FACULTAD DE INGENIERÍAS Y CIENCIAS AGROPECUARIAS

DISEÑO E IMPLEMENTACIÓN DE UN PROTOTIPO DE DISPOSITIVO PARA AUTOMÓVILES QUE PERMITA VISUALIZAR EN EL PARABRISAS, INFORMACIÓN DE UTILIDAD PARA SU CONDUCCIÓN

AUTOR

Jorge Richard Ortega Poveda

AÑO

2017

FACULTAD DE INGENIERÍAS Y CIENCIAS AGROPECUARIAS

DISEÑO E IMPLEMENTACIÓN DE UN PROTOTIPO DE DISPOSITIVO PARA AUTOMÓVILES QUE PERMITA VISUALIZAR EN EL PARABRISAS, INFORMACIÓN DE UTILIDAD PARA SU CONDUCCIÓN

Trabajo de Titulación presentado en conformidad con los requisitos establecidos para optar por el título de Ingeniero en Redes y Telecomunicaciones.

Profesor Guía:

MBA. José Julio Freire Cabrera

Autor:

Jorge Richard Ortega Poveda

Año:

2017

DECLARACIÓN DEL PROFESOR GUÍA

"Declaro haber dirigido el trabajo, Diseño e implementación de un prototipo de dispositivo para automóviles que permita visualizar en el parabrisas, información de utilidad para su conducción, a través de reuniones periódicas con el estudiante Jorge Richard Ortega Poveda, en el semestre 2018-1, orientando sus conocimientos y competencias para un eficiente desarrollo del tema escogido y dando cumplimiento a todas las disposiciones vigentes que regulan los Trabajos de Titulación".

José Julio Freire Cabrera
Magíster en Gerencia Empresarial
C.I. 1709731457

DECLARACIÓN DEL PROFESOR CORRECTOR

"Declaro haber revisado este trabajo, Diseño e implementación de un prototipo de dispositivo para automóviles que permita visualizar en el parabrisas, información de utilidad para su conducción, del estudiante Jorge Richard Ortega Poveda, en el semestre 2018-1, dando cumplimiento a todas las disposiciones vigentes que regulan los Trabajos de Titulación".

Iván Ricardo Sánchez Salazar
Magíster en Calidad, Seguridad y Ambiente
C.I. 1803456142

DECLARACIÓN DE AUTORÍA DEL ESTUDIANTE

"Declaro que este trabajo es original, de mi autoría, que se han citado las fuentes correspondientes y que en su ejecución se respetaron las disposiciones legales que protegen los derechos de autor vigentes."

Jorge Richard Ortega Poveda
C.I. 1715196604

AGRADECIMIENTOS

A mi Docente Guía para este Trabajo de Titulación, por su apoyo constante para la consecución de esta meta.

A la UDLA y la planta Docente de la FICA, por ser quienes han permitido a través de la educación brindada, el constituirme en un profesional.

A toda mi Familia, por su apoyo, paciencia y sacrificio, en todo este largo y ajetreado camino.

DEDICATORIA

A Evelyn, mi amada compañera de vida. A Sofía y Matías, mis regalos de Dios y motivación para seguir adelante. A mis Padres, Mariana y Jorge, por siempre mis ejemplos a seguir.

RESUMEN

El presente trabajo describe en detalle el proceso de desarrollo de un prototipo que utiliza los sensores del vehículo para permitir al conductor obtener información del estado del mismo, en el parabrisas. Se trata de una aplicación práctica de los conocimientos adquiridos a lo largo del curso de la carrera en la UDLA.

Para el diseño se aplica una metodología inductiva pues se basa en la revisión de proyectos similares complementado con la investigación teórica, para generar una propuesta que generaliza la experiencia de los ejemplos analizados acorde a los objetivos señalados.

Con este antecedente, y teniendo como resultado el diseño, se procede a implementar el prototipo de dispositivo, en donde haciendo uso de la metodología experimental, se valida la eficiencia del diseño, con la finalidad de ir ajustando dicho diseño y en consecuencia el prototipo implementado.

El documento incluye: Una Introducción, en donde se define los antecedentes, justificación y objetivos del trabajo. Un primer capítulo, denominado Análisis de Tecnologías de asistencia a conductores de vehículos, que hace una reseña de la problemática a afrontar y de las alternativas de solución. Un segundo capítulo, Diseño del Prototipo, que documenta precisamente este proceso. Un tercer capítulo, de Implementación y pruebas del Prototipo, que corresponde al proceso de documentación de la construcción del prototipo diseñado. Un cuarto capítulo, Análisis de Costos y Resultados, que incluye un desglose de los costos incurridos. Y, una parte final, en donde se detalla las Conclusiones y Recomendaciones a las que se llega luego de la elaboración del presente trabajo. Adicionalmente, se citan las Referencias, así como también, se incluyen los Anexos, con información relevante a detalle.

ABSTRACT

The present work describes in detail the development process of a prototype that uses the sensors of the vehicle to allow the driver to obtain information of its state, in the windshield. It is a practical application of the knowledge acquired throughout the course of the UDLA career.

For the design an inductive methodology is applied since it is based on the review of similar projects complemented with theoretical research, to generate a proposal that generalizes the experience of the analyzed examples according to the raised objectives.

With this background, and resulting in the design, we proceed to implement the device prototype, where using the experimental methodology, the design efficiency is validated, in order to adjust this design and consequently the implemented prototype.

The document includes: An Introduction, where the background, justification and objectives of the work are defined. A first chapter, called Analysis of Assistance Technologies for vehicle drivers, which gives an overview of the problems to be faced and its alternative solutions. A second chapter, Design of the Prototype, which documents precisely this process. A third chapter, Implementation and testing of the Prototype, which corresponds to the documentation process of the construction of the designed prototype. A fourth chapter, Analysis of Costs and Results, which includes a breakdown of the costs incurred. And a final chapter, which details the conclusions and recommendations that are reached after the preparation of this work. Additionally, the References are cited, as well as, the Annexes are included, with relevant information in detail.

INDICE

INDICE DE FIGURAS

INDICE DE TABLAS

1. CAPÍTULO I. INTRODUCCIÓN

1.1. Antecedentes

Hoy en día el desarrollo de la tecnología ha permitido pasos agigantados en la difusión de diseños y construcción de equipos especializados, que se orientan a prestar ayuda y facilidades en las tareas de los seres humanos.

Existen varios proyectos, impulsados por instituciones académicas de renombre en el campo de las ciencias tecnológicas, que pretenden dar la pauta o bases para la generación de soluciones innovadoras no asociadas al ámbito comercial, sin que eso descarte que un siguiente paso, para los prototipos que efectivamente demuestren su eficacia, sea la producción a nivel industrial en escalas de masificación. Uno de estos, es el proyecto “Arduino” que inició en el año 2005 como un proyecto para estudiantes en el Instituto IVREA, ubicado en Ivrea (Italia) y que básicamente se constituye en una plataforma de creación de prototipos de código abierto basada en hardware y software fáciles de usar (más adelante en este documento se desarrolla mayor explicación del tema); otro ejemplo es la fundación “Raspberry Pi” fundada en Caldecote, South Cambridgeshire, Reino Unido en el 2009, como una asociación caritativa que es regulada por la comisión de caridad de Inglaterra y Gales, y tiene como principal objetivo el animar a los niños a aprender informática, el dispositivo que esta organización respalda es el Raspberry Pi, que es un computador de placa reducida, única o simple, de bajo costo, desarrollado con el objetivo de estimular la enseñanza de ciencias de la computación en las escuelas (más información y con mayor detalle se puede encontrar en la web oficial https://www.raspberrypi.org/).

Los ejemplos citados, constituyen herramientas que complementan hardware y software con gran versatilidad, cuyo objetivo final es ser la base para la construcción de prototipos tecnológicos a nivel académico.

El auto inteligente, es un concepto que se viene desarrollando desde inicios del siglo XX, y de manera primordial apunta a la generación de sistemas autónomos. Sin embargo, también forman parte de este concepto, todos los componentes o dispositivos que permiten facilitar la maniobrabilidad de los automóviles, por ejemplo, sistemas de navegación satelital, sistemas de parqueo, sistemas de diagnóstico continuo del auto, etc. Este tipo de componentes permiten que el conductor tenga cada vez menos cosas en qué preocuparse o a las cuales prestar atención al mismo tiempo que conduce.

La electrónica y telemática cumplen un rol fundamental en la cristalización del concepto de auto inteligente antes mencionado. Es así, que los automóviles vienen equipados con cada vez mejores sistemas de control y automatización de sus funciones, un ejemplo de ello es la incorporación de las unidades de control de energía o ECU, por sus siglas en inglés (Energy Control Unit), las cuales procesan información recolectada a través de sensores ubicados en los diferentes sistemas del auto, para mediante una comparación determinar su estado óptimo de funcionamiento. A su vez, la ECU forma parte del sistema de diagnóstico a bordo u OBD, por sus siglas en inglés (On Board Diagnosis System), el cual se encarga de visualizar en el tablero del automóvil, de manera luminosa o sonora, la información adquirida por la ECU a través de los respectivos sensores.

Con estos antecedentes, se propone el presente proyecto, constituido por un prototipo de dispositivo HUD (Head Up Display), cuyo concepto se aborda con más detalle más adelante en este documento, y que básicamente permite la adquisición y visualización de información relacionada con el estado del automóvil en el parabrisas del mismo, evitando que la mirada del conductor tenga que dirigirse al tablero de instrumentos del vehículo y así pierda la concentración en el camino que recorre. Se incluye el análisis de la opción de integración con otros dispositivos que generen información de importancia que se requiera que sea visualizada para quien conduce el vehículo. De esta

manera, se pretende aportar tanto a la disminución de accidentes como a la mejora de la experiencia de conducción, haciéndola más cómoda y segura.

1.2. Alcance

Este proyecto será enfocado a aquellos automóviles de la gama media y baja y que a su vez se incluyan en los segmentos de turismo o SUV (Sport Utility Vehicle); y, considerando adicionalmente que deben contar con un puerto OBD II, que es la interfaz sobre la cual se desarrollará el diseño del prototipo. En primera instancia, para conseguir la visualización de la información, se realizará una investigación de los códigos que proporciona el computador de los automóviles, a través de su puerto de diagnóstico, lo cual también permitirá delimitar qué información acerca del estado del auto es posible mostrar con el dispositivo; esto implica también, que este proyecto restringe su funcionalidad a aquellos automóviles que cuenten con el mencionado puerto, es decir que tengan computador a bordo. Como resultado de esta investigación, se programará el módulo para el reconocimiento y visualización de los códigos referidos previamente.

Por otro lado, la visualización de la información del auto que se logre identificar se planifica hacerla a través de un método de reflexión en el parabrisas del automóvil, con la finalidad de mantener el concepto de una pantalla transparente, que es el que caracteriza a los sistemas HUD. Para el efecto, la propuesta es hacer uso de una pantalla luminosa que haga las veces de proyector, generando las imágenes que por reflexión serán visibles en el parabrisas.

Y de acuerdo a esto se incluirá el respectivo análisis de costo del dispositivo descrito, toda vez que el mismo haya sido implementado, con el objeto de contar con los parámetros reales que permitan determinar la rentabilidad del proyecto.

1.3. Justificación

La implementación del prototipo que se propone con el presente trabajo, pretende ser una alternativa para aquellos profesionales y aficionados a la electrónica y telecomunicaciones, al constituirse y consolidarse como una solución de bajo costo, en comparación con sus similares de tipo comercial, para mejorar las condiciones de conducción de los automóviles que no vienen equipados con dispositivos HUD de fábrica. Esto, dado que se ha establecido que dichos dispositivos, puesto que permiten trasladar la información que se proporciona en el tablero de los automóviles al área de visualización del conductor, contribuyen a evitar distracciones en el proceso de manejo del automotor y por ende disminuyen la tasa de accidentes a causa de este particular.

Adicionalmente, se estima que toda la base de conocimiento que se genere para la implementación de éste prototipo, se constituya en un referente para el desarrollo de nuevas propuestas a un siguiente nivel, para la comunidad universitaria de la UDLA, es decir que esta investigación permita a otros compañeros visionarios, ampliar la presente idea dándole nuevas aplicaciones, o generando incluso otras posibilidades de negocio, manteniendo la visión de apoyo a la sociedad.

Para lograr este objetivo, se aplicará los conceptos adquiridos en el transcurso de la carrera dentro de las aulas de la universidad, y se procurará presentarlo a la sociedad como un proyecto de bajo costo para proveer de una mejora a una de las herramientas más utilizadas por las personas en la actualidad, que es el automóvil. Así mismo, se consolidará la posición de la Universidad al destacar el enfoque de su investigación en el desarrollo de prototipos de dispositivos que la sociedad requiere y le pueden ser de gran utilidad en temas de seguridad con una baja inversión.

1.4. Objetivo General

Desarrollar un prototipo utilizando los sensores del vehículo que permitan al conductor obtener información del estado del mismo, en el parabrisas.

1.5. Objetivos específicos

- Analizar las tecnologías existentes para asistencia al conductor en el proceso de manejo del automóvil.

- Diseñar el prototipo de dispositivo que permita visualizar de manera eficiente la información indispensable para la conducción del automóvil en su parabrisas.

- Implementar y efectuar pruebas del prototipo diseñado.

- Analizar los resultados para establecer mejoras y recomendaciones para el dispositivo implementado.

1.6. Estructura del documento

Para cumplir con los objetivos planteados, el presente documento se ha desarrollado de la siguiente manera:

- Introducción, en donde se define los antecedentes, justificación y objetivos del trabajo.

- Un primer capítulo, denominado Análisis de Tecnologías de asistencia a conductores de vehículos, en donde se hace una reseña de la problemática

surgida en torno a las distracciones en las que incurren las personas al conducir vehículos; y así mismo, de las alternativas a las que se puede recurrir para la reducción de las mencionadas distracciones.

- Un segundo capítulo, Diseño del Prototipo, en donde se documenta el proceso de diseño del prototipo, incluyendo el detalle de los conceptos técnicos y características de los elementos utilizados en esta fase.
- El tercer capítulo, de Implementación y pruebas del Prototipo, corresponde al proceso de documentación de la construcción del prototipo. Incluye los resultados trascendentes que generen las pruebas de funcionalidad.
- En el cuarto capítulo, Análisis de Costos y Resultados, incluye un desglose de los costos incurridos en la implementación del proyecto, determinando a priori su conveniencia. También muestra un resumen de los resultados obtenidos considerando diferentes factores
- Un capítulo quinto y final, detalla las Conclusiones y Recomendaciones a las que se llega luego de la elaboración del presente trabajo.

Adicionalmente, se citan las Referencias de los libros y páginas web que han servido para la consulta. Así también, forma parte del documento los Anexos, en donde se hace constar el detalle de los códigos de las aplicaciones generadas como parte de este trabajo y otros datos adicionales.

2. CAPÍTULO II. Análisis de Tecnologías de asistencia a conductores de vehículos

En esta primera parte se expone una reseña de la problemática surgida en torno a las distracciones a las que se ven expuestos hoy en día los conductores de vehículos, de manera particular se hace referencia a aquellas distracciones en las que incurren a razón de enterarse del estado de los mecanismos del automóvil: nivel de gasolina, temperatura del motor, velocidad, kilometraje, etc., los cuales son visibles en el tablero de instrumentos del vehículo que generalmente se ubica en la parte posterior del volante de este.

Por lo antes señalado y para llegar a cubrir el objetivo, se inicia mostrando algunos conceptos básicos del citado tablero de instrumentos, sus variables, formas de adquisición de datos para estas variables; y, por otro lado, se hace mención a la siniestralidad generada por eventos distractores en la conducción de automóviles; finalmente como cierre a esta primera parte se plantean los conceptos principales respecto a dispositivos “Head Up Display” o HUD.

2.1. El tablero de instrumentos del automóvil

El tablero de instrumentos de un automóvil está compuesto por un conjunto de indicadores del estado de funcionamiento de un vehículo, generalmente se ubica en la parte frontal del conductor por debajo del parabrisas, en modelos más actuales o dependiendo de su diseño también se los puede encontrar en la parte intermedia entre el airbag del pasajero y el volante. Además de los indicadores, el tablero también cuenta con una serie de luces testigo o también conocidos como testigos luminosos, como por ejemplo indicador de temperatura elevada del motor, bajo nivel de gasolina, testigo de presión de aceite, de carga de la batería, de accionamiento de luces intermitente, entre otros.

2.1.1. Indicadores del tablero de instrumentos del automóvil

El tablero, como se mencionó, contiene varios indicadores como el velocímetro para conocer la velocidad del vehículo, el tacómetro para vigilar las revoluciones del motor, la temperatura del refrigerante, el nivel de combustible, todos estos dependiendo del modelo o año de fabricación del vehículo se los puede identificar por una característica forma de relojes analógicos, en otros casos en vehículos más nuevos, ya utilizan pantallas digitales de números, o también se puede encontrar una combinación de ambos.

2.1.2. Adquisición de datos para el tablero de instrumentos del automóvil

La información que el tablero de instrumentos del automóvil muestra al conductor proviene de su computador, el cual a su vez permite esa visualización a través de la compilación de los datos que proporcionan los diferentes sensores de cada sistema del auto. Es así que, por ejemplo, para el caso de la temperatura, la computadora toma la información del sensor que se ubica en el sistema de refrigeración y la convierte en datos visibles para el usuario.

2.2. Accidentalidad por distracciones en automóviles

La distracción visual y cognitiva, según la Organización Mundial de la Salud (OMS) es muy peligrosa. Un artículo publicado por el diario El Comercio, que se basa en el “Informe sobre la situación mundial de la seguridad vial 2015” de la OMS, menciona que un conductor distraído tiene una capacidad de reacción más lenta, aún más en aquellos casos en los que requiere frenar, también presenta dificultades tanto para mantener el vehículo en el carril como para guardar la distancia de seguridad entre vehículos.

En el país, según estadísticas de la Agencia Nacional de Tránsito, más de la mitad de los siniestros que ocurren en las vías se dan por la distracción del conductor, lo cual incluye el uso del celular, distraer la vista del camino por algo interno como algún indicador del tablero, entre otros.

La OMS ha determinado que la mínima distracción visual de alrededor de 3 segundos, dependiendo de la velocidad del automóvil, puede representar que se recorra entre 60 y 180 metros de camino sin tener conciencia de ello, es decir que en ese tramo no se tiene idea de lo que aparece en el camino y consecuentemente no deja tiempo a la reacción del conductor.

2.3. Dispositivos de visualización frontal

Se refiere a aquellos dispositivos en los que la visualización o proyección de la información se consigue sin que se interfiera o cambie el punto de visión habitual del usuario. Esto marca la diferencia en comparación con los sistemas convencionales, como el caso del tablero de instrumentos del automóvil, y precisamente en este ámbito de la conducción de automóviles permite disminuir e incluso eliminar aquel problema antes analizado con respecto a las distracciones que pueden ser causa de siniestros.

Con esta visión general, es posible citar ejemplos específicos para clarificar las potencialidades y usos que se ha dado a este tipo de sistemas; se tiene, por ejemplo, interfaces en el mundo de los videojuegos, en el cual es posible observar en alguna parte de la pantalla o directamente al frente del jugador, información sobre el juego o la actividad que esté realizando, sin que tenga que desviar por completo la mirada del juego; obviamente en este caso particular, la necesidad que se cubre a diferencia de la propuesta que se está desarrollando, es la de ubicar toda la “acción” en el espacio disponible de una pantalla.

Otra aplicación práctica se puede ver en la aeronáutica, y en particular en vuelos con pilotos militares, los cuales requieren monitorear variables de vuelo constantemente sin desviar su atención del rumbo u objetivo. Esta herramienta fue diseñada en los cascos que utilizan para proyectar los datos a una pantalla o a parabrisas del avión, evitando que los pilotos desvíen su atención y campo de visión.

2.3.1. Sistemas HUD

La visualización head-up o simplemente HUD (head-up display) término traducido como visualización cabeza-arriba o visualización frontal, alude a los sistemas o dispositivos que se sirven de una pantalla transparente colocada dentro del ángulo de visión del usuario para presentarle información de tal forma que éste no requiera cambiar su punto de vista para ubicarla. Se hace referencia al HUD como un sistema porque no constan solo de una pantalla, y ésta realmente es sólo la interfaz hacia el usuario en donde se muestra la información detallada sobre un suceso a un usuario, la cual ha sido previamente adquirida por diferentes medios, pudiendo ser estos sensores, etc.

En la actualidad existen ya automóviles que vienen con sistemas HUD incorporados, generalmente del segmento más alto económicamente hablando, en estos se han implementado tanto sistemas sencillos, en los que únicamente se muestran alertas puntuales, hasta los más sofisticados que se acoplan incluso a los sistemas de navegación indicando rutas y otro tipo de información.

Figura 1. Ejemplo de sistema HUD. Proyección de pantalla PIONNER Nav Gate.
Tomado de (Gabatek, s.f.)

Se puede concluir que el propósito principal de estos sistemas es ayudar al usuario a mantener su visión enfocada y recibir datos importantes de la actividad que realiza al mismo tiempo.

2.4. Las tecnologías de asistencia a conductores

Hoy en día, se usa la denominación de tecnologías de asistencia a conductores para referirse a los diferentes sistemas, generalmente electrónicos, que permiten al conductor maniobrar de manera mucho más segura el vehículo. Es así que se pueden mencionar como ejemplos, los nuevos sistemas de frenado, asistencia audiovisual para parqueo, control de velocidad, entre otros. Claro está, que estos sistemas se pueden catalogar como los más novedosos e innovadores, sin embargo, si se tiene en cuenta que la esencia de los mismos es la seguridad y confort del conductor, no quedan fuera los clásicos elementos que ya son costumbre en los autos, como por ejemplo el tablero de instrumentos.

Al enfocarnos en el tablero de instrumentos, es que se descubre que hay también una gran evolución de los modelos y métodos que presentan estos sistemas, así por ejemplo se ha pasado a indicadores digitales con pantallas

más vistosas incluso personalizables, y un paso más adelante los ya mencionados sistemas HUD, pensados como un aporte tanto a la comodidad como a la seguridad del proceso de conducción.

Básicamente se puede decir que los componentes que han experimentado cambios, es decir que han variado en el transcurso del tiempo y que se relacionan directamente con los sistemas de asistencia a conductores son:

- **Computadora del automóvil**, que es en donde los datos de los diferentes sensores del sistema son recopilados e incluso interpretados cuando se comparan con escalas predefinidas en los diferentes procesos que se encuentran programados en este elemento. Sus variables de entrada son entonces los sensores de medición de cada uno de los aspectos a valorar, como por ejemplo, el sensor de temperatura del motor, el sistema de sensores que miden las revoluciones a las que gira el motor, o también el conjunto de sensores que determinan la velocidad de movimiento del automóvil.

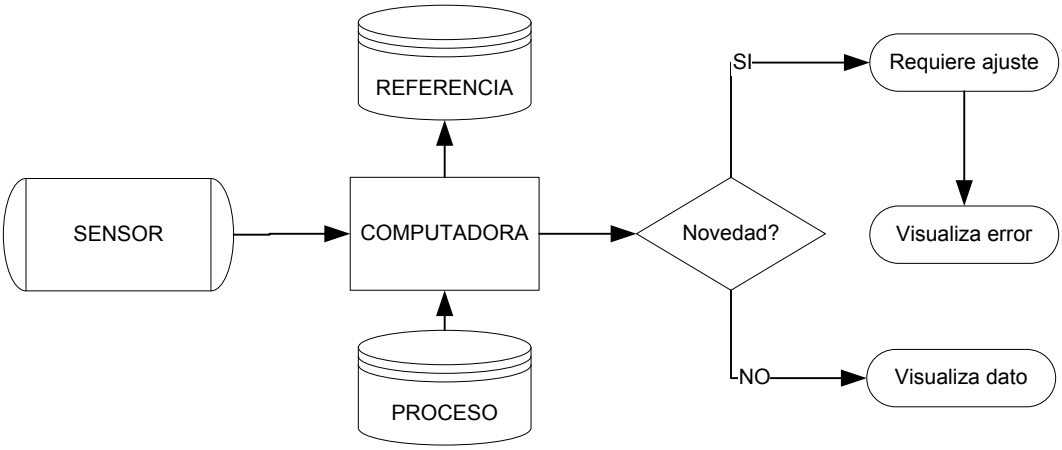


Figura 2. Proceso general que sigue la computadora de automóvil.

- **Dispositivo o método de visualización**, que en la mayoría de los casos de automóviles corresponde al tablero de instrumentos previamente descrito. Es el medio por el cual la información que ha sido procesada por el computador del automóvil, se transmite de manera visual y entendible al conductor. Usando los mismos ejemplos que para la computadora, correspondería a la aguja de la temperatura (que varía en una escala lineal), la aguja de las revoluciones (conocido como tacómetro que también se ubica en una escala a modo de reloj lineal incremental) y la aguja de la velocidad (o velocímetro, también en una escala del tipo reloj). En sistemas más avanzados como por ejemplo, los HUD, estos datos pueden ser visualizados de manera digital, es decir con valores numéricos y no necesariamente dentro de una escala.

Figura 3. Entrega de información a través del tablero

- Adicionalmente se podría incluir también al módulo que procesa la información, que si bien es cierto en gran parte de los casos es la propia computadora, se lo destaca porque desde la perspectiva del sistema de asistencia al conductor, no es precisamente un proceso más de la misma sino que puede ser manejado por un elemento adicional que hace el requerimiento de la información atendiendo a las necesidades puntuales del conductor, como ocurre en los sistemas HUD, en donde a diferencia del tablero de instrumentos, únicamente se visualiza la información prioritaria del estado del automóvil al usuario, y la misma puede ser programada en función de darle dicha prioridad. Justo en el caso de los ejemplos que se han venido citando, sería la selección de la información de temperatura, revoluciones y velocidad.

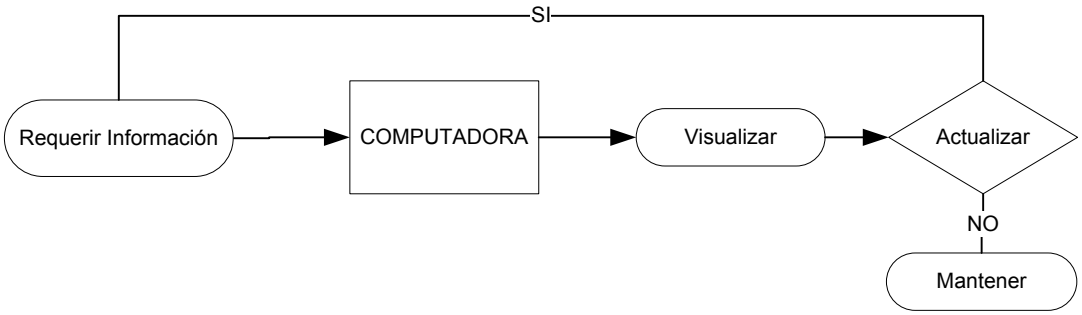


Figura 4. Proceso de visualización de información

Con estos antecedentes, se procede a hacer una evaluación de la conveniencia de la implementación de este prototipo, a través de un análisis FODA, como sigue:

Tabla 1.

FODA del proyecto

FORTALEZAS	**DEBILIDADES**
• Brinda seguridad en la conducción. • Propuesta de uso de materiales económicos y fáciles de adquirir. • Se cuenta con las herramientas, conocimiento y experiencia para dar soporte y desarrollo a largo plazo.	• La calibración de los elementos del sistema toma cierto tiempo de manera inicial. • Es necesario ir alimentando una base de datos con las configuraciones propias de cada marca de vehículo.

OPORTUNIDADES	AMENAZAS
• Posibilidad de ampliar funcionalidades para atender demandas de segmentos como el transporte comercial, por ejemplo integrando el sistema de taxímetro. • La curva de aprendizaje se estima que en el corto plazo minimiza significativamente el tiempo de producción.	• Existen dispositivos en el mercado producidos de manera industrial que cumplen funciones similares y que van reduciendo su costo • Ya existen vehículos que incorporan sistemas del mismo tipo desde la fábrica

Nota: En esta tabla se resume el resultado de un análisis tanto de factores externos como internos que influyen o pueden derivar de la implementación del proyecto.

En consecuencia, se puede determinar que la implementación de este prototipo tiene muchas más ventajas que limitantes, considerando como uno de los factores de mayor peso, las posibilidades u oportunidades que se pueden aprovechar si se toma en cuenta este prototipo como un paso inicial en un desarrollo a gran escala.

3. CAPÍTULO III. Diseño del Prototipo

Para el diseño del prototipo se determinó la necesidad de trabajar en tres etapas o módulos complementarios que son:

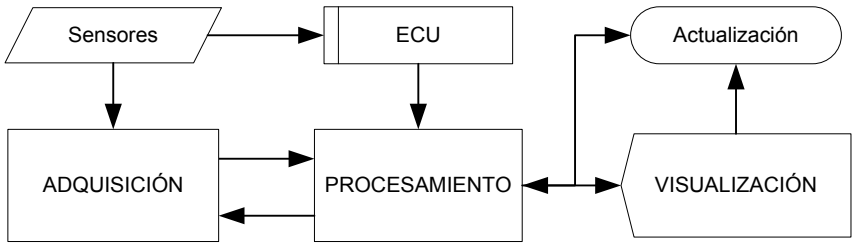


Figura 5. Diagrama de bloques inicial de la solución

3.1. Adquisición de datos

Para poder conocer la información que la computadora del automóvil recopila y genera a través de los diferentes sensores de su sistema, es necesario poder conectar un dispositivo externo al auto, situación que es característica cuando se quiere realizar un diagnóstico del funcionamiento de dicho sistema o cuando ya se ha generado una alerta de error que no se puede identificar únicamente con los indicadores del tablero de instrumentos. Para esta finalidad, existe un dispositivo cuyas especificaciones se detallan a continuación, así como algunos otros conceptos importantes que permiten ir configurando la solución.

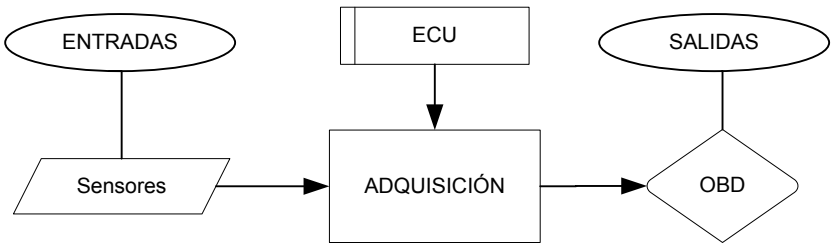


Figura 6. Diagrama de bloques del proceso de Adquisición

3.1.1. El módulo ELM327

El módulo ELM327 ha sido diseñado para actuar como un puente entre el puerto OBD-II y una interfaz serial que cumpla con el estándar RS232. El

núcleo del módulo ELM327 es el microcontrolador del mismo nombre que incorpora el programa de interpretación de los códigos generador por la ECU. La datasheet del fabricante se incorpora como Anexo 2.

En la actualidad, se exige que todos los automóviles nuevos que sean producidos dispongan de una interfaz, por medio de la cual utilizando un equipo de prueba se pueda obtener un diagnóstico con toda la información necesaria del automóvil.

Una de las características que hacen al ELM327 muy útil es la capacidad que tiene de interpretar hasta doce protocolos OBD, provee también soporte para comunicaciones de alta velocidad, un bajo consumo en modo de espera y brinda incluso la posibilidad de personalizar sus funciones.

La configuración de este módulo se la realiza mediante comandos AT y logra una tasa de transferencia de información de 500 Kbps. Con estas prestaciones, sus aplicaciones más difundidas son en el campo automotriz, para la lectura, control y manipulación de códigos de falla a través de escáner.

3.1.2. La computadora del automóvil

Es una unidad de control electrónico o ECU (sigla en inglés de Electronic Control Unit) que recopila, asocia e interpreta varios aspectos que intervienen en la operación de combustión del motor, movimiento y otros sistemas de un automóvil. Las primeras ECU que se implementaron eran exclusivas de motor y se encargaban de controlar la cantidad de combustible inyectado en cada cilindro de acuerdo con cada ciclo de motor, en los años previos al uso de las ECU éste trabajo era realizado de manera electromecánica por un carburador o bomba de inyección. Con el paso de los años y la evolución de la tecnología, se empezaron a fabricar componentes adicionales que mejoraron el funcionamiento de las ECU, y en la actualidad se ha logrado tener el control de

varios factores de la combustión como el punto de ignición, la regulación del tiempo de apertura y cierre de las válvulas, el nivel de impulso del turbocompresor (en motores a diésel), así como también el control de otros periféricos. Así mismo, se ha logrado hoy en día por medio de las ECU controlar de manera más precisa principalmente la cantidad de combustible, el punto de presión y temperatura, y varios parámetros que se encuentran monitorizados en el motor a través de sensores. Los sensores se agrupan en sensores de motor y transmisión (por ejemplo el sensor de presión del aire); sensores de seguridad (por ejemplo el sensor de airbag); y sensores de confort (por ejemplo el sensor de lluvia).

Las principales funciones de una ECU automotriz son las siguientes:

- **Control de la inyección de combustible**: mediante parámetros y sensores calcula el nivel de combustible que debe inyectar al motor según la cantidad de aire que ingrese al mismo, o disminuir el mismo según la necesidad, por ejemplo, si el acelerador está presionado a fondo, el ECU recibe esta información y permitirá que la entrada de aire al motor sea mayor; o si el motor al arrancar no ha alcanzado la temperatura suficiente, la ECU inyectará más combustible al motor hasta que el motor esté a temperatura óptima.

- **Control del tiempo de inyección**: el ECU detecta la necesidad de chispa para iniciar la combustión y éste puede ajustar el tiempo exacto de la chispa con lo que provee de más potencia al motor y un disminuye el gasto de combustible. Otro ejemplo, en vehículos de transmisión manual, cuando el motor no ha alcanzado las revoluciones necesarias para realizar un giro, el ECU impide que los pistones puedan moverse hasta que no se haya producido la chispa adecuada. En vehículos de transmisión automática, la ECU reducirá el movimiento que realiza la transmisión.

- **Control de la distribución de válvulas**: para los vehículos con distribución de válvulas, la ECU controla el tiempo en el que las válvulas deben abrirse para optimizar el flujo de aire con lo que aumenta la potencia del motor.

- **Control de arranque**: actualmente se está suprimiendo el motor de arranque, y la ECU controla el tiempo en el que se producen una inyección e ignición para arrancar el motor, lo que disminuye el consumo de combustible y eficiencia al motor.

Las antiguas ECU estaban sistematizadas para única y exclusivamente ciertas funciones, basadas únicamente en el motor. Con el desarrollo tecnológico, las ECU instaladas en casi todos los vehículos son programables, esto quiere decir, que permite leer los códigos de error y modificar parámetros según las adecuaciones al vehículo. Estás ECU se pueden monitorear, programar o mapear con una interfaz USB conectada a una computadora o escáner. La ECU usualmente monitorea:

- **Ignición**: indica a la bujía cuando debe disparar la chispa.

- **Límite de revoluciones**: Define el máximo número de revoluciones por minuto que el motor puede alcanzar. Más allá de este límite se corta la entrada de combustible.

- **Correcta temperatura del agua**: Permite la adicción de combustible extra cuando el motor está frío.

- **Alimentación de combustible temporal**: Gestiona un mayor aporte de combustible cuando el acelerador es presionado.

- **Modificador de baja presión en el combustible**: aumenta el tiempo en el que actúa la bujía para compensar una pérdida en la presión del combustible.

- **Sensor de oxígeno**: Permite obtener datos permanentes del escape y así modificar la entrada de combustible para conseguir una combustión ideal.

Algunos modelos actuales incluyen otras funcionalidades más específicas como control de salida, limitación de la potencia del motor en la primera marcha para evitar la rotura de éste, u otras como:

- **Control de pérdidas**: Configura el comportamiento del waste gate (que es como se conoce a la válvula para el control de presión del compresor del turbo, utilizado en motores a diésel) del turbo, controlando el boost (hace referencia al aumento temporal de la presión del compresor para aumentar fuerza., también en relación a motores a diésel).

- **Inyección Banked**: Configura el comportamiento del doble de inyectores por cilindro, usado para conseguir una inyección de combustible más precisa y para atomizar en un alto rango de RPM.

- **Tiempo variable de levas**: Le dice a la ECU como controlar las variables temporales en las levas de entrada y escape.

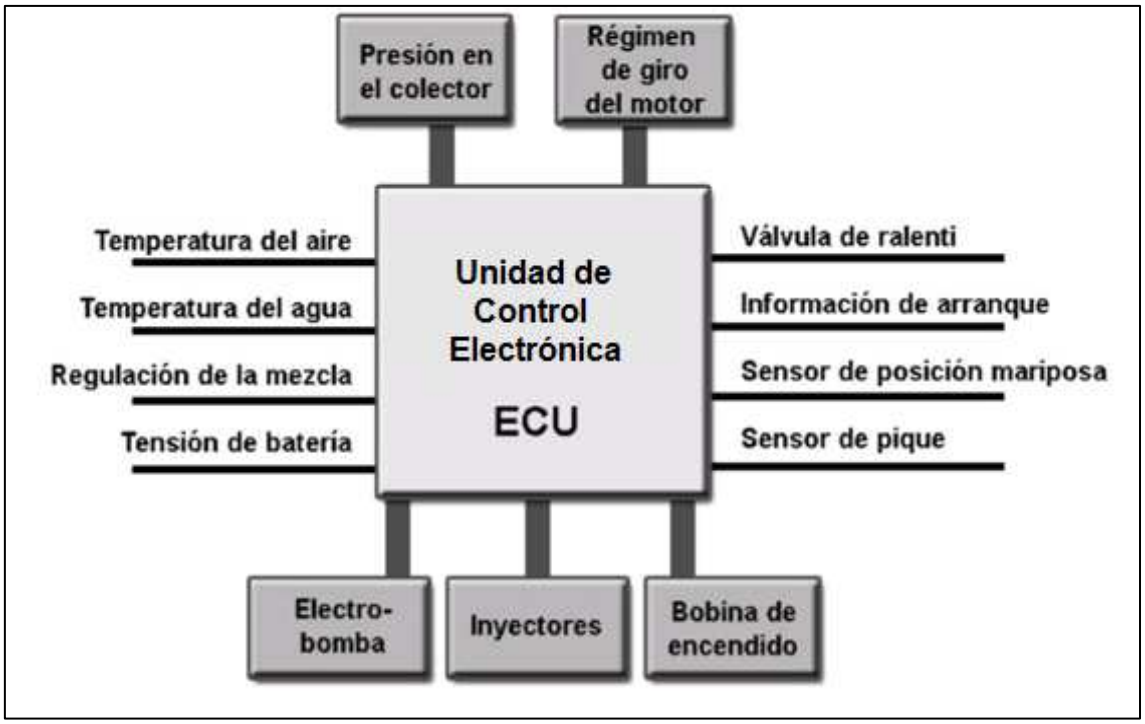


Figura 7. Diagrama de bloques de la ECU.
Tomado de Vallejo, 2012, p. 66

Así también para reprogramar o modificar una ECU actualmente se acopla un sistema de diagnóstico externo a través de un puerto interno estandarizado en la mayoría de vehículos y diseñado con este propósito.

A continuación, se describe los bloques con su funcionamiento en el circuito de una ECU automotriz:

Bloque de Entrada:

Lo constituyen todos los circuitos que se encuentran como receptores de las diferentes señales que van a ingresar a la ECU y antes de que lleguen al microprocesador. Por ejemplo, filtros, amplificadores, conversores análogos a digital, etc. Las señales que van a ingresar al microprocesador son adquiridas y tratadas por todos estos circuitos.

Bloque de Procesamiento:

Lo constituyen todos los circuitos que permiten realizar las diferentes funciones programadas. Se podría decir que es la parte lógica de la ECU, porque interpreta la información recopilada y entregada por el bloque de entrada y la procesa para realizar los diferentes procesos requeridos para el correcto funcionamiento del automóvil. Está conformado por el procesador, memorias y demás componentes involucrados en la ejecución del software.

Bloque de Salida:

Son otro conjunto de circuitos que se encuentran entre las salidas del microprocesador y los diferentes elementos actuadores, y cuya finalidad es adaptar las señales que se transmiten a dichos elementos, dependiendo de sus características particulares.

Bloque de Soporte:

Se conoce así al conjunto de componentes que tienen como función alimentar con la energía necesaria y adecuada a los circuitos internos de los diferentes bloques citados previamente.

3.1.3. OBD

Para obtener la información del ECU del automóvil se hará uso de un módulo ELM327 para conectarse al puerto OBD (On Board Diagnosis) que es la interfaz que viene instalada en los autos para poder acceder a la computadora a través de los dispositivos conocidos como escáner automotriz. Este módulo permite convertir la señal que se obtiene del puerto OBD en una señal serializada de más fácil tratamiento para una siguiente etapa del proyecto y viceversa.

Una primera versión del sistema OBD, únicamente permitía monitorizar los componentes del automotor relacionados con las emisiones, y no eran del todo útiles porque sus mediciones atendían a una calibración determinada y específica para un solo nivel de emisiones. Actualmente, a partir del año 2002 ya se usa una segunda versión del OBD, conocida como OBD-II, que utiliza los mismos protocolos y códigos, pero que incorpora muchas más variables de monitoreo del correcto funcionamiento del motor del automóvil. Además, es a la cual está enfocado el diseño del presente prototipo.

Utilizando el sistema OBD II, existen 9 modos de interacción con la ECU, y que de manera general son:

1. Obtención de datos actualizados
2. Acceso a cuadro de datos congelados
3. Obtención de los códigos de falla
4. Borrado de códigos de falla y valores almacenados
5. Resultado de las pruebas de los sensores de oxígeno
6. Resultado de las pruebas de otros sensores
7. Muestra de códigos de falla pendientes
8. Control de funcionamiento de componentes
9. Información del automóvil (Vallejo, Club Saber Electrónica, 2012)

Así mismo, se debe tomar en cuenta que la interacción por el puerto OBD utiliza diversos protocolos dependiendo del fabricante del automóvil, sin embargo, cabe señalar que los códigos que se transmiten a través de éstos protocolos, si son estandarizados. Los protocolos más usuales y sus características más importantes, se resumen en la siguiente tabla:

Tabla 2.

Protocolos de comunicación con ECU.

PROTOCOLO	FABRICANTE	COMUNICACIÓN	VELOCIDAD DE TRANSMISIÓN	VOLTAJE
SAE J1850 PWM	Ford, Lincoln y Mercury	Modulación de ancho de pulso (PWM)	41.6 Kbps	0-5 V en modo diferencial
SAE J1850 VPW	General Motors	Modulación por ancho de pulso variable (VPW)	10.4-41.6 Kbps	2.2 V – 0L 8 V – 1L
ISO 9141-2	Europeos, asiáticos, Chrysler, Jeep y Dodge	Similar al estándar RS-232	10.4 Kbps	0 – 12 V (se ajustan al voltaje de la batería)
ISO 14230 KWP (Key Word Protocol)	Europeos y asiáticos	Similar al estándar RS-232	1.2 - 10.4 Kbps	0 – 12 V (se ajustan al voltaje de la batería)
ISO 15765 CAN	Compañía Bosch	Red de área del controlador	250 - 500 Kbps	2.5 – 5 V (CANH) 2.5 – 0V (CANL)
SAE J1939	Volvo, Renault	Red con Unidad de Datos de Protocolo (PDU)	250 - 500 Kbps	0-5 V en modo diferencial

Adaptado de Simbaña, 2016, p. 14

En cuanto a los citados códigos, un listado de los más utilizados de ellos, que son más conocidos como PID (del inglés Process Identification), se incluye en el Anexo 1.

3.1.4. Conector DLC

Físicamente el puerto OBD-II que permite conectar una interfaz para acceder a la computadora del automóvil, es un conector de diagnóstico conocido como DLC (Data Link Conector), el cual está basado en el estándar SAE J1962 y es aceptado por todos los fabricantes de automóviles, consta de 16 pines y

físicamente presenta una característica forma trapezoidal; generalmente se encuentra ubicado por debajo del tablero de instrumentos y en el lado del conductor, en otros casos puede estar ubicado debajo del cenicero (vehículos asiáticos y europeos). A través de este puerto es posible acceder y recuperar datos de la ECU para a su vez poder mostrarlos en un equipo de diagnóstico como lo es por ejemplo un escáner automotriz. Por otro lado también permite ingresar códigos de control a la ECU, con la finalidad de manipular los datos en aplicaciones mucho más especializadas.

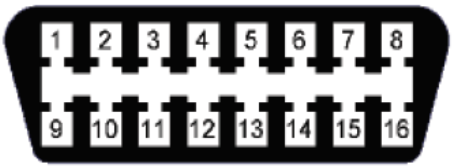


Figura 8. Conector DLC.
Tomado de Simbaña, 2016, p. 12

El uso de cada uno de los pines del conector DLC es el que se muestra en la siguiente tabla:

Tabla 3.
Descripción de pines del conector DLC

PIN	USO
1	Uso del fabricante
2	Bus (+) J1850 VPM (Variable Pulse Modulation) y PWM (Pulse Width Modulation)
3	Uso del fabricante
4	Tierra (chasis)
5	Señal de tierra
6	Bus de datos CAN (Controller Area Network) alto (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 bajo (J-2284)
15	Línea L ISO 9141-2
16	Voltaje de batería

Adaptado de Simbaña, 2016, p. 13

En base a la tabla precedente, la configuración de los pines se mostraría gráficamente de la siguiente manera:

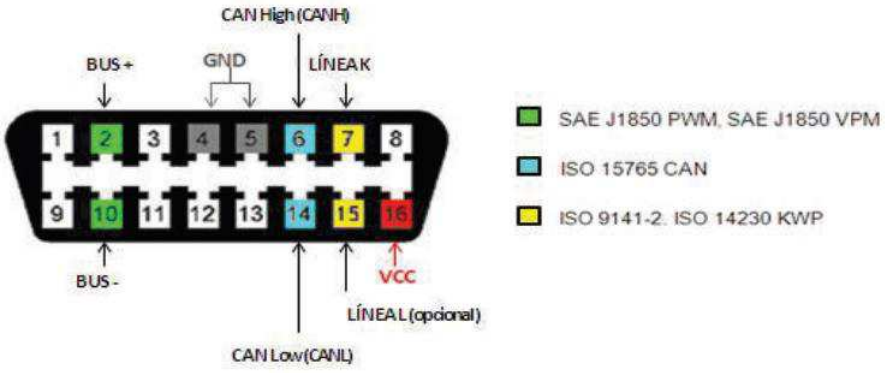


Figura 9. Configuración de pines DLC.
Tomado de Simbaña, 2016, p. 14

3.1.5. Mensajes en OBD-II

La trama de los mensajes que se transmiten y reciben a través del puerto OBD-II tienen un número de bytes para la cabecera y otro para la información, en donde dicha longitud viene determinada por el protocolo del que se esté haciendo uso, siendo una especificación para el J1850 e ISO y otra para el CAN.

El formato del mensaje para el protocolo J1850 e ISO, comprende 3 bytes de cabecera al inicio de la trama (incluye los parámetros de prioridad, receptor y transmisor), un máximo de 7 bytes de datos y 2 bytes de checksum (suma para la verificación de integridad de datos) al final de dicha trama.

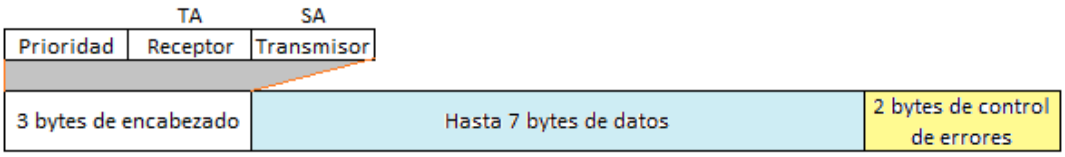


Figura 10. Estructura de la trama del mensaje.
Adaptado de Simbaña, 2016, p. 15

Para este caso, cada uno de los parámetros que conforman el mensaje tienen la siguientes funciones:

- Prioridad: información sobre la importancia del mensaje
- Receptor: indica la dirección de destino del mensaje
- Transmisor: indica la dirección del origen del mensaje
- Datos: es la información que se envía, ya sea un parámetro o petición
- Control de errores (Checksum): detecta errores en el envío, para solicitar retransmisión del mensaje de ser necesario.

Por otro lado para el protocolo CAN, el formato del mensaje guarda una estructura similar, con la única variación del encabezado. Los bytes de cabecera son llamados ID bits, los cuales en el estándar CAN pueden ser 11 o 29 bits.

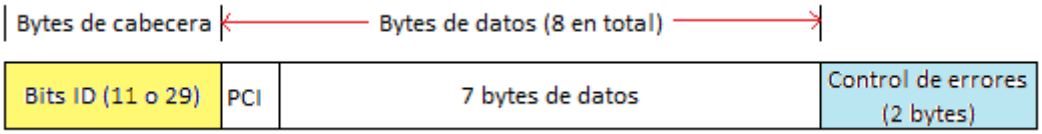


Figura 11. Formato de trama de mensaje CAN.
Adaptado de Simbaña,2016, p.16

En este caso, cada uno de los parámetros que conforman el mensaje tienen la siguientes funciones principales:

- Bits de identificación (ID bits): indican la información sobre el tipo de mensaje, prioridad del protocolo CAN utilizado, dirección destino y dirección fuente.
- PCI: proporciona la longitud en bytes del campo de información de la trama. Generalmente el valor es de 7 bytes
- El campo de datos y control de errores cumplen el mismo funcionamiento del formato del mensaje anterior (del protocolo ISO)

3.1.6. Uso del módulo ELM327

Este módulo tiene varias presentaciones comerciales, cuya diferencia una de otra es en el tipo de tecnología para realizar el puente con la interfaz serial, por esta razón es necesario realizar una comparación para determinar qué opción es la más recomendable para los fines buscados.

Tabla 4.

Tabla comparativa de tecnologías de enlace

CARACTERÍSTICA	ZIGBEE	BLUETOOTH	WIFI	USB
Estándar	IEEE 802.15.4	IEEE 802.15.1	IEEE 802.11 b/g/n	USB 3.0
Frecuencia	2.4 GHz	2.4 GHz	2.4 / 3.6 / 5 GHz	N/A
Velocidad Transmisión	250 Kbps	3 Mbps	1 Gbps	5 Gbps
Consumo	30mA	40mA	200mA	1000mA
Consumo (espera)	23uA	200uA	50mA	150mA
Alcance	100m	10m	300m	N/A
Topología	Ad-hoc, estrella, malla	Ad-hoc	Ad-hoc, estrella, malla	Punto - punto
Configuración	Fácil	Media	Media	Fácil
Aplicaciones	Sensores	Dispositivos móviles	Redes inalámbricas	Conexión directa

Nota: En esta tabla se muestran las características principales de cada tecnología, con la finalidad de poder facilitar una comparación entre las mismas.

Adaptado de Simbaña, 2016, p. 46

De la evaluación realizada en la tabla anterior, se puede concluir que la conexión a través del puerto USB con cable serial es la más destacada por las características superiores que representa en cuanto a confiabilidad y velocidad de los datos transmitidos, así como su fácil configuración. Sin embargo, considerando las particularidades del proyecto en cuestión, se ha optado por utilizar la tecnología bluetooth por su mínimo consumo, su adecuada velocidad de transmisión, alcance y topología para el caso del prototipo en desarrollo. Además otras de las ventajas es su reducido costo, la casi universal compatibilidad con otros dispositivos y de hecho la eliminación de interfaces

físicas que se constituyan en un problema para el espacio de aplicación del proyecto, que es el interior del automóvil.

3.1.7. Requerimiento de información

Es necesario la revisión de opciones para la adquisición y procesamiento de los datos del prototipo planteado, para el efecto se han considerado de manera prioritaria, atendiendo a las capacidades de escalabilidad y compatibilidad demostrada con dispositivos del tipo de sensores y actuadores (en aplicaciones domóticas por ejemplo) dos posibilidades, que son las siguientes:

La plataforma ARDUINO

El proyecto “Arduino” inició en el año 2005 como un proyecto para estudiantes a razón del cierre del Instituto de Diseño Interactivo IVREA, ubicado en Ivream (Italia). El nombre Arduino hace referencia al sitio en el que se reunían los integrantes del equipo fundador, Bar di Re Arduino, llamado así en honor del Rey Arduino que vivió en el año 1002. Arduino es una plataforma electrónica desarrollada con software y hardware libre y abierto, de bajo costo, del tipo plug and play (conectar y usar), que se constituye de una placa con un microcontrolador Atmel AVR, que brinda un acceso modular a sus puertos y que gracias a un entorno de desarrollo sencillo permite la creación de programas de manera rápida, ya que originalmente fue concebido como un sistema académico en arquitectura abierta (Reyes Cortés & Cid Monjarráz, 2015, p. 12). Las opciones de aplicación son múltiples, ya que se puede leer información tanto digital como analógica de variables como la temperatura, presión, humedad, a través de los sensores adecuados, en base a lo cual se puede controlar otros dispositivos electrónicos como motores, actuadores o simplemente visualizar información en pantallas.

El lenguaje de programación para Arduino se basa en Wiring (una plataforma desarrollada para programar el microcontrolador) y el entorno de desarrollo se basa en Processing, que como se mencionó previamente, es libre y gratuito.

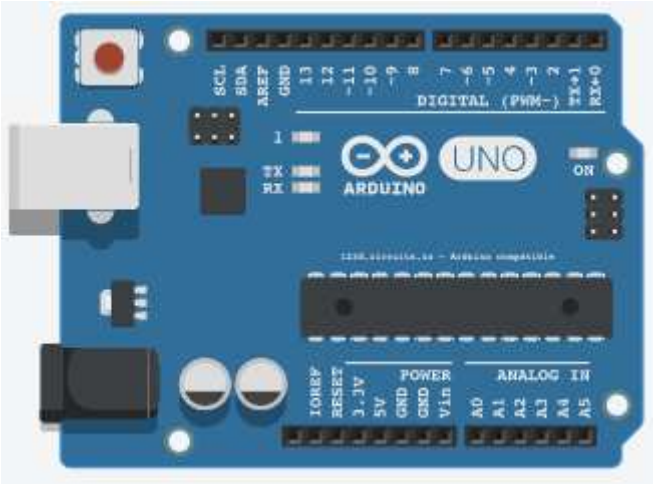


Figura 12. Placa Arduino UNO.
Tomado de Autodesk, s.f.

Actualmente existen varias opciones de placas Arduino, acorde a la necesidad del usuario, que básicamente difieren en capacidad de memoria, número de entradas y salidas, tipos de interfaz y otras funcionalidades adicionales como botón de reset (reinicio completo de la Placa).

Es importante también resaltar que la placa Arduino tiene flexibilidad para el uso de lo que se conoce como shield, que son módulos individuales que permiten ampliar funcionalidades al dispositivo en su versión original, por ejemplo se tiene shield para añadir conectividad bluetooth, wifi, conectar displays, lcd, TFT, etc

RASPBERRY PI

Es un módulo programable con aplicaciones similares a Arduino, sin embargo cumple con muchas más características que lo ubican en un segmento más cercano al de una computadora pues permite un procesamiento de información mucho más a detalle, no sólo cumpliendo un proceso basado en instrucciones simples, sino también realizando cálculos y tratamiento de información mucho más elaborado.

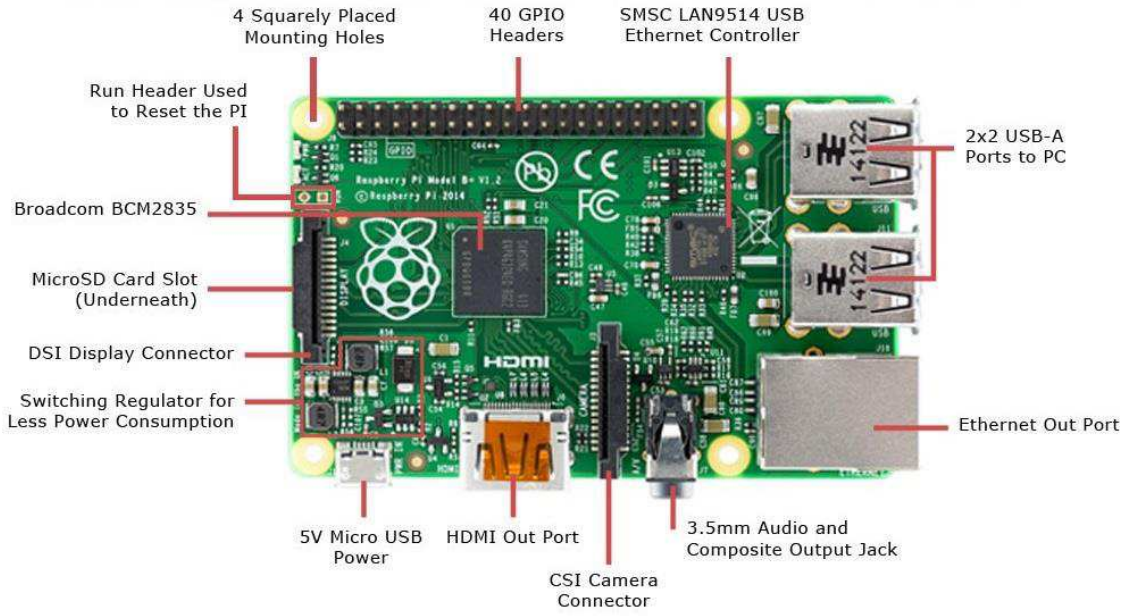


Figura 13. Placa de un Raspberry PI.
Tomado de Crespo, 2017

Sus características principales son las de poseer un microprocesador de 700Mhz, con memoria RAM de 256 MB, una potencia mínima de consumo de 3.5W y la presencia de puertos para dispositivos de red, VGA, USB. Además cuenta con un sistema operativo multitarea Linux.

En base a la conceptualización previa hecha sobre la plataforma Arduino, en el sentido de que permite obtener información y posteriormente visualizarla, es

que se ha seleccionado a la misma para cumplir con esta función en el proyecto que se está diseñando. Sin embargo es importante mostrar que dado que también se consideró la opción de utilizar Raspberry Pi, se realizó la siguiente tabla comparativa, para elegir la alternativa más adecuada:

Tabla 5.

Comparativo de Raspberry con Arduino

CARACTERÍSTICA	RASPBERRY	ARDUINO
Procesamiento	Procesador 386. 1.2Ghz	Micro ATMEL
Memoria RAM	1 GB	
Tipo	Miniordenador	Microcontrolador
Interfaces	Limitado - USB	Variado – múltiples shields
Alimentación	5V	5V
Consumo	2A	0.5ª
Consumo (espera)	500A	200uA
Configuración	Fácil	Fácil
Aplicaciones	Procesamiento, requiere inicializar mediante Sistema Operativo	Interfaces, actuadores, control. Ejecuta aplicaciones con sólo conectar.

Nota: En esta tabla se resume las principales características de cada uno de los tipos de placas considerados para la implementación del presente prototipo.

Con esta premisa es que se ha seleccionado y usa la plataforma Arduino para generar el requerimiento de información a través del módulo ELM327 a la computadora del automóvil, y a su vez, cuando esta información sea recibida se la mostrará a través del dispositivo de visualización. El proceso en sí

requiere el envío de un código que representa la solicitud de datos, y una vez realizado es necesario esperar la respuesta de la computadora.

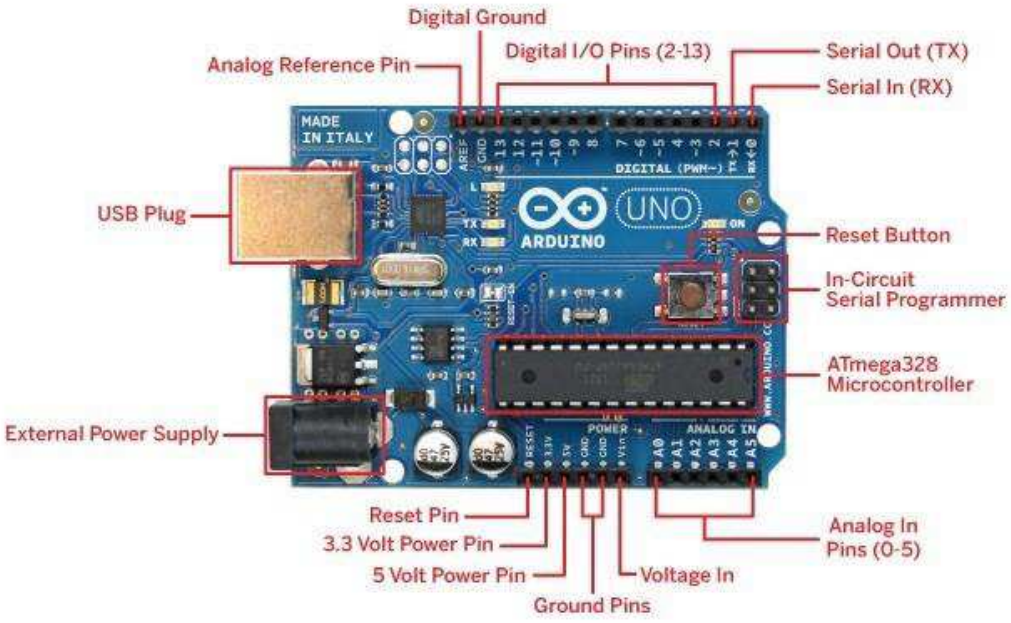


Figura 14. Arduino UNO y sus partes
Tomado de Crespo, 2017

A través de este método se va a obtener de manera inicial la velocidad del automóvil, las revoluciones del motor y la temperatura del mismo.

Arduino permite la interconexión del dispositivo de adquisición de datos utilizando un shield que incluye una interfaz bluetooth, y entregar la misma a través de una pantalla que haga legible la información compilada, básicamente se sigue el siguiente diagrama de bloques:

Figura 15. Diagrama de bloques de adquisición de datos

3.2. Procesamiento

En una siguiente etapa se realiza el requerimiento y procesamiento de la información obtenida de la ECU del automóvil. Para ello se utilizará el módulo Arduino con interfaz bluetooth.

En este módulo se debe crear un programa para lectura cíclica de códigos, el cual hace requerimiento, recibe respuesta, la almacena y en conjunto la transmite al siguiente módulo para su visualización.

En base a lo analizado en el capítulo precedente, se va a trabajar con el modo 1 del sistema OBD-II para la medición de los parámetros que se va a visualizar.

El modo 1 como ya se había citado, permite acceder a las medidas producidas por los sensores del motor y demás sistemas del automóvil en tiempo real. Para obtener las medidas es necesario enviar una solicitud a la ECU a través de un código PID, y posteriormente esperar a que la ECU envíe la correspondiente respuesta.

Las medidas que se pueden obtener a través de este modo incluyen: estado de combustible, temperatura del refrigerante del motor, presión del múltiple de admisión, RPM del motor, velocidad del automóvil, temperatura del aire de admisión, posición absoluta del acelerador, entre otros (Simbaña, 2015).

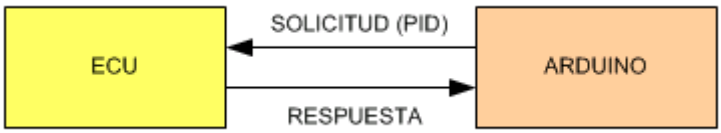


Figura 16. Procesamiento de Información

3.3. Visualización

Para poder mostrar la información al conductor del automóvil se plantean diferentes opciones, siendo las que más se acoplan a las necesidades particulares del presente proyecto, las que se analizan a continuación:

3.3.1. Matriz de Led

Figura 17. Matriz de Leds

Una matriz de led no es más que un arreglo de leds consolidado en un solo dispositivo. En el mercado actual se los puede encontrar en diversas dimensiones que se elige de acuerdo a la aplicación que se le quiera dar, así mismo están disponibles en uno o en varios colores.

Su costo es bajo, y existe incluso la posibilidad de construir un arreglo con varios módulos.

El principal inconveniente para esta opción es que su manejo o control requiere de programas mucho más elaborados que otras opciones, dado que en la generalidad de los casos, se controla de led en led el encendido o apagado. Esta particularidad implica que se le dedique más tiempo a la programación, además de que el programa requiere más recursos en cuanto a memoria y procesamiento, así como la necesidad de un mayor número de pines para su conexión y control.

3.3.2. Pantalla LCD

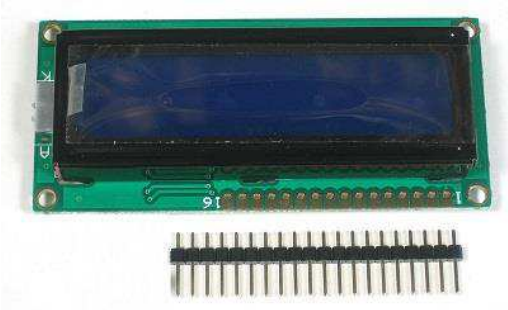


Figura 18. Pantalla LCD

La pantalla LCD (Liquid Cristal Display) funciona de tal manera que se activan los segmentos que tienen un componente líquido aprisionado entre dos cristales, con el voltaje aplicado. Sus aplicaciones son muy difundidas pues básicamente fueron la evolución en los equipos de video como las televisiones desde el tradicional tubo de rayos catódicos. En su versión más simple se las utiliza en dispositivos como relojes pequeños, calculadoras básicas, o como displays de datos sencillos. El costo es relativamente bajo y su configuración no implica mayor complejidad.

Para el caso de Arduino es posible conseguirlo en una shield para adaptarlo a cualquier proyecto. El costo es relativamente bajo y su configuración no implica mayor complejidad. El inconveniente que se presenta en la utilización de esta opción está en el tamaño que se añade al módulo que contiene la placa Arduino.

3.3.3. Pantalla TFT

Figura 19. Pantalla TFT

La pantalla TFT utiliza básicamente una tecnología LCD, con el valor añadido de que permite la interacción táctil del usuario con lo que se presenta a través de ella. La aplicación más difundida son los teléfonos móviles de nueva generación.

También se los puede conseguir en el mercado como una shield para Arduino, con un nivel de configuración de dificultad media, sin embargo su limitante es el precio elevado con respecto a las otras opciones.

3.3.4. Interfaz Teléfono Móvil

Una opción que se ha considerado posible para el prototipo, es el uso de la pantalla del teléfono celular inteligente como interfaz para la presentación de los datos. Para el efecto, una de las mayores ventajas es la disponibilidad de este tipo de equipos, lo cual representaría un costo directo para el prototipo de cero, pues se presume como un componente previamente disponible.

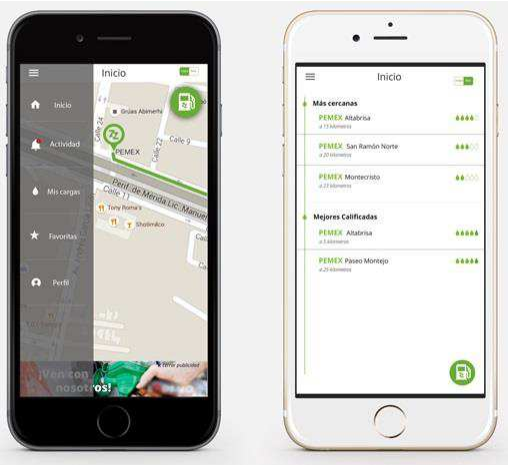


Figura 20. Uso de pantalla celular a través de aplicación Android

Para poder utilizar esta opción es necesario el diseño e implementación de una aplicación en Android para poder darle al teléfono la funcionalidad de proyectar la información generada por la ECU, con la particularidad ya mencionada de que esta debe tener un sentido invertido de tal forma que por reflexión en el parabrisas se logre el propósito ya establecido. Sin embargo se convierte en la opción elegida por el costo prácticamente nulo que representa al proyecto y la versatilidad de uso de diferentes dispositivos únicamente utilizando la aplicación desarrollada.

En este aspecto, es importante citar que se conoce como Android al sistema operativo utilizado por la mayoría de teléfonos inteligentes de la actualidad. De

manera similar y en este caso, también hace referencia a los programas desarrollados para funcionar en esta plataforma.

3.4. Prototipo

Una vez consideradas y evaluadas las diferentes opciones para las diferentes fases del prototipo, se determina que las características finales del prototipo en elaboración son las siguientes:

Tabla 6.
Componentes del Prototipo diseñado

ETAPA	CARACTERÍSTICA	DETALLE
ADQUISICIÓN	DISPOSITIVO	Módulo ELM327 bluetooth
	CONECTIVIDAD	Conexión física del módulo al puerto OBD-II del automóvil
	FUNCIONALIDAD	Interfaz para transmitir los requerimientos y recibir los datos generados por el automóvil
PROCESAMIENTO	DISPOSITIVO	Placa Arduino UNO con shield Bluetooth del tipo esclavo y otra del tipo maestro.
	CONECTIVIDAD	Conexión vía bluetooth con el módulo ELM327 y por método similar con el dispositivo celular. Usa cable USB para energizarse.
	FUNCIONALIDAD	Plataforma que contiene y ejecuta el programa que establece las conexiones por vía bluetooth tanto con la interfaz de adquisición de datos como con la de visualización de información. En el mismo se genera los requerimientos hacia el

		computador del automóvil y se procesa las respuestas recibidas, organizando la información para poder ser visualizada.
VISUALIZACIÓN	DISPOSITIVO	Teléfono móvil inteligente con sistema operativo Android
	CONECTIVIDAD	Conexión vía bluetooth con la placa Arduino
	FUNCIONALIDAD	Visualizar por medio de la aplicación desarrollada para el efecto, la información generada por la computadora del automóvil y procesada por la placa Arduino, que le es transmitida por la interfaz bluetooth.

Nota: Resumen de las características que debe tener el prototipo.

Con base en las diferentes etapas que se ha determinado que conforman el prototipo, se procede a establecer el flujo del proceso que deberán cumplir las mencionadas etapas para obtener los resultados esperados:

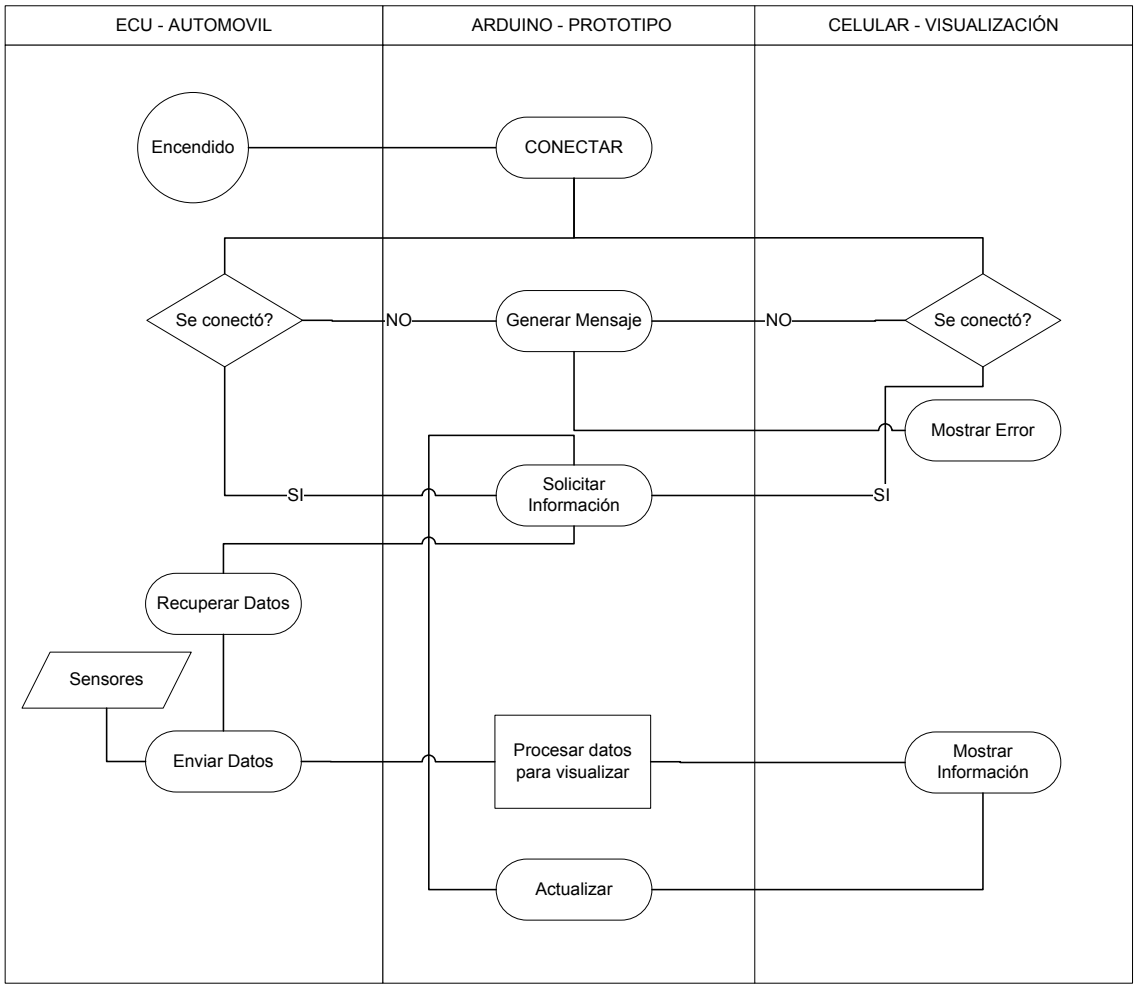


Figura 21. Flujo del proceso de los diferentes módulos que forman parte del prototipo.

4. CAPÍTULO IV. Implementación y pruebas del Prototipo

Para la construcción del prototipo se distinguen dos partes importantes, una a nivel de hardware y otra en cuanto a software. Para el efecto es necesario señalar la concepción gráfica del proyecto, que se muestra a continuación:

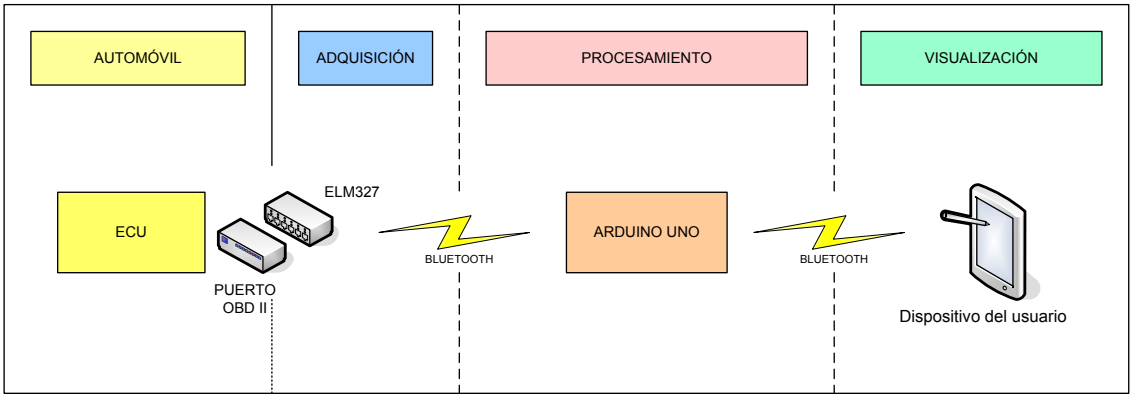


Figura 22. Esquema de bloques del prototipo diseñado

Tanto la fase de procesamiento de los datos como la de visualización requieren que se programe aplicaciones para los respectivos dispositivos, es decir, para el Arduino y para el teléfono móvil. En la figura que se muestra a continuación se representa de manera gráfica esta necesidad:

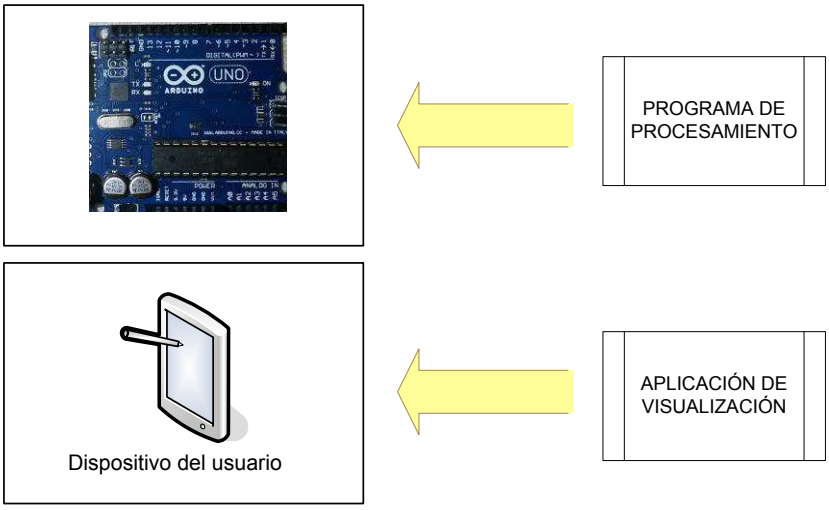


Figura 23. Programación necesaria

4.1. Programación para el Arduino

En el caso del Arduino, es necesario programar el procesamiento de los datos recopilados a través del ELM327 por el puerto OBD II. Para ello, la aplicación

que se utiliza es Arduino UNO versión 1.8.5, que es una distribución que se obtiene de la página oficial de Arduino.

El programa va a permitir:

- Establecer la conexión vía bluetooth con el ELM327
- Establecer la conexión vía bluetooth con el dispositivo móvil del usuario.
- Requerir los datos de la ECU del automóvil
- Ordenar la información de respuesta del ECU del automóvil para transmitirla al dispositivo móvil.

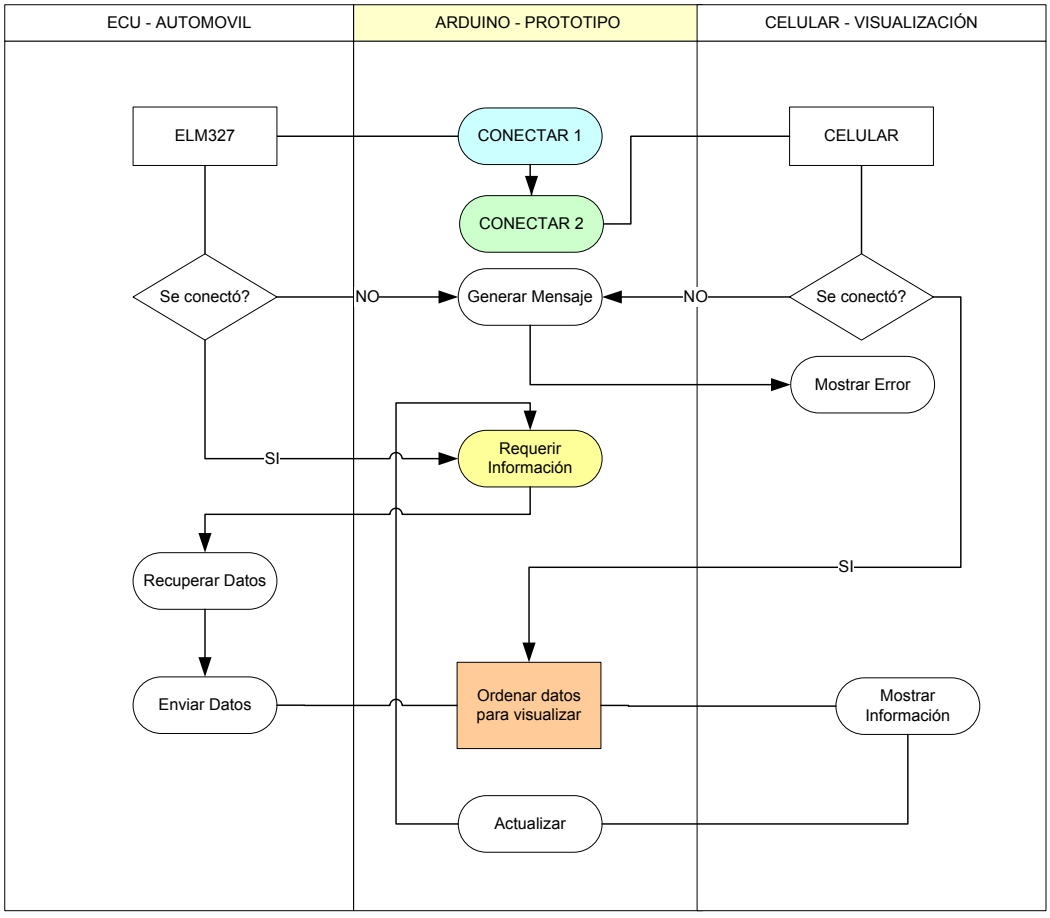


Figura 24. Programa de Arduino

La programación del Arduino se realiza en un solo archivo con extensión .ino que debe ser cargado a través del puerto serial en el módulo respectivo. Dicho archivo, como ya se había señalado, se desarrolla en un entorno Java, por lo cual guarda la estructura básica de los archivos de código de este lenguaje, es decir:

- Llamado a las librerías a utilizarse
- Definición de la clase, sus variables e inicialización
- Definición de métodos y funciones de la clase
- Desarrollo de la función principal

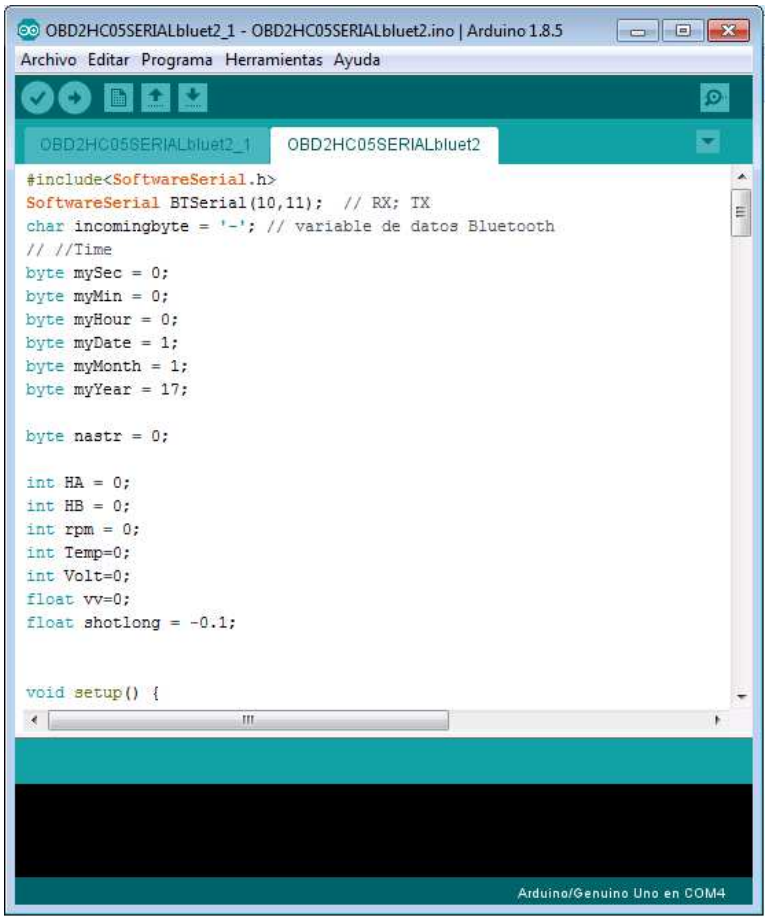


Figura 25. Pantalla de desarrollo de Arduino

El código completo se encuentra en los Anexos del presente trabajo.

4.2. Programación para el dispositivo móvil

En lo que refiere al dispositivo móvil, es necesario desarrollar un programa para la visualización de la información que el Arduino transfiere vía bluetooth.

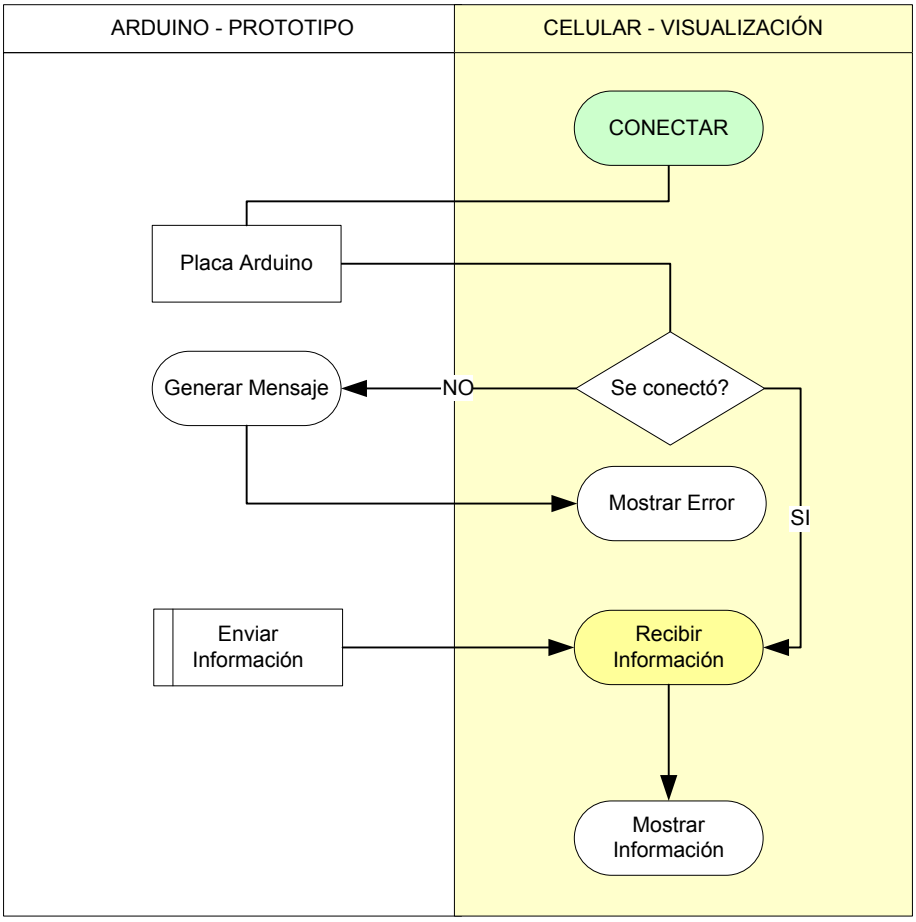


Figura 26. Programa en Android para Dispositivo Móvil

La aplicación debe ser desarrollada en Android, para lo cual se utilizó el programa Android Studio en su versión 3.0.1.

El lenguaje para el desarrollo de la aplicación Android es Java, por lo tanto al igual que con Arduino, mantiene la misma estructura de los archivos del programa. Adicionalmente Android Studio tienen una interfaz gráfica que permite diseñar el entorno de la aplicación Android de manera mucho más intuitiva, similar a otros entornos como Visual Studio.

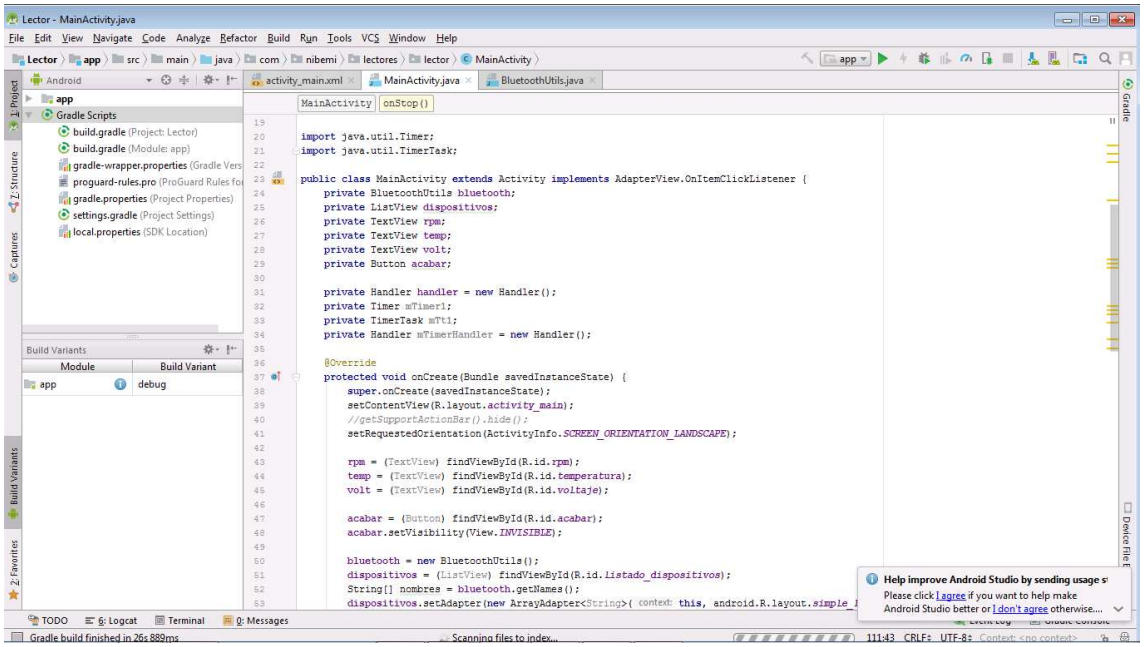


Figura 27. Entorno de desarrollo de Android Studio

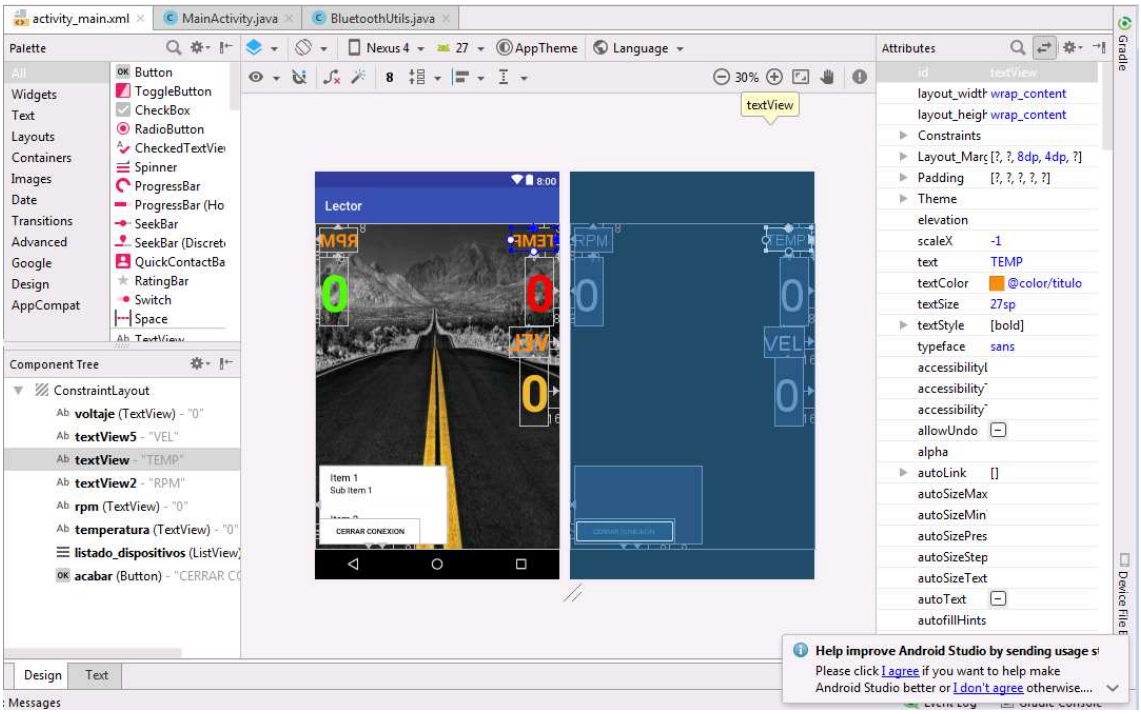


Figura 28. Entorno de diseño de interfaz gráfica de la aplicación Android

El código completo de la aplicación que visualiza los datos enviados vía bluetooth por el Arduino, se encuentra en los Anexos del presente trabajo.

4.3. Esquema de conexión

ara el funcionamiento del Arduino de la manera planificada, es necesario conectar los módulos o shields de conectividad bluetooth.

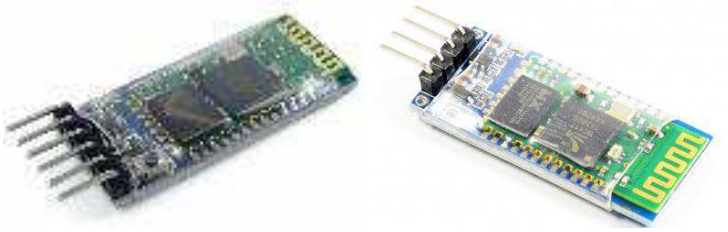


Figura 29. Shield bluetooth HC05 y HC06 respectivamente

Estos módulos requieren ser alimentados con 5V y conectar la los pines de recepción y transmisión. El módulo HC06 será el que se va a utilizar como esclavo para transmitir la información al dispositivo móvil, mientras que el HC05 funcionará como maestro para poder solicitar y enviar datos a la ECU del automóvil a través del ELM327. El esquema de conexión es el siguiente:

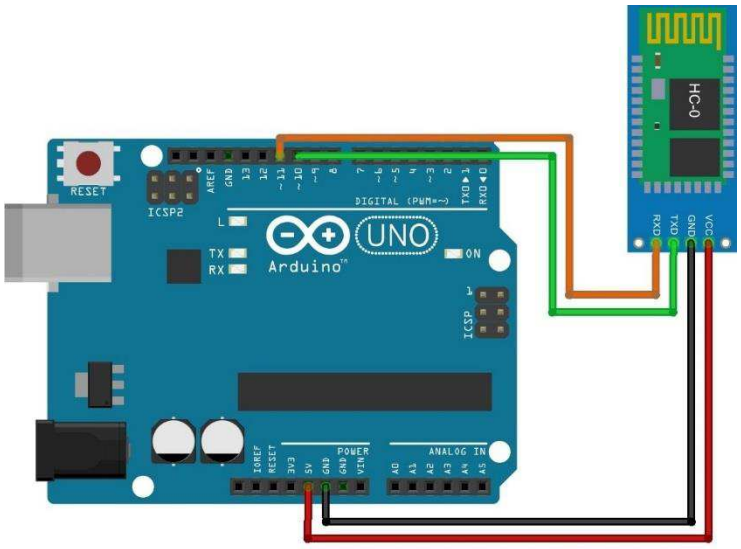


Figura 30. Diagrama de conexión del Shield HC05

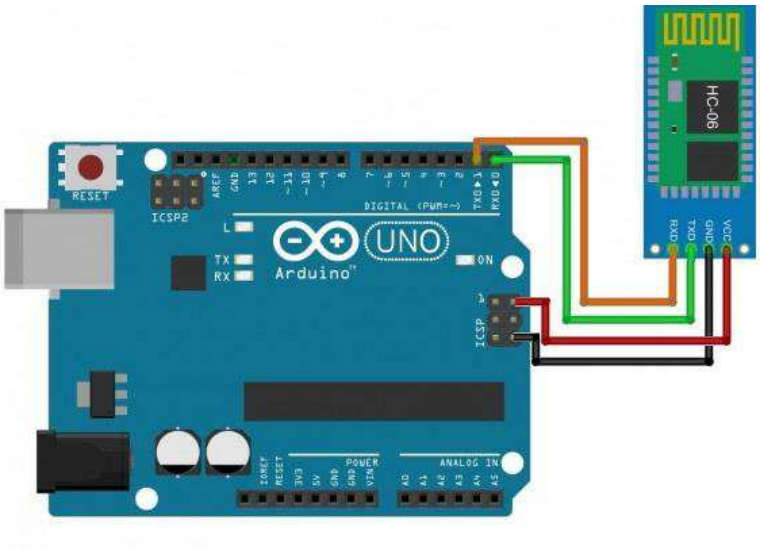


Figura 31. Diagrama de conexión del Shield HC06

Como resultado de las conexiones físicas se tiene la siguiente figura:

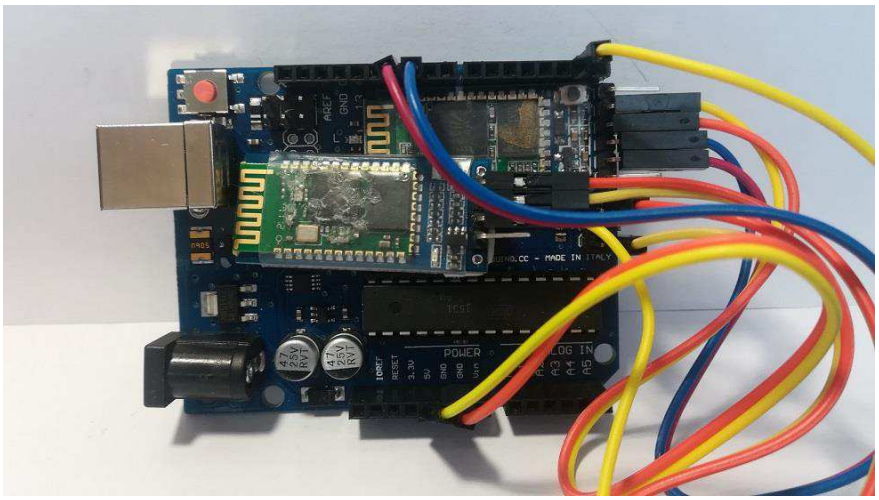


Figura 32. Arduino con los shields bluetooth conectados

Para mantener las conexiones y proteger el módulo, se consideró necesario incrustarlo en un case, que viene prefabricado para la placa Arduino, en plástico transparente, quedando finalmente con el siguiente aspecto:

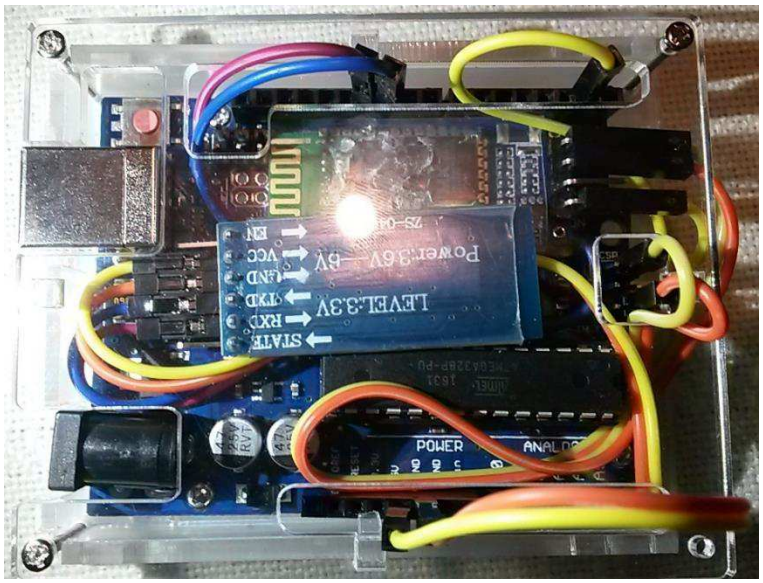


Figura 33. Arduino incrustado en un case plástico para protección

4.4. Pruebas del prototipo

Una vez realizadas las conexiones físicas y cargado el programa en el Arduino, así como instalada la aplicación Android en el dispositivo móvil, se procede a realizar las pruebas respectivas para lo cual, es necesario instalar el prototipo en el automóvil para la observación de su funcionamiento.

En primer lugar se instala el Arduino en un lugar en el que tenga acceso a energía mediante un puerto USB, y de tal manera que no corra peligro de moverse o ser manipulado o expuesto a algún daño accidental.

Figura 34. Instalación del Arduino cerca de una fuente de energía

A continuación se identifica el puerto OBD II del automóvil, y se coloca en él, el módulo ELM327. Generalmente este puerto se encuentra en el lado del conductor y debajo del volante.

Figura 35. Localización del puerto OBD II.
El punto de vista que muestra la fotografía es desde abajo del volante.

Figura 36. Conexión del dispositivo ELM327 en el puerto OBD II

Finalmente en el dispositivo móvil del usuario, se accede a la aplicación denominada Lector, con la conectividad bluetooth activada. En el primer uso pedirá una clave para establecer la conexión entre el dispositivo y el módulo HC06, por defecto ésta es “1234”.

Figura 37. Pantalla de visualización del prototipo implementada en el teléfono móvil

Como se puede observar en la imagen anterior, la pantalla muestra la información invertida, esto es hecho a propósito considerando que la intención es mostrar la información en el parabrisas del vehículo, lo cual se consigue por reflexión, misma que se logra ubicando el teléfono móvil en la parte plana y horizontal que tiene la consola frontal al conductor y pasajero, por donde generalmente se encuentran las rendijas de salida de aire para el parabrisas delantero.

En la figura a continuación, se muestra el principio de reflexión aplicado, en donde la imagen generada en el teléfono móvil es visible en una superficie colocada con cierto grado de inclinación por sobre el mismo. Es notorio además, que la información que se visualiza en dicha superficie, y que para el caso particular de este proyecto es el parabrisas del automóvil, se puede interpretar adecuadamente pues la imagen ya no se encuentra invertida como lo está en su generador (el dispositivo móvil).

Figura 38. Vista del reflejo de la aplicación Android

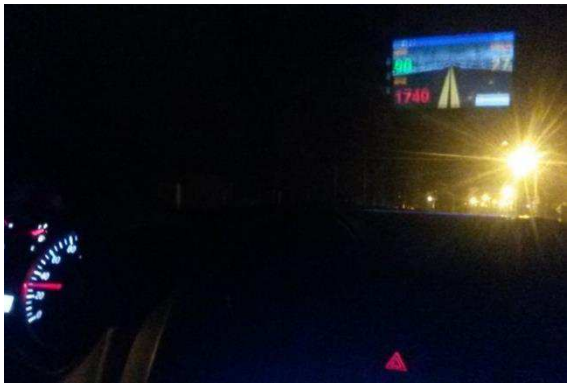
A continuación se muestran los resultados de las pruebas realizadas con el automóvil en movimiento, haciendo una consideración importante para la interpretación de las imágenes, que es la de establecer que los indicadores del tablero del automóvil utilizado para las pruebas son del tipo análogo, con agujas y escala discreta, lo cual hace casi imposible determinar valores precisos, sin que esto desvirtúe las pruebas porque el rango de tolerancia de error en la interpretación del usuario no va más allá del ±5%:

Tabla 7.

Matriz de pruebas aplicadas al prototipo

Variable	Prototipo	Tablero	Imagen
Velocidad (km/h)	11	10	
	29	30	
	44	45	

	1026	1000	
Revoluciones (rpm)	2306	2300	
	1999	2000	
Temperatura (grados C)	5	~0 mínimo	

	90	80-90 medio	

Nota: La tabla muestra los valores obtenidos para cada variable considerada en este proyecto, tanto a través de los instrumentos del propio vehículo como del prototipo. En la imagen se ha capturado estas dos fuentes de información para cada uno de los casos.

Cabe señalar que en lo que respecta a la variación de la temperatura del automóvil, esta no experimenta cambios significativos en un recorrido normal, y el indicador del tablero tiene una escala de tres estados nada más lo cual hace muy complicado el hacer un análisis de dicha variación, sin embargo el resultado que muestra el prototipo es satisfactorio porque se asocia al nivel de temperatura medio en la escala del indicador que es el valor relativamente constante y adecuado para el funcionamiento normal del automóvil, cuando éste ya está en marcha, y en el caso de arranque en frío, mientras el indicador del tablero se ubica en la mínima posición, el prototipo señala un valor de 5 °C. Así mismo, se conoce que la temperatura de trabajo normal de un automóvil oscila entre los 80 y 90 grados centígrados, según indica el manual de servicio del mismo. Esto representa la mitad de la escala en el indicador del tablero y la concordancia con los valores obtenidos a través del prototipo.

Así mismo, se ha podido observar que la variable de velocidad es congruente con los cambios inducidos en la conducción del automóvil durante las pruebas, la precisión del prototipo es mucho mayor que la del indicador del tablero, lo

cual es una evidente ventaja, pero da un mínimo margen de percepción subjetiva al determinar el valor que el tablero muestra para establecer la comparación con el prototipo.

Algo similar ocurre con las revoluciones del motor, que si bien el indicador del tablero tiene un mayor detalle, no llega a la precisión del prototipo, y sin embargo se denota una gran correlación entre las dos variables.

Como información relevante también se cita que las pruebas que se documentan en el presente trabajo se realizaron en un automóvil tipo jeep, marca Great Wall, modelo Haval H1, fabricado en el año 2013. Así mismo se señala que las mismas pruebas se aplicaron, sin resultados congruentes o con fallas, a los siguientes vehículos: camioneta Mazda BT-50 2012, auto Hyundai Accent 2010, jeep Toyota Rav 4 2004 y jeep Chevrolet Grand Vitara SZ 2007.

5. CAPÍTULO V. Análisis de Costos y Resultados

A continuación se muestra un análisis de costos incurridos en la implementación del proyecto, que básicamente es la determinación de los valores de los elementos utilizados para la implementación del prototipo:

Tabla 8.
Valores de elementos utilizados en el prototipo

CANTIDAD	ELEMENTO	VALOR UNITARIO	V. TOTAL
1	Placa Arduino UNO R3	12,00	12,00
1	Case Arduino	4,00	4,00

1	Shield bluetooth maestro HC05	7,00	7,00
1	Shield bluetooth esclavo HC06	7,00	7,00
1	Interfaz bluetooth ELM327 OBD II	25,00	25,00
7	Cables Dupont	0,03	0,18
		TOTAL	**55,18**

Nota: La tabla muestra todos los elementos adquiridos para la implementación del prototipo, así como la cantidad utilizada de cada uno de ellos, el valor individual y el valor total, en dólares de los Estados Unidos de Norteamérica.

La inversión económica realizada es media en relación con otros dispositivos, sin considerar el tiempo empleado en la programación del Arduino. En el país no hay un dispositivo producido de manera comercial que cumpla con las características del prototipo implementado, por esta razón y como relación de comparación se consideró dispositivos de funciones similares que se los puede adquirir por internet en otros países como Estados Unidos.

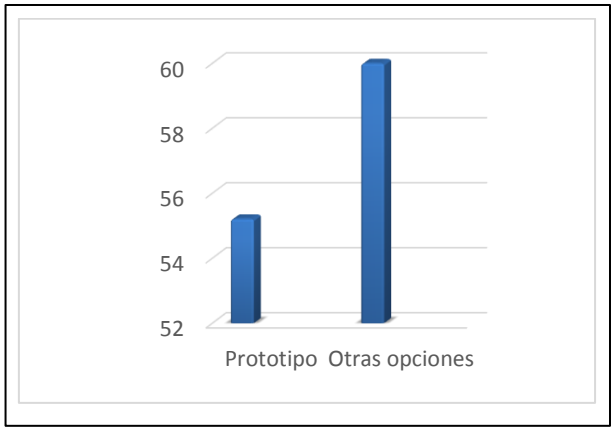


Figura 39. Comparación de nivel de precios

En la comparación realizada los costos aparentemente no son tan elevados ni con mucha diferencia, pero si analizamos los factores externos que implica la

adquisición de las otras opciones en el exterior, dichos productos quedan en desventaja ya que deben realizar pagos adicionales como aranceles, salvaguardas y empacado y transporte (lo que se conoce como Courier), por lo que podría resultar beneficiosa la adquisición del prototipo realizado. Para lograr mayor criterio ante estos factores expuestos, se realiza el siguiente análisis FODA:

Tabla 9.
Análisis FODA del prototipo implementado

FORTALEZAS	**DEBILIDADES**
• Único dispositivo en el país con las funciones descritas. • Materiales fáciles y económicos de adquirir. • Se cuenta con las herramientas, conocimiento y experiencia para programar Arduino.	La configuración del computador de cada vehículo según su marca demora el tiempo de programación del Arduino.
OPORTUNIDADES	**AMENAZAS**
• Utilizar al Arduino como herramienta útil aplicable a vehículos de servicios públicos como buses, taxis, etc. • Potenciar el uso de Arduino con la programación de más funciones aplicables a los vehículos.	• El desarrollo tecnológico que puede dejar de lado al Arduino y generar nuevas herramientas con dimensiones más adaptables y fáciles de usar e implementar. • Disminución de costos para la adquisición de productos en el

	exterior.

Recalcando que el diseño del prototipo fue con la necesidad de aprendizaje más no con fines comerciales, ante el análisis realizado se puede determinar que al establecerlo como un producto comercial tiene muchas ventajas económicas y personales para los conductores de vehículos. Así también al establecerlo como un producto comercial su costo disminuiría, ya que la compra de materiales al por mayor disminuye el costo de adquisición, lo que no ocurrió en la elaboración del prototipo al comprar materiales por unidad.

Los resultados obtenidos se pueden analizar desde la perspectiva de los factores involucrados de la siguiente manera:

1. *Tiempo.* Es decir cuanta dedicación exclusiva ha requerido la implementación del prototipo, descartando o excluyendo la curva de aprendizaje para lograr el diseño final dado que, si bien es cierto se hizo necesaria la inversión de mucho tiempo (aproximadamente 4 semanas) para el manejo de las herramientas necesarias (software y hardware), también es claro el hecho de que el proyecto está definido como diseño del prototipo y en consideración a ello se estima que el tiempo de 4 semanas se puede encasillar como de una valoración media dentro de la calificación en cuanto a proyectos de similar envergadura y complejidad.

2. *Dificultad*. En este aspecto se pretende valorar cuán difícil constituyó la implementación del prototipo, desde su armado hasta su instalación y puesta en funcionamiento, tampoco incluye el tema de diseño en donde se realiza la programación, pero si el proceso de pruebas y ajustes al prototipo. En este sentido, el resultado de puede cuantificar como de un nivel medio, pues si bien es cierto la destreza para la parte física de conexiones y armado juega un papel fundamental y vuelve sencilla la labor, también se

requirió de solvencia conceptual para la solución de inconvenientes a nivel de software, que requirieron de un buen tiempo de investigación, por las características específicas de los automóviles de prueba.

3. *Costo*. Este factor refiere a cuánto dinero ha sido necesario invertir en la implementación del prototipo, y sobre la base de los precios de equipos comerciales con similares funcionalidades que según se pudo investigar en sitios como Amazon o EBay (portales internacionales de venta de productos considerados en razón de no existir un mercado de referencia en Ecuador), que muestran equipos de un promedio de USD $ 60,00, se deduce que el costo del prototipo es de un nivel medio porque a pesar de sobrepasar el promedio antes mencionado, utiliza un elemento que es el Arduino que tiene prestaciones mucho mayores a las necesarias para esta aplicación específica, pero que se justifica por tratarse de un prototipo sujeto a pruebas y nuevos desarrollos.

4. *Conveniencia*. En función de la utilidad que tiene el prototipo, cuán conveniente es su uso. En este aspecto, se toma en cuenta lo señalado de manera inicial en el presente documento, en cuanto a la necesidad de soluciones tecnológicas que permitan disminuir o incluso eliminar los factores de distracción en la conducción de un automóvil. Desde esta perspectiva se valora la conveniencia del uso del prototipo en la vida cotidiana como de un nivel alto, incluso considerando un factor de seguridad.

5. *Precisión*. Qué nivel de precisión tienen los datos mostrados por el prototipo? Este se considera un factor importante para la fiabilidad del prototipo, pues cuan preciso sea determina directamente cuan confiable es la información que el mismo proporciona; y, desde este punto de vista y considerando los resultados obtenidos en las pruebas, se puede valorar este factor como de un nivel alto.

6. *Objetivo*. Aquí se pretende valorar si se ha llegado a alcanzar el objetivo planteado para el prototipo, es decir el de desarrollar un prototipo utilizando los sensores del vehículo que permitan al conductor obtener información del estado del mismo, en el parabrisas. Después de las pruebas realizadas, se puede concluir que dicho objetivo ha sido cumplido.

En la siguiente tabla se muestra un resumen de lo previamente analizado:

Tabla 10.
Resumen de resultados obtenidos

FACTOR	VALORACIÓN
Tiempo	Medio
Dificultad	Medio
Costo	Medio
Conveniencia	Alta
Precisión	Alta
Objetivo	Cumple

Nota: Los factores que se describen en la tabla, se consideran los más influyentes a la hora de decidir la implementación de un prototipo, como el que se ha mostrado en el presente documento.

6. Conclusiones y Recomendaciones

6.1. Conclusiones

Después de realizado el proyecto se ha llegado a las siguientes conclusiones:

Las tecnologías existentes para asistencia al conductor en el proceso de manejo del automóvil, aún no tienen la diversificación necesaria. Si bien es cierto, algunos autos incorporan equipamiento de serie de este tipo, en su mayoría representan un costo más elevado para las personas, lo cual es una razón para su escaza difusión y uso.

El prototipo de dispositivo diseñado permite visualizar de manera eficiente la información indispensable para la conducción del automóvil en el parabrisas, inclusive de manera mucho más precisa que los propios indicadores del mismo, si se consideran las variables implementadas que en este caso han sido la velocidad, temperatura y revoluciones del motor. En este punto es importante señalar que los datos que se obtienen son proporcionados por la propia computadora del automóvil, es decir los generados por los sensores incorporados, lo cual da cuenta de una funcionalidad no aprovechada en los sistemas de los automóviles.

Al implementar y efectuar pruebas del prototipo diseñado, se puede concluir que el prototipo muestra gran versatilidad para su aplicación e instalación, de hecho, el tiempo invertido en este aspecto es mínimo lo cual se convierte en una gran ventaja. A pesar de ello, es importante considerar que el estudio de las diferentes variables involucradas en el diseño, así como la prueba de alternativas y determinación de especificidades para el prototipo final, requiere de mucho tiempo de dedicación para ir alineando las piezas y conceptos necesarios para conseguir el objetivo planteado.

6.2. Recomendaciones

Así mismo, se ha llegado a determinar las siguientes recomendaciones, al respecto del prototipo implementado y la experiencia adquirida en este proceso:

Como se pudo observar en los resultados, el prototipo tiene muchas buenas características que lo hacen una alternativa muy práctica y económica a los avanzados sistemas que incorporan los automóviles de la actualidad, sin embargo, también es pertinente señalar la necesidad de una investigación previa de los protocolos utilizados por las diferentes marcas de automotores con la finalidad de adecuar el código de la aplicación del Arduino, pues su efectividad depende de la correcta identificación del protocolo de comunicación que maneja la computadora del automóvil.

Es recomendable una siguiente fase de desarrollo del prototipo, en donde se pueda dar mayores prestaciones al mismo, por ejemplo a la par de solucionar la transferencia del sistema de un protocolo a otro, se podría incluir temas como la selección a discreción del usuario de las variables que desea visualizar, así como la personalización del aspecto gráfico de la aplicación del teléfono móvil.

REFERENCIAS

Autodesk (s.f.). *Arduino real time simulator.* Recuperado el 10 de febrero de 2018, de https://circuits.io/

Bermúdez, J. A., y Ríos, J. (2015). *Diseño de una interfaz head up display (hud) para casco de motocicleta*. Recuperado el 05 de junio de 2017 de http://repositorio.utp.edu.co/dspace/handle/11059/5631

Corona Ramírez, L. G., Abarca Jiménez, G., & Mares Carreño, J. (2014). *Sensores y actuadores: aplicaciones con Arduino.* (1ª ed.). México, D.F., México: Patria.

Crespo, J. E. (2017). *Aprendiendo Arduino.* Recuperado el 10 de febrero de 2018, de https://aprendiendoarduino.wordpress.com

Gabatek (s.f.). *Pioneer trae el futuro a los carros con realidad aumentada a gran escala.* Recuperado el 10 de febrero de 2018, de http://gabatek.com/tecnologica/tecnologia-hud/

Garrido Pedraza, J. (2015). *Fundamentos de Arduino.* Recuperado el 25 de junio de 2017, de http://aprender.tdrobotica.co/cursos/guia-arduino/

Hernández Olarte, Y. (2013). *Guía Básica de Arduino.* Recuperado el 20 de junio de 2017, de www.freelibros.or

López López, G. G., y Romero Barreno, P. A. (2014). *Diseño e implementación de un sistema de simulación para la asistencia en la conducción utilizando realimentación de fuerza en los comando de dirección y aceleración.* Recuperado el 09 de junio de 2017, de http://bibdigital.epn.edu.ec/handle/15000/8694

OBD Con. (2017). *OBD-II ResourceOBD-II software resource for developers and users. OBD-II and PIDs.* Recuperado el 13 de diciembre de 2017, de: http://obdcon.sourceforge.net/2010/06/obd-ii-pids/

Reyes Cortés, F., & Cid Monjarráz, J. (2015). *Arduino: Aplicaciones en robótica, mecatrónica e ingenierías.* México D.F., México: Alfaomega Grupo Editor.

Simbaña, W. (2015). *Diseño e implementación de una solución telemática basada en OBD-II (On-Board Diagnostic) que permita obtener y*

procesar la información de los sensores del motor de un automóvil. Recuperado el 11 de junio de 2017 de http://bibdigital.epn.edu.ec/handle/15000/10659

Simbaña, W., Caiza, J., y Chávez, D. (2016). *Diseño e Implementación de un Sistema de Monitoreo Remoto del Motor de un Vehículo basado en Obd-II y la plataforma Arduino.* Recuperado el 15 de junio de 2017 de http://www.revistapolitecnica.epn.edu.ec/ojs2/index.php/revista_politecnica2/article/view/573/pdf

Torrente, O. (2013). *Arduino, curso práctico de formación.* (1ª ed.). México D.F., México: Alfaomega Grupo Editor.

Vallejo, H. D. (2012). *Electrónica del Automóvil V.4* (Vol. 4). México D.F., México: QUARK S.R.I.

Vallejo, H. D. (2012). *Electrónica del Automóvil V.5* (Vol. 5). México D.F., México: QUARK S.R.I.

Velásquez, A. (2013). *Android OBD-II.* Recuperado el 15 de julio de 2017 de https://uvadoc.uva.es/bitstream/10324/4084/1/TFG-B.359.pdf

ANEXOS

Anexo 1

Tabla 11.

Lista de PID más usuales.

Mode (hex)	PID (hex)	Data bytes returned	Description	Min val	Max val	Units	Formula
01	00	4	PIDs supported [01 - 20]				Bit encoded [A7..D0] == [PID 0x01..PID 0x20]
01	01	4	Monitor status since DTCs cleared. (Includes malfunction indicator lamp (MIL) status and number of DTCs.)				Bit encoded.
01	02	8	Freeze				
01	03	2	Fuel system status				Bit encoded.
01	04	1	Calculated engine load value	0	100	%	A*100/255
01	05	1	Engine coolant temperature	-40	215	°C	A-40
01	06	1	Short term fuel % trim—Bank 1	-100 (Rich)	99.22 (Lean)	%	(A-128) * 100/128
01	07	1	Long term fuel % trim—Bank 1	-100 (Rich	99.22 (Lean)	%	(A-128) * 100/128

				)			
01	08	1	Short term fuel % trim—Bank 2	-100 (Rich)	99.22 (Lean)	%	(A-128) * 100/128
01	09	1	Long term fuel % trim—Bank 2	-100 (Rich)	99.22 (Lean)	%	(A-128) * 100/128
01	0A	1	Fuel pressure	0	765	kPa (gauge)	A*3
01	0B	1	Intake manifold absolute pressure	0	255	kPa (absolute)	A
01	0C	2	Engine RPM	0	16,383.75	rpm	((A*256)+B)/4
01	0D	1	Vehicle speed	0	255	km/h	A
01	0E	1	Timing advance	-64	63.5	° relative to #1 cylinder	A/2 – 64
01	0F	1	Intake air temperature	-40	215	°C	A-40
01	10	2	MAF air flow rate	0	655.35	g/s	((A*256)+B) / 100
01	11	1	Throttle position	0	100	%	A*100/255
01	12	1	Commanded secondary air status				Bit encoded
01	13	1	Oxygen sensors				[A0..A3] == Bank 1,

			present				Sensors 1-4. [A4..A7] == Bank 2…
01	14	2	Bank 1, Sensor 1:Oxygen sensor voltage, Short term fuel trim	0-100(lean)	1.27599.2(rich)	Volts %	A * 0.005(B-128) * 100/128 (if B==0xFF, sensor is not used in trim calc)
01	15	2	Bank 1, Sensor 2:Oxygen sensor voltage, Short term fuel trim	0-100(lean)	1.27599.2(rich)	Volts %	A * 0.005(B-128) * 100/128 (if B==0xFF, sensor is not used in trim calc)
01	16	2	Bank 1, Sensor 3:Oxygen sensor voltage, Short term fuel trim	0-100(lean)	1.27599.2(rich)	Volts %	A * 0.005(B-128) * 100/128 (if B==0xFF, sensor is not used in trim calc)
01	17	2	Bank 1, Sensor 4:Oxygen sensor voltage, Short term fuel trim	0-100(lean)	1.27599.2(rich)	Volts %	A * 0.005(B-128) * 100/128 (if B==0xFF, sensor is not used in trim calc)
01	18	2	Bank 2, Sensor 1:Oxygen sensor voltage, Short term fuel trim	0-100(lean)	1.27599.2(rich)	Volts %	A * 0.005(B-128) * 100/128 (if B==0xFF, sensor is not used in trim calc)
01	19	2	Bank 2, Sensor 2:Oxygen sensor voltage, Short term fuel trim	0-100(lean)	1.27599.2(rich)	Volts %	A * 0.005(B-128) * 100/128 (if B==0xFF, sensor is not used in trim calc)
01	1A	2	Bank 2, Sensor 3:Oxygen sensor voltage, Short term fuel trim	0-100(lean)	1.27599.2(rich)	Volts %	A * 0.005(B-128) * 100/128 (if B==0xFF, sensor is not used in trim calc)

01	1B	2	Bank 2, Sensor 4:Oxygen sensor voltage, Short term fuel trim	0-100(lean)	1.27599.2(rich)	Volts %	A * 0.005(B-128) * 100/128 (if B==0xFF, sensor is not used in trim calc)
01	1C	1	OBD standards this vehicle conforms to				Bit encoded.
01	1D	1	Oxygen sensors present				Similar to PID 13, but [A0..A7] == [B1S1, B1S2, B2S1, B2S2, B3S1, B3S2, B4S1, B4S2]
01	1E	1	Auxiliary input status				A0 == Power Take Off (PTO) status (1 == active)[A1..A7] not used
01	1F	2	Run time since engine start	0	65,535	seconds	(A*256)+B
01	20	4	PIDs supported 21-40				Bit encoded [A7..D0] == [PID 0x21..PID 0x40]
01	21	2	Distance traveled with malfunction indicator lamp (MIL) on	0	65,535	km	(A*256)+B
01	22	2	Fuel Rail Pressure (relative to manifold vacuum)	0	5177.265	kPa	(((A*256)+B) * 10) / 128
01	23	2	Fuel Rail Pressure (diesel)	0	655350	kPa (gauge)	((A*256)+B) * 10

01	24	4	O2S1_WR_lambda(1):Equivalence Ratio Voltage	00	28	N/AV	((A*256)+B)/32768((C*256)+D)/8192
01	25	4	O2S2_WR_lambda(1):Equivalence Ratio Voltage	00	28	N/AV	((A*256)+B)/32768((C*256)+D)/8192
01	26	4	O2S3_WR_lambda(1):Equivalence Ratio Voltage	00	28	N/AV	((A*256)+B)/32768((C*256)+D)/8192
01	27	4	O2S4_WR_lambda(1):Equivalence Ratio Voltage	00	28	N/AV	((A*256)+B)/32768((C*256)+D)/8192
01	28	4	O2S5_WR_lambda(1):Equivalence Ratio Voltage	00	28	N/AV	((A*256)+B)/32768((C*256)+D)/8192
01	29	4	O2S6_WR_lambda(1):Equivalence Ratio Voltage	00	28	N/AV	((A*256)+B)/32768((C*256)+D)/8192
01	2A	4	O2S7_WR_lambda(1):Equivalence Ratio Voltage	00	28	N/AV	((A*256)+B)/32768((C*256)+D)/8192
01	2B	4	O2S8_WR_lambda(1):Equivalence	00	28	N/AV	((A*256)+B)/32768((

			Ratio Voltage				C*256)+D)/8192
01	2C	1	Commanded EGR	0	100	%	100*A/255
01	2D	1	EGR Error	-100	99.22	%	(A-128) * 100/128
01	2E	1	Commanded evaporative purge	0	100	%	100*A/255
01	2F	1	Fuel Level Input	0	100	%	100*A/255
01	30	1	# of warm-ups since codes cleared	0	255	N/A	A
01	31	2	Distance traveled since codes cleared	0	65,535	km	(A*256)+B
01	32	2	Evap. System Vapor Pressure	-8,192	8,192	Pa	((A*256)+B)/4 (A is signed)
01	33	1	Barometric pressure	0	255	kPa (Absolute)	A
01	34	4	O2S1_WR_lambda(1):Equivalence Ratio Current	0-128	2128	N/AmA	((A*256)+B)/32768((C*256)+D)/256 – 128
01	35	4	O2S2_WR_lambda(1):Equivalence Ratio Current	0-128	2128	N/AmA	((A*256)+B)/32768((C*256)+D)/256 – 128
01	36	4	O2S3_WR_lambda(1):Equivalence Ratio	0-128	2128	N/AmA	((A*256)+B)/327685((C*256)+D)/256 – 128

			Current				
01	37	4	O2S4_WR_lambda(1):Equivalence Ratio Current	0-128	2128	N/AmA	((A*256)+B)/32768((C*256)+D)/256 – 128
01	38	4	O2S5_WR_lambda(1):Equivalence Ratio Current	0-128	2128	N/AmA	((A*256)+B)/32768((C*256)+D)/256 – 128
01	39	4	O2S6_WR_lambda(1):Equivalence Ratio Current	0-128	2128	N/AmA	((A*256)+B)/32768((C*256)+D)/256 – 128
01	3A	4	O2S7_WR_lambda(1):Equivalence Ratio Current	0-128	2128	N/AmA	((A*256)+B)/32768((C*256)+D)/256 – 128
01	3B	4	O2S8_WR_lambda(1):Equivalence Ratio Current	0-128	2128	N/AmA	((A*256)+B)/32768((C*256)+D)/256 – 128
01	3C	2	Catalyst TemperatureBank 1, Sensor 1	-40	6,513.5	°C	((A*256)+B)/10 – 40
01	3D	2	Catalyst TemperatureBank 2, Sensor 1	-40	6,513.5	°C	((A*256)+B)/10 – 40
01	3E	2	Catalyst TemperatureBank 1, Sensor 2	-40	6,513.5	°C	((A*256)+B)/10 – 40

01	3F	2	Catalyst TemperatureBank 2, Sensor 2	-40	6,513.5	°C	((A*256)+B)/10 – 40
01	40	4	PIDs supported 41-60				Bit encoded [A7..D0] == [PID 0x41..PID 0x60]
01	41	4	Monitor status this drive cycle				Bit encoded.
01	42	2	Control module voltage	0	65.535	V	((A*256)+B)/1000
01	43	2	Absolute load value	0	25,700	%	((A*256)+B)*100/255
01	44	2	Command equivalence ratio	0	2	N/A	((A*256)+B)/32768
01	45	1	Relative throttle position	0	100	%	A*100/255
01	46	1	Ambient air temperature	-40	215	°C	A-40
01	47	1	Absolute throttle position B	0	100	%	A*100/255
01	48	1	Absolute throttle position C	0	100	%	A*100/255
01	49	1	Accelerator pedal position D	0	100	%	A*100/255
01	4A	1	Accelerator pedal position E	0	100	%	A*100/255
01	4B	1	Accelerator pedal position F	0	100	%	A*100/255
01	4C	1	Commanded throttle	0	100	%	A*100/255

			actuator				
01	4D	2	Time run with MIL on	0	65,535	minute	(A*256)+B
01	4E	2	Time since trouble codes cleared	0	65,535	minute	(A*256)+B
01	51	1	Fuel Type				From fuel type table
01	52	1	Ethanol fuel %	0	100	%	A*100/255
01	53	2	Absoulute Evap system Vapour Pressure	0	327675	kpa	1/200 per bit
01	C3	?	?	?	?	?	Returns numerous data, including Drive Condition ID and Engine Speed*
01	C4	?	?	?	?	?	B5 is Engine Idle RequestB6 is Engine Stop Request*
02	02	2	Freeze frame trouble code				BCD encoded
03	N/A	n*6	Request trouble codes				3 codes per message frame, BCD encoded.
04	N/A	0	Clear trouble codes / Malfunction indicator lamp (MIL) / Check engine light				Clears all stored trouble codes and turns the MIL off.
05	0100		OBD Monitor IDs supported ($01 – $20)				

05	0101		O2 Sensor Monitor Bank 1 Sensor 1	0.00	1.275	Volts	0.005 Rich to lean sensor threshold voltage
05	0102		O2 Sensor Monitor Bank 1 Sensor 2	0.00	1.275	Volts	0.005 Rich to lean sensor threshold voltage
05	0103		O2 Sensor Monitor Bank 1 Sensor 3	0.00	1.275	Volts	0.005 Rich to lean sensor threshold voltage
05	0104		O2 Sensor Monitor Bank 1 Sensor 4	0.00	1.275	Volts	0.005 Rich to lean sensor threshold voltage
05	0105		O2 Sensor Monitor Bank 2 Sensor 1	0.00	1.275	Volts	0.005 Rich to lean sensor threshold voltage
05	0106		O2 Sensor Monitor Bank 2 Sensor 2	0.00	1.275	Volts	0.005 Rich to lean sensor threshold voltage
05	0107		O2 Sensor Monitor Bank 2 Sensor 3	0.00	1.275	Volts	0.005 Rich to lean sensor threshold voltage
05	0108		O2 Sensor Monitor Bank 2 Sensor 4	0.00	1.275	Volts	0.005 Rich to lean sensor threshold voltage
05	0109		O2 Sensor Monitor Bank 3 Sensor 1	0.00	1.275	Volts	0.005 Rich to lean sensor threshold voltage
05	010A		O2 Sensor Monitor Bank 3 Sensor 2	0.00	1.275	Volts	0.005 Rich to lean sensor threshold voltage

05	010B		O2 Sensor Monitor Bank 3 Sensor 3	0.00	1.275	Volts	0.005 Rich to lean sensor threshold voltage
05	010C		O2 Sensor Monitor Bank 3 Sensor 4	0.00	1.275	Volts	0.005 Rich to lean sensor threshold voltage
05	010D		O2 Sensor Monitor Bank 4 Sensor 1	0.00	1.275	Volts	0.005 Rich to lean sensor threshold voltage
05	010E		O2 Sensor Monitor Bank 4 Sensor 2	0.00	1.275	Volts	0.005 Rich to lean sensor threshold voltage
05	010F		O2 Sensor Monitor Bank 4 Sensor 3	0.00	1.275	Volts	0.005 Rich to lean sensor threshold voltage
05	0110		O2 Sensor Monitor Bank 4 Sensor 4	0.00	1.275	Volts	0.005 Rich to lean sensor threshold voltage
05	0201		O2 Sensor Monitor Bank 1 Sensor 1	0.00	1.275	Volts	0.005 Lean to Rich sensor threshold voltage
05	0202		O2 Sensor Monitor Bank 1 Sensor 2	0.00	1.275	Volts	0.005 Lean to Rich sensor threshold voltage
05	0203		O2 Sensor Monitor Bank 1 Sensor 3	0.00	1.275	Volts	0.005 Lean to Rich sensor threshold voltage
05	0204		O2 Sensor Monitor Bank 1 Sensor 4	0.00	1.275	Volts	0.005 Lean to Rich sensor threshold voltage

05	0205		O2 Sensor Monitor Bank 2 Sensor 1	0.00	1.275	Volts	0.005 Lean to Rich sensor threshold voltage
05	0206		O2 Sensor Monitor Bank 2 Sensor 2	0.00	1.275	Volts	0.005 Lean to Rich sensor threshold voltage
05	0207		O2 Sensor Monitor Bank 2 Sensor 3	0.00	1.275	Volts	0.005 Lean to Rich sensor threshold voltage
05	0208		O2 Sensor Monitor Bank 2 Sensor 4	0.00	1.275	Volts	0.005 Lean to Rich sensor threshold voltage
05	0209		O2 Sensor Monitor Bank 3 Sensor 1	0.00	1.275	Volts	0.005 Lean to Rich sensor threshold voltage
05	020A		O2 Sensor Monitor Bank 3 Sensor 2	0.00	1.275	Volts	0.005 Lean to Rich sensor threshold voltage
05	020B		O2 Sensor Monitor Bank 3 Sensor 3	0.00	1.275	Volts	0.005 Lean to Rich sensor threshold voltage
05	020C		O2 Sensor Monitor Bank 3 Sensor 4	0.00	1.275	Volts	0.005 Lean to Rich sensor threshold voltage
05	020D		O2 Sensor Monitor Bank 4 Sensor 1	0.00	1.275	Volts	0.005 Lean to Rich sensor threshold voltage
05	020E		O2 Sensor Monitor Bank 4 Sensor 2	0.00	1.275	Volts	0.005 Lean to Rich sensor threshold voltage

05	020F		O2 Sensor Monitor Bank 4 Sensor 3	0.00	1.275	Volts	0.005 Lean to Rich sensor threshold voltage
05	0210		O2 Sensor Monitor Bank 4 Sensor 4	0.00	1.275	Volts	0.005 Lean to Rich sensor threshold voltage
09	00	4	mode 9 supported PIDs 01 to 20				Bit encoded
09	02	5×5	Vehicle identification number (VIN)				Returns 5 lines, A is line ordering flag, B-E ASCII coded VIN digits.
09	04	varies	calibration ID				Returns multiple lines, ASCII coded
09	06	4	calibration				

Nota: En la columna de fórmula, las letras A, B, C, etc. representan el equivalente decimal del primer, segundo, tercer, etc. bytes de datos. Donde aparece una (?), Información contradictoria o incompleta estaba disponible. En el 2006 SAE HS-3000 se puede verificar estos hechos. (OBD Con, 2017)

Recuperado el 13 de Diciembre de 2017 de http://obdcon.sourceforge.net/2010/06/obd-ii-pids/

Anexo 2

ELM327
OBD to RS232 Interpreter

Description

Almost all of the automobiles produced today are required, by law, to provide an interface for the connection of diagnostic test equipment. The data transfer on these interfaces follow several standards, but none of them are directly usable by PCs or smart devices. The ELM327 is designed to act as a bridge between these On-Board Diagnostics (OBD) ports and a standard RS232 serial interface.

In addition to being able to automatically detect and interpret nine OBD protocols, the ELM327 also provides support for high speed communications, a low power sleep mode, and the J1939 truck and bus standard. It is also completely customizable, should you wish to alter it to more closely suit your needs.

The following pages discuss all of the ELM327's features in detail, how to use it and configure it, as well as providing some background information on the protocols that are supported. There are also schematic diagrams and tips to help you to interface to microprocessors, construct a basic scan tool, and to use the low power mode.

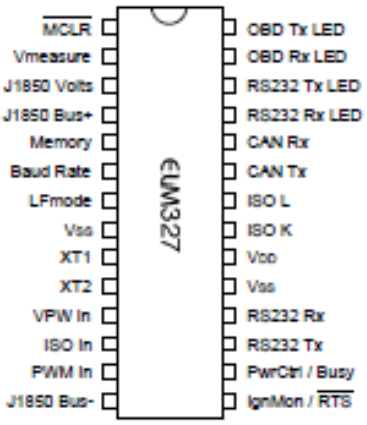
Applications

- Diagnostic trouble code readers
- Automotive scan tools
- Teaching aids

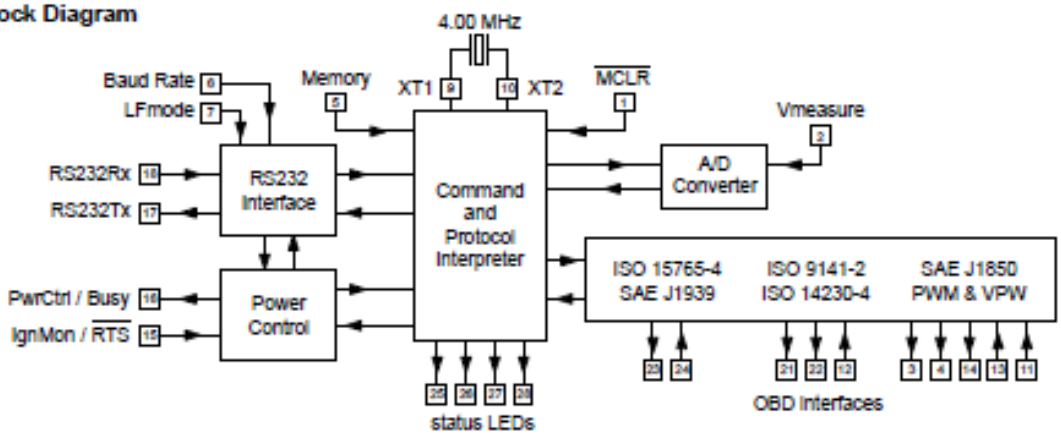
Features

- Power Control with standby mode
- Universal serial (RS232) interface
- Automatically searches for protocols
- Fully configurable with AT commands
- Low power CMOS design

Connection Diagram
PDIP and SOIC
(top view)



Block Diagram



Contents

Contents

Pin Descriptions

MCLR (pin 1)

A momentary (>2µsec) logic low applied to this input will reset the ELM327. If unused, this pin should be connected to a logic high (V_{DD}) level.

Vmeasure (pin 2)

This analog input is used to measure a 0 to 5V signal that is applied to it. Care must be taken to prevent the voltage from going outside of the supply levels of the ELM327, or damage may occur. If it is not used, this pin should be tied to either V_{DD} or V_{SS}.

J1850 Volts (pin 3)

This output can be used to control a voltage supply for the J1850 Bus+ output. The pin normally outputs a logic high level when a nominal 8V is required (for J1850 VPW), and a low level for 5V (for J1850 PWM), but this can be changed with PP 12. If this switching capability is not required for your application, this output can be left open-circuited.

J1850 Bus+ (pin 4)

This active high output is used to drive the J1850 Bus+ Line to an active level. Note that this signal does not have to be used for the Bus- Line (as was the case for the ELM320), since a separate J1850 Bus- drive output is provided on pin 14.

Memory (pin 5)

This input controls the default state of the memory option. If this pin is at a high level during power-up or reset, the memory function will be enabled by default. If it is at a low level, then the default will be to have it disabled. Memory can always be enabled or disabled with the AT M1 and AT M0 commands.

Baud Rate (pin 6)

This input controls the baud rate of the RS232 interface. If it is at a high level during power-up or reset, the baud rate will be set to 38400 (or the rate that has been set by PP 0C). If at a low level, the baud rate will be initialized to 9600 bps.

LFmode (pin 7)

This input is used to select the default linefeed mode to be used after a power-up or system reset. If it is at a high level, then by default messages sent by the ELM327 will be terminated with both a carriage return and a linefeed character. If it is at a low level, lines will be terminated by a carriage return only. This behaviour can always be modified by issuing an AT L1 or AT L0 command.

V_{SS} (pin 8)

Circuit common must be connected to this pin.

XT1 (pin 9) and XT2 (pin 10)

A 4.000 MHz oscillator crystal is connected between these two pins. Loading capacitors as required by the crystal (typically 27pF each) will also need to be connected between each of these pins and circuit common (Vss).

Note that this device has not been configured for operation with an external oscillator – it expects a crystal to be connected to these pins. Use of an external clock source is not recommended. Also, note that this oscillator is turned off when in the Low Power or 'standby' mode of operation.

VPW In (pin 11)

This is the active high input for the J1850 VPW data signal. When at rest (bus recessive) this pin should be at a low logic level. This input has Schmitt trigger wave shaping, so no special amplification is required.

ISO In (pin 12)

This is the active low input for the ISO 9141 and ISO 14230 data signal. It is derived from the K Line, and should be at a high logic level when at rest (bus recessive). No special amplification is required, as this input has Schmitt trigger wave shaping.

PWM In (pin 13)

This is the active low input for the J1850 PWM data signal. It should normally be at a high level when at rest (ie. bus recessive). This input has Schmitt trigger wave shaping, so no special amplification is required.

J1850 Bus- (pin 14)

This active high output is used to drive the J1850 Bus- Line to an active (dominant) level for J1850 PWM applications. If unused, this output can be left open-circuited.

Pin Descriptions (continued)

IgnMon / $\overline{\text{RTS}}$ (pin 15)

This input pin can serve one of two functions, depending on how the Power Control options (PP 0E) are set.

If both bit 7 and bit 2 of PP 0E are '1's, this pin will act as an Ignition Monitor. This will result in a switch to the Low Power mode of operation, should the IgnMon signal go to a low level, as would happen if the vehicle's ignition were turned off. An internal 'debounce' timer is used to ensure that the ELM327 does not shut down for noise at the input.

When the voltage at pin 15 is again restored to a high level, and a time of 1 or 5 seconds (as set by PP 0E bit 1) passes, the ELM327 will perform a 'Warm Start' and return to normal operation. A low to high transition at pin 15 will in fact restore normal operation, regardless of the setting of PP 0E bit 2, or whether pin 15 was the initial cause for the low power mode. This feature allows a system to control how and when it switches to low power standby operation, but still have automatic wakeup by the ignition voltage, or even by a pushbutton.

If either bit 7 or bit 2 of PP 0E are '0', this pin will function as an active low 'Request To Send' input. This can be used to interrupt the OBD processing in order to send a new command, or as previously mentioned, to highlight the fact that the ignition has been turned off. Normally kept at a high level, this input is brought low for attention, and should remain so until the Busy line (pin 16) indicates that the ELM327 is no longer busy, or until a prompt character is received (if pin 16 is being used for power control).

This input has Schmitt trigger wave shaping. By default, pin 15 acts as the $\overline{\text{RTS}}$ interrupt input.

PwrCtrl / Busy (pin 16)

This output pin can serve one of two functions, depending on how the Power Control options (PP 0E) are set.

If bit 7 of PP 0E is a '1' (the default), this pin will function as a Power Control output. The normal state of the pin will be as set by PP 0E bit 6, and the pin will remain in that state until the ELM327 switches to the Low Power mode of operation, when the output changes to the opposite level. This output is typically used to control enable inputs, but may also be used for relay circuits, etc. with suitable buffering. The discussion on page 84 ('Modifications for Low Power Standby Operation') provides more detail on how to use this output.

If bit 7 of PP 0E is a '0', pin 16 will function as a 'Busy' output, showing when the ELM327 is actively processing a command (the output will be at a high level), or when it is idle, ready to receive commands (the output will be low).

By default, bit 7 of PP 0E is '1', so pin 16 provides the Power Control function.

RS232Tx (pin 17)

This is the RS232 data transmit output. The signal level is compatible with most interface ICs (the output is high when idle), and there is sufficient current drive to allow interfacing using only a PNP transistor, if desired.

RS232Rx (pin 18)

This is the RS232 receive data input. The signal level is compatible with most interface ICs (when at idle, the level should be high), but can be used with other interfaces as well, since the input has Schmitt trigger wave shaping.

V_{SS} (pin 19)

Circuit common must be connected to this pin.

V_{DD} (pin 20)

This pin is the positive supply pin, and should always be the most positive point in the circuit. Internal circuitry connected to this pin is used to provide power on reset of the ELM327 processor, so an external reset signal is not required. Refer to the Electrical Characteristics section for further information.

ISO K (pin 21) and ISO L (pin 22)

These are the active high output signals which are used to drive the ISO 9141 and ISO 14230 buses to an active (dominant) level. Many new vehicles do not require the L Line – if yours does not, you can simply leave pin 22 open-circuited.

CAN Tx (pin 23) and CAN Rx (pin 24)

These are the two CAN interface signals that must be connected to a CAN transceiver IC (see the

Pin Descriptions (continued)

Example Applications section for more information). If unused, pin 24 must be connected to a logic high (V_{DD}) level.

RS232 Rx LED (pin 25), RS232 Tx LED (pin 26), OBD Rx LED (pin 27) and OBD Tx LED (pin 28)

These four output pins are normally high, and are driven to low levels when the ELM327 is transmitting or receiving data. These outputs are suitable for directly driving most LEDs through current limiting resistors, or interfacing to other logic circuits. If unused, these pins may be left open-circuited.

Note that pin 28 can also be used to turn off all of the Programmable Parameters, if you can not do so by using the normal interface - see page 69 for details.

Unused Pins

When people only want to implement a portion of what the ELM327 is capable of, they often ask what to do with the unused pins. The rule is that unused outputs may be left open-circuited with nothing connected to them, but unused inputs must be terminated. The ELM327 is a CMOS integrated circuit that can not have any inputs left floating (or you might damage the IC). Connect unused inputs as follows:

Pin	1	2	5	6	7	11	12	13	15	18	24
Level	H	H*	H*	H*	H*	H*	L*	L*	H	H	H

Note that the inputs that are shown with an asterisk (*) may be connected to either a High (V_{DD}) or a Low (V_{SS}) level, but the level shown is preferred.

Absolute Maximum Ratings

Storage Temperature..... -65°C to +150°C

Ambient Temperature with Power Applied..... -40°C to +85°C

Voltage on V_{DD} with respect to V_{SS}..... -0.3V to +7.5V

Voltage on any other pin with respect to V_{SS}..... -0.3V to (V_{DD} + 0.3V)

Note:

These values are given as a design guideline only. The ability to operate to these levels is neither inferred nor recommended, and stresses beyond those listed here will likely damage the device.

Electrical Characteristics

All values are for operation at 25°C and a 5V supply, unless otherwise noted. For further information, refer to note 1 below.

Characteristic		Minimum	Typical	Maximum	Units	Conditions
Supply voltage, V_{DD}		4.2	5.0	5.5	V	
V_{DD} rate of rise		0.05			V/ms	see note 2
Average current, I_{DD}	normal		12		mA	ELM327 device only - does not include any load currents
	low power		0.15		mA	
Input logic levels	low	V_{SS}		0.8	V	Pins 5, 6, 7, and 24 only
	high	3.0		V_{DD}	V	
Schmitt trigger input thresholds	rising		2.9	4.0	V	Pins 1, 11, 12, 13, 15 and 18 only
	falling	1.0	1.5		V	
Output low voltage			0.3		V	current (sink) = 10 mA
Output high voltage			4.4		V	current (source) = 10 mA
Brown-out reset voltage		2.65	2.79	2.93	V	
A/D conversion time			9		msec	AT RV to beginning of response
Pin 18 wake pulse duration		128			µsec	to wake from Low Power mode
IgnMon debounce time		50	65		msec	
AT LP to PwrCtrl output time			1.0		sec	
LP ALERT to PwrCtrl output time			2.0		sec	
Reset time	AT Z		800		msec	Measured from the end of the command to the start of the ID message (ELM327 v2.1)
	AT WS		2		msec	

Notes:

1. This integrated circuit is based on Microchip Technology Inc.'s PIC18F2480 device. For more detailed device specifications, and possibly clarification of those given, please refer to the Microchip documentation (available at www.microchip.com).
2. This spec must be met in order to ensure that a correct power on reset occurs. It is quite easily achieved using most common types of supplies, but may be violated if one uses a slowly varying supply voltage, as may be obtained through direct connection to solar cells or some charge pump circuits.

Overview

The following describes how to use the ELM327 to obtain information from your vehicle.

We begin by discussing just how to 'talk' to the IC using a PC, then explain how to change options using 'AT' commands, and finally we show how to use the ELM327 to obtain trouble codes (and reset them). For the more advanced experimenters, there are also sections on how to use some of the programmable features of this integrated circuit as well.

Using the ELM327 is not as daunting as it first seems. Many users will never need to issue an 'AT' command, adjust timeouts, or change the headers. For most, all that is required is a PC or smart device with a terminal program (such as HyperTerminal or ZTerm), and a little knowledge of OBD commands, which we will provide in the following sections...

Communicating with the ELM327

The ELM327 expects to communicate with a PC through an RS232 serial connection. Although modern computers do not usually provide a serial connection such as this, there are several ways in which a 'virtual serial port' can be created. The most common devices are USB to RS232 adapters, but there are several others such as PC cards, ethernet devices, or Bluetooth to serial adapters.

No matter how you physically connect to the ELM327, you will need a way to send and receive data. The simplest method is to use one of the many 'terminal' programs that are available (HyperTerminal, ZTerm, etc.), to allow typing the characters directly from your keyboard.

To use a terminal program, you will need to adjust several settings. First, ensure that your software is set to use the proper 'COM' port, and that you have chosen the proper data rate - this will be either 9600 baud (if pin 6 = 0V at power up), or 38400 baud (if PP 0C has not been changed). If you select the wrong 'COM' port, you will not be able to send or receive any data. If you select the wrong data rate, the information that you send and receive will be all garbled, and unreadable by you or the ELM327. Don't forget to also set your connection for 8 data bits, no parity bits, and 1 stop bit, and to set it for the proper 'line end' mode. All of the responses from the ELM327 are terminated with a single carriage return character and, optionally, a linefeed character (depending on your settings).

Properly connected and powered, the ELM327 will energize the four LED outputs in sequence (as a lamp test) and will then send the message:

```
ELM327 v2.1

>
```

In addition to identifying the version of this IC, receiving this string is a good way to confirm that the computer connections and terminal software settings are correct (however, at this point no communications have taken place with the vehicle, so the state of that connection is still unknown).

The '>' character that is shown on the second line is the ELM327's prompt character. It indicates that the device is in the idle state, ready to receive characters on the RS232 port. If you did not see the identification string, you might try resetting the IC again with the AT Z (reset) command. Simply type the letters A T and Z (spaces are optional), then press the return key:

```
>AT Z
```

That should cause the leds to flash again, and the identification string to be printed. If you see strange looking characters, then check your baud rate - you have likely set it incorrectly.

Characters sent from the computer can either be intended for the ELM327's internal use, or for reformatting and passing on to the vehicle. The ELM327 can quickly determine where the received characters are to be directed by monitoring the contents of the message. Commands that are intended for the ELM327's internal use will begin with the characters 'AT', while OBD commands for the vehicle are only allowed to contain the ASCII codes for hexadecimal digits (0 to 9 and A to F).

Whether it is an 'AT' type internal command or a hex string for the OBD bus, all messages to the ELM327 must be terminated with a carriage return character (hex '0D') before it will be acted upon. The one exception is when an incomplete string is sent and no carriage return appears. In this case, an internal timer will automatically abort the incomplete message after about 20 seconds, and the ELM327 will print a single question mark ('?') to show that the input was not understood (and was not acted upon).

Messages that are not understood by the ELM327 (syntax errors) will always be signalled by a single

Communicating with the ELM327 (continued)

question mark. These include incomplete messages, incorrect AT commands, or invalid hexadecimal digit strings, but are not an indication of whether or not the message was understood by the vehicle. One must keep in mind that the ELM327 is a protocol interpreter that makes no attempt to assess the OBD messages that you send for validity – it only ensures that hexadecimal digits were received, combined into bytes, then sent out the OBD port, and it does not know if a message sent to the vehicle was in error.

While processing OBD commands, the ELM327 will continually monitor for either an active RTS input, or an RS232 character received. Either one will interrupt the IC, quickly returning control to the user, while possibly aborting any initiation, etc. that was in progress. After generating a signal to interrupt the ELM327, software should always wait for either the prompt character ('>' or hex 3E), or a low level on the Busy output before beginning to send the next command.

Finally, it should be noted that the ELM327 is not case-sensitive, so the commands 'ATZ', 'atz', and 'AtZ' are all exactly the same to the ELM327. All commands may be entered as you prefer, as no one method is faster or better. The ELM327 also ignores space characters and all control characters (tab, etc.), so they can be inserted anywhere in the input if that improves readability.

One other feature of the ELM327 is the ability to repeat any command (AT or OBD) when only a single carriage return character is received. If you have sent a command (for example, 01 0C to obtain the rpm), you do not have to resend the entire command in order to resend the request to the vehicle - simply send a carriage return character, and the ELM327 will repeat the command for you. The memory buffer only remembers one command though, and there is no provision in the current ELM327 to provide storage for any more.

Please Note:

There is a very small chance that NULL characters (byte value 00) may occasionally be inserted into the RS232 data that is transmitted by the ELM327.

Microchip Technology has reported that some ICs which use the same EUSART as in the ELM327 may, under very specific (and rare) conditions, insert an extra byte (always of value 00) into the transmitted data. If you are using a terminal program to view the data, you should select the 'hide control characters' option if it is available, and if you are writing software for the ELM327, then ignore incoming bytes that are of value 00 (ie. remove NULLs).

Estas corresponden sólo a las primeras páginas del DataSheet, de la 1 a la 8, para más detalle se deberá recurrir a la página web www.elmelectronics.com.

Anexo 3

Código del programa del Arduino:

```
#include<SoftwareSerial.h>
SoftwareSerial BTSerial(10,11);  // RX; TX
char incomingbyte = '-'; // variable de datos Bluetooth
// //Time
byte mySec = 0;
byte myMin = 0;
byte myHour = 0;
byte myDate = 1;
byte myMonth = 1;
byte myYear = 17;

byte nastr = 0;

int HA = 0;
int HB = 0;
int rpm = 0;
int Temp=0;
int Volt=0;
float vv=0;
float shotlong = -0.1;


void setup() {

  delay(300);
  delay(1000);

  BTSerial.begin(38400); //BLUETOOTH MASTER QUE SE CONECTA CON EL OBD2
38400
  Serial.begin(9600);  // BLUETOOTH QUE SE CONECTA CON LA APP 9600

  delay(1000);
  nastrELM();
}//setup

void  nastrELM()
{
  // //

  AT_ELM("at i\r");
  //AT_ELM("at d\r");
  AT_ELM("at z\r");
  AT_ELM("at d\r");
  AT_ELM("at at 1\r");
  //AT_ELM("at st 9\r");
  AT_ELM("at e0\r");
  //AT_ELM("at sp 5\r");                  //5-FAST INIT    10 AUTOMATIC
  AT_ELM("at tp 5\r");
  AT_ELM("at ib10\r");
  AT_ELM("at dp\r");
```

```
  //AT_ELM("at st 19\r");   // 19 ->25 *4mc = 100mc
  //AT_ELM("at at 2\r");
  //AT_ELM("at st 25\r");   // 19 ->25 *4mc = 100mc
  nastr = 1;
  BTSerial.flush();

  // //
  //Serial.println("------end nastrELM------");
}// end nastrELM


void loop()
{
  //Serial.println("\n------- loop -");
  delay(50);
  if (nastr == 0) nastrELM();
  AT_ELM("at i\r"); //inicializandoELM !!!
  AT_ELM("at dp\r");
  //if (nastr == 0) nastrELM();
  AT_ELM("at rv\r");
  AT_ELM("at i\r");
  BTSerial.flush();
   AT_ELM("01 14\r");
  //AT_ELM("01 00\r");
  AT_ELM("01 05\r"); //Temperatura del refrigerante; A-40; 27-40=-13
  //AT_ELM("01 0F\r"); //Intake air temperature; A-40; 30-40=-10
  //AT_ELM("01   11\r");   //Posición   del   acelerador;   A*100/255;
37*100/255=14,51 %
  AT_ELM("01 14\r");
  AT_ELM("01 06\r"); // (B-128)*100/128 Short
  AT_ELM("01 14\r");
  AT_ELM("01 07\r"); // (B-128)*100/128 Long
  //AT_ELM("01 0B\r"); //Presión del colector de admisión; A; 101
  //AT_ELM("01 10\r"); //Flujo de aire ((A*256)+B)/100
  AT_ELM("01 14\r"); // A/200  :  (B-128) * 100/128 (if B==$FF, sensor
no usado)
  AT_ELM("01 0C\r"); //Velocidad del motor; ((A*256)+B)/4 ; 0
  //AT_ELM("01 04\r"); //Carga nominal del motor: A*100/255;
  AT_ELM("01 14\r"); // A/200  :  (B-128) * 100/128 (if B==$FF, sensor
no usado)
  AT_ELM("01 15\r"); //
  AT_ELM("01 14\r");
}//loop


void AT_ELM (char bufIn[40])
{
  long ttime = 0;
  int i = 0;
  int j = 0;
  char bufOut[40];
  for(int j=0; j<39; j++) bufOut[j]='-';
  BTSerial.print(bufIn);
  ttime = millis();

  while(incomingbyte !='>')
  {
```

```
  while (BTSerial.available() > 0) // verifica si el puerto esta
disponible
    {
      incomingbyte = BTSerial.read(); // lee los datos del puerto
      bufOut[i++] = incomingbyte;
      if (i >37) break;
    }
    if ( millis()-ttime >2000) break;
  }
  BTSerial.flush();
  bufOut[i++]='\0';
  incomingbyte ='-';

  if(bufOut[0]=='N' && bufOut[1]=='O') {
    nastr = 0;
  }

//recepcion y decodificacion de datos provenientes del OBD2
//ELM
  if  ((bufOut[0]=='a'  &&    bufOut[1]=='t'  &&    bufOut[2]==' ')  ||
(bufOut[0]=='0' &&  bufOut[1]=='1' &&  bufOut[2]==' '))
      {
            //Serial.print("???");
            nastr = 0;
   } // Afinación !!!

  if (bufOut[17]=='F' &&  bufOut[18]=='A' &&  bufOut[19]=='S')
     { //Serial.println("F");
   }

  if (bufOut[0]=='E' &&  bufOut[1]=='L' &&  bufOut[2]=='M')
    { //Serial.println("E");
  }

  if (bufOut[2]=='.' &&  bufOut[4]=='V')
    {
     //Serial.print("    VOLT");
     Volt=(((bufOut[0]-48)*100)+((bufOut[1]-48)*10)+(bufOut[3]-48));
    } //12,0

  if ( bufOut[1]=='.' &&  bufOut[3]=='V' )
    {

    }  //13,0

  //EBU
  if (bufOut[0]=='4' && bufOut[1]=='1')
      {

      if (bufOut[3]=='0' && bufOut[4]=='5') // T OG
      {
        HA = Hex2ToByte (bufOut, 6);
        Temp=HA-40;
      }

      if (bufOut[3]=='0' && bufOut[4]=='6') //short
      {
        HA = Hex2ToByte (bufOut, 6);
```

```
        shotlong = (float(HA)-128)*100/128;    //shotlong = -1.66;

      }

      if (bufOut[3]=='0' && bufOut[4]=='7') //long
      {
        HA = Hex2ToByte (bufOut, 6);
        shotlong = (float(HA)-128)*100/128;
      }

      if (bufOut[3]=='0' && bufOut[4]=='C') //rpm
      {
        HA = Hex2ToByte (bufOut, 6);
        HB = Hex2ToByte (bufOut, 9);
        rpm = ((HA*256)+HB)/4;
      }

      if (bufOut[3]=='1' && bufOut[4]=='4') //rpm
      {
        HA = Hex2ToByte (bufOut, 6);
        shotlong = float(HA)/200;
      }

      if (bufOut[3]=='1' && bufOut[4]=='5') //rpm
      {
        HA = Hex2ToByte (bufOut, 6);
        shotlong = float(HA)/200;
      }

          }
    vv=Volt/10.0;
    Serial.print(rpm);
    Serial.print(":");
    Serial.print(Temp);
    Serial.print(":");
    Serial.print(vv);
    Serial.print("\n");

}//AT_ELM

int Hex2ToByte (char buf[], byte n)//CONVERSION HEXADECIMAL A BYTE
{
  int y=0;
  if (buf[n]-'0' >16) { y = y + (buf[n]-('A' - 10)) << 4  ;}
  if (buf[n]-'0' <16) { y = y + (buf[n]-'0') << 4  ;}
  if (buf[n+1]-'0' >16) { y = y + buf[n+1]-('A' - 10);}
  if (buf[n+1]-'0' <16) { y = y + buf[n+1]-'0';}
  return y;
}

// //
int freeRam () {
  extern int __heap_start, *__brkval;
  int v;
  return (int) &v - (__brkval == 0 ? (int) &__heap_start : (int)
__brkval);
}
```

Anexo 4

Código de la aplicación Android:

MainActivity.java

```
package com.nibemi.lectores.lector;

import android.app.Activity;
import android.content.pm.ActivityInfo;
import android.os.AsyncTask;
import android.os.Handler;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.Toast;

import org.w3c.dom.Text;

import java.util.Timer;
import java.util.TimerTask;

public class MainActivity extends Activity implements
AdapterView.OnItemClickListener {
    private BluetoothUtils bluetooth;
    private ListView dispositivos;
    private TextView rpm;
    private TextView temp;
    private TextView volt;
    private Button acabar;

    private Handler handler = new Handler();
    private Timer mTimer1;
    private TimerTask mTt1;
    private Handler mTimerHandler = new Handler();

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        //getSupportActionBar().hide();

setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);

        rpm = (TextView) findViewById(R.id.rpm);
        temp = (TextView) findViewById(R.id.temperatura);
```

```
        volt = (TextView) findViewById(R.id.voltaje);

        acabar = (Button) findViewById(R.id.acabar);
        acabar.setVisibility(View.INVISIBLE);

        bluetooth = new BluetoothUtils();
        dispositivos = (ListView)
findViewById(R.id.listado_dispositivos);
        String[] nombres = bluetooth.getNames();
        dispositivos.setAdapter(new ArrayAdapter<String>(this,
android.R.layout.simple_list_item_1, nombres));
        dispositivos.setOnItemClickListener(this);

        acabar.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                if (v.getId()==R.id.acabar) {
                    Toast.makeText(getApplicationContext(),"Conexion
Terminada... Adios!", Toast.LENGTH_SHORT).show();
                    bluetooth.disconnect();
                    finish();
                }
            }
        });
    }

    // Cuando cerramos la app desconectamos
    protected void onPause() {
        cancelTimer();
        super.onPause();
        bluetooth.disconnect();
    }
    // Conexion a un dispositivo vinculado
    public void onItemClick(AdapterView<?> ag, View v, int index, long
id) {
        //Conectamos con el elemento pulsado
        if (bluetooth.connect(index))
            Toast.makeText(this, "Conectado correctamente",
Toast.LENGTH_SHORT).show();
            startTimer();
            dispositivos.setVisibility(View.INVISIBLE);
            acabar.setVisibility(View.VISIBLE);
    }

    Runnable runnable = new Runnable() {
        @Override
        public void run() {
            String recibe = bluetooth.recibir();
            try{
                String[] separated = recibe.split(":");

                rpm.setText(separated[0]);
                temp.setText(separated[1]);
                volt.setText(separated[2]);
            }catch(Exception e){
                // cadena vacia
            }
            startTimer();
        }
```

```
    };

    public void startTimer() {
        handler.postDelayed(runnable, 1000);
    }

    public void cancelTimer() {
        handler.removeCallbacks(runnable);
    }

    @Override
    protected void onStop() {
        super.onStop();
        handler.removeCallbacks(runnable);
    }
}
```

```
package com.nibemi.lectores.lector;

import java.io.IOException;
import java.io.InputStream;
import java.util.ArrayList;
import java.util.UUID;

import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothSocket;
import android.os.Bundle;
import android.widget.TextView;

public class BluetoothUtils {

    // id de la aplicación para la conexión con bluetooth
    private static final String UUID_CODE = "00001101-0000-1000-8000-
00805F9B34FB";

    // declaración de variables para uso de bluetooth
    private ArrayList<BluetoothDevice> devices;
    private BluetoothAdapter adapter;

    // variable socket para el envio, recepción y conexión del
bluetooth
    private BluetoothSocket socket;

    public BluetoothUtils() {
        devices = new ArrayList<BluetoothDevice>();

        //Obtenemos el dispositivo bluetooth del terminal
        adapter = BluetoothAdapter.getDefaultAdapter();
```

```
        //Si no hay dispositivo terminamos
        if (adapter == null)
            return;

        //Obtenemos todos los dispositivos bluetooth
        //vinculados
        for (BluetoothDevice d : adapter.getBondedDevices())
            devices.add(d);

    }

    /**
     * Devuelve el nombre de los dispositivos
     * vinculados para poder seleccionarlo
     * @return
     */
    public String[] getNames() {
        String names[] = new String[devices.size()];

        for (int i = 0; i < devices.size(); i++)
            names[i] = devices.get(i).getName();

        return names;
    }
```

BluetoothUtils.java

```
    /**
     * Método para conectar a un dispositivo Bluetooth
     * según su posición en la lista.
     * @param index
     * @return si ha conectado o no
     */
    public boolean connect(int index) {
        if (index < 0 || index >= devices.size())
            return false;

        try {
            //Obtenemos el dispositivo según su posición
            BluetoothDevice device = devices.get(index);
            //Conectamos con el dispositivo
            socket =
device.createInsecureRfcommSocketToServiceRecord(UUID.fromString(UUID_
CODE));
            socket.connect();
            return true;

        } catch (IOException e) {
            //Si ha ocurrido algún error devolvemos false
            e.printStackTrace();
            return false;
```

```
        }


    }

    /**
     * Método para desconectar
     */
    public void disconnect() {
        if (isConnected()) {
            try {
                socket.close();
            } catch (IOException e) {

            }
        }
    }


    /**
     * Método que nos indica si está o no conectado
     * el socket bluetooth
     */
    public boolean isConnected() {
        if (socket == null)
            return false;

        return socket.isConnected();
    }


    /**
     * Método para enviar un dato mediante bluetooth hacia el arduino
     * @param dato
     */
    public void send(int dato) {

        //Si el socket está a null es que no hemos conectado
        if (socket == null)
            return;


        try {
            socket.getOutputStream().write(dato);

        } catch (IOException e) {}
    }

    /**
     * Método para recibir un dato o string mediante bluetooth desde
el arduino
     */

    public String recibir(){
        if (socket==null){
            return "0";
        }
```

```
        final byte delimiter=10;
        int position=0;
        String envio="";
        byte[] readBuffer=new byte[1024];

        try{
            int bytesAvailable= socket.getInputStream().available();
            if (bytesAvailable>0){
                byte[] packetBytes = new byte[bytesAvailable];
                socket.getInputStream().read(packetBytes);

                for (int i=0; i<bytesAvailable; i++){
                    byte b= packetBytes[i];
                    if(b == delimiter) {
                        byte[] encodedBytes = new byte[position];
                        System.arraycopy(readBuffer, 0, encodedBytes,
0, encodedBytes.length);
                        final String data = new String(encodedBytes,
"US-ASCII");
                        position = 0;
                        envio=data;
                    }
                    else {
                        readBuffer[position++] = b;
                    }
                }


            }

            return envio;

        }catch (IOException e){ return "0"; }

    }


}

    public static BluetoothUtils newInstance() {

        Bundle args = new Bundle();

        BluetoothUtils fragment = new BluetoothUtils();
        fragment.setArguments(args);
        return fragment;
    }
```