2	DATA3	DATA4	DATA5	DATA6	DATA7
	Header	Longitud del mensaje	Número adicional de bytes	Modo	PID	Respuesta A	Respuesta B			
OC	7DF	8	02	01	0C					
	7E8	8	02	41	0C	00	00			
14	7DF	8	02	01	14					
	7E8	8	02	41	14	5B	80			
0B	7DF	8	02	01	0B					
	7E8	8	02	41	0B	00				

Las líneas de azul muestran los mensajes de petición de información, mientras que las líneas de verde muestran las respuestas, estos datos serán constantes, los de color amarillo son las respuestas A y B, las mismas que varían dependiendo de las condiciones a las que se encuentre el vehículo. Los espacios en blanco son rellenados automáticamente por la ECU, puesto que estos datos no interfieren en nada tanto para la escritura como para la lectura.

1.2.2 Datos de lectura de códigos de falla (Modo 3 de diagnóstico).

En este punto fue necesario provocar fallas en el vehículo desconectando varios sensores para que la ECU determine un error y por ende proporcione códigos de falla o DTC. Este proceso se lo realiza progresivamente, es decir, primero leyendo un solo DTC, luego dos y de esta manera hasta leer tantos códigos como sean necesarios. Este punto es indispensable para la lectura de DTC's ya que dependiendo de la cantidad de estos variará la longitud del mensaje. En lecturas de tres y más códigos de falla se puede determinar que ya no hay una sola línea de respuesta, sino que son dos líneas. Estas no se pueden visualizar de manera simultánea sino que es necesario enviar el código de petición desde los dos programas usados. En este caso se realizó lecturas hasta de 5 códigos de falla. Para la escritura en el programa Hércules, basta solo con escribir el modo de diagnóstico (3) que es el encargado de leer los DTC. De igual manera se realiza una comparación con el software CAN Bus Analyzer para determinar el mensaje completo de escritura y lectura para posteriormente hacer un análisis de cada uno de los datos contenidos dentro del mensaje.

1.2.2.1 Lectura de 1 y 2 códigos de falla

La lectura de 1 y 2 DTC's tienen cierta similitud, tomando en cuenta el espacio disponible en cada línea de datos que son 8 bytes, lo suficiente para contener los dos DTC's en la misma. La figura 3.2 y 3.3 muestran la lectura de uno y dos códigos de falla respectivamente, haciendo la misma comparación que se hizo con los datos del modo 1 de diagnóstico entre los dos software utilizados.

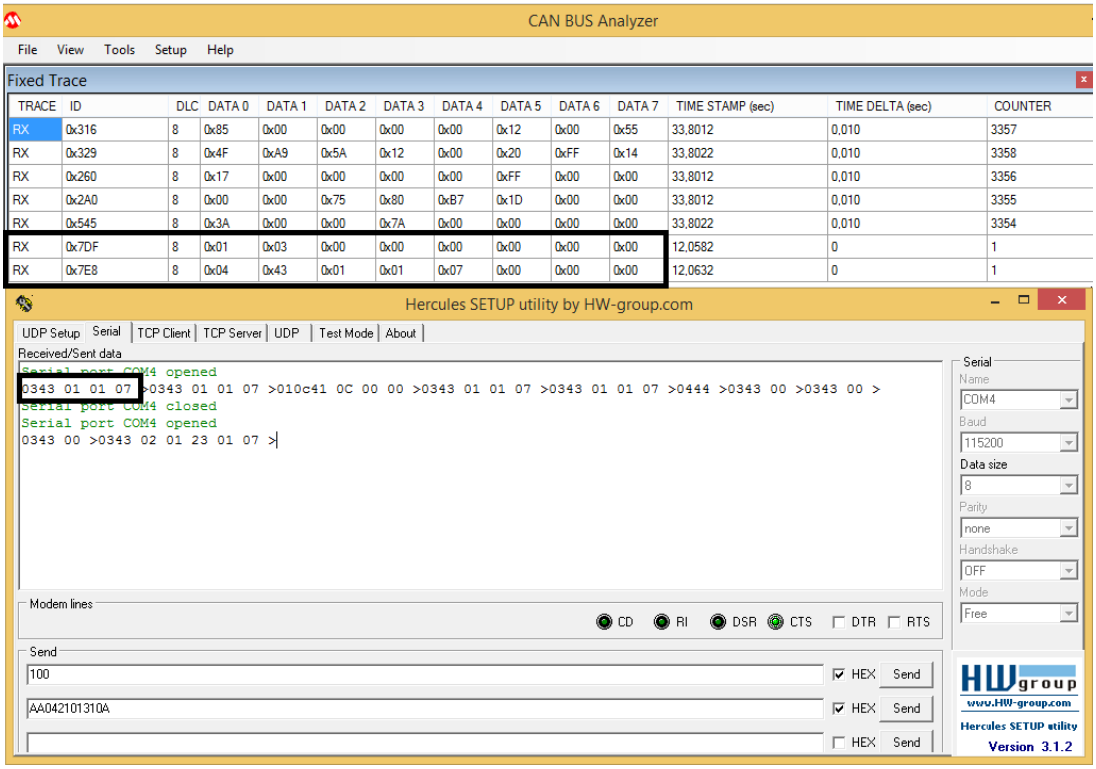


Figura 3.10 Lectura de 1 DTC

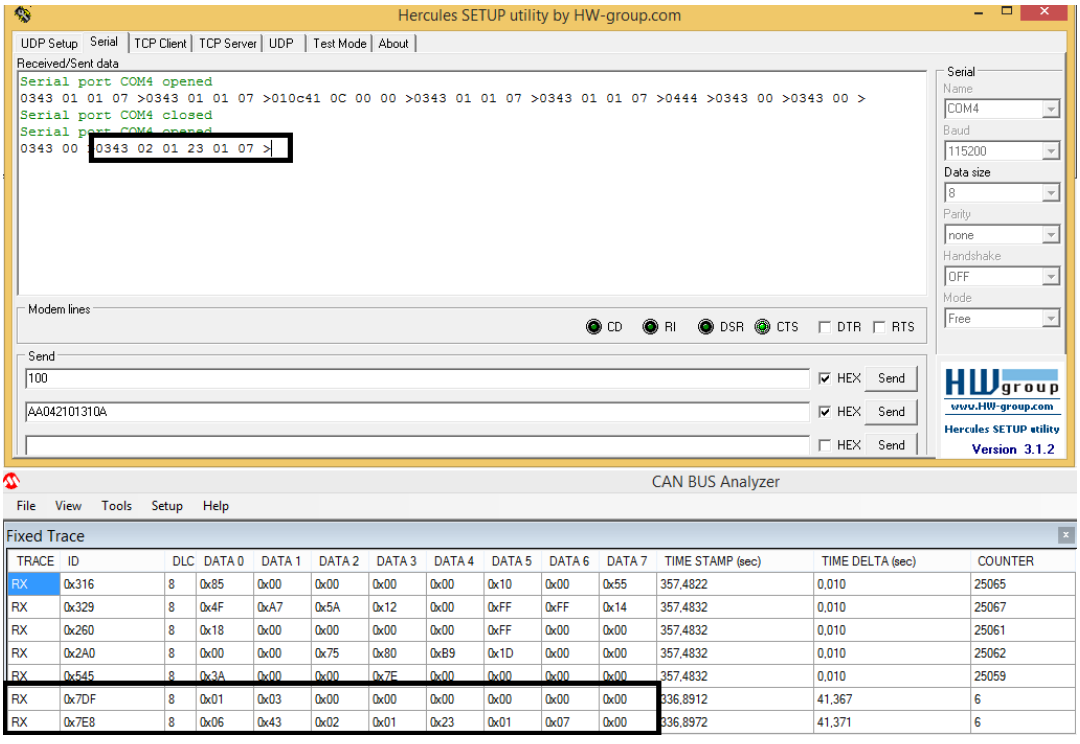


Figura 3.11 Lectura de 2 DTC's

La tabla 3.5 presenta la extracción de los datos correspondientes a las figuras 3.10 y 3.11 con su respectivo análisis de los códigos hexadecimales contenidos.

Tabla 3.4 Extracción de datos de 1 y 2 DTC's

#DTC	ID	DLC	DATA0	DATA1	DATA2	DATA3	DATA4	DATA5	DATA6	DATA7
	Header	Longitud del mensaje	Número adicional de bytes	Modo	Respuesta al número de códigos leidos	Código 1		Código 2		
1	7DF	8	01	03						
	7E8	8	04	43	01	01	07			
2	7DF	8	01	03						
	7E8	8	06	43	02	01	23	01	07	

Los datos marcados de color azul corresponden al código de escritura en la red CAN para solicitar los códigos de falla, como se puede observar es idéntico en ambos casos. Por otro lado los datos de color verde corresponden a la lectura, donde se puede apreciar que la única diferencia entre los dos casos es el número adicional de bytes. Los datos de color lila corresponden al número de códigos leídos y por último en color amarillo están los DTC, como se observa a cada código le corresponde dos datos hexadecimales.

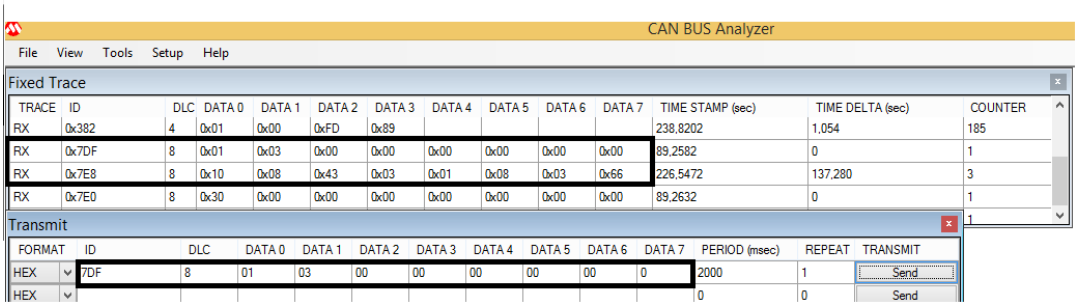
1.2.2.2 Lectura de 3, 4 y 5 códigos de falla

En lo que respecta a la lectura de 3, 4, y 5 DTC's existieron inconvenientes puesto que, dentro de la misma línea del *header* de lectura no hay suficientes espacios para contener todos los DTC's, es por esto que la ECU responde en dos líneas diferentes usando el mismo *header* pero con mensajes distintos. Al tener una respuesta de este tipo y cuya velocidad es demasiado rápida para poder leerla se optó por enviar los datos de petición primero por la opción *transmit* del CAN Bus Analyzer y luego desde el *Hyperterminal* Hércules para poder visualizar las dos líneas de respuesta en tiempo real, puesto que enviando desde un único software no se logró realizar dicha visualización.

Lectura de 3 códigos de falla

En la primera línea de respuesta se tienen los datos propios a: la longitud del mensaje, número adicional de bytes, réplica al modo de diagnóstico, número de DTC's encontrados y dos de ellos. Mientras que en la segunda línea se presenta el otro código de falla restante.

PRIMERA LÍNEA



SEGUNDA LÍNEA

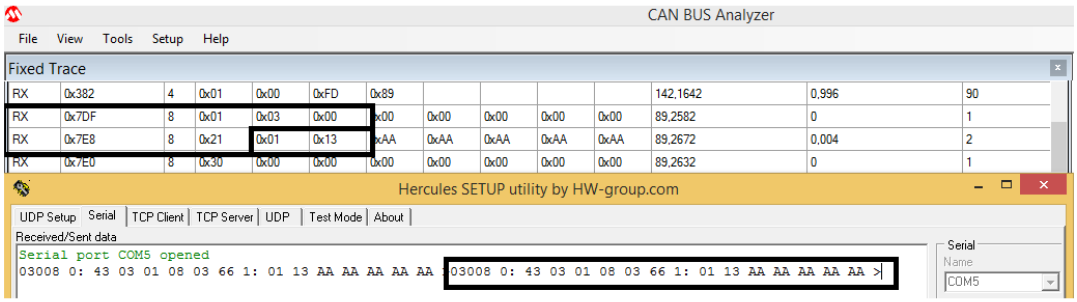
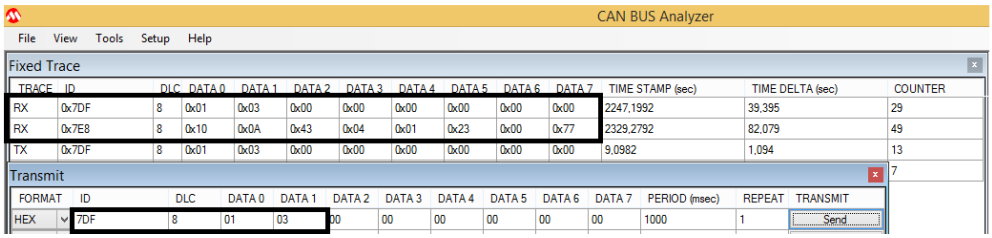


Figura 3.12 Lectura de 3 DTC's

Lectura de 4 códigos de falla

La primera línea es similar a la de 3 DTC, con la diferencia que cambia el valor del número adicional de bytes. Por otro lado en la segunda línea de respuesta se tienen dos códigos de falla más.

PRIMERA LÍNEA



SEGUNDA LÍNEA

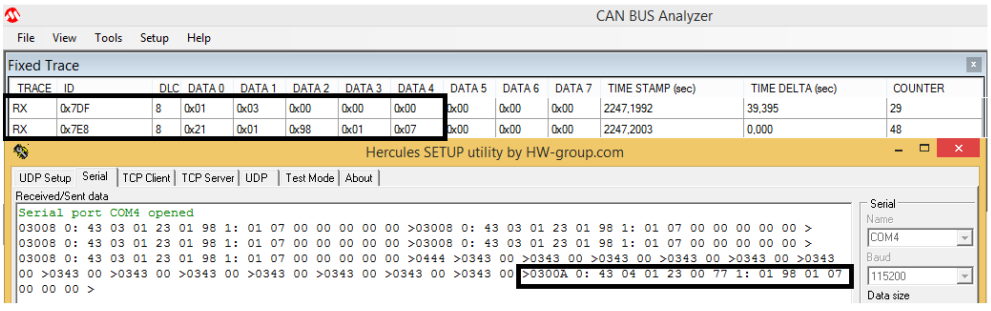


Figura 3.13 Lectura de 4 DTC's

Lectura de 5 códigos de falla

Al igual que las anteriores la primera línea es similar, cambiando el mismo valor del número adicional de bytes. Mientras que en la segunda línea casi llega a llenarse con los tres códigos de falla restantes.

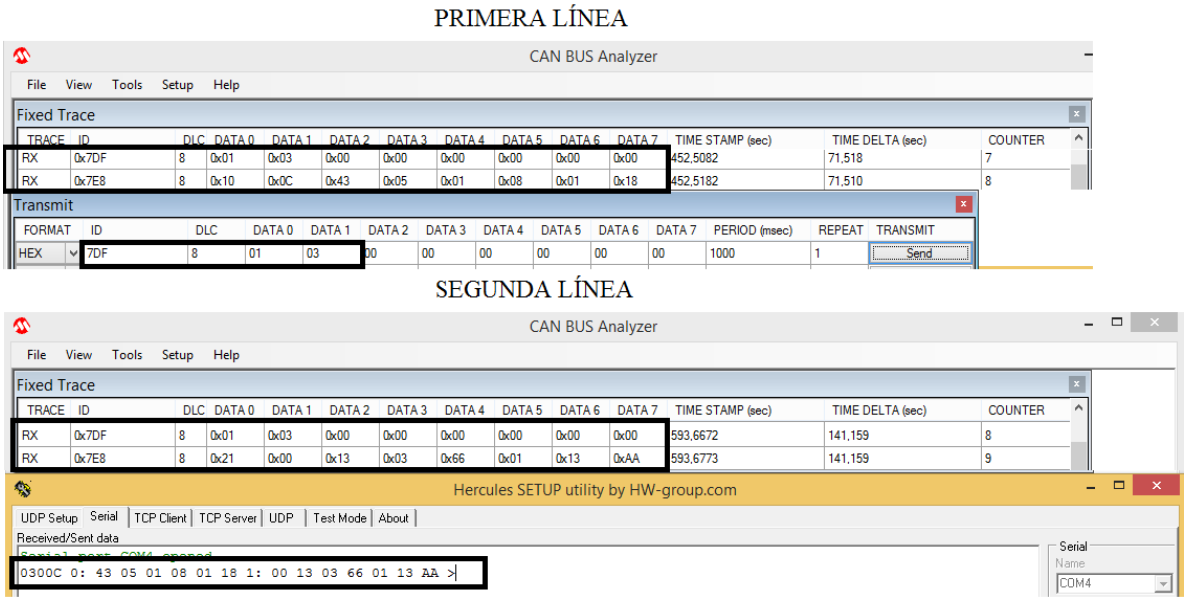


Figura 3.14 Lectura de 5 DTC's

En la tabla 3.6 se puede apreciar los datos de las dos líneas correspondientes a la lectura de los códigos de falla. El color azul se tiene los datos de escritura que son los mismos para la lectura de 1 y 2 DTC's. En color rojo se tiene la primera línea de lectura, donde se observa que apareció un dato adicional. Este dato indica que el mensaje total tiene 16 bytes (10 convertido a decimal es 16), es decir dos líneas de 8 bytes. Los siguientes valores en estas líneas al igual que en casos anteriores representan el número adicional de bytes y la respuesta al modo. Seguido están en color morado el número de DTC's leídos. Para continuar se tienen los datos en color amarillo los cuales al ser agrupados en pares representan los propios códigos de falla. En cuanto a las segundas líneas de lectura dispuestas en color verde, se tiene que el *header* y el DLC es el mismo que en las primeras líneas. Seguido a esto se tiene un valor que indica la continuación de la lectura de los datos.

Tabla 3.5 Extracción de datos de lectura de 3, 4 y 5 DTC's

		ID	DLC	DATA0	DATA1	DATA2	DATA3	DATA4	DATA5	DATA6	DATA7
3 DTC		7DF	8	01	03						
	1ra línea	7E8	8	10	08	43	03	01	08	03	66
	2da línea	7E8	8	21	01	13					
4 DTC		7DF	8	01	03						
	1ra línea	7E8	8	10	0A	43	04	01	23	00	77
	2da línea	7E8	8	21	01	98	01	07			
5 DTC		7DF	8	01	03						
	1ra línea	7E8	8	10	0C	43	05	01	08	01	18
	2da línea	7E8	8	21	00	13	03	66	01	13	

1.2.3 Datos de borrado de códigos de falla (Modo 4 de diagnóstico).

El modo 4 de diagnóstico es el encargado de borrar los códigos de falla existentes en el vehículo. Al igual que el modo 3 para escribir este código en la red CAN del vehículo únicamente se los hace escribiendo el modo 04 como se indica en la figura 3.4.

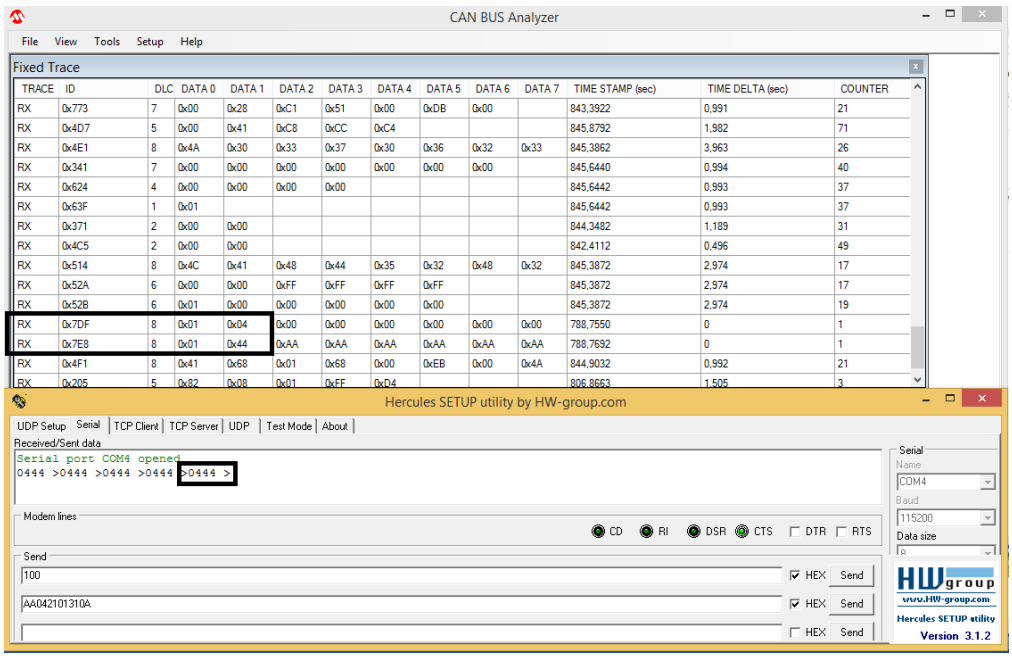


Figura 3.15 Borrado de códigos de falla.

La tabla 3.7 muestra la extracción de los datos correspondiente a este modo de diagnóstico, donde se puede apreciar que en el *header* de respuesta no se tienen datos específicos, sino que únicamente se presenta la respuesta al modo utilizado.

Tabla 3.6 Extracción de datos de borrado de códigos de falla

ID	DLC	DATA0	DATA1	DATA2	DATA3	DATA4	DATA5	DATA6	DATA7
Header	Longitud del mensaje	Número adicional de bytes	Modo						
7DF	8	01	04						
7E8	8	01	44						

Para corroborar que los DTC fueron borrados se volvió a enviar el modo 3 donde efectivamente se pudo apreciar que ya no se tenían códigos grabados además de poder visualizar que la luz MIL se apagó en el tablero del vehículo.

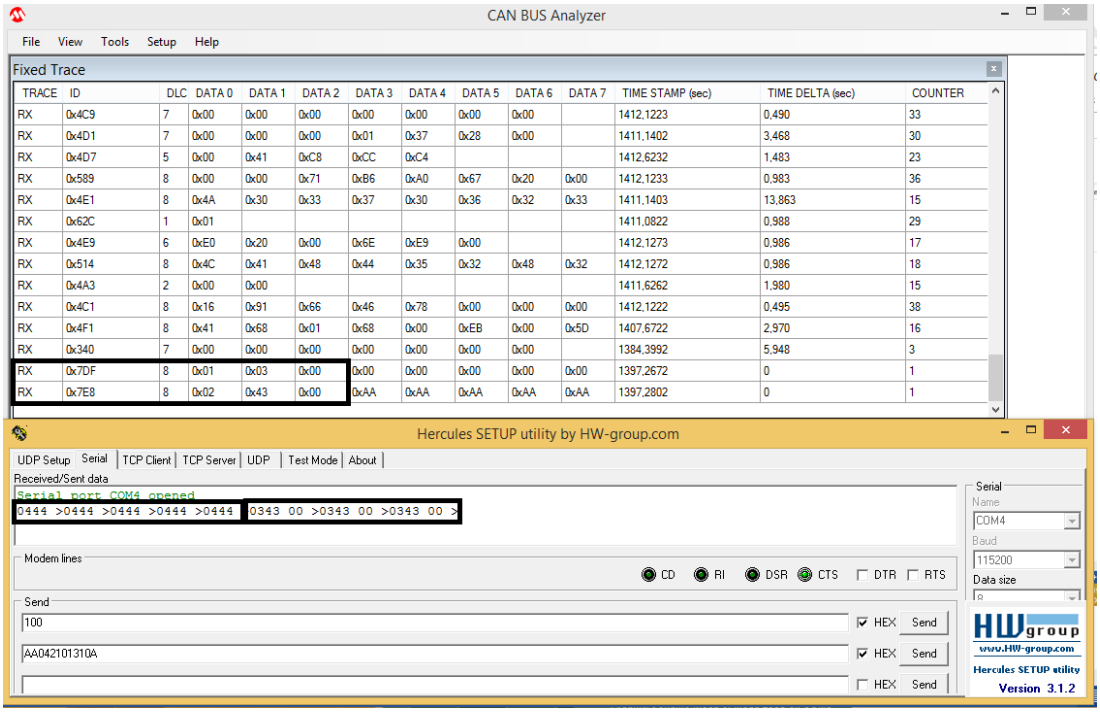


Figura 3.16 Comprobación de borrado de códigos de falla

1.3 RESULTADOS DE CÓDIGOS DE PROGRAMACIÓN

Luego de la obtención de los datos necesarios para enviar mensajes a la red CAN que soliciten información (modo 1 y 3), o de realizar una acción (modo 4) a la ECU, se realizó la programación tanto en Arduino IDE como en LabVIEW para que realice las mismas acciones de manera automática con solo presionar diferentes botones contenidos dentro de una interfaz.

1.3.1 Código de programación en Arduino IDE

Como se mencionó en el capítulo anterior, se realizaron códigos individuales para cada acción requerida, para después unirlos en un solo que los contenga a todos estos. En esta sección se presentará extractos del código final elaborado en este software. El código final se lo puede visualizar en el anexo I.

1.3.1.1 Definición de variables y librerías

La librería que se usó en la interfaz fue (CAN.h), esta proporcionó los elementos necesarios para cumplir con las funciones establecidas. Además en esta sección se especifica el uso de direcciones estándar.

```
#include <CAN.h>                          //incluir la librería usada

const bool useStandardAddressing = true;  //usar direcciones estándar
```

1.3.1.2 Void setup

Como se mencionó anteriormente este conjunto se ejecuta una sola vez y se mantiene así durante el funcionamiento del programa.

```
void setup() {
  Serial.begin(9600);                     //iniciar comunicación serial

  Serial.println("UTN SCANN");            // mensaje de confirmación de
inicio de comunicación serial


  if (!CAN.begin(500E3)) {                //iniciar el CAN bus

    Serial.println("INICIO FALLIDO");
    while (1);
  }
  if (useStandardAddressing) {
    CAN.filter(0x7e8);                    //definir header de lectura
  } else {
    CAN.filterExtended(0x18daf110);       //header de lectura extedido
  }
}
```

Como se puede observar en el extracto anterior se inicia la comunicación serial a una velocidad de 9600 braudios, si esta parte inicia correctamente el programa imprimirá el mensaje de confirmación. Por otro lado se inicia el CAN bus a una velocidad de 500kb por segundo, que es la velocidad de trabajo del CAN Bus Shield. De no lograr la iniciación del CAN Bus se imprime un mensaje indicando que el inicio falló. Además se define el *header* de lectura tanto en su dirección estándar como en su dirección extendida.

1.3.1.3 Void loop

En este bloque se encuentran las acciones que va a realizar el programa, todas ellas dirigidas por una letra que será la encargada de iniciar su ejecución. Para esto se creó caracteres a los que posteriormente se asignaron letras dependiendo de la acción a realizar.

Modo 1 de diagnóstico

En cuanto al modo 1 respecta, se tienen tres diferentes acciones a realizarse. Es por esto que se requiere una letra para cada PID usado en este modo. La letra A estará ligada al PID 14, correspondiente al voltaje del sensor de oxígeno y ajuste de combustible a corto plazo. La letra B dará paso al PID 0B, correspondiente a la presión absoluta en el colector de admisión. Por ultimo en este modo la letra C activa el PID 0C, de revoluciones del motor. Para los tres casos la escritura es similar ya que únicamente cambia el PID. Mientras que la lectura varía dependiendo del número adicional de bytes y de las fórmulas propias de cada PID.

La lectura de los datos se lo hace de forma jerárquica, es decir cada "CAN.read()" lee datos diferentes, si hay dos de estos comandos leerá dos datos (DATA 1 y DATA 2), de esta manera se realiza las lecturas de todos los datos necesarios. Para obtener los valores netos se optó, para mayor exactitud, presentarlos con decimales, es por esto que en cada fórmula se especifica la función "*float*". Al usar la función "*float*" todas las constantes que se encuentran dentro de la fórmula se les añaden un decimal para que no existan errores de cálculo. Existen datos que solo son leídos y hay datos que aparte de ser leídos son impresos luego de pasar por la fórmula indicada. En la tabla 3.7 se presenta el extracto del código de cada función.

Tabla 3.7 Extracto del código de las funciones del modo 1

PID 14	PID 0B	PID 0C
Voltaje del sensor de oxígeno y ajuste de combustible	Presión absoluta en el colector de admisión	RPM del motor
`if (letra1 == 'A') { if (useStandardAddressing) { CAN.beginPacket(0x7df, 8); } else { CAN.beginExtendedPacket (0x18db33f1, 8); } CAN.write(0x02); CAN.write(0x01); CAN.write(0x14); CAN.endPacket(); while (CAN.parsePacket() == 0 || CAN.read() < 3 || CAN.read() != 0x41 || CAN.read() != 0x14); float VOLT = (CAN.read() / 200.0); float AJUSTE = ((CAN.read() * 0.78125) - 100.0); Serial.print("VOLT SENSOR OXIGENO (V) = "); Serial.print(VOLT); Serial.print(" AJUSTE (%)"); Serial.print(AJUSTE); }`	`if (letra2 == 'B') { if (useStandardAddressing) { CAN.beginPacket(0x7df, 8); } else { CAN.beginExtendedPacket (0x18db33f1, 8); } CAN.write(0x02); CAN.write(0x01); CAN.write(0x0B); CAN.endPacket(); while(CAN.parsePacket() == 0 || CAN.read() < 2 || CAN.read() != 0x41 || CAN.read() != 0x0B); float PRESION = (CAN.read()); Serial.print("PRESION ABSOLUTA EN EL COLECTOR (kPa)= "); Serial.print(PRESION); }`	`if (letra3 == 'C') { if (useStandardAddressing) { CAN.beginPacket(0x7df, 8); } else { CAN.beginExtendedPacket (0x18db33f1, 8); } CAN.write(0x02); CAN.write(0x01); CAN.write(0x0c); CAN.endPacket(); while (CAN.parsePacket() == 0 || CAN.read() < 3 || CAN.read() != 0x41 || CAN.read() != 0x0c); float rpm = ((CAN.read() * 256.0) + CAN.read()) / 4.0; Serial.print("RPM motor (rpm) = "); Serial.print(rpm); }`

Modo 3 de diagnóstico

Dentro de la lectura de códigos de falla se determinó que no es posible realizar la lectura de tres o más DTC's, esto porque no es posible leer las dos líneas del mismo *header* de manera simultánea, puesto que las respuestas son demasiado rápidas como para poder hacer la lectura respectiva. Por esta razón la interfaz está programada para realizar lecturas de hasta dos DTC's.

```
if (letra4 == 'D') {                          //condición para la letra D
                                               //ESCRITURA DEL MENSAJE
   if (useStandardAddressing) {
     CAN.beginPacket(0x7df, 8);                //header de escritura
   } else {
     CAN.beginExtendedPacket(0x18db33f1, 8);//header extendido
   }
   CAN.write(0x01);                           //número adicional de bytes
   CAN.write(0x03);                           //modo de diagnóstico
   CAN.endPacket();
```

El conjunto anterior muestra la definición de la letra que ejecutará la lectura de códigos de falla junto con el mensaje de petición de los mismos. Este mensaje servirá para cuando no hayan DTC's y cuando si estén presentes.

En el caso de no presenciar DTC's únicamente se mostrará un mensaje confirmando ese estado, como se muestra a continuación.

```
while (CAN.parsePacket());
    CAN.read();                              //número adicional de bytes
    CAN.read();                              //respuesta al modo

    int C0 = (CAN.read());                   //lectura del dato
    if (C0 = -1) {
      Serial.println("NO HAY CODIGOS DE FALLA");// impresión de mensaje
cuando no existen DTC
    }
```

Por otro lado cuando la ECU tiene presentes códigos de falla proporciona una respuesta diferente a cuando no los hay, es por esto que el código cambia cuando es necesario leer datos para imprimirlos. Al realizar lecturas tanto de uno como de dos códigos de falla lo único que varía es la longitud del mensaje como se muestra en la tabla 3.8, donde se presenta los extractos de los códigos de lectura de uno y dos DTC's.

Como se mencionó anteriormente la escritura del mensaje es la misma para este modo. Además en los dos casos el primer valor en aparecer es el conteo de número de DTC's hallados. Los valores siguientes son propiamente los elementos que conforman cada código de falla, donde cada uno de ellos está compuesto por dos datos. Cada mensaje es leído individualmente y además es necesario hacer una comparación para que los valores menores que 10, es decir los que tienen un solo digito, tengan un 0 delante de ellos, ya que en el mensaje es necesario tener los datos de esta forma para su correcta interpretación. En esta sección también es indispensable que los datos leídos sean impresos en su forma original, es decir en modo hexadecimal, dado que el CAN Bus Shield realiza la transformación a decimal de manera automática. Por esto, después de cada orden de impresión de datos se añade el comando "HEX" para que el software realice la conversión.

Tabla 3.8 Códigos de programación de lectura de 1 y 2 DTC's

Código de lectura de 1 DTC	Código de lectura de 2 DCT's

`while (CAN.parsePacket() == 0 ||`
`CAN.read() < 3 ||`
`CAN.read() != 0x43);`
 	
`Serial.print("Número de codigos encontrados = ");`
`int C1 = (CAN.read());`
`if (C1 < 16) {`
`Serial.print("0");`
`}`
`Serial.print(C1, HEX);`
 	
`Serial.print(" Codigos encontrados = P");`
 	
`int C2 = (CAN.read());`
`if (C2 < 16) {`
`Serial.print("0");`
`}`
`Serial.print(C2, HEX);`
 	
`int C3 = (CAN.read());`
`if (C3 < 16) {`
`Serial.print("0");`
`}`
`Serial.print(C3, HEX);`
	`while (CAN.parsePacket() == 0 ||`
	`CAN.read() < 05 ||`
	`CAN.read() != 0x43);`
	
	`Serial.print("Número de codigos encontrados = ");`
	`int C4 = (CAN.read());`
	`if (C4 < 16) {`
	`Serial.print("0");`
	`}`
	`Serial.print(C4, HEX);`
	
	`Serial.print(" Codigos encontrados = P");`
	
	`int C5 = (CAN.read());`
	`if (C5 < 16) {`
	`Serial.print("0");`
	`}`
	`Serial.print(C5, HEX);`
	
	`int C6 = (CAN.read());`
	`if (C6 < 16) {`
	`Serial.print("0");`
	`}`
	`Serial.print(C6, HEX);`
	
	`Serial.print(" Codigos encontrados = P");`
	
	`int C7 = (CAN.read());`
	`if (C7 < 16) {`
	`Serial.print("0");`
	`}`
	`Serial.print(C7, HEX);`
	
	`int C8 = (CAN.read());`
	`if (C8 < 16) {`
	`Serial.print("0");`
	`}`
	`Serial.print(C8, HEX);`
	
	`}`

Modo 4 de diagnóstico

Para el borrado de códigos de falla basta solo con escribir el modo en la red CAN, en esta parte al no obtener ningún tipo de respuesta que sea necesaria presentar en forma de datos,

exclusivamente se limitó a imprimir un mensaje que confirme que la acción del borrado de DTC's fue ejecutada correctamente.

```
if (letra5 == 'E') {
    if (useStandardAddressing) {
      CAN.beginPacket(0x7df, 8);
    } else {
      CAN.beginExtendedPacket(0x18db33f1, 8);
    }
    CAN.write(0x01);
    CAN.write(0x04);
    CAN.endPacket();

    Serial.print("CODIGOS BORRADOS");

  }

}
```

A diferencia de los otros modos de diagnóstico, en este no es necesario escribir el código de lectura de datos.

1.3.2 Código de programación en LabVIEW

La interfaz elaborada en LabVIEW es la encargada de escribir las letras que dan paso a las acciones anteriormente descritas y a receptar los datos del monitor serial del Arduino.

1.3.2.1 Diagrama de bloques

En la figura 3.20 se puede apreciar la programación gráfica de la interfaz donde se ha dividido todas las secciones por medio de un recuadro.

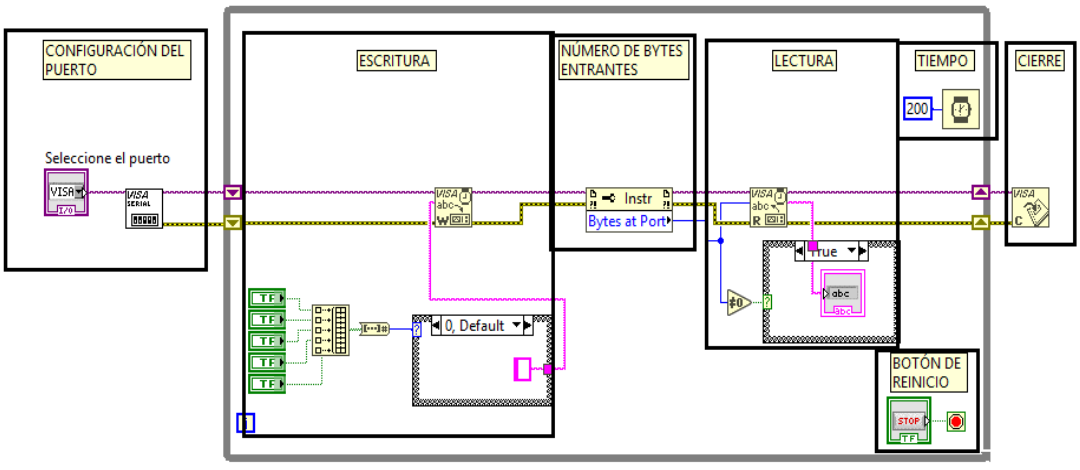


Figura 3.17 Programación en LabVIEW (diagrama de bloques)

Configuración del puerto

Dentro de la configuración del puerto únicamente es necesario añadir un control en la opción "*VISA resourse name*" el cual permite seleccionar el puerto que está usando el Arduino para enviar y recibir datos. Las demás opciones de configuración del puerto no es necesario que estén incluidas porque su configuración ya se lo realizó en el Arduino IDE.

Escritura

Para la escritura se crearon botones los cuales con de tipo "OK", es decir funcionan como un *switch* de encendido y apagado. Estos botones son de tipo booleano que proporcionan dados de 0 y 1 o de verdadero y falso. Seguido de estos botones es necesario construir un *array* que es la colección de datos del mismo tipo. Es necesario convertir el *array* a un número añadiendo un conversor. Dentro de una estructura de caso se definirá que dato escribe cada letra en el buffer de escritura, este punto se detalla en la tabla 3.9. Los elementos utilizados para la escritura se muestran en la figura 3.21.

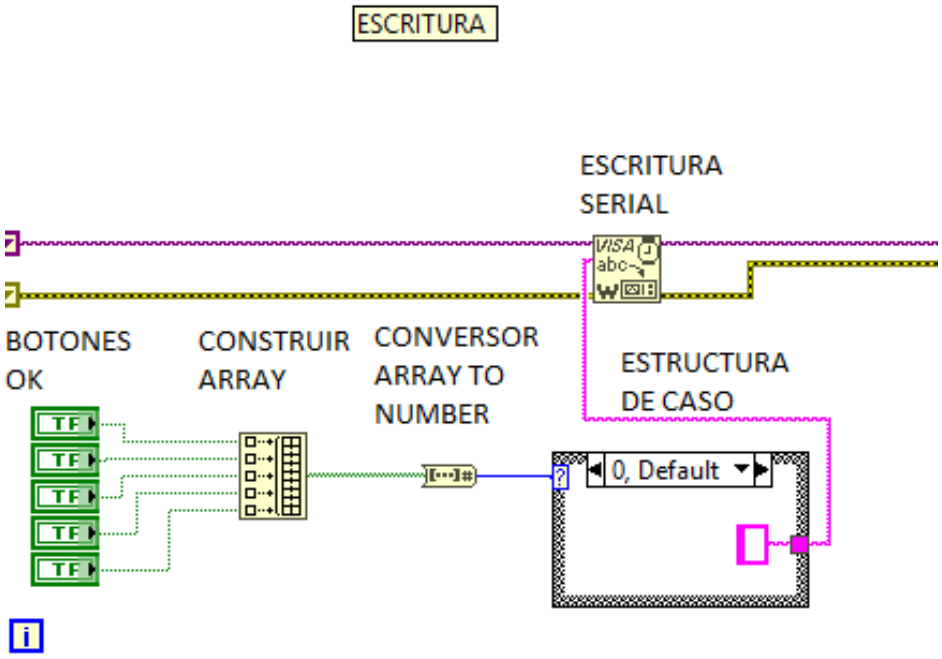
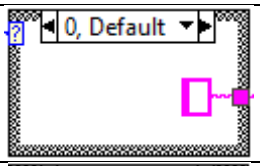
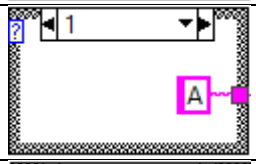
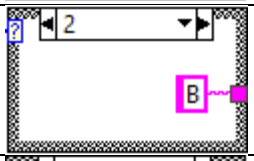
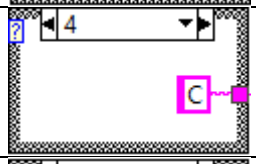
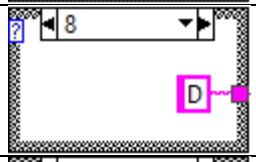
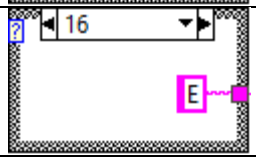


Figura 3.18 Extracto de código de escritura LabVIEW

Tabla 3.9 Estructura de casos para cada función

CASO		LETRA	FUNCIÓN
0		Ninguna	Ninguna
1		A	Voltaje del sensor de oxígeno y ajuste de combustible
2		B	Presión absoluta en el colector de admisión
4		C	RPM motor
8		D	Lectura de códigos de falla
16		E	Borrado de códigos de falla

Cada fila de la tabla 3.9 indica el número del caso que al ser activado por cada “botón OK” escribirá una letra en el buffer de escritura serial dando paso a las diferentes funciones.

Número de bytes entrantes al puerto

Esta función es necesaria y útil al momento de estar realizando lecturas de datos, lo que hace es identificar la cantidad de bytes que llegan al puerto. De esta manera se puede realizar una comparación para que en caso de que no estén llegando datos al puerto se genere un error y se detenga la lectura y la impresión de datos. Su conexión y ubicación dentro de la programación se puede observar en la figura 3.20.

Lectura e impresión

Una vez determinado el si hay o no datos en el puerto, es indispensable hacer una restricción para que el programa funcione solamente cuando existan datos en el buffer. Por otro lado se añadió otra estructura de caso pero en esta situación de verdadero y falso, esta trabaja en conjunto con el condicionante diferente de 0. Dentro de la estructura se añade un indicador que será el que muestre los datos en el panel frontal.

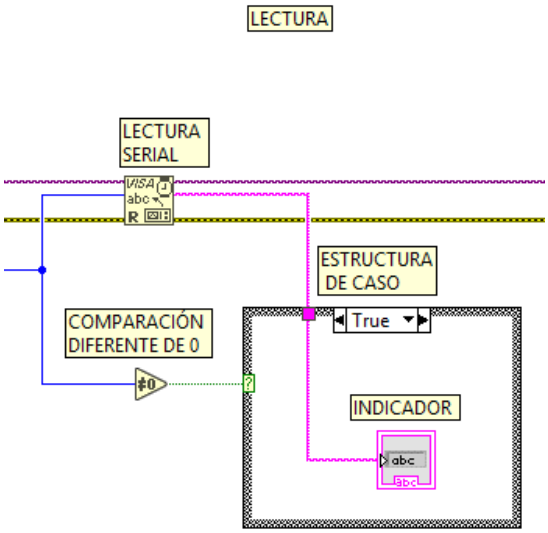


Figura 3.19 Extracto del código en LabVIEW (lectura)

El comparador está anclado a la estructura de caso para que en caso de que el número de bytes en el puerto sean 0, la estructura proporcione un caso falso que conlleva a que el indicador no presente datos. En el caso contrario de existir datos en el buffer se tendrá un caso verdadero y por ende el indicador mostrará los datos en el panel frontal.

Tiempo

Al haber colocado un ciclo "*while*" (recuadro gris de la figura 3.20) que contienen los elementos que se requiere que actúen de manera repetitiva es necesario especificar en tiempo para cada ciclo. Por este motivo se añade un "*wait*" o tiempo de espera, este trabaja en milisegundos que se añaden de acuerdo a las necesidades. En este caso se dispuso un tiempo de 200 ms para la repetición del ciclo.

Botón Stop

Este botón es necesario para detener o reiniciar el ciclo "*while*". Existen casos en que se tienen errores que producen una falla en el correcto funcionamiento del programa, provocando que este quede congelado. En dichos casos una de las soluciones para continuar con el funcionamiento de la interfaz es presionando el botón Stop.

1.3.2.2 Panel frontal

Como se mencionó anteriormente el panel frontal es la ventana donde prácticamente se puede interactuar con la interfaz. Está compuesta por botones e indicadores que han sido programados desde el diagrama de bloques. La figura 3.23 muestra la venta del panel frontal con todas sus herramientas.

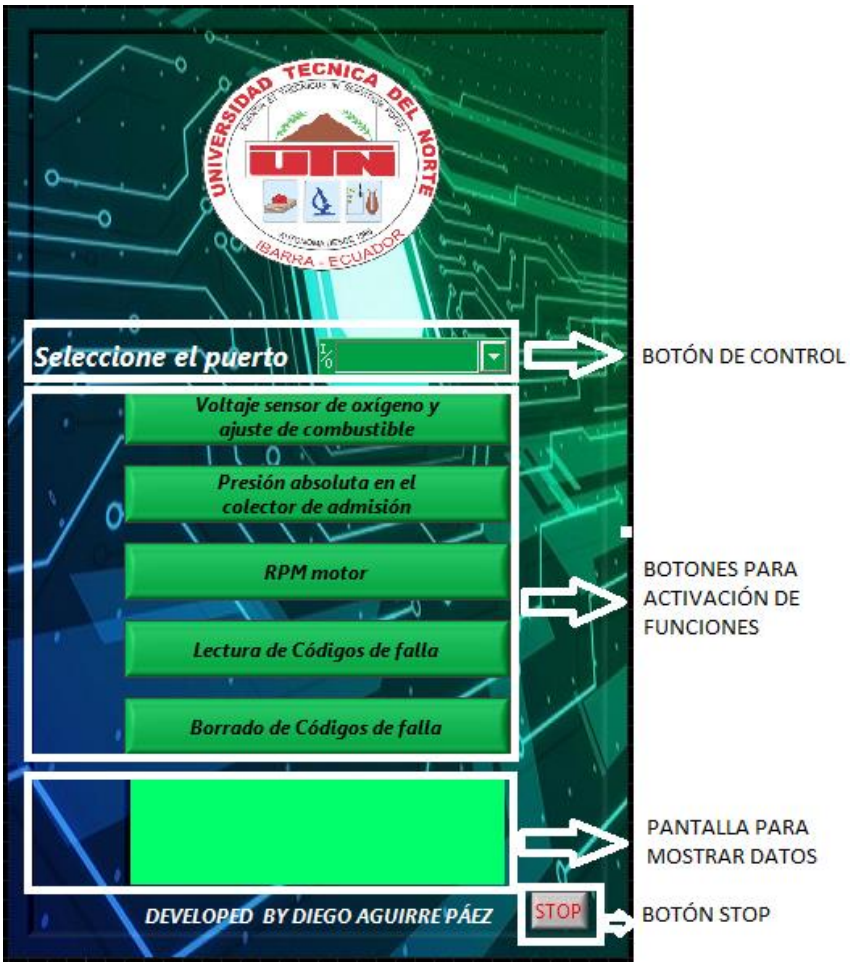


Figura 3.20 Panel frontal en LabVIEW

Para empezar se tiene un control, este permite seleccionar el puerto el cual está usando el Arduino para poder iniciar la comunicación serial. Seguidamente están los botones de activación de las funciones programadas en el Arduino. Luego se tiene la pantalla donde se mostraran los datos obtenidos luego de la ejecución de las funciones anteriores. Por último se encuentra el botón stop para detener o reiniciar el programa. Los botones de activación de voltaje del sensor de oxígeno y ajuste de combustible, presión absoluta en el colector de admisión y RPM del motor son de acción mecánica que al dar clic se quedan activos hasta que se los desactiva con otro clic. Mientras que los botones de lectura y borrado de códigos de falla son del tipo pulsador, es decir están activos mientras se mantiene presionado el clic.

1.4 RESULTADOS DE PRUEBAS DE INTERFAZ

Después de haber realizado las programaciones en los dos software indicados, se realizaron las pruebas correspondientes para verificar el funcionamiento de la interfaz. Para empezar es necesario hacer las conexiones correspondientes, donde por un lado se conecta desde el puerto OBD-II del vehículo al conector DB9 del CAN Bus Shield, y por otro lado se conecta el Arduino con la computadora por medio de un cable USB como se muestra e la figura 3.24.

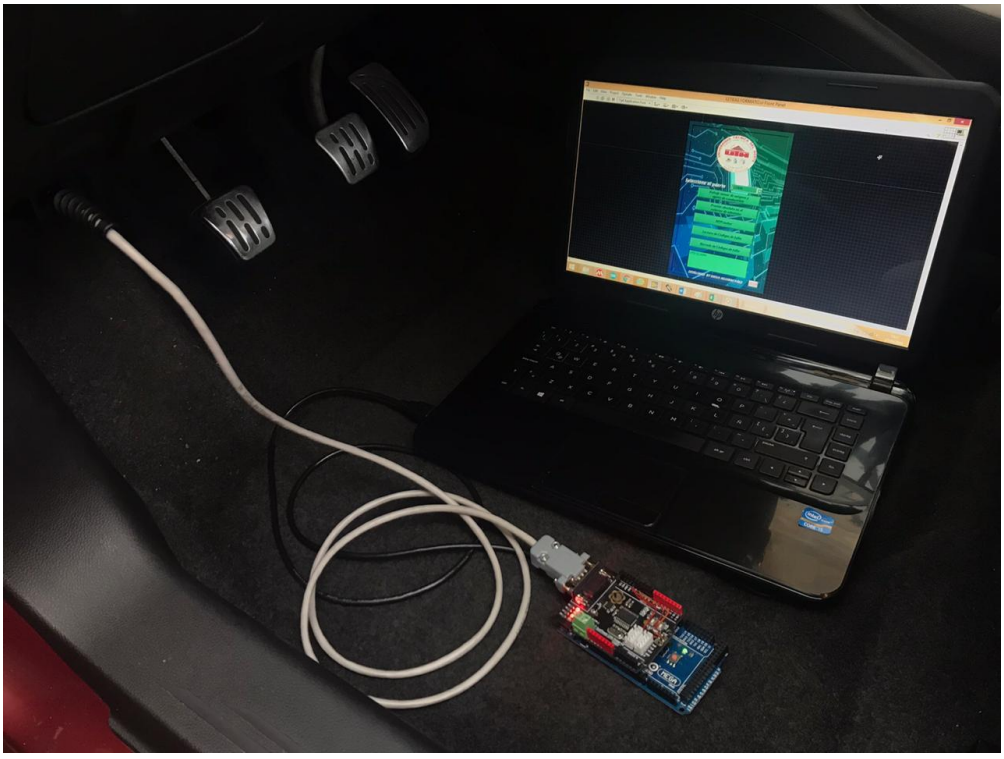


Figura 3.21 Conexión del dispositivo

Una vez conectado y con la interfaz abierta el primer paso es configurar el puerto. En el caso de no aparecer dicho puerto se debe dar clic en la opción "*refresh*" hasta que aparezca el puerto requerido. En el caso de no aparecer el puerto requerido se debe revisar que el puerto no esté ocupado en otro programa como por ejemplo Arduino IDE. Realizada la configuración del puerto se puede correr el programa donde en la pantalla de muestreo de datos debe aparecer un mensaje con el nombre de la interfaz indicando que está lista para ser usada.

Figura 3.22 Selección del puerto y mensaje de confirmación

1.4.1 Pruebas de funcionamiento del modo 1 de diagnóstico

Para la lectura de datos en tiempo real se tienen los tres primeros botones que deben ser activados uno por uno de manera individual. Al ser activados los botones cambian de color y se puede observar los datos obtenidos en la pantalla de muestreo.

Figura 3.23 Prueba de Voltaje del sensor de oxígeno y ajuste de combustible

En el caso de las pruebas de presión absoluta en el colector e admisión se puede apreciar que conforme se acelera el vehículo la presión varía, dando a entender el correcto funcionamiento del dispositivo

Figura 3.24 Prueba de presión absoluta en el colector de admisión

En la prueba de RPM's del motor es más apreciable el correcto funcionamiento del dispositivo, ya que se puede comparar las lecturas con el contador de revoluciones del tablero del vehículo.

Figura 3.25 Prueba de RPM del motor

Como se puede observar en las figuras anteriores, los valores obtenidos están dentro de los rangos admisibles. Además con otras pruebas se observó la variación de los datos cuando se cambian las condiciones de manejo del vehículo.

1.4.2 Pruebas de funcionamiento del modo 3 de diagnóstico

Como se mencionó anteriormente la interfaz está programada para leer hasta dos códigos de falla de manera consecutiva, además de presentar un mensaje en caso de no existir ninguno. En la pantalla de muestreo de datos también se puede observar que se presenta el número de códigos de falla encontrados. En las figuras 3.26 y 3.37 se observa a la interfaz realizando la lectura de uno y dos DTC's respectivamente, los mismos que fueron provocados al desconectar varios sensores dentro del vehículo.

Figura 3.26 Lectura de 1 DTC

A diferencia de la figura anterior, se puede apreciar la variación en canto al número de DTC's encontrados y además se puede observar que se está leyendo otro código de falla.

Figura 3.27 Lectura de 2 DTC

1.4.3 Pruebas de funcionamiento del modo 4 de diagnóstico

En el modo 4 de diagnóstico basta con obtener el mensaje de confirmación del borrado de los DTC's, para posteriormente volver a probar con el botón de lectura para verificar que efectivamente los códigos hayan sido borrados.

Figura 3.28 Borrado de Códigos de falla

Luego de haber borrado los códigos de falla se vuelve a dar lectura y como se puede observar en la figura 3.29 estos ya no están presentes.

Figura 3.29 Comprobación

1.5 DISCUSIÓN DE FUNCIONAMIENTO DE INTERFAZ DE DIAGNÓSTICO OBD-II

Una vez realizadas las pruebas con la interfaz se puede indicar los parámetros de funcionamiento de la misma. En los puntos siguientes se describen unas de las acciones que podrían generar errores y por ende provocar fallos en la lectura de datos.

- Inicialmente se debe seguir el orden anteriormente indicado para la configuración y ejecución del programa en cuestión. De caso contrario pueden existir errores que no permitan su correcta operación.
- No se pueden ejecutar dos funciones al mismo tiempo, esto debido a que dentro de la programación gráfica en el diagrama de bloques del LabVIEW, se especificó este punto al momento de crear los casos que se escriben las variables en la comunicación serial.
- Luego de realizar la lectura de códigos de falla se puede tener fallos, puesto a que el programa detecta valores que en algunos casos son interpretados como datos de error, los que provocan que la interfaz quede congelada momentáneamente. Esto se soluciona presionando el botón de stop que reiniciará el software y se podrán ejecutar nuevamente todas las funciones.
- Es importante que el puerto que se vaya utilizar no esté abierto en ningún otro programa, por ejemplo, al tener abierto el Arduino IDE con el monitor serial. Esto provoca que no se pueda enviar ni recibir datos a la interfaz.
- Otro punto importante es verificar la condición de los cables de conexión. Estos pueden tender a romper sus conexiones internas causando que no se transmitan los datos.

En las demás condiciones se esperaría un correcto funcionamiento de la interfaz.

CAPÍTULO IV

4. CONCLUSIONES Y RECOMENDACIONES

4.1 CONCLUSIONES

- Los *headers* de escritura y lectura de datos en la red CAN fueron identificados como 0x7DF y 0x7E8 respectivamente. Por un lado, en el *header* 0x7DF se escriben los mensajes hexadecimales que transitan por la red CAN desde el *scanner* hasta la ECU para acceder a los diferentes modos de diagnóstico. Mientras que por otro lado, en el *header* 0x7E8 se obtuvieron los datos de respuesta provenientes de la ECU, los cuales posteriormente fueron presentados en la interfaz.
- Al trabajar con vehículos OBD-II genéricos es suficiente conectar un dispositivo de diagnóstico a los pines correspondientes a CAN High, CAN Low y tierra de señal, en este caso identificados como pines 6, 14 y 5 del puerto OBD-II respectivamente. De este modo se puede obtener los datos que transitan por la red CAN en ese instante.
- Se consiguió realizar la infiltración a la red CAN del vehículo para tener acceso a los datos que transitan por la misma y además se logró escribir mensajes sobre ella para poder controlar ciertas funciones de diagnóstico al igual que un *scanner* automotriz convencional.
- La programación en Arduino se la realizó de tal manera que cumpla con las funciones establecidas realizando en envío y recepción de datos de la misma manera que al utilizar la opción "*transmit*" en el CAN Bus Analyzer, esto para no tener errores al momento del muestreo de datos. Además, se enlazó dichos datos del monitor serial del Arduino para que sean importados y exportados desde la interfaz en LabVIEW para su presentación.

4.2 RECOMENDACIONES

- En caso de fallas al usar los dispositivos de diagnóstico se debe revisar los cables de conexión, puesto que pueden existir rupturas internas de los cables o en los sockets, lo que produce que los datos no sean transmitidos en ningún sentido.
- Continuar con este proyecto implementando las demás funciones de lectura de datos que pueden ser soportadas dentro de vehículos con OBD-II genérico.
- Es necesario complementar este este proyecto usando dispositivos de análisis de redes CAN con mejores prestaciones para que se puedan añadir más funciones como es el caso de prueba de actuadores o lectura de más de dos DTC´s simultáneamente.
- Además, se podría perfeccionar este proyecto haciendo que el *scanner* presentado no sea únicamente para vehículos OBD-II genérico, sino que funcione como un *scanner* automotriz multimarca como los que se presentan en el mercado.

BIBLIOGRAFÍA

1. Autosoporte. (s.f.). *Autosoporte.* Obtenido de Autosoporte: http://www.autosoporte.com/index.php

2. CERVANTES, I., & ESPINOZA, S. (2010). *ESCÁNER AUTOMOTRÍZ DE PANTALLA TÁCTIL.* MÉXICO D.F.

3. Denton, T. (2016). *Sistemas Eléctrico y Electrónico del Automóvil. Tecnología automotriz: mantenimiento y reparación de vehículos* . España : MARCOMBO, S.A. .

4. Gutiérrez Gómez, A. E. (2017). *ANALIZADOR DE REDES CAN.* Jalisco.

5. Martínez Mas, A. (2012). *Analizador de redes CAN.* Valencia .

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

7. Microchip Technology, I. (2011). *Microchip CAN BUS Analyzer.* Obtenido de Microchip CAN BUS Analyzer: https://ww1.microchip.com/downloads/en/DeviceDoc/51848B.pdf

8. *NATIONAL INSTRUMENTS* . (02 de 02 de 2011). Obtenido de NATIONAL INSTRUMENTS : http://www.ni.com/white-paper/2732/es/#toc1

9. Olimex. (s.f.). *ARDUINO.cl*. Obtenido de http://arduino.cl/arduino-mega-2560/

10. Ortiz López, J. C. (2014). *DISEÑO DE ESCÁNER AUTOMOTRIZ OBDII MULTIPROTOCOLO.* GUATEMALA .

11. Sánchez Carrizo , J. (2017). *Simulador de una ECU y diagnóstico mediante CAN y OBD-II.* Cuenca .

12. Senplades. (2013). *Plan Nacional del Buen Vivir 2013-2017.* Quito.

13. Senplades. (2017). *Plan Nacional de Desarrollo 2017-2021-Toda una Vida.* Quito .

14. Simbaña, W., Caiza , J., Chávez , D., & López , G. (2016). Diseño e Implementación de un Sistema de Monitoreo Remoto del. *Revista Politécnica* .

ANEXOS

ANEXO I

Código de programación en Arduino IDE

```
#include <CAN.h>                              //incluir la librería usada


const bool useStandardAddressing = true;  // usar direcciones estándar

void setup() {
  Serial.begin(9600);                         //iniciar comunicación serial a
9600 braudios

  Serial.println("DIEGO SCANN");              // mensaje de confirmación de
inicio de comunicación serial


  if (!CAN.begin(500E3)) {                    //iniciar el CAN bus a 500 kb
por segundo
    Serial.println("INICIO FALLIDO");
    while (1);
  }

  if (useStandardAddressing) {
    CAN.filter(0x7e8);                        //definir header de lectura
  } else {
    CAN.filterExtended(0x18daf110);           //header de lectura extedido
  }
}

void loop() {

  char letra1 = Serial.read();                //creación de caracteres por
cada acción
  char letra2 = Serial.read();
  char letra3 = Serial.read();
  char letra4 = Serial.read();
  char letra5 = Serial.read();

  if (letra1 == 'A') {                        // condición para la letra A
    if (useStandardAddressing) {              // ESCRITURA DEL MENSAJE
      CAN.beginPacket(0x7df, 8);              //definir header de escritura y
el DLC
    } else {
      CAN.beginExtendedPacket(0x18db33f1, 8);// header extendido
    }
    CAN.write(0x02);                          //número adicional de bytes
    CAN.write(0x01);                          //modo de
diagnóstico
    CAN.write(0x14);                          //PID
    CAN.endPacket();

                                              //LECTURA DE DATOS
    while (CAN.parsePacket() == 0 ||          //leer datos diferentes de 0
           CAN.read() < 3 ||                  //número adicional de bytes
```

```
          CAN.read() != 0x41 ||          //respuesta al modo de
diagnóstico
          CAN.read() != 0x14);           //respuesta al PID

    float VOLT = (CAN.read() / 200.0);              //fórmula propia del
PID
    float AJUSTE = ((CAN.read() * 0.78125) - 100.0);//fórmula propia del
PID

    Serial.print("VOLT SENSOR OXIGENO (V) = ");     //impresión del
rótulo y unidad
    Serial.print(VOLT);                             //impresión del valor

    Serial.print("        AJUSTE (%)");             //impresión del
rótulo y unidad
    Serial.print(AJUSTE);                           //impresión del valor

  }

  if (letra2 == 'B') {                      //condición para la letra B
    if (useStandardAddressing) {            //ESCRITURA DEL MENSAJE
      CAN.beginPacket(0x7df, 8);            //definir header de escritura y
DLC
    } else {
      CAN.beginExtendedPacket(0x18db33f1, 8);//header extendido
    }
    CAN.write(0x02);                        //número adicional de bytes
    CAN.write(0x01);                        //modo de diagnóstico
    CAN.write(0x0B);                        //PID
    CAN.endPacket();

                                            //LECTURA DE DATOS
    while (CAN.parsePacket() == 0 ||        //leer valores diferentes de 0
           CAN.read() < 2 ||                //número adicional de bytes
           CAN.read() != 0x41 ||            //respuesta al modo
           CAN.read() != 0x0B);             //respuesta al PID

    float PRESION = (CAN.read());           //operación propia del PID

    Serial.print("PRESION ABSOLUTA EN EL COLECTOR (kPa)= ");//impresión
del rótulo y unidad
    Serial.print(PRESION);                  //impresión del valor

  }



  if (letra3 == 'C') {                      //condición para la letra C
    if (useStandardAddressing) {            //ESCRITURA DEL MENSAJE
      CAN.beginPacket(0x7df, 8);            //definir header de escritura y
DLC
    } else {
      CAN.beginExtendedPacket(0x18db33f1, 8);//header extendido
    }
    CAN.write(0x02);                        //número adicional de bytes
    CAN.write(0x01);                        //modo de diagnóstico
    CAN.write(0x0c);                        //PID
    CAN.endPacket();
                                            //LECTURA DE DATOS
    while (CAN.parsePacket() == 0 ||        //leer datos diferentes de 0
```

```
            CAN.read() < 3 ||               //número adicional de bytes
            CAN.read() != 0x41 ||           //respuesta al modo
            CAN.read() != 0x0c);            //respuesta al PID

    float rpm = ((CAN.read() * 256.0) + CAN.read()) / 4.0;//fórmula
propia del PID

    Serial.print("RPM motor (rpm) = ");     //impresión del rótulo y unidad
    Serial.print(rpm);                      //impresión del dato

  }

  //CÓDIGOS DE FALLA
  if (letra4 == 'D') {                      //condición para la letra D
                                            //ESCRITURA DEL MENSAJE
    if (useStandardAddressing) {
      CAN.beginPacket(0x7df, 8);            //header de escritura
    } else {
      CAN.beginExtendedPacket(0x18db33f1, 8);//header extendido
    }
    CAN.write(0x01);                        //número adicional de bytes
    CAN.write(0x03);                        //modo de diagnóstico
    CAN.endPacket();


    while (CAN.parsePacket());
    CAN.read();                             //número adicional de bytes
    CAN.read();                             //respuesta al modo

    int C0 = (CAN.read());                  //lectura del dato
    if (C0 = -1) {
      Serial.println("NO HAY CODIGOS DE FALLA");// impresión de mensaje
cuando no existen DTC
    }

    // LECTURA 1 CODIGO

    if (useStandardAddressing) {            //ESCRITURA DEL MENSAJE
      CAN.beginPacket(0x7df, 8);            //header de escritura
    } else {
      CAN.beginExtendedPacket(0x18db33f1, 8);//header extendido
    }
    CAN.write(0x01);                        //número adicional de bytes
    CAN.write(0x03);                        //modo de diagnóstico
    CAN.endPacket();

                                          //LECTURA DE DATOS
    while (CAN.parsePacket() == 0 ||      //leer datos diferentes de 0
           CAN.read() < 3 ||              //número adicional de bytes
           CAN.read() != 0x43);           //respuesta al PID

    Serial.print("Número de codigos encontrados = ");//impresión de
rótulo
    int C1 = (CAN.read());                //lectura de dato
    if (C1 < 16) {                        //valores menores de 10
      Serial.print("0");                  //imprimir 0
    }
    Serial.print(C1, HEX);                //imprimir dato en hexadecimal

    Serial.print("      Codigos encontrados = P");//imprimir rótulo
```

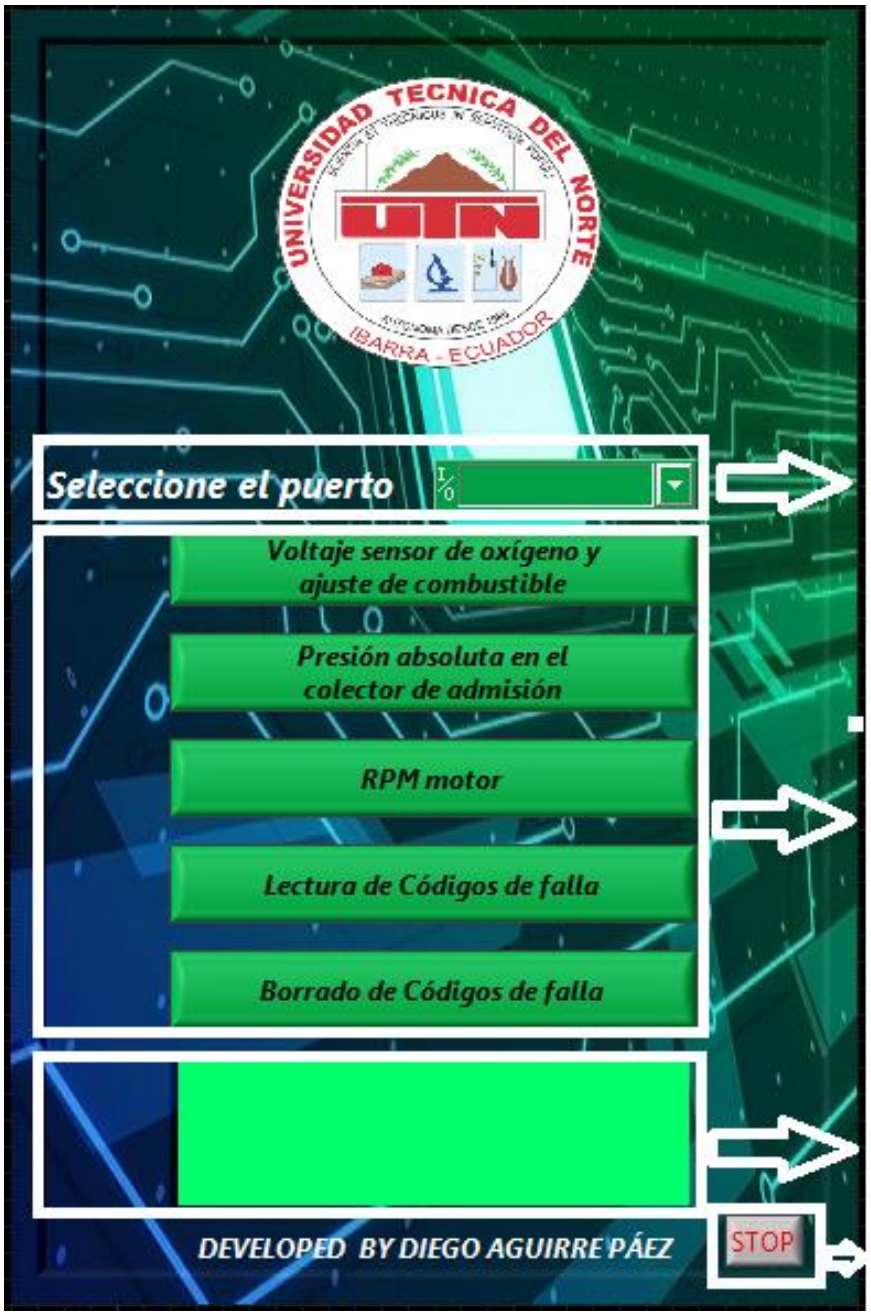
```
    int C2 = (CAN.read());                  //lectura de dato
    if (C2 < 16) {                          //valores menores que 10
      Serial.print("0");                    //imprimir 0
    }
    Serial.print(C2, HEX);                  //imprimir valor en hexadecimal

    int C3 = (CAN.read());                  //lectura de dato
    if (C3 < 16) {                          //valores menores de 10
      Serial.print("0");                    //imprimir 0
    }
    Serial.print(C3, HEX);                  //imprimir dato en hexadecimal

    //LECTURA 2 CODIGOS
                                            //ESCRITURA DEL MENSAJE
    if (useStandardAddressing) {
      CAN.beginPacket(0x7df, 8);            //header de lectura
    } else {
      CAN.beginExtendedPacket(0x18db33f1, 8);//header extendido
    }
    CAN.write(0x01);                        //número adicional de bytes
    CAN.write(0x03);                        //modo de diagnóstico
    CAN.endPacket();


    while (CAN.parsePacket() == 0 ||        //leer datos diferentes de 0
           CAN.read() < 05 ||               //número adicional de bytes
           CAN.read() != 0x43);             //respuesta al modo


    Serial.print("Número de codigos encontrados = ");//impresión de
rótulo
    int C4 = (CAN.read());                  //lectura de dato
    if (C4 < 16) {                          //valores menores que 10
      Serial.print("0");                    //imprimir 0
    }
    Serial.print(C4, HEX);                  //imprimir dato en hexadecimal

    Serial.print("      Codigos encontrados = P");//impresión de rótulo

    int C5 = (CAN.read());                  //lectura de dato
    if (C5 < 16) {                          //valores menores de 10
      Serial.print("0");                    //imprimir 0
    }
    Serial.print(C5, HEX);                  //imprimir dato en hexadecimal

    int C6 = (CAN.read());                  //lectura de dato
    if (C6 < 16) {                          //valores menores a 10
      Serial.print("0");                    //imprimir 0
    }
    Serial.print(C6, HEX);                  //imprimir dato en hexadecimal


    Serial.print("        Codigos encontrados = P");//impresión de rótulo

    int C7 = (CAN.read());                  //lectura de dato
    if (C7 < 16) {                          //valores menores a 10
      Serial.print("0");                    //imprimir 0
    }
    Serial.print(C7, HEX);                  //imprimir dato en hexadecimal
```

```
    int C8 = (CAN.read());                //lectura de dato
    if (C8 < 16) {                        //valores menores a 10
      Serial.print("0");                 //imprimir 0
    }
    Serial.print(C8, HEX);                //imprimir dato en hexadecimal

  }

  if (letra5 == 'E') {                    //condición para la letra E
    if (useStandardAddressing) {          //ESCRITURA DEL MENSAJE
      CAN.beginPacket(0x7df, 8);          //definir header de escritura y
DLC
    } else {
      CAN.beginExtendedPacket(0x18db33f1, 8);//header extendido
    }
    CAN.write(0x01);                      //número adicional de bytes
    CAN.write(0x04);                      //modo de diagnóstico
    CAN.endPacket();

    Serial.print("CODIGOS BORRADOS");     //mensaje de correcta ejecución

  }

}
```

ANEXO II

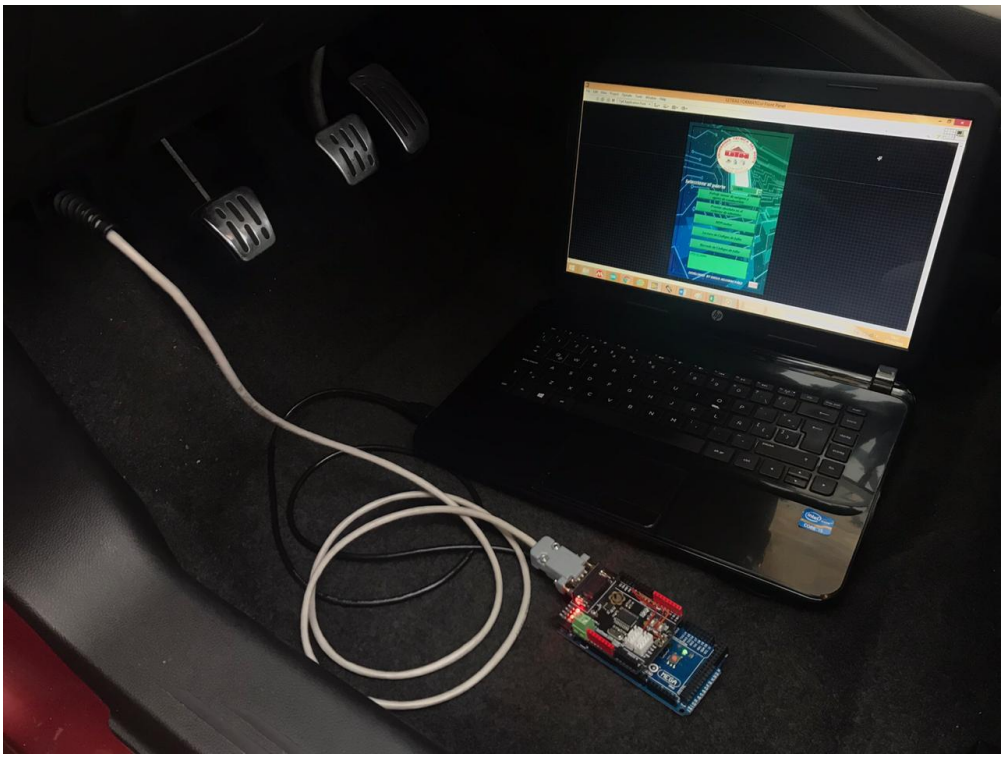
Manual de usuario de la interfaz UTN SCANN

Elementos de la interfaz



Conexión del dispositivo

Por un lado se conecta desde el puerto OBD-II del vehículo al conector DB9 del UTN SCANN, y por otro lado se conecta el Arduino con la computadora por medio de un cable USB como se muestra a continuación.

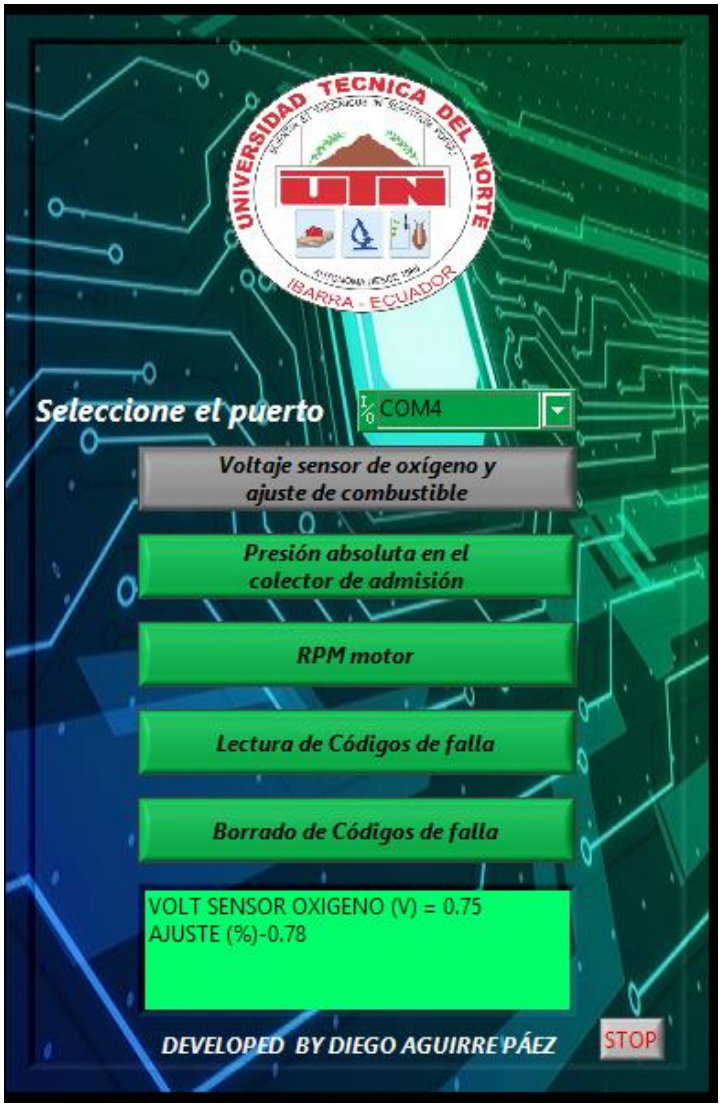


Una vez conectado y con la interfaz abierta el primer paso es configurar el puerto. En el caso de no aparecer dicho puerto se debe dar clic en la opción "*refresh*" hasta que aparezca el puerto requerido. En el caso de no aparecer el puerto requerido se debe revisar que el puerto no esté ocupado en otro programa como por ejemplo Arduino IDE. Realizada la configuración del puerto se puede correr el programa donde en la pantalla de muestreo de datos debe aparecer un mensaje con el nombre de la interfaz "UTN SCANN" indicando que está lista para ser usada. De lo contrario aparecerá un mensaje "INICIO FALLIDO" indicando un problema en el funcionamiento del programa. En el caso de no aparecer ningún mensaje podría tratarse de problemas en la conexión del dispositivo o a que el puerto seleccionado no es el correcto o está siendo usado en otro programa.

FUNCIONES

Lectura de datos

Para la lectura de datos en tiempo real se tienen los tres primeros botones que deben ser activados uno por uno de manera individual. Al ser activados los botones cambian de color y se puede observar los datos obtenidos en la pantalla de muestreo. Estos botones se quedarán activados automáticamente hasta que el operador los desactive. Una vez desactivada la función el programa mantendrá grabado el último dato leído.



Lectura de códigos de falla

La interfaz está programada para leer hasta dos códigos de falla de manera consecutiva, además de presentar un mensaje en caso de no existir ninguno. En la pantalla de muestreo de datos también se puede observar que se presenta el número de códigos de falla encontrados. Esta función se ejecutará únicamente al presionar el botón correspondiente, es decir, no permanecerá activa. Sin embargo, los datos leídos permanecerán grabados en la pantalla de muestreo.

Borrado de códigos de falla

Únicamente se obtendrá un mensaje confirmando la ejecución de la función.

SOLUCIÓN DE PROBLEMAS

Seguir las siguientes recomendaciones en el caso de presentar problemas con el uso de la interfaz:

- Inicialmente se debe seguir el orden anteriormente indicado para la configuración y ejecución del programa en cuestión. De caso contrario pueden existir errores que no permitan su correcta operación.
- No se pueden ejecutar dos funciones al mismo tiempo.
- Luego de realizar la lectura de códigos de falla se puede tener fallos, puesto a que el programa detecta valores que en algunos casos son interpretados como datos de error, los que provocan que la interfaz quede congelada momentáneamente. Esto se soluciona presionando el botón de stop que reiniciará el software y se podrán ejecutar nuevamente todas las funciones.
- Es importante que el puerto que se vaya utilizar no esté abierto en ningún otro programa, por ejemplo, al tener abierto el Arduino IDE con el monitor serial. Esto provoca que no se pueda enviar ni recibir datos a la interfaz.
- Otro punto importante es verificar la condición de los cables de conexión. Estos pueden tender a romper sus conexiones internas causando que no se transmitan los datos.

En las demás condiciones se esperaría un correcto funcionamiento de la interfaz.