UNIVERSIDAD TECNOLÓGICA EQUINOCCIAL

FACULTAD DE CIENCIAS DE LA INGENIERÍA
CARRERA DE INGENIERÍA MECATRÓNICA

SISTEMA DE SEGUIMIENTO EN TIEMPO REAL Y OBTENCIÓN DE DATOS DE UN AUTOMÓVIL VÍA REMOTA

TRABAJO PREVIO A LA OBTENCIÓN DEL TÍTULO
DE INGENIERO EN MECATRÓNICA

JOSÉ GERARDO ACOSTA ARIAS

DIRECTORA: ING. JENNY JÁCOME

QUITO - ECUADOR

2015

DECLARACIÓN

Yo **JOSE GERARDO ACOSTA ARIAS**, declaro que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

La Universidad Tecnológica Equinoccial puede hacer uso de los derechos correspondientes a este trabajo, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normativa institucional vigente.

_____

José Gerardo Acosta Arias

C.I. 1003241856

CERTIFICACIÓN

Certifico que el presente trabajo que lleva por título "**Sistema de seguimiento en tiempo real y obtención de datos de un automóvil vía remota**", que, para aspirar al título de **Ingeniero en Mecatrónica** fue desarrollado por **José Gerardo Acosta Arias**, bajo mi dirección y supervisión, en la Facultad de Ciencias de la Ingeniería; y cumple con las condiciones requeridas por el reglamento de Trabajos de Titulación artículos 18 y 25.

_____

Jenny Jácome

DIRECTORA DEL TRABAJO

C.I. 1802820736

AGRADECIMIENTOS

Primeramente quiero agradecer a Dios por haberme permitido culminar este objetivo muy importante, además de haberme dado fuerza y salud para continuar día a día superando obstáculos y brindándome sabiduría para compartir con los demás.

Agradezco a mi madre Luz María Acosta y a mi abuelo Gerardo Acosta (+) por ser un ejemplo de superación y un gran apoyo durante el transcurso de mi vida personal y profesional.

De igual manera agradezco a mi directora de tesis, Ingeniera Jenny Jácome, quien me brindó sus conocimientos durante todo el transcurso de mi carrera universitaria, quien fue una guía para desarrollar este proyecto y una gran persona.

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE ANEXOS

RESUMEN

En el proyecto se diseñó un sistema de monitoreo vehicular en tiempo real y de obtención de datos de telemetría con envío de un reporte de parámetros por correo electrónico, posee una interfaz visual amigable y sencilla para el manejo de los usuarios y está contenida en una estructura anti-golpes realizada en fundición de plástico en una impresora 3D.

La interfaz gráfica visual fue desarrollada en Windows Presentation Fundation C# por su completa factibilidad, compatibilidad y eficiencia al enlazar paquetes de desarrollos dentro del entorno Windows, además de tener una base sólida y soporte en línea que permite al desarrollador elaborar proyectos de mucho interés para el cliente y estar actualizados con novedades sobre vínculos con otras plataformas de desarrollo o software adicionales que complementan los proyectos de ingeniería.

El sistema se ejecuta en una laptop con Windows desde la versión 7 en adelante como sistema operativo primordial base; este dispositivo debe permanecer fijo en un soporte estático; el equipo estuvo conectado a la red en todo momento gracias a un router USB Wifi con plan de datos activo, para la ubicación satelital se adaptó una antena GPS USB.

Se consideró el SDK de Bing Maps que permite al usuario visualizar el mapa del globo terráqueo con la finalidad de desarrollar proyectos de seguimiento, ubicación, localización y en algunos casos de seguridad, gracias a que el paquete de desarrollo permite conocer datos de ubicación del vehículo instantáneamente de manera visual gracias a la interfaz gráfica.

El dispositivo adicional que este proyecto utilizó es un escáner OBD (On Board Diagnostics) el cual se lo ha empleado con la finalidad de obtener los parámetros automotrices como: velocidad, presiones del motor, torque, nivel

de combustible, consumo de combustible, temperatura, entre otros parámetros.

El escáner OBD se enlazó con una placa de desarrollo Arduino MEGA 2560 que permitió manipular los datos proporcionados por el escáner para que al ser enviados al computador puedan alimentar una tabla de datos junto a los parámetros de posición para posteriormente ser enviada remotamente mediante un correo electrónico a un administrador que tiene acceso a estos datos en todo momento y planificar revisiones y mantenimientos del vehículo.

ABSTRACT

In this project was designed, programmed and developed a vehicle tracking system through an optimal visual interface for users, this Project was developed in Windows Presentation Fundation by C# for its feasibility, compatibility and efficiency by linking packages developments within the Windows environment, besides having a solid base and online support that allows the developer to develop projects of great interest to the client and be updated with news about links with other development platforms or additional software that complement engineer projects. Bing Maps SDK was used in this project. It lets you use the global map to develop tracking, location, and in some cases safety projects. This development package let us to know the location data used by vehicle in real-time, also we will have a report of location on a dastabase with the automotive parameters.

An additional device that this project will use is an OBD scanner (On Board Diagnostics), who permits to have the completely automotive parameters such as speed, engine load, torque, fuel level, temperatures, and some percentages among many other parameters. The OBD scanner will be linked to a development board Arduino MEGA 2560 that allow us to manipulate the data from the scanner to the computer and feed the database table, then are sent to a server remote who will have access to this data at all times and plan preventive and corrective actions to the vehicle.

The GUI developed in WPF C# remains on a laptop with Windows 8 as a primary operating system. The laptop will stay static on the vehicle, the equipment will be connected to the network at all times thanks to a USB Wifi router with active data plan and will be connected to satellite USB GPS antenna to get location data.

1. INTRODUCCIÓN

En la actualidad, gracias al avance tecnológico se puede encontrar dispositivos diseñados para realizar cualquier función o actividad para satisfacer las demandas del mercado, estos dispositivos pueden ser aplicados en sistemas de procesos, plantas automatizadas, máquinas, entre otros, con la finalidad de transmitir índices de parámetros o permitir monitorear estos sistemas para tomar acciones preventivas.

Debido a la globalización, empresas privadas en el Ecuador han empezado a importar tecnología con la finalidad de comercializar proyectos personalizados para otras empresas, lo que ha generado que el costo de las soluciones sea elevado. En el país existe una gran demanda de tecnología por parte de las empresas ya que desempeña un papel muy importante en las actividades industriales además de ser una ventaja competitiva, se puede recordar que las empresas que no se adapten al cambio y a la modernización de su sistema de trabajo quedan rezagadas de la competencia y fracasan en el tiempo.

Los vehículos están expuestos a gasto de combustible innecesario, no se tiene un monitoreo constante del lugar por donde el vehículo se encuentra, además de riesgos de seguridad a causa de la delincuencia presente en todo lugar. Por otra parte el administrador de los vehículos no posee un monitoreo de datos de telemetría lo que causa pérdida de tiempo, inhabilitación del vehículo por mantenimiento y pérdida de dinero por gastos mecánicos.

Se pretende utilizar un dispositivo de conexión satelital (GPS) y otro de obtención de datos de telemetría (OBD II) con la finalidad de entregar datos de posición exactos y parámetros de estado del vehículo en cualquier lugar de manera remota para indicar al administrador sobre el estado del vehículo,

con la finalidad de programar sus mantenimientos preventivos y correctivos, además de planificar presupuestos a invertir en los vehículos.

El objetivo de este proyecto es desarrollar un sistema de plataforma abierta que permite ir realizando mejoras de funcionamiento constante según avance la tecnología, satisfaciendo los requerimientos del cliente y demostrando que la tecnología que aparentemente es muy avanzada, costosa y difícil de manipular, puede estar al alcance de nuestras manos.

Los proyectos de enlace satelital colaboran con la adquisición de datos sobre las coordenadas de posicionamiento gracias a las ventajas de un GPS incorporado que permiten durante todo momento solucionar inconvenientes de ubicación de los vehículos y colaborando con el monitoreo de parámetros de ubicación que portan estos sistemas y pueden ser de gran ayuda en algunas circunstancias.

OBJETIVO PRINCIPAL

Desarrollar el sistema de seguimiento vehicular en tiempo real, junto a la recopilación de datos de ubicación y telemetría del vehículo proporcionado por el sensor de ubicación y el escáner automotriz para ser almacenados en un archivo y enviados por correo electrónico hacia un dispositivo administrador.

OBJETIVOS ESPECÍFICOS

Además se debe tomar en cuenta los siguientes puntos clave para el desarrollo de este proyecto que son:

- Realizar una correcta comunicación entre la placa electrónica, el escáner automotriz y la antena GPS con la laptop del usuario, la cual recibirá los parámetros conjuntamente.
- Obtener los datos de telemetría como kilometraje, velocidad, RPM, acelerador, temperatura del refrigerante, tiempo de admisión, presión del combustible, nivel de combustible, presión barométrica, tiempo de recorrido, distancia, latitud, longitud , hora y fecha exactos para ser almacenados en la unidad C: dentro del ordenador.
- Diseñar una interfaz visual de fácil manejo y comprensión para el usuario final.
- Realizar pruebas de funcionamiento tanto en un vehículo a diésel o a gasolina como en un simulador automotriz.
- Construir un contenedor adecuado para el proyecto que evite golpes internos de la placa de desarrollo.

2. MARCO TEÓRICO

Es necesario revisar la fundamentación teórica sobre temas específicos que se incluyen en el desarrollo de este proyecto con la finalidad de aclarar dudas que puedan surgir al lector sobre el funcionamiento del dispositivo.

2.1 DISEÑO CAD (DISEÑO ASISTIDO POR COMPUTADORA)

Es el uso de software para crear representaciones gráficas de objetos en dos o en tres dimensiones. El software CAD puede ser especializado para usos y aplicaciones específicas. CAD es utilizado para la animación computacional y efectos especiales en películas, publicidad y productos de diferentes industrias, donde el software realiza cálculos para determinar una forma y tamaño óptimo para una variedad de productos y aplicaciones de diseño industrial. (SIEMENS, 2015)

CAD son las siglas de Computer Aided Design, en castellano, diseño asistido por computador. Se trata de una tecnología de software aplicada al diseño de geometrías para realizar el diseño de piezas y conjuntos en 3D y sus planos 2D. De esta manera se sustituyen las herramientas tradicionales del diseñador como son la mesa de dibujo, la regla y escuadra, lápices, borradores, rapidógrafos y compases, por computadores.

Como sus siglas indican, este software no deja de ser un asistente para el desarrollo de las tareas del diseñador, quien tiene los conocimientos tanto para el manejo del software CAD como de las especificaciones técnicas de diseño y amplios conocimientos de dibujo técnico.

Inicialmente en las herramientas CAD se podía realizar el diseño en el plano conformado por las coordenadas X y Y en el espacio, llamado bidimensional (2D) que es la representación de una persona dibujando sobre el papel, posteriormente los diferentes tipos de software CAD han evolucionado a tres

dimensiones de diseño (3D), donde se ha añadido la coordenada Z, ver en la figura1 un ejemplo de modelado CAD en 3D. (Sanz, 2009)

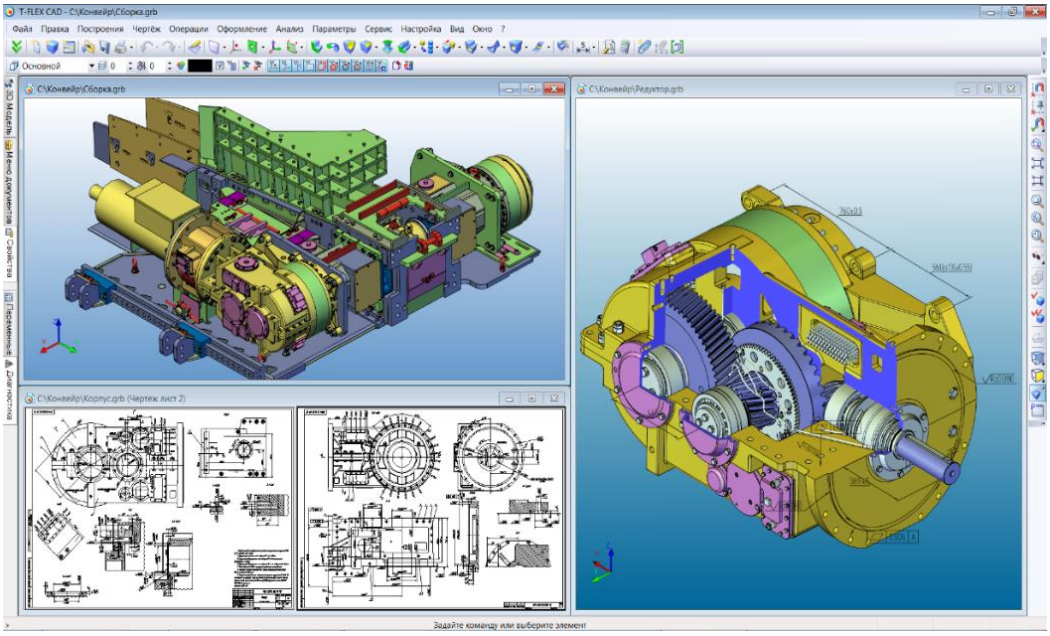


Figura 1 Diseño CAD de motor en tres coordenadas

Fuente: (Vasquez, 2008)

El cambio de herramientas tradicionales hasta el software CAD actual posee las siguientes ventajas:

- Sustitución de enormes cantidades de planos en papel (normal, poliéster, entre otros tipos), economizando en espacio y tiempo, ya que el soporte pasó a ser almacenado en el disco duro del ordenador, dispositivos extraíbles o en servidores de almacenamiento.
- Eficiencia en el desarrollo de productos con mayor precisión y velocidad.

Actualmente el CAD se ha convertido en algo más que el desarrollo de un proyecto o de un sólido en un papel, para pasar a ser una parte imprescindible de cualquier proceso de industrialización de un producto. Existen multitud de procesos de fabricación que no podrían llevarse a cabo sin un archivo CAD,

tales como el desarrollo de bosquejos rápidamente, el CAM (Manufactura Asistida por Computadora), el control dimensional, etc.

2.2 POSICIONAMIENTO SALTELITAL GPS

Los objetivos iniciales del GPS (Global Positioning System) en español Sistema de Geo-Posición, fueron militares. El surgimiento de este sistema de gran utilidad mundial fue similar al de la actual Internet, creado e instalado a comienzos de los años 60 por el Departamento de Defensa, Transporte y la Agencia Espacial Norteamericana. Sin embargo, sus utilidades están hoy relacionadas con servicios que permiten mejorar la calidad de vida de las personas. (Eduardo Huerta, 2005)

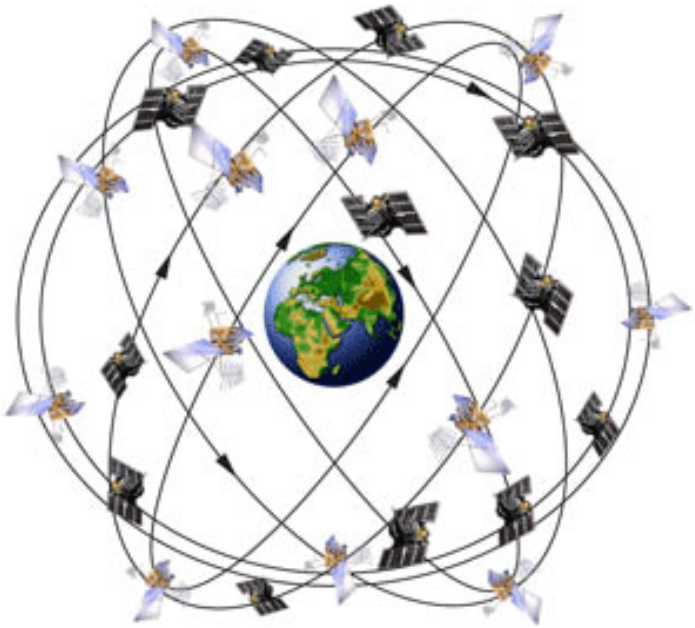


Figura 2 Representación de una red de satélites

Fuente: (Eduardo Huerta, 2005)

El GPS funciona mediante una red de 24 satélites que permiten determinar en todo el mundo la posición de una persona, un vehículo o una nave, con una precisión muy exacta, ver en la figura 2.

El GPS funciona a través de un Sistema Global de Navegación por Satélite (GNSS) que permite establecer la posición de una persona, un vehículo o una nave en todo el mundo a través de la longitud, latitud y altura.

Los elementos que lo componen son:

- Sistema de satélites: formado por 24 unidades con trayectorias sincronizadas para cubrir toda la superficie del globo. Más concretamente, repartidos en 6 planos orbitales de 4 satélites cada uno.
- Estaciones terrestres: envían información de registro a los satélites para controlar las órbitas.
- Terminales receptores: indican la posición en la que se encuentran las personas, vehículos u objetos. A estas terminales se las conoce como Unidades GPS y pueden adquirirse en los comercios.

2.2.1 Funcionamiento del GPS

Cuando hace falta establecer la posición de un determinado objeto o persona que portan una unidad GPS, el terminal receptor utiliza un sistema de localización automática a través de, como mínimo, cuatro satélites de la red.

De ellos recibe una señal indicando la posición y el reloj que marca cada uno. En base a estas señales, el receptor sincroniza el reloj del GPS y calcula el retraso de las señales y la distancia a los satélites. Por triangulación, computando cada una de las cuatro señales respecto al punto de medición, se determina la posición relativa respecto a los cuatro satélites. Conociendo además las coordenadas o posición de cada uno de ellos a través de la señal que emiten se obtiene la posición absoluta o coordenadas reales del punto de medición. A su vez, se consigue una exactitud extrema en el reloj del GPS, similar a la de los relojes atómicos que desde la tierra sincronizan a los satélites. De esta manera, conocidas las distancias, se determina fácilmente la propia posición relativa respecto a los cuatro satélites.

En la actualidad existen dos sistemas mundialmente reconocidos de Geo Posicionamiento Satelital, el primero creado por los Estados Unidos, llamado GPS y el segundo sistema creado por Rusia, llamado el GLONASS; para evitar dependencia de estos sistemas, la unión europea está trabajando en el lanzamiento de su propio dispositivo de ubicación satelital llamado Galileo.

2.3 ESCANER DE DIAGNÓSTICOS AUTOMOTRIZ OBD (ON BOARD DIAGNOSTICS)

En el transcurso de los años setenta y ochenta algunos fabricantes de la industria automotriz usaron dispositivos electrónicos para el control y el diagnóstico de problemas y errores automotrices. Inicialmente se los utilizó para conocer y controlar datos de emisiones proporcionadas por el vehículo y ajustarlas a estándares establecidos para la época, conforme el avance tecnológico estos dispositivos se volvieron más robustos y sofisticados, al llegar los años 90 surgió el estándar OBD (On Board Diagnostic) y posteriormente el OBD II perteneciente a la segunda generación de dispositivos, este escáner es un sistema que permite diagnosticar por medio del análisis de datos los errores de funcionamiento que el vehículo presenta sin la necesidad de desmontar físicamente las partes vinculadas al problema.

Esta codificación se encuentra instalada en la actualidad en todos los vehículos industriales y de turismo ligeros que se producen y comercializan desde el año 1996, una característica muy importante es que es un sistema estandarizado y permite de manera fácil, ver que errores se han producido en un vehículo cualquiera utilizando una única codificación y claro está, un conector estandarizado (Concepción, 2010).

Existen dispositivos OBD desarrollados especialmente para presentar una alta compatibilidad con plataformas de desarrollo abierto como Arduino con la finalidad que el diseñador la pueda vincular como una herramienta importante para sus proyectos de ingeniería. El dispositivo OBD II compatible con Arduino

trabaja como un medio de adquisición de información de los parámetros del vehículo, consta de una librería de programación donde está configurado los protocolos de comunicación para tipo y marca de cada vehículo, además posee una fuente de poder convertida y regulada desde el puerto OBD II para todas las versiones de Arduino.

Dentro de las características más representativas se tiene:

- Conectividad directa con el puerto OBD del automóvil.
- Interfaz de datos en serie (UART o I2C).
- Alta eficiencia DC-DC módulos de 5v/3v DC y salidas de 2A.
- Soporta CAN bus, protocolos KWP2000 y ISO9141-2.
- Rápido (sobre los 100Hz) acceso a OBD-II PIDs
- Válidos de un vehículo de protocolos ECU.
- Acelerómetro embebido 3-ejes, 3-ejes giroscopio y sensor de temperatura (solo modelo B).

2.3.1 Compatibilidad automotriz

Se debe mantener conectado el dispositivo dentro del puerto OBD del vehículo, el cual está situado comúnmente debajo del volante o por sus alrededores. Las versiones de manufactura soportadas por el dispositivo de acuerdo al año y país son:

- Estados Unidos (Gas) 1996 en adelante
- Estados Unidos (Diesel) 2004 en adelante
- Canadá (Gas) 1998 en adelante
- Europa + UK (Gas) 2001 en adelante
- Europa + UK (Diesel) 2004 en adelante

- Australia + NZ (Gas/Diesel) 2006 en adelante

Los vehículos poseen una certificación que asegura la conectividad con el dispositivo OBD y OBD II, es un sticker que los vehículos integran en la parte superior interna del volante y asegura la total transferencia de datos a través del puerto OBD, como se muestra en la figura 3:

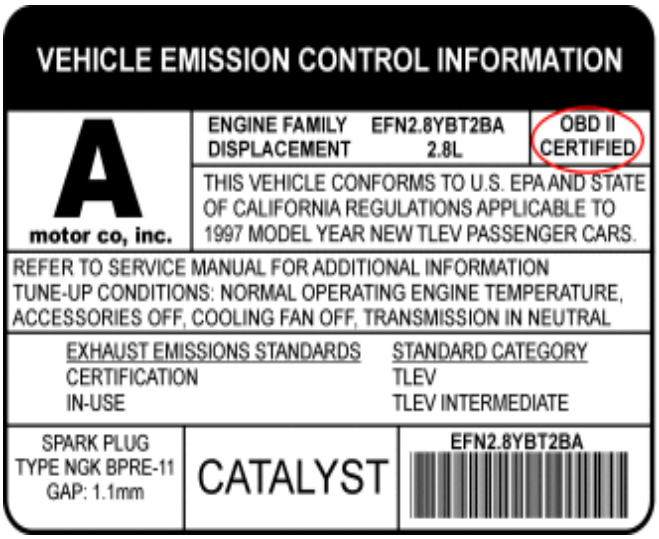


Figura 3 Información de emisión de datos de control del vehículo

Fuente: (Martínez, 2011)

2.3.2 Protocolos de comunicación

Existen tres protocolos de comunicación del sistema OBD II que dependen del vehículo emisor. Cada marca ha optado por cierto estándar a utilizar y lo ha incorporado en la fabricación y producción de cada vehículo, A continuación se observa que tipos de protocolos pueden utilizar los vehículos dependiendo de la marca y lugar de producción:

- ISO 9141-2 en vehículos Europeos, Asiáticos y Chrysler con variantes.
- SAE J1850 VPW que significa Ancho de Pulso Variable (Variable Pulse Width) y lo utiliza GM USA
- SAE J1850 PWM que indica Modulación Ancho de Pulso (Pulse Width Modulatión) utilizado por Ford USA.

- KWP 1281 y KWP (Key Word Protocol) 2000 utilizado por el grupo VAG.
- ISO 14230 que lo utiliza Renault.

Cada uno de los protocolos señalados anteriormente requiere de su correcta configuración y un especial tratamiento de información para poder conectar el OBD II con el PC.

Se requieren interfaces de conexión diferentes. Esto no es del todo exacto, ya que existe la posibilidad de fabricar un interfaz de conexión del OBDII con el PC, capaz de utilizar todos los protocolos e incluso seleccionar automáticamente cual es el protocolo utilizado por el vehículo a conectar (Martínez, 2011).

Dentro de los protocolos indicados anteriormente se subdividen ciertos protocolos con pequeñas variantes que dependen del escáner automotriz, los cuales se indican a continuación:

- CAN 500Kbps/11bit
- CAN 250Kbps/11bit
- CAN 500Kbps/29bit
- CAN 250Kbps/29bit
- ISO9141-2
- KWP2000 Fast
- KWP2000 5Kbps

Los dispositivos OBD II son compatibles con la mayoría de versiones de Arduino, el dispositivo OBD II funciona correctamente tanto al trabajar con 5v y con 3.3v. La versión de 5v es completamente compatible con las placas de desarrollo Arduino de 8 bits, entre los cuales se tiene a los modelos: UNO, Duemilanove, Leonardo, Micro, Nano, Mini, Pro Mini, MEGA 1280/2560/ADK.

Para la versión de 3.3v es compatible con la mayoría de modelos de 32 bits que incluyen al Arduino DUE y al YÚN.

2.3.3 Descripción de parámetros y conexión

El dispositivo OBD II debe permanecer conectado en el puerto del vehículo, hasta estar muy ajustado, a continuación se muestra en la figura 4 el escáner OBD II, el puerto en el vehículo y el dispositivo montado en el vehículo:

Figura 4 Escáner y puerto automotriz

Fuente: (Bryan, 2014)

En uno de sus extremos, el escáner OBD II consta de cuatro cables conectores para adaptarlos a la placa Arduino separados en dos pares, el primer par de cables corresponde a las líneas de poder que son Vcc (rojo) y GND (negro) y el otro par de cables corresponden a las líneas de datos Rx o SDA (amarillo) y Tx o SCL (blanco o azul) como se observa en la figura 5. Estos cables deben ser conectados a los puertos del Arduino correspondientes por medio de una placa auxiliar UART o directamente.

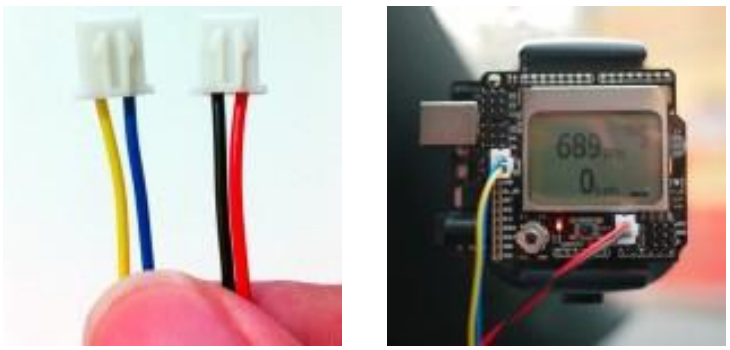


Figura 5 Cables de conectividad del escáner

Fuente: (Bryan, 2014)

La librería para Arduino desarrollada especialmente para el escáner automotriz tiene soporte constante, por lo que provee un fácil uso de la API para recuperar datos en tiempo real desde el vehículo.

Los comandos de comunicación que la librería OBD para Arduino integra para la obtención de parámetros de telemetría son:

Motor:

- PID_RPM – Motor RPM (rpm).
- PID_ENGINE_LOAD – Cálculo de carga de motor (%).
- PID_COOLANT_TEMP – Temperatura de enfriamiento del motor (°C).
- PID_ABSOLUTE_ENGINE_LOAD – Carga absoluta del motor (%).
- PID_TIMING_ADVANCE – Tiempo de avance (°).
- PID_ENGINE_OIL_TEMP – Temperatura de aceite del motor (°C).
- PID_ENGINE_TORQUE_PERCENTAGE – Porcentaje de torque del motor (%).
- PID_ENGINE_REF_TORQUE – Torque referido del motor (Nm).

Admisión/Escape:

- PID_INTAKE_TEMP – Temperatura del tanque (°C).
- PID_INTAKE_PRESSURE – Presión absoluta del tanque de admisión (kPa).
- PID_MAF_FLOW – MAF –Presión de flujo (grams/s).
- PID_BAROMETRIC – Presión barométrica (kPa).

Speed/Time:

- PID_SPEED – Velocidad del vehículo (km/h).
- PID_RUNTIME – Tiempo de recorrido del motor (second).
- PID_DISTANCE – Distancia recorrida del vehículo (km).

Conductor:

- PID_THROTTLE – Posición de la mariposa (%).
- PID_AMBIENT_TEMP – Temperatura ambiente (°C).

Sistema Eléctrico:

- PID_CONTROL_MODULE_VOLTAGE – Control de voltaje del vehículo (V).
- PID_HYBRID_BATTERY_PERCENTAGE – Batería híbrida vida útil (%).

2.4 LENGUAJES DE PROGRAMACIÓN

Son los lenguajes utilizados por el programador para generar las instrucciones necesarias para la ejecución de algoritmos y obtener un programa nativo o base (Medina, 2000).

A continuación se puede observar la figura 6, donde muestra una relación entre los tipos de lenguaje:

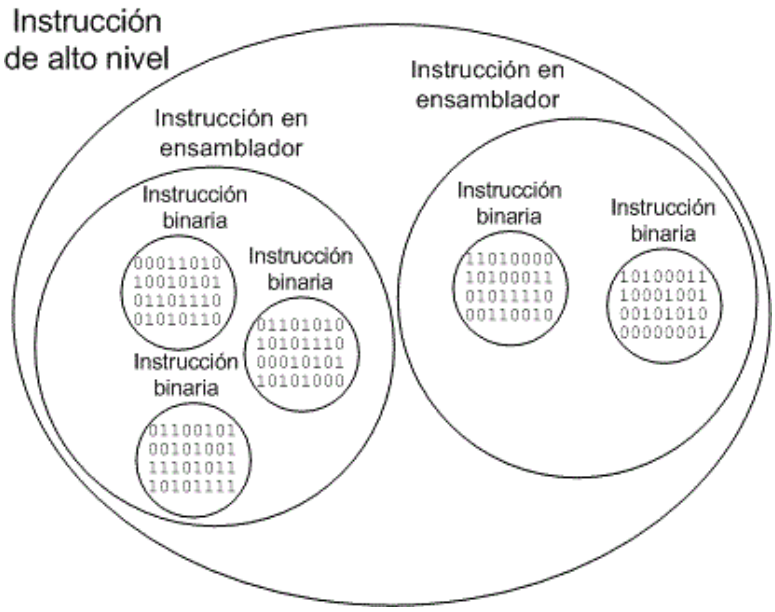


Figura 6 Relación entre las instrucciones de alto nivel y ensamblador

Fuente: (Medina, 2000)

Existen una amplia gama de lenguajes de programación según su gramática y sus tipos de instrucciones, estos son:

De alto nivel: son aquellos lenguajes de estructura próxima al lenguaje utilizado por los humanos. Son de fácil comprensión y colaboran para el diseño de programas en reducidas líneas de programación. Sus programas se pueden ejecutar en una diversa cantidad de ordenadores y requieren de un proceso de compilación. Algunos ejemplos lenguajes de alto nivel son: Ada, Basic, C#, Cobol, Fortran, Matlab, Pascal, PHP, Python, Java, Visual Basic.Net.

Características de este tipo de lenguaje:

- Algunos requieren que la máquina cliente posea una determinada plataforma.
- Escribir un código válido para diversas máquinas y, posiblemente, sistemas operativos.
- Genera un código más sencillo y comprensible.
- Reducción de velocidad al ceder el trabajo de bajo nivel a la máquina.

De bajo nivel o ensamblador: Son los lenguajes en el cual las instrucciones son directamente comprendidas y ejecutadas por el ordenador. Son de difícil interpretación para los humanos que desconocen del lenguaje y no poseen una traducción en concreto, solo se debe cambiar la sintaxis para conseguir el programa objeto, este procedimiento es conocido como "**ensamblado"**, y es realizado por un programa llamado **ensamblador"**, que equivale a la fase de compilación de los lenguajes de alto nivel.

Características de este tipo de lenguaje:

- Abstracción - Mínima por depender completamente de la técnica del hardware.

- Adaptación - Máxima entre programación y aprovechamiento del recurso de la máquina.
- Portabilidad - Mínima por estar restringido a las especificaciones del fabricante.
- Uso - Requiere de la máxima atención y de una organización estructurada en base a los planos del hardware y del objetivo del software.
- Velocidad - Máxima al contar con un acceso directo a los recursos, sin capas intermedias.

2.5 LENGUAJE DE PROGRAMACIÓN C

La sintaxis de C# es parecida a C, C++ o Java porque está basada en signos de llaves, lo que la hace expresiva, sencilla y fácil de aprender. La programación en la sintaxis de C# resulta más sencilla que en la sintaxis de C++ porque simplifica muchas de sus complejidades además de proporcionar mejores características como son los tipos de valores NULL, enumeraciones, delegados, expresiones lambda y acceso directo a memoria que las otras sintaxis no poseen.

C# acepta métodos y tipos genéricos, estas opciones prestan un óptimo rendimiento y seguridad a tipos e iteradores que permiten al usuario personalizar las definiciones y comportamientos de las iteraciones (Katrib, 2009).

C# también es un lenguaje orientado a objetos ya que está abierto a aceptar conceptos de encapsulación, herencia y poliformismo. Variables y métodos, incluido el main es el inicio de la aplicación, se encuentran encapsulados dentro de sus clases.

En C#, una estructura es como una clase sencilla; es un tipo asignado en la pila que puede implementar interfaces pero que no admite la herencia (Microsoft, 2015).

C# facilita el desarrollo de componentes de software, algunos componentes son:

- Firmas de métodos encapsulados llamados delegados, que habilitan notificaciones de eventos con seguridad de tipos.
- Propiedades, son los descriptores de acceso para variables miembro privadas.
- Atributos, que proporcionan metadatos descriptivos de tipos en tiempo de ejecución.
- Líneas de documentaciones en XML.
- Si el diseñador necesita interactuar con Windows, como objetos COM o archivos DLL nativos de Win32, se los puede vincular en C# mediante un proceso llamado "interoperabilidad", este proceso permite a los programas de C# para que realicen prácticamente las mismas tareas que una aplicación C++.
- La fase de compilación en C# es más simple en comparación de C y C++, y de mayor flexibilidad que la fase de compilación en Java pero guardan similitud entre los entornos de programación.

Los programas de C# se ejecutan en .NET Framework, un componente que forma parte de Windows y que incluye un sistema de ejecución virtual denominado Common Language Runtime (CLR) y un conjunto unificado de bibliotecas de clases. CLR es la implementación comercial de Microsoft de CLI (Common Language Infrastructure), un estándar internacional que constituye la base para crear entornos de ejecución y desarrollo en los que los lenguajes y las bibliotecas trabajan juntos sin ningún problema. El código fuente escrito en C# se compila en un lenguaje intermedio (IL) conforme con la especificación CLI. El código de lenguaje intermedio y recursos tales como mapas de bits y cadenas se almacenan en disco en un archivo ejecutable denominado ensamblado, cuya extensión es .exe o .dll generalmente. Un

ensamblado contiene un manifiesto que proporciona información sobre los tipos, la versión, la referencia cultural y los requisitos de seguridad del ensamblado, ver en la figura 7 un diagrama de las fases de ejecución y compilación de un programa en C# (Narváez, 2008).

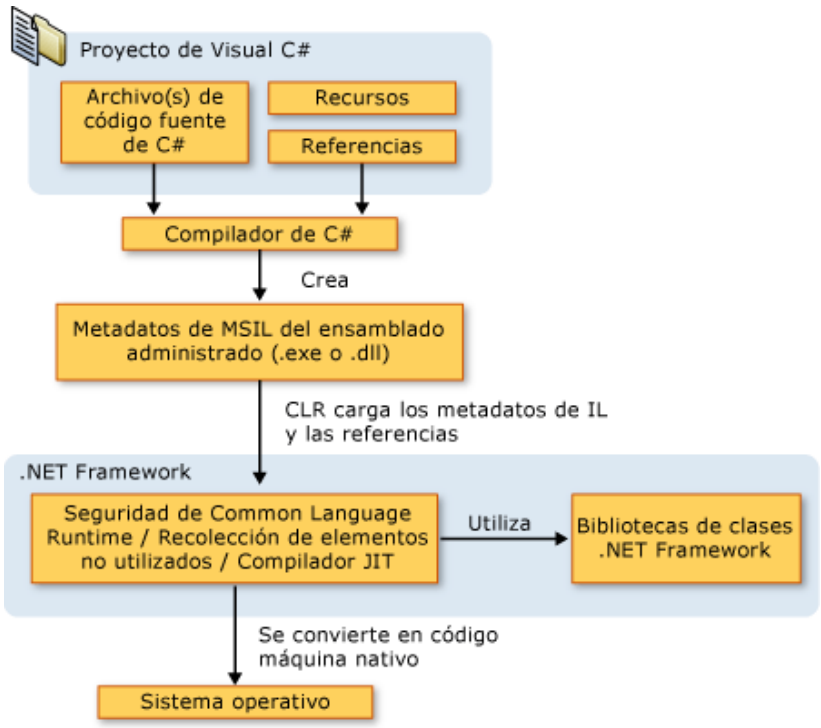


Figura 7 Diagrama de relación de compilación y ejecución

Fuente: (Microsoft, 2015)

2.6 WINDOWS PRESENTATION FUNDATION (WPF)

Windows Presentation Fundation es una interfaz para que el usuario realice sus aplicaciones clientes personalizadas con partes interactivas permitiendo un amplio desarrollo de aplicaciones, con la integración de menús interactivos, botones animados, nuevos recursos de diseño, controladores, posibilidad de generar animaciones, enlace de datos, seguridad, documentos, y otras opciones adicionales. Es un subconjunto de .NET Framework que integra Asp.NET o WinForms, por lo que la estructura de programación resulta familiar al programador. WPF usa al lenguaje XAML para realizar la programación del

diseño de la interfaz visual y el contenido lógico se lo desarrolla a través del lenguaje de programación de alto nivel C#.

WPF permite al diseñador crear proyectos vinculados a la manipulación de archivos de algunas extensiones para vincularlos con bases de datos, software de edición de archivos de texto (.txt), plantillas de datos (.xls), archivos de valores separados por coma (.csv), y otros más (MacDonald, 2012).

El formato CSV es muy sencillo y no indica un juego de caracteres concreto, ni cómo van situados los bytes, ni el formato para el salto de línea. Estos puntos deben indicarse muchas veces al abrir el archivo, por ejemplo, con una hoja de cálculo (Shafranovich, 2005) . Ejemplo:

1, 0, 0, 0, 1, 0, -1, 0, 0, 0, -1, 0, 1, 0, 0
1, 0, 0, 0, 1, 0, -1, 0, 0, 0, -1, 0, 1, 0, 0

2.7 KIT DE DESARROLLO DE SOFTWARE (SDK)

En ocasiones es necesario el uso de herramientas ya desarrolladas con la finalidad de ahorrar tiempo y esfuerzo al trabajo realizado, sin alterar el resultado final y sin descuidar la calidad del proyecto.

Un paquete de desarrollo permite elevar la capacidad de resolución de problemas del programa realizado, complementando al proyecto con funciones previamente elaboradas que podrían ser elementales.

Un Kit de desarrollo de software o SDK (Software Development Kit) es un paquete de herramientas que colaboran al desarrollo de aplicaciones para un entorno particular.

Son muchos los recursos que puede contener un SDK. Se detallan a continuación algunos de ellos:

Una interfaz de programación de aplicaciones (API). Se trata de un conjunto de funciones, rutinas, estructuras de datos, clases y variables que permiten manipular el mecanismo de la plataforma sin conocerlo internamente (Mariano, 2013).

Un entorno de desarrollo integrado (IDE). Es un editor que permite desarrollar el código fuente del programa, y brinda una interfaz amigable, contiene los siguientes elementos:

- **Debugger**. Permite al diseñador probar el programa durante cada paso de su ejecución.
- **Compilador**. Realiza la traducción del programa al lenguaje máquina, creando un programa ejecutable.
- **Código de ejemplo y otra documentación**. Es el punto de partida para el desarrollo de aplicaciones.
- **Un emulador del entorno**. Permite visualizar el resultado final del proyecto en el dispositivo que sea ejecutado.

En la actualidad, plataformas como los sistemas operativos Android, iOS y Windows Phone ofrecen kits para desarrollar software que funcione sobre sus entornos, y muchas redes sociales tienen SDK específicos para desarrollar todo tipo de aplicaciones en diferentes lenguajes (Mariano, 2013).

2.8 ARDUINO MEGA 2650

Arduino MEGA 2650 es una placa de desarrollo libre que integra un microcontrolador ATmega1280 como se indica en la figura 8, consta de 54

pines Input /Output digitales, 14 sirven como salidas PWM, 16 son entradas analógicas, consta de 4 puertos serial, tiene un cristal oscilador de 16MHz, una conexión USB, tiene un Jack de poder, conexión ICSP y un botón de reseteo. Se puede decir que es una placa completa que posee todo lo necesario para dar soporte a su microcontrolador, es de fácil uso, se lo conecta al ordenador a través del cable USB, alimentado por un cargador AC a DC o con una batería. Posee completa compatibilidad con todas las placas auxiliares diseñadas para Arduino. El esquema eléctrico de la placa de desarrollo se lo puede apreciar en el Anexo 3

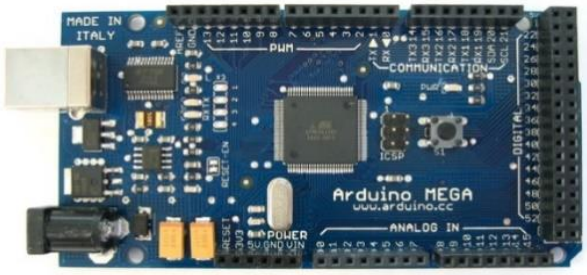


Figura 8 Arduino MEGA 2650
Fuente: (Arduino, 2015)

2.8.1 Características

Tabla 1 Parámetros de funcionamiento de la placa de desarrollo

Microcontrolador	ATmega1280
Voltage de Operación	5V
Voltage recomendado de entrada	7-12V
Límite de entrada de voltaje	6-20V
Digital I/O Pines	54 (of which 15 provide PWM output)
Pines de entradas analógicas	16
DC Corriente por I/O Pin	40 mA
DC Corriente por 3.3V Pin	50 mA
Memoria Flash	128 KB de los cuales 4 KB usados por el arrancador
SRAM	8 KB
EEPROM	4 KB
Velocidad Reloj	16MHz

Fuente: (Arduino, 2015)

2.8.2 Suministro de energía

La alimentación de la placa Arduino MEGA se da por conexión de USB o mediante una fuente externa por lo que la selección del suministro de energía es automática. El adaptador es conectado por los plugs de 2.1 mm dentro de los jacks de poder, si se ha usado una batería se debe conectar los cables a los pines Gnd y Vin del conector de poder.

El suministro de voltaje que una fuente externa proporciona puede ser de 6 a 20 voltios pero hay que tomar en cuenta que el suministro no debe tener 5v porque el funcionamiento de la placa puede ser inestable y si es mayor a 12v, el regulador de voltaje se puede sobrecargar y dañar la placa, por lo que se recomienda que el desarrollador proporcione un voltaje entre el rango de 7v a 12v.

La placa de desarrollo consta de los siguientes pines:

- **VIN.** Conocido como el pin de entrada de voltaje a la placa de desarrollo Arduino cuando se usa una fuente de poder externa.
- 5**V.** Es el pin de suministro de voltaje de 5v que puede ser proporcionado externamente por una fuente o a través del cable USB del computador.
- **3V3.** Es el pin de suministro de voltaje de 3.3v que puede ser proporcionado externamente por una fuente o a través del cable USB del computador.
- **GND**. Es el pin que corresponde a tierra.

2.8.3 Entradas y salidas

Los 54 pines digitales de la placa Arduino MEGA sirven como pines de entrada y salida, utilizando las funciones:

- pinMode(),
- digitalWrite(),
- digitalRead()

Estos pines operan a 5v con una corriente de 40mA, integran un resistor interno que se encuentra desconectado predeterminadamente cuyo valor es de 20 a 50 KΩ. Además la placa consta de:

- **Serial:** 0 (RX) and 1 (TX); Serial 1: 19 (RX) and 18 (TX); Serial 2: 17 (RX) and 16 (TX); Serial 3: 15 (RX) and 14 (TX).
- **Interruptores externos:** 2 (interruptor 0), 3 (interruptor 1), 18 (interruptor 5), 19 (interruptor 4), 20 (interruptor 3), and 21 (interruptor 2).
- **PWM:** analogWrite().
- **SPI:** 50 (MISO), 51 (MOSI), 52 (SCK), 53 (SS).
- **LED: 13.** Tiene un LED integrado a la salida digital 13 auxiliar con el que se puede probar el funcionamiento de la placa.
- **I2C:** 20 (SDA) y 21 (SCL). Pines que corresponden a la configuración de la comunicación I2C
- **AREF.** Voltaje de referencia para las entradas analógicas.
- **Reset.** Da un pulso de reset al microcontrolador. Sirve para volver a arrancar el programa del dispositivo

2.9 COMUNICACIÓN UART (UNIVERSAL ASINCRÓNICO TRANSMISOR-RECEPTOR) (SERIE)

El receptor/transmisor asíncrono universal (Universal Asynchronous Receiver/Transmitter, UART) Es un dispositivo importante cuando de comunicaciones en serie se habla, su función principal es la conversión de datos serie a paralelo al ser de entrada y de conversión de datos paralelo a

serie al ser de salida. En la figura 9 se muestra el esquema general de la comunicación serie.

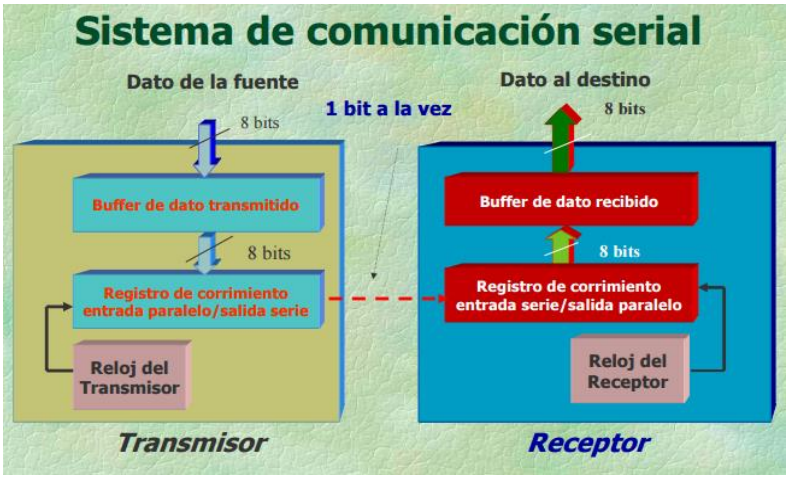


Figura 9 Diagrama del sistema de comunicación serie

Fuente: (Málaga, 2014)

Aunque la transmisión es realmente sincrónica, tradicionalmente se les ha llamado asíncrona para hacer referencia al hecho de que existe flexibilidad para la transmisión de la siguiente palabra de datos (Málaga, 2014).

Dentro de las funciones principales del chip UART son las de manejar las interrupciones de los dispositivos que se encuentran integrados por su conexión serie y de la conversión a datos paralelos, que son transmitidos por un bus de enlace a datos de tipo serie, para realizar la transmisión dúplex a través de sus puertos.

El controlador UART toma cadenas de datos para ser transmitidas en secuencia. Un segundo UART arma las cadenas de bits en bytes completos por medio de un cable único o por otros medios, la transmisión de datos por paralelo resulta ser de mayor efectividad porque intervienen multiples canales de transferencia de datos. El UART es utilizado para convertir la información transmitida entre la forma paralela y en secuencia en cada punto de enlace.

Cada UART contiene un registro de desplazamiento que es el método de conversión entre las forma serie y paralelo.

El UART no genera directamente o recibe las señales externas entre los diferentes módulos del equipo. Usualmente se usan dispositivos de interfaz separados para convertir las señales de nivel lógico del UART hacia y desde los niveles de señalización externos. (Rodríguez, 2007)

Los parámetros que se debe tomar en cuenta y definir correctamente para elaborar la interfaz de comunicaciones son:

- Sincronismo entre el receptor y el transmisor.
- Codificación de los datos.
- Prioridad en el envío de los bits.
- Tasa de envío de datos.
- Señales “handshaking”.
- Señales eléctricas de los valores lógicos

3. METODOLOGÍA

La metodología mecatrónica es un diseño concurrente que integra disciplinas como el diseño mecánico, eléctrico, de hardware y software que permiten al estudiante o profesional integrar procedimientos que complementan el desarrollo del proyecto simultáneamente para obtener un prototipo físico y virtual que ha sido expuesto a sus respectivas pruebas de funcionamiento y corrección de procedimientos defectuosos para llegar a la construcción del equipo, para el desarrollo de este proyecto se ha realizado el siguiente diagrama sobre la metodología mecatrónica utilizada en este proyecto como se aprecia en la figura 10.

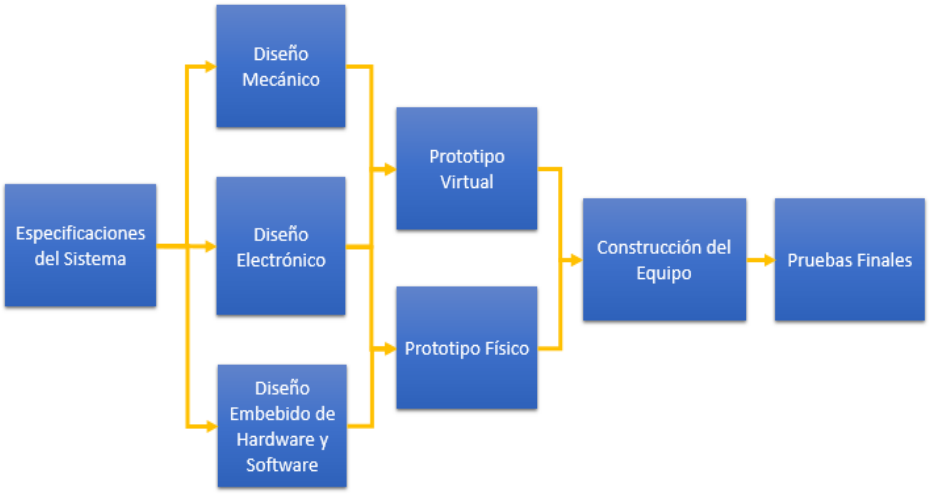


Figura 10 Diseño concurrente de la Mecatrónica

Dentro de algunas características que proporciona el diseño mecatrónico se tiene:

- Integración de componentes (hardware).
- Diseño compacto.
- Mecanismos simples.
- Comunicación física o inalámbrica.
- Integración mediante procesos informáticos (software).
- Construcción elástica.
- Retroalimentación programable.
- Precisión a través de medidas.

- Supervisión con diagnóstico de error.

El proceso concurrente de metodología mecatrónica mostrado en la figura 10, hace énfasis en el desarrollo integrativo y de forma paralela con la finalidad de realizar un diseño de mayor robustez y complejidad pero es claro para hallar posibles errores y solucionarlos de una manera efectiva y rápida.

Para establecer el proceso de diseño que este proyecto adopta se considera las siguientes etapas de investigación establecidas en la figura 11.

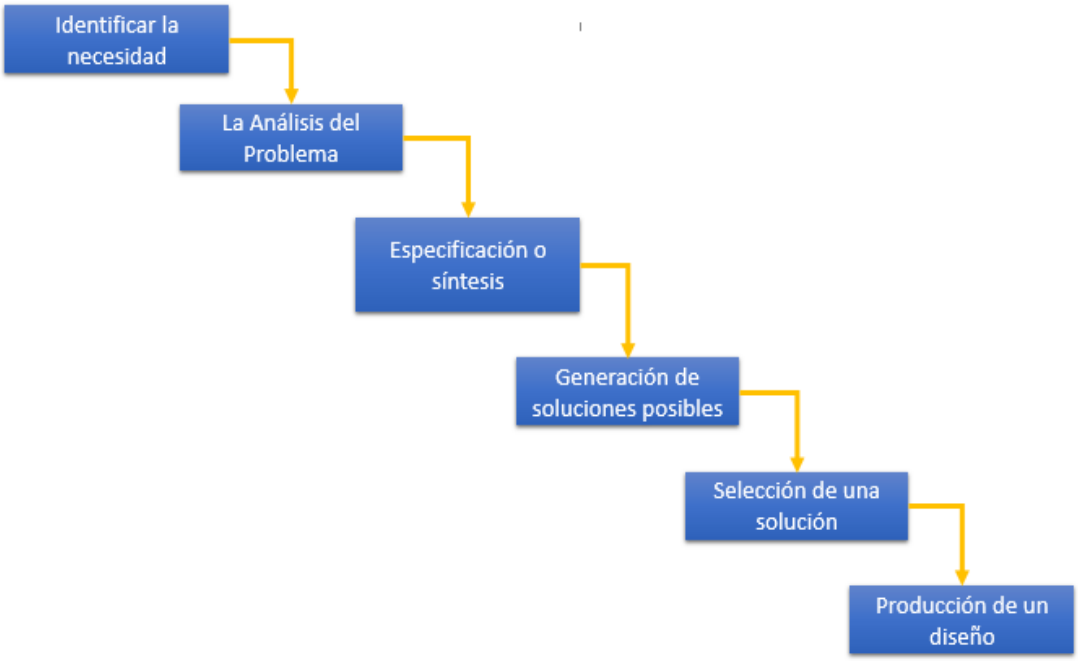


Figura 11 Proceso de diseño para sistemas

(Bolton, 2010)

El inicio del proceso de diseño se establece encontrando una necesidad que tiene el consumidor, se la lleva a cabo mediante procedimientos de investigación de mercado con la finalidad de encontrar posibles clientes potenciales, una vez identificada la necesidad es importante investigar sobre la naturaleza del problema e identificarlo con exactitud, esto podría ayudar a ganar tiempo de diseño y la satisfacción de la necesidad.

A continuación se procede a realizar una síntesis de los requerimientos de diseño, tomando en cuenta criterios de calidad y las posibles vías de solución

dependiendo el problema planteado, además se especifica todas las características que se quiere que el diseño tenga, encontrando las soluciones de diseño que faciliten los procedimientos para su rápida finalización. Las soluciones se evalúan mediante procedimientos de modelado o simulaciones y la de mayor calificación es la que se toma para finalizar con el desarrollo del prototipo diseñado y sus respectivas pruebas de funcionamiento.

3.1 PARÁMETROS DE FUNCIONABILIDAD

A continuación se realiza un análisis del funcionamiento del proyecto, tomando en cuenta criterios de necesidad del cliente conjuntamente a decisiones de criterios de ingeniería que pueden complementar la eficiente construcción del dispositivo.

3.1.1 Dimensionamiento

Es importante conocer el dimensionamiento del dispositivo ya que se encontrará ubicado en algún lugar por donde se encuentra el puerto OBD del vehículo por lo que tiene que ser un elemento pequeño que no recurra a un gran espacio ni moleste a los pedales y se debe conocer su tiempo de operación continuo, en la siguiente tabla se indica una ficha técnica del dispositivo.

Tabla 2 Ficha técnica del dispositivo

Parámetros	(mm)	(°C)	(días)
Largo	115		
Ancho	60		
Altura	35		
Temperatura		30 aprox.	
Trabajo			5 aprox.

Se puede observar en la tabla que las dimensiones del dispositivo son permitidas, no es un elemento grande que pueda ocasionar molestias al conductor, la temperatura adecuada para un óptimo funcionamiento es de 30 grados aproximadamente y un período de trabajo de 5 días para evitar excesivo desgaste del microcontrolador.

3.1.2 Requerimientos técnicos y del cliente

Dentro de los requerimientos técnicos a tomar en cuenta para el desarrollo del proyecto por parte del cliente y del diseñador se tiene:

- **Confiabilidad:** El usuario debe estar seguro que los datos proporcionados por el dispositivo son reales y puede vincularlos a otros servicios externos a este proyecto.
- **Control:** El usuario debe conocer sobre el procedimiento a seguir para manipular el software
- **Costo:** Este parámetro depende de la cantidad de dinero a invertir para el desarrollo de este proyecto por parte del cliente final.
- **Funcionalidad:** El dispositivo debe realizar los procedimientos requeridos por el cliente en cualquier momento que sea utilizado el software.
- **Mantenimiento:** El dispositivo no debe estar sellado con su estructura para realizar mantenimientos preventivos o correctivos de acuerdo a un período de revisión designado por el diseñador.
- **Operatividad:** El dispositivo debe ser utilizado estrictamente para realizar el trabajo por el que fue desarrollado y no se utilizarán los equipos para otras actividades.
- **Seguridad:** Se necesita fijar la estructura del dispositivo, junto al equipo de monitoreo en el vehículo, la placa de desarrollo debe estar protegida en caso de cortocircuitos.

- **Tamaño:** La estructura debe poseer dimensiones permisibles para que no ocupe espacios exagerados en el vehículo.

Dentro de los criterios establecidos por el cliente a tomar en cuenta para el desarrollo y selección de alternativa se tiene:

- **Buen precio:** El dispositivo realizado y el desarrollo de software debe ser económico o competente con el establecido en el mercado.
- **Fácil manejo:** El software debe ser de simple interacción con el usuario, controlable, apreciable y su interfaz visual clara y concisa.
- **Fácil instalación:** La implementación del dispositivo no debe generar molestias al conductor y tiene que ser de simple instalación.
- **Larga vida útil:** Los elementos que componen el dispositivo deben resistir a desgastes por el uso diario y permanente.
- **Repuestos económicos:** Los repuestos o partes utilizadas en este proyecto deben ser comercializados en el mercado tecnológico y estar al alcance del cliente a precios cómodos.

3.1.3 Restricciones del dispositivo

- **Tamaño:** El dispositivo puede ser instalado en cualquier tipo de vehículo siempre y cuando este no sobrepase dimensiones establecidas en la tabla 2.
- **Compatibilidad:** la instalación del dispositivo debe ser para vehículos que posean el puerto de conexión del escáner OBD, además de que se puedan enlazar con la placa de desarrollo mediante la configuración de los protocolos establecidos por el chip ELM 327 que el escáner integra.
- **Material de la estructura:** Debe ser de construida de cualquier material excepto de chapa metálica para evitar cortocircuitos internos entre las conexiones realizadas.

- **Sistemas operativos:** El dispositivo trabaja en sistemas operativos Windows de versión 7 en adelante.

3.2 ANÁLISIS Y SELECCIÓN DE ALTERNATIVA

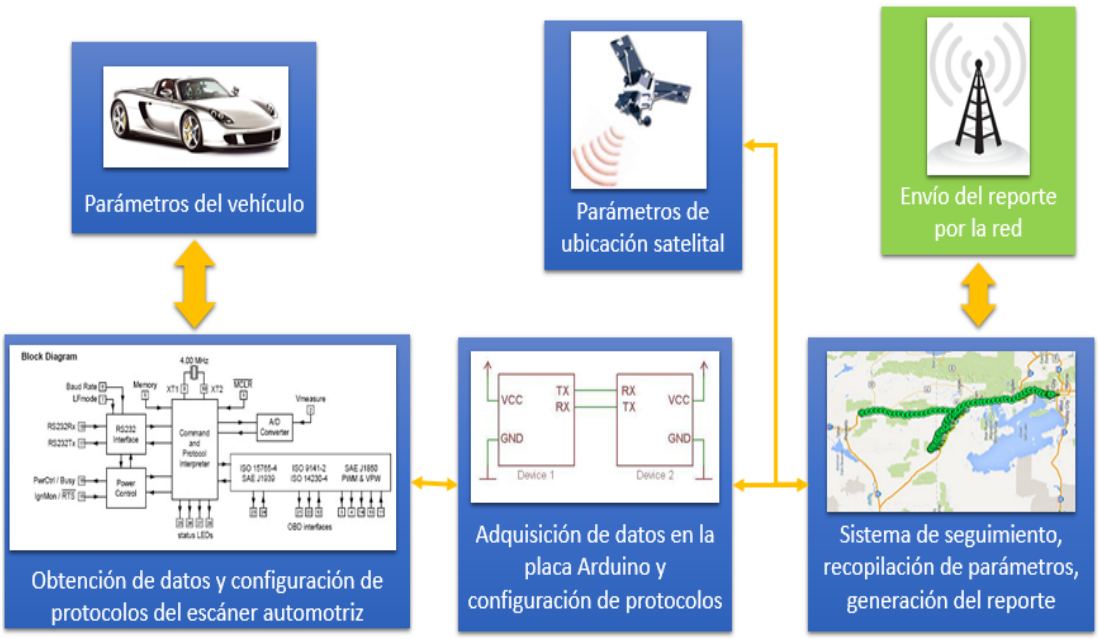


Figura 12 Proceso de funcionamiento del proyecto

A continuación se evaluaron tres alternativas para desarrollar el dispositivo que cumplen con la satisfacción de la necesidad tanto del desarrollador como del cliente final pero pudieron ser elaboradas con elementos de diferente material, costo, tamaño, medio de programación, soportes, entre otros parámetros de diseño, que fueron analizadas dentro de las alternativas hasta llegar a la seleccionada bajo el siguiente proceso de funcionamiento de la figura 12.

3.2.1 Primera Alternativa

Dentro de esta alternativa se estableció como medio de obtención de datos de telemetría el OBD I de manufactura china como muestra la figura 13, que

posee las ventajas de comunicación inalámbrica por medio de un dispositivo Bluetooth, lo que evita la conexión cableada por la parte inferior del vehículo, para recibir los datos del OBD I se debe instalar a la placa de desarrollo un dispositivo Bluetooth para enlazar la comunicación y poderla manipular.

Figura 13 OBD I con Bluetooth incorporado

Fuente: (ScanTool.net, 2015)

Para la manipulación de datos y configuración de los protocolos de comunicación y generación de la cadena de datos String se utilizó la placa de desarrollo Beagleboard con el modelo Beaglebone Black ver la figura 14, que se conectará al computador a una interfaz visual GUI desarrollada en el lenguaje de programación C++.

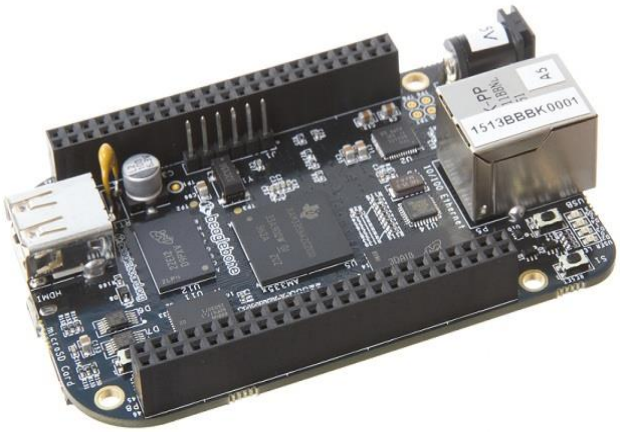


Figura 14 Beagleboard Beaglebone Black

Fuente: (Beagleboard.org, 2015)

La adquisición de datos de ubicación del vehículo se la obtuvo mediante la placa Adafruit GPS Logger Shield como se muestra en la figura 15, de compatibilidad con la placa Beaglebone Black, existe soporte para enlace de dispositivos en la red.

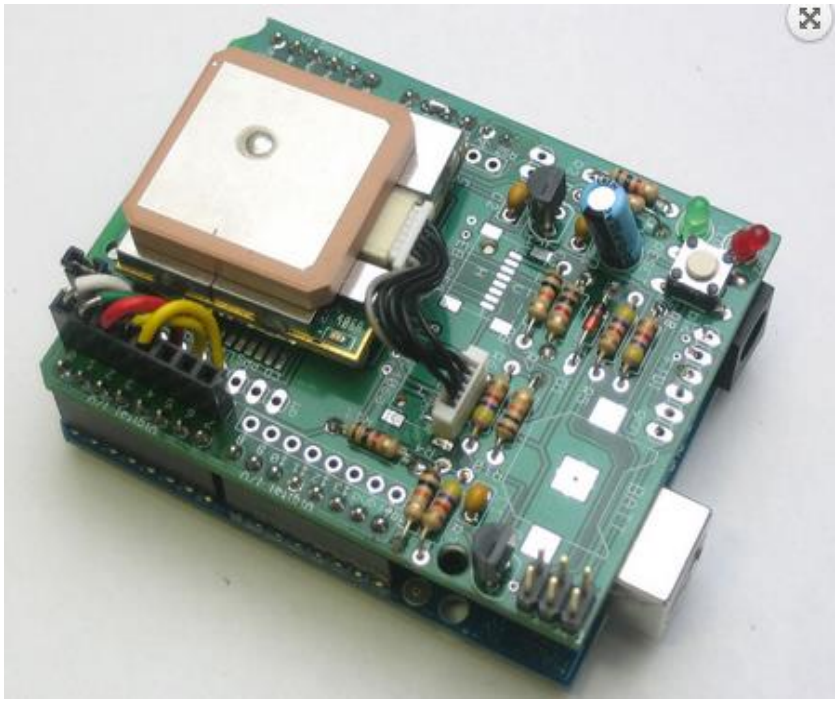


Figura 15 Adafruit GPS Logger Shield

Fuente: (openhacks.com, 2015)

3.2.2 Segunda alternativa

Para esta alternativa se consideró algunos cambios que difieren de la anterior haciendo enfoque en el soporte que se pueda encontrar en la red y una mayor comercialización de dispositivos en el mercado, como el escáner automotriz de manufactura australiana, es un OBD II de marca Freematics con más ventajas tanto en comunicación como en parámetros vehiculares que proporciona con respecto al dispositivo de la anterior alternativa pero es utilizado a través de cable de datos mediante puerto serie, el escáner automotriz utilizado en esta alternativa se lo puede apreciar en la siguiente figura 16.

Figura 16 OBD II de Freematics

Fuente: (Freematics, 2015)

El medio de comunicación para tratar los parámetros obtenidos se dio mediante una placa de desarrollo Arduino con su modelo MEGA 2560, la ventaja de esta placa es que integra cuatro puertos serie que pueden ser utilizados para la recepción de datos de algunos dispositivos al mismo tiempo, sin crear problemas de comunicación, el dispositivo se lo puede apreciar en la figura 17.

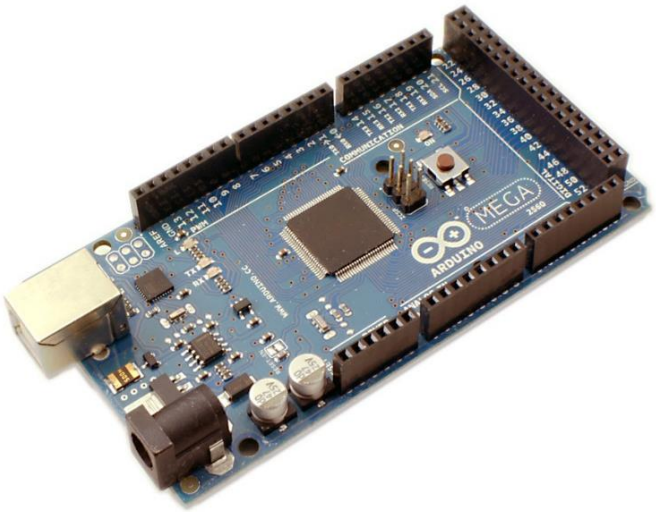


Figura 17 Arduino MEGA 2560

Fuente: (Arduino, 2015)

El lenguaje de programación empleado en esta alternativa es en WPF C# debido al gran soporte que posee en la red, en sitios oficiales y aficionados, además que su entorno de programación permite vincular paquetes de desarrollo que permiten al programador ganar algo de tiempo en la programación. El dispositivo de ubicación satelital empleado en esta alternativa es la antena GPS USB Globalsat BU-353-S4, de alta precisión y recomendable para el desarrollo de proyectos de enlace satelital desde cualquier tipo de vehículo tanto terrestres, aéreos y marítimos, se puede apreciar este dispositivo en la figura 18.

Figura 18 GPS USB Globalsat BU-353-S4

Fuente: (USGlobalSat, 2015)

3.2.3 Tercera alternativa

La última alternativa tiene similitud a la anterior porque usa el mismo dispositivo OBD I de la primera alternativa, ver la figura 13 y una placa de desarrollo Arduino ADK, que es similar en un 95% al modelo MEGA 2560, se analizó la posibilidad de realizar el procedimiento buscado con otros elementos que puedan realizar el mismo proceso de funcionamiento, la interfaz visual fue desarrollada en el lenguaje de programación C#.

Para la adquisición de los datos de posición satelital se utilizó la placa GPS Shield V1.0 GPS Module Breakout Board como se aprecia en la figura 19, esta

placa es de compatibilidad con Arduino, se enlazó directamente a la placa de desarrollo y no a la laptop donde se encuentra desarrollado el software y la interfaz visual.

Figura 19 GPS Shield V1.0 GPS Module Breakout Board

Fuente: (openhacks.com, 2015)

3.2.4 Análisis de los criterios de diseño mediante una casa de calidad

Para proceder al análisis de las alternativas anteriormente descritas se plantea una metodología de selección de alternativas muy útil y necesaria llamada casa de calidad.

Esta herramienta considera el criterio del cliente, de ingeniería y pondera cada uno de ellos. Las expectativas y necesidades de los clientes son recolectadas mediante técnicas de investigación de mercados: entrevistas, encuestas, exposiciones, entre otras. Los criterios del cliente que han sido tomados en cuenta en nuestra casa de calidad son: buen precio, larga vida útil, fácil manejo, fácil instalación, repuestos baratos; y los criterios de ingeniería empleados para la construcción de nuestra casa de calidad son: seguridad, confiabilidad, funcionalidad, tamaño, operatividad, costo, mantenimiento, control.

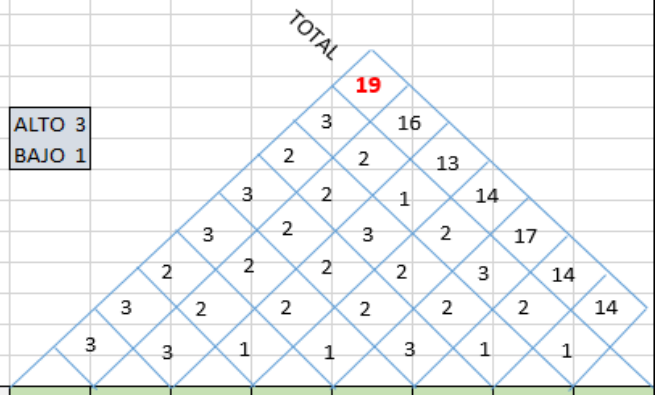
Los factores de ponderación para el análisis de factibilidad técnica se tomaron de la siguiente manera:

Tabla 3 Factores de ponderación

Parámetro	Valor
5	Excelente
4	Muy bueno
3	Bueno
2	Regular
1	Malo

Tabla 4 Análisis de los criterios de ingeniería y del cliente

CASA DE CALIDAD	CARACTERÍSTICAS DE INGENIERÍA



Atribuciones del Cliente	Importancia Relativa	Seguridad	Confiabilidad	Funcionalidad	Tamaño	Operatividad	Costo	Mantenimiento	Control
Buen precio	9	4	4	5	0	4	5	4	3
Fácil manejo	6	4	4	4	1	2	4	3	3
Fácil instalación	4	3	2	2	2	3	5	4	4
Larga vida útil	6	4	3	4	1	3	5	4	3
Repuestos baratos	5	0	3	3	0	3	5	3	1
TOTAL	30	15	16	18	4	15	24	18	14

De la elaboración de la casa de calidad se puede conocer que el enfoque de mayor importancia para el cliente es el precio de igual manera para el criterio de ingeniería hace un enfoque en el costo del proyecto. Al relacionar entre los criterios de ingeniería, el parámetro de mayor importancia es la seguridad, por lo que la selección de las alternativas se basará en esta selección de parámetros de calidad: costo, funcionabilidad, mantenimiento y seguridad.

3.2.5 Selección de viabilidad de las alternativas

A continuación se analizó las alternativas anteriormente planteadas con relación a los criterios del cliente para conocer cuál es la alternativa más viable para su selección.

Tabla 5 Selección de alternativas

Atribuciones del cliente	Alternativas		
	Primera Alternativa	Segunda Alternativa	Tercera Alternativa
Buen Precio	4	5	3
Fácil Manejo	5	5	4
Fácil Instalación	5	5	4
Larga vida útil	5	5	5
Repuestos baratos	3	4	2
TOTAL	22	**24**	18

3.2.6 Alternativa seleccionada

La segunda alternativa es la mayor puntuada y es la que se seleccionó, los elementos que contiene esta alternativa cumplen con los requisitos del cliente, son económicos, duraderos, comercializados. Además que los elementos de esta alternativa tienen basto soporte en la red, son compatibles entre sí y el desarrollo de la interfaz visual es clara y de fácil manejo para el usuario, por lo que complementan y validan la selección de la alternativa.

4 DISEÑO Y DESARROLLO

A continuación se indica los pasos del desarrollo del proyecto realizado de acuerdo a la metodología descrita y establecida en el capítulo anterior, explicados a detalle para conocer los procedimientos tomados para obtener el resultado buscado.

Algunos aspectos de factibilidad para el desarrollo del proyecto representativos que se tomaron en cuenta son:

- Los recursos de programación están al alcance de todos los aficionados a la programación gracias a que en el presente año (2015), Microsoft Developer Studio con su principal herramienta de desarrollo Microsoft Studio creó una versión gratuita del software, para que proyectos de ingeniería sean desarrollados bajo esta plataforma.
- Arduino permite al diseñador realizar proyectos de ingeniería económicos, su plataforma es de código abierto posee abundante información de soporte para el desarrollo de proyectos y los modelos de sus placas son comúnmente comercializados en el mercado nacional.
- No se adquirió licencia alguna para desarrollar de este proyecto.

4.1 ANALISIS MECÁNICO

Su objetivo primordial es proporcionar una o varias soluciones para desarrollar un elemento físico de forma que satisfaga los requisitos y restricciones establecidas por el diseñador y el cliente final.

Para la construcción del contenedor del dispositivo se tomó ciertas consideraciones de diseño para su elaboración que son: tener un espacio reducido, poseer un cableado mínimo, ser resistente a movimientos bruscos o a golpes y no generar incomodidad al conductor.

Se inicia la construcción de la estructura realizando el modelado CAD ideal de nuestro contenedor que posteriormente se lo imprimió en 3D, las medidas para este diseño se indican en la Tabla 2, con estos datos se realizó el diseño de la estructura base en SolidWorks como se puede observar en la figuras 20, 21, y 22.

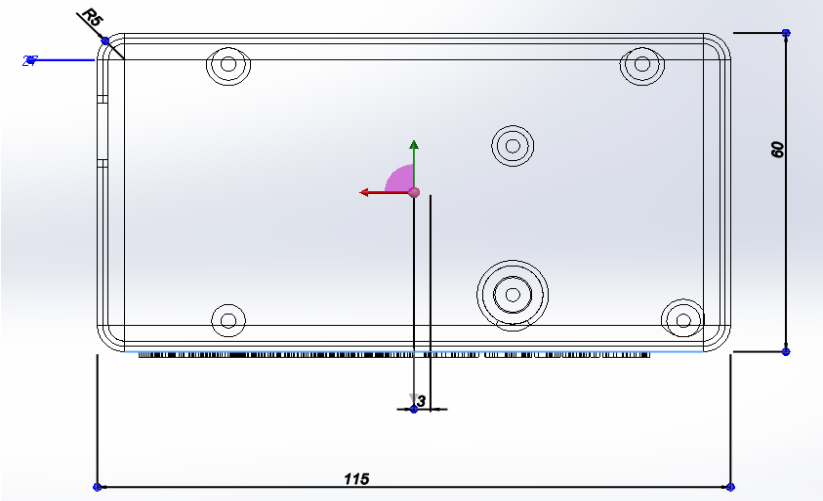


Figura 20 Vista superior de la estructura base diseñada

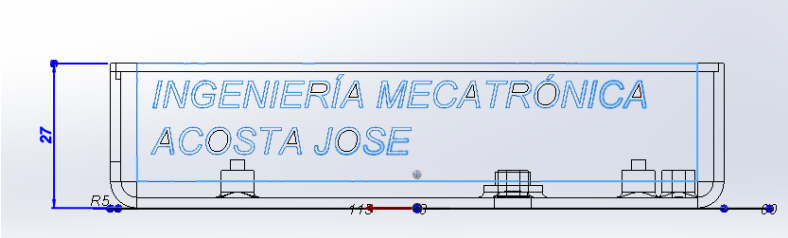


Figura 21 Vista frontal de la estructura base diseñada

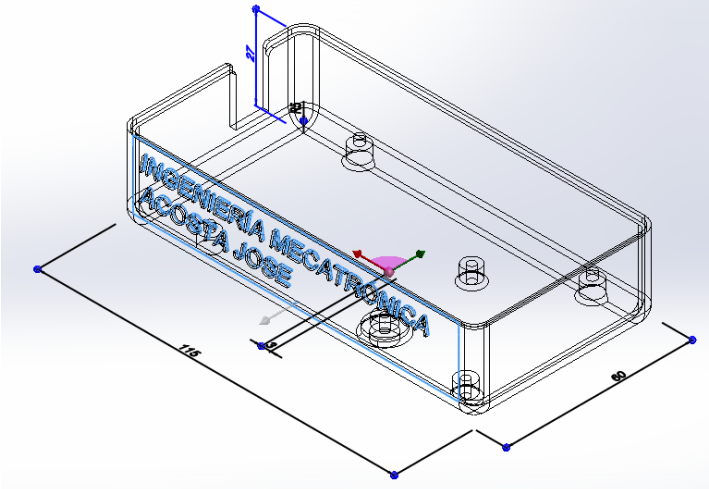


Figura 22 Vista isométrica de la estructura base diseñada

La tapa de la estructura base fue diseñada con dimensiones proporcionales a la estructura base como se muestra en las figuras 23, y 24.

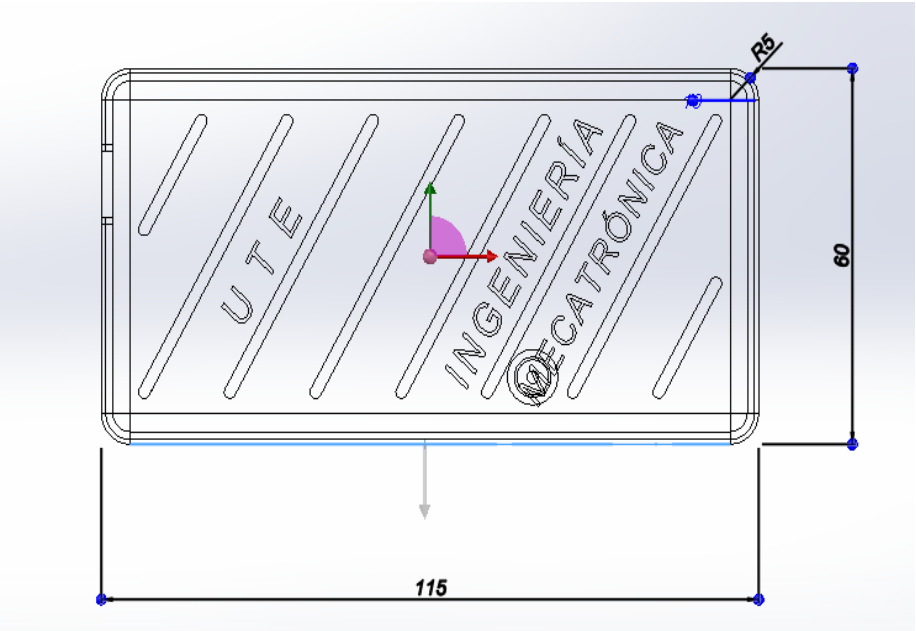


Figura 23 Vista superior de la tapa de la estructura base

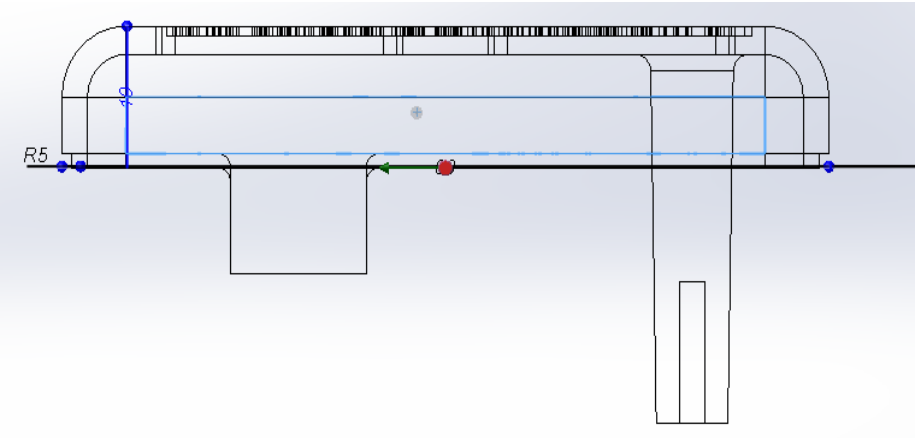


Figura 24 Vista lateral de la tapa de la estructura base

Cabe indicar que la estructura posee ranuras internas con la finalidad de que la placa de desarrollo permanezca fija al ser ajustadas por medio de tornillos de 2 mm diámetro, el resultado del diseño se lo puede apreciar en la figura 25.

Figura 25 Diseño isométrico del diseño CAD de la estructura

Para finalizar el diseño CAD realizado del modelo con las dimensiones establecidas se procedió a imprimirlo en la impresora 3D del Laboratorio de Mecanismos de la Universidad Tecnológica Equinoccial, el costo del material y del servicio de esta impresión fue de 20 dólares.

4.2 ANÁLISIS ELECTRÓNICO

Este análisis permite al diseñador poner en funcionamiento su ingenio y creatividad para resolver problemas de la ingeniería electrónica implementando nuevas estrategias de procesamiento de información mediante el uso de nuevos componentes y/o desarrollando proyectos a partir de circuitos integrados que realizan aplicaciones específicas o placas de desarrollo pre-programables.

Dentro de los elementos electrónicos utilizados en este proyecto se destaca la placa de desarrollo Arduino modelo MEGA 2650 y el escáner automotriz OBD II de la marca Freematics que se soldaron para constituir un solo elemento de adquisición de datos como se aprecia en la figura 26.

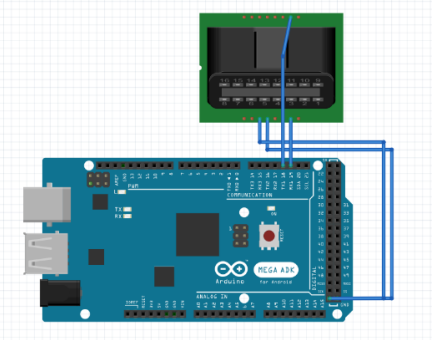


Figura 26 Conectividad entre el Arduino MEGA 2650 y el escáner OBD II

4.2.1 Configuración de los protocolos de comunicación OBD II

Es necesario contar con la correcta configuración de los protocolos de comunicación del dispositivo para obtener los parámetros automotrices, la programación de la librería OBD.h se la puede apreciar en el Anexo 5, donde se indica la configuración automática de reconocimiento de protocolos de comunicación del escáner OBD II.

El escáner automotriz cuenta con un elemento interno, el chip ELM327 el cual es el encargado de enlazar y entregar los datos de telemetría a cualquier dispositivo electrónico externo por medio de comunicación serie, el chip trabaja bajo 5v o 3,3v dependiendo de la necesidad del desarrollador del proyecto, en nuestro caso se utilizó 5v, ver la figura 27.

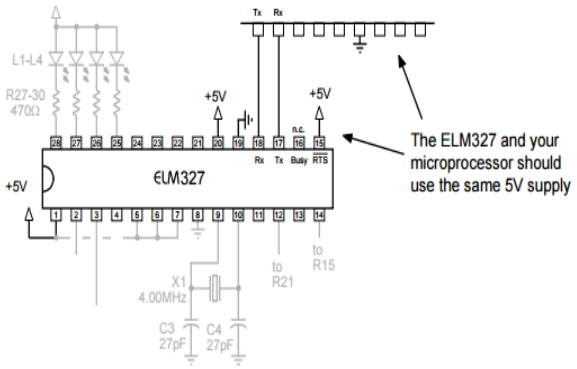


Figura 27 Conexión entre ELM327 y un dispositivo externo bajo 5v

Fuente: (elmelectronics.com, 2013)

Para la programación de la placa de desarrollo Arduino 2650, se utilizó la librería <OBD.h>, esta librería es desarrollada por la marca Freematics para comunicar el dispositivo OBD y tener acceso una gran cantidad de parámetros requeridos, estos parámetros son los siguientes:

Motor:

- PID_RPM
- PID_ENGINE_LOAD
- PID_COOLANT_TEMP
- PID_ENGINE_LOAD
- PID_ABSOLUTE_ENGINE_LOAD
- PID_TIMING_ADVANCE
- PID_ENGINE_OIL_TEMP
- PID_ENGINE_TORQUE_PERCENTAGE
- PID_ENGINE_REF_TORQUE

Presiones y flujos:

- PID_INTAKE_TEMP
- PID_INTAKE_PRESSURE
- PID_MAF_FLOW
- PID_BAROMETRIC

Velocidad/ Tiempo

- PID_SPEED
- PID_RUNTIME
- PID_DISTANCE

Conducción:

- PID_THROTTLE

- PID_AMBIENT_TEMP

Systema Eléctrico:

- PID_CONTROL_MODULE_VOLTAGE
- PID_HYBRID_BATTERY_PERCENTAGE

Además, se utilizó la librería <Wire.h> que permite vincular al escáner automotriz con la placa de desarrollo mediante comunicación UART (Transmisión-Recepción Asincrónica Universal o comúnmente conocida como puerto Serie) e I2C (Inter Circuitos Integrados) se puede apreciar en la figura 28.

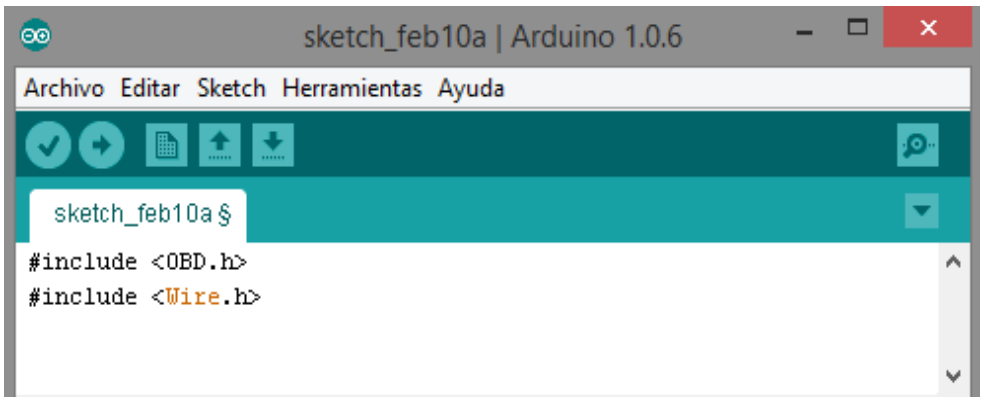


Figura 28 Uso de la librería Wire.h

4.2.2 Adquisición de parámetros de telemetría

Para obtener los datos necesarios del vehículo es primordial realizar una conexión adecuada entre los pines de uno de los puertos serie de la placa de desarrollo y los pines de comunicación del escáner automotriz, para esto se revisó el datasheet del dispositivo OBD II donde se identificó los pines de conectividad para la obtención de datos, ver la figura 29 y los terminales de conexión que permiten la comunicación entre el escáner y el vehículo como se indica en la figura 30.

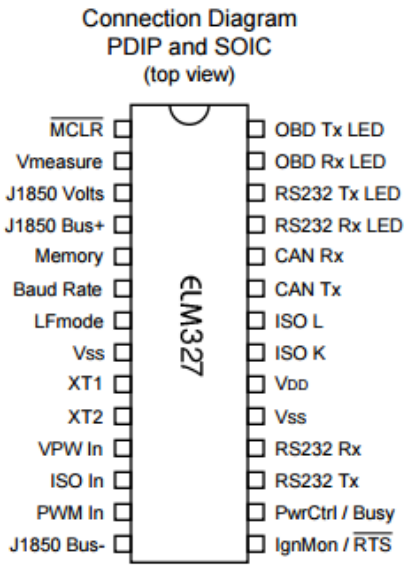


Figura 29 Diagrama de conexión del Chip ELM327

Los pines del terminal del puerto OBD II lleva la configuración mostrada en la figura 30 que puede depender de ciertas marcas, modelos, y lugar de comercialización o manufactura de los vehículos. Por ejemplo existen algunas marcas que desarrollan su propio escáner de parámetros predefinido para que solo ese vehículo pueda usarlo y no escáner ajenos a la marca, además puede existir ciertos problemas de compatibilidad o problemas físicos debido al tipo de ranura del puerto OBD que se maneja, lo que evita el poder conectarlos. Una breve descripción de éste puerto se indica en la figura 30 junto a su representación real.

Figura 30 Pines del terminal de conexión del OBD II

Fuente: (http://e-auto.com.mx/)

El otro extremo el escáner consta de cuatro pines para enlazarlo con la placa Arduino MEGA 2650, con la correcta conexión de estos pines el diseñador tendrá al alcance la información que necesita sobre el vehículo que se ha enlazado, ver la figura 31. Los cables son los siguientes:

- Amarillo.- Rx (Hacia el Pin Tx de la placa)
- Azul o Blanco.- Tx (Hacia el Pin Rx de la placa)
- Rojo.- Vcc (Hacia el Pin Vcc 5v de la placa)
- Negro Gnd (Hacia el Pin Gnd de la placa)

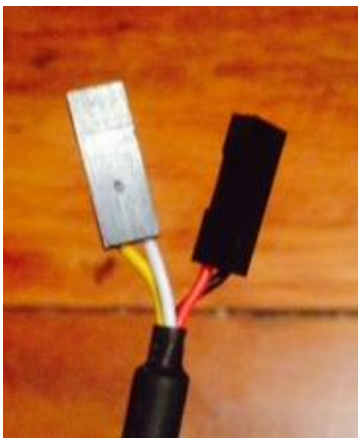


Figura 31 Pines escáner OBD II

Debido a que los datos entregados por el escáner son demasiados, se ha escogido de acuerdo a su relevancia, diez de los más importantes parámetros según la necesidad del cliente:

- Velocidad (PID_SPEED)
- RPM del motor (PID_RPM)
- Posición de aceleración (PID_THROTTLE)
- Carga absoluta del motor(PID_ABS_ENGINE_LOAD)
- Temperatura en tanque (PID_INTAKE_TEMP)
- Presión del Combustible (PID_FUEL_PRESSURE)

- Nivel de combustible (PID_FUEL_LEVEL)
- Presión barométrica (PID_BAROMETRIC)
- Tiempo de carrera del motor (PID_RUNTIME)
- Distancia (PID_DISTANCE)

4.2.3 Programación de la placa Arduino MEGA 2650

El desarrollo de la programación realizado para la adquisición de los datos de telemetría se rige bajo la programación indicada en el flujograma representado en la figura 32:

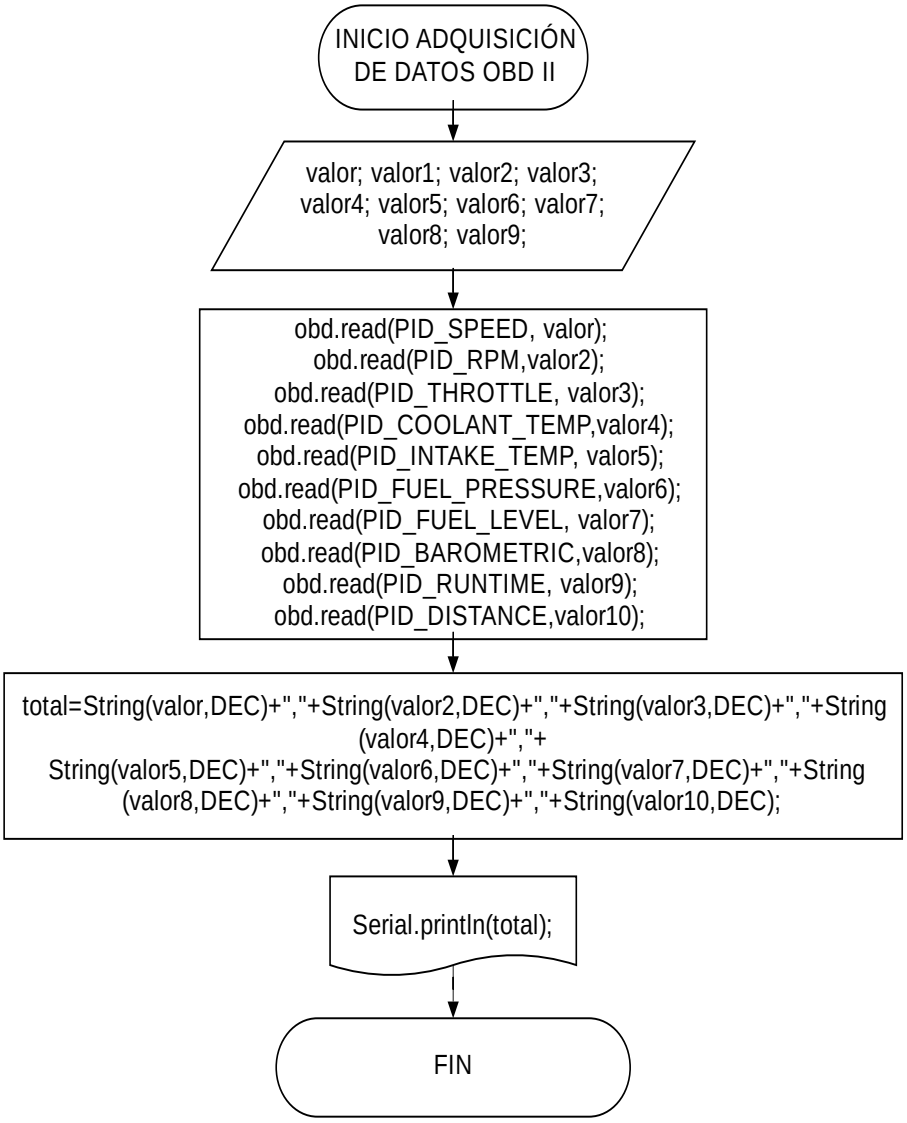


Figura 32 Diagrama de flujo de la adquisición de datos

4.3 ANÁLISIS DEL DISEÑO Y DESARROLLO DEL SOFTWARE

El diseño y el desarrollo del software son importantes para la resolución de problemas o satisfacer necesidades de clientes independientemente del software de programación creado y el lenguaje en el cual se lo ha desarrollado.

Se parte por analizar e identificar los requisitos para posteriormente establecer una plataforma para la construcción (desarrollo de código y pruebas de funcionamiento); a la par del software, el desarrollo del hardware debe ajustarse ampliamente a las necesidades del programa realizado.

4.3.1 Aplicación en Windows Presentation Fundation C#

Para poder desarrollar la interfaz visual sin ningún tipo de inconveniente es necesario tener conocimientos sobre programación y uso de la sintaxis XAML para desarrollar el entorno gráfico del sistema, además tener el software adecuado y las librerías correspondientes para lo que se pretenda programar. Para que no sea tan compleja la programación de todas las aplicaciones del proyecto al mismo tiempo, se lo ha realizado por separado, esto con la finalidad que resulte de mayor comodidad para el programador vincularlas cuando todas estén funcionando correctamente. Se empezó desde lo más elemental que es el desarrollo del entorno visual, hasta la construcción de una estructura adecuada para la placa de desarrollo.

4.3.2 Diseño de la interfaz visual

Se debe realizar un entorno visual claro y simple para el usuario final y de fácil operatividad, la interfaz visual tiene dos pestañas de contenido que

representen las utilidades de este proyecto. La primera pestaña se la nombró como “SISTEMA DE SEGUIMIENTO” que contiene el sistema de posicionamiento en tiempo real del vehículo cada cierto tiempo, consta de un botón llamado “Conectar” que da inicio al sistema completo, el mismo botón se convierte en otro llamado “Desconectar”, el cual da por finalizado el sistema, tanto el seguimiento como la adquisición de datos de posición y de telemetría. El botón de “Finalizar” crea el archivo recopilado de datos de seguimiento y envía automáticamente el reporte por correo electrónico con el archivo creado como adjunto, ver la figura 33.

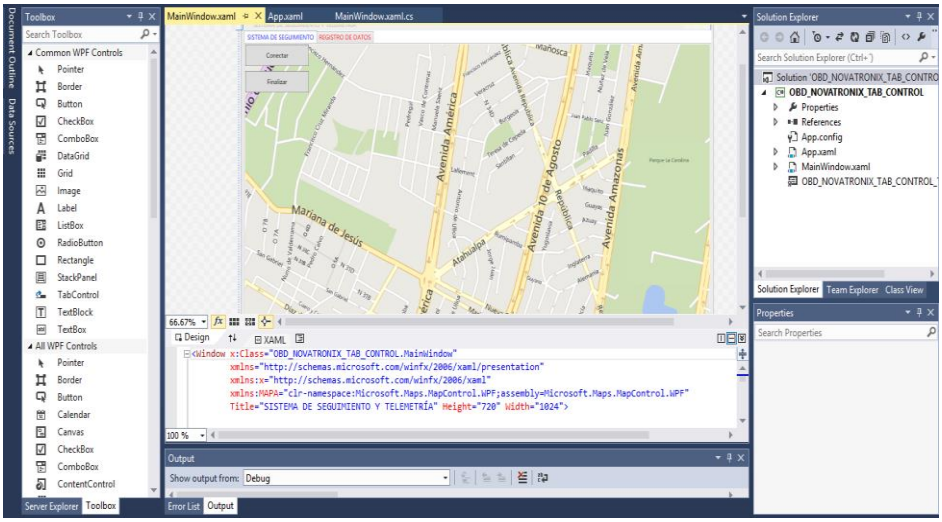


Figura 33 Representación de la interfaz visual el sistema de seguimiento

Programación XAML de la primera pestaña del sistema de seguimiento de la interfaz visual:

```
<TabItem>
        <TabItem.Header>
                <StackPanel Orientation="Horizontal">
                        <TextBlock Text="SISTEMA DE SEGUIMIENTO" Foreground="Blue" />
                </StackPanel>
        </TabItem.Header>
        <MAPA:Map x:Name="myMap" CredentialsProvider="AnrEB8TwfV9IjlUv9PLRvLhzmevm9NhgQO-
        F3PPYFCqg9567Dv0zEXDDkqhnv0vw" Center="-0.184459,-78.493334" ZoomLevel="16" Grid.Row="1">
                <Button Content="Conectar" Name="Connect_btn" HorizontalAlignment="Left" Height="39"
                VerticalAlignment="Top" Width="138" Click="Connect_btn_Click"/>
                <RichTextBox Name="Commdata" Margin="0,615,0,0" VerticalScrollBarVisibility="Auto"
                Height="66" VerticalAlignment="Top"/>
                <Button Content="Finalizar" Name="Finalizar_btn" HorizontalAlignment="Left" Height="39"
                VerticalAlignment="Top" Width="138" Margin="0,48,0,0" Click="Finalizar_btn_Click"/>
```

```
        </MAPA:Map>
</TabItem>
```

La siguiente pestaña se llama REGISTRO DE DATOS que corresponde a la recopilación los parámetros en tiempo real de datos obtenidos por el escáner OBD y de posición satelital proporcionados por el GPS dentro de una tabla personalizada (DataGrid) que adquiere datos cada tres segundos, los datos aquí recopilados son almacenados en un archivo en el disco duro o en cualquier unidad extraíble y posteriormente enviados mediante correo electrónico al administrador como se muestra en la figura 34.

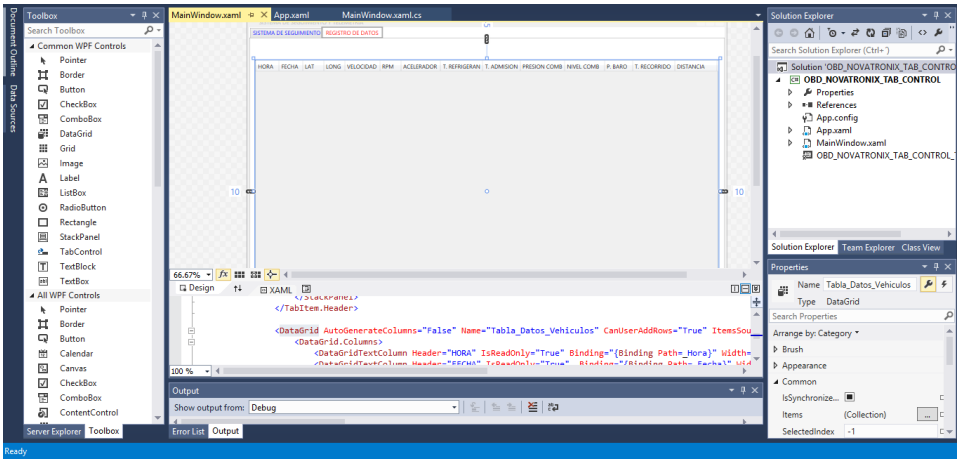


Figura 34 Representación de la interfaz visual de la recepción y envío de datos recopilados

Programación de la segunda pestaña de registro de datos de la interfaz visual:

```
<TabItem>
        <TabItem.Header>
                <StackPanel Orientation="Horizontal">
                        <TextBlock Text="REGISTRO DE DATOS" Foreground="Red" />
                </StackPanel>
        </TabItem.Header>

        <DataGrid AutoGenerateColumns="False" Name="Tabla_Datos_Vehiculos" CanUserAddRows="True"
        ItemsSource="{Binding TestBinding}" Margin="10,45,10,55" >
        <DataGrid.Columns>
        <DataGridTextColumn Header="HORA" IsReadOnly="True" Binding="{Binding Path=_Hora}"
        Width="50"></DataGridTextColumn>
        <DataGridTextColumn Header="FECHA" IsReadOnly="True"  Binding="{Binding Path=_Fecha}"
        Width="50"></DataGridTextColumn>
        <DataGridTextColumn Header="LAT" IsReadOnly="True"  Binding="{Binding Path=_Lat}"
        Width="45"></DataGridTextColumn>
```

```
            <DataGridTextColumn Header="LONG" IsReadOnly="True"  Binding="{Binding Path=_Long}"
            Width="45"></DataGridTextColumn>
            <DataGridTextColumn Header="VELOCIDAD" IsReadOnly="True"  Binding="{Binding Path=_Velocidad}"
            Width="75"></DataGridTextColumn>
            <DataGridTextColumn Header="RPM" IsReadOnly="True"  Binding="{Binding Path=_RPM}"
            Width="45"></DataGridTextColumn>
            <DataGridTextColumn Header="ACELERADOR" IsReadOnly="True"  Binding="{Binding Path=_Acelerador}"
            Width="85"></DataGridTextColumn>
            <DataGridTextColumn Header="T. REFRIGERANTE" IsReadOnly="True"  Binding="{Binding Path=_Trefri}"
            Width="90"></DataGridTextColumn>
            <DataGridTextColumn Header="T. ADMISION" IsReadOnly="True"  Binding="{Binding Path=_Tadmi}"
            Width="80"></DataGridTextColumn>
            <DataGridTextColumn Header="PRESION COMB" IsReadOnly="True"  Binding="{Binding Path=_Pcomb}"
            Width="95"></DataGridTextColumn>
            <DataGridTextColumn Header="NIVEL COMB" IsReadOnly="True"  Binding="{Binding Path=_Ncomb}"
            Width="85"></DataGridTextColumn>
            <DataGridTextColumn Header="P. BARO" IsReadOnly="True"  Binding="{Binding Path=_Pbaro}"
            Width="60"></DataGridTextColumn>
            <DataGridTextColumn Header="T. RECORRIDO" IsReadOnly="True"  Binding="{Binding Path=_Treco}"
            Width="90"></DataGridTextColumn>
            <DataGridTextColumn Header="DISTANCIA" IsReadOnly="True"  Binding="{Binding Path=_Distancia}"
            Width="90"></DataGridTextColumn>
            </DataGrid.Columns>
            </DataGrid>
</TabItem>
```

Se recomienda ver el Anexo 3 que contiene la programación XAML completa de la interfaz visual.

4.3.3 Comunicación del GPS y del OBD a la interfaz visual

Para este procedimiento se debe conectar la antena GPS y la placa de desarrollo a cualquier puerto de comunicación del equipo que se encuentre libre, en el entorno de programación se utilizó la librería:

- “using System.IO.Ports;”.

A continuación se procedió a revisar el puerto en el cual el GPS y la placa de desarrollo se encuentran conectados y se los vinculó a la interfaz gráfica, se configuraron los siguientes parámetros de comunicación tanto para el GPS como para la placa de desarrollo: radio de comunicación de bits, bits de

paridad, handshake, entre otros parámetros, se lo puede apreciar en el flujograma de la figura 35.

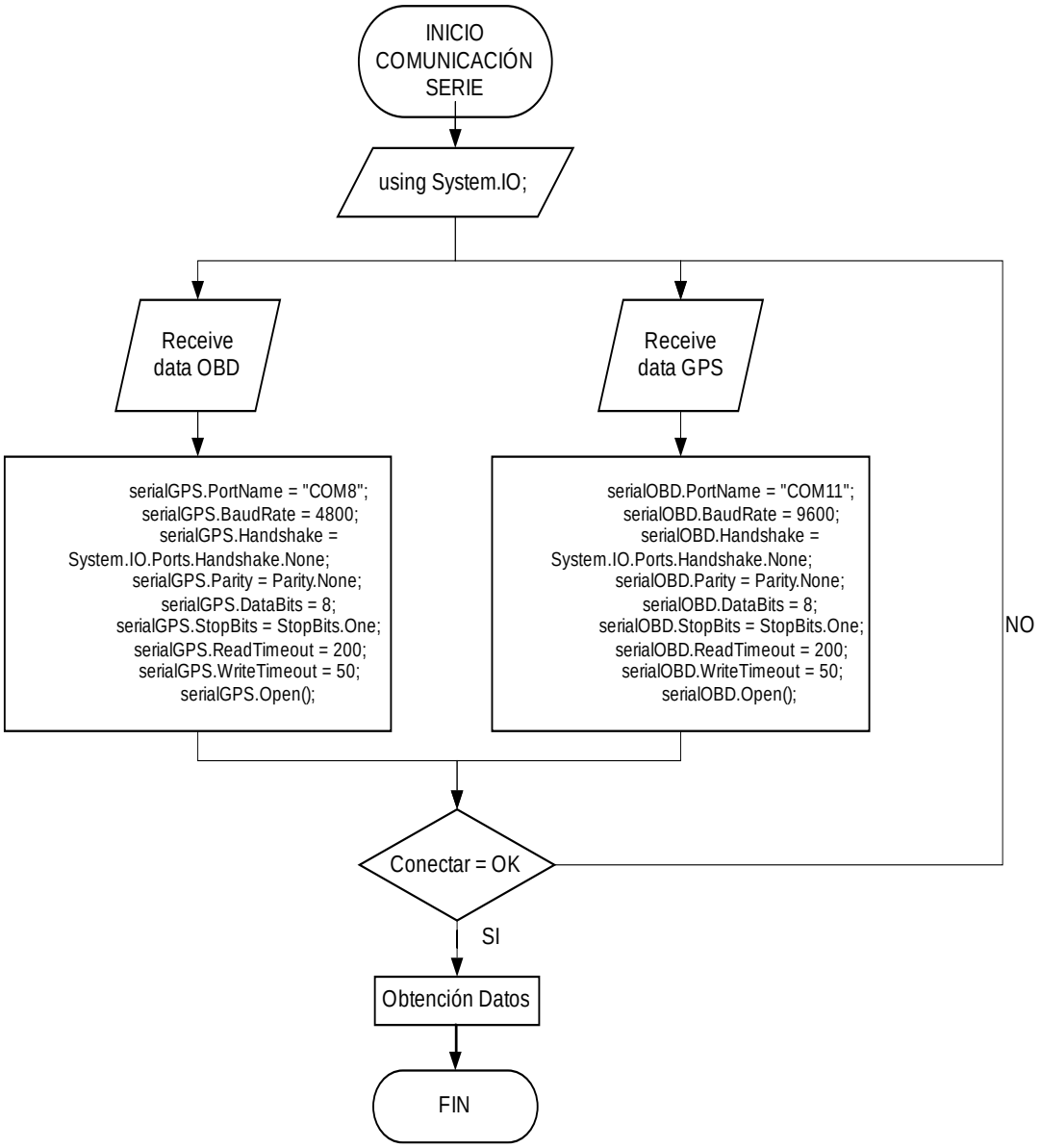


Figura 35 Flujograma de configuración de comunicación con los dispositivos

4.3.4 Adquisición de datos obtenidos por el GPS

Al utilizar el GPS, los datos arrojados tienen un formato inapropiado para almacenarlos en una tabla de datos por lo que se pretende manipular el formato con la finalidad de que los datos obtenidos sean más apreciables para el usuario quien supervisa el monitoreo de los parámetros de ubicación.

El GPS proporciona algunos estándares de comunicación mediante cadenas de datos, el más importante de ellos es el protocolo “$GPRMC”, que proporciona datos de ubicación satelital además de otros parámetros secundarios que no son de importante necesidad. A continuación se muestra una cadena de datos GPRMC proporcionado por el GPS donde se explica cada parámetro detalladamente:

$GPRMC,055956.00,A,2029.49050,N,10316.32055,W,000.00,000.0,111113,08.0,E,D*27

Cada parámetro significa:

- $GPRMC.- Esta variable indica el formato de la trama, en este caso es GPRMC.
- 055956.00.- Hora UTC en la que se obtuvo la posición del GPS (hhmmss).
- A.- Estatus de la trama A significa que es una trama válida, 9 que es una trama de caché y V significa que es una trama inválida.
- 2029.49050.- Latitud.
- N.- Hemisferio (N: Norte, S: Sur).
- 10316.32055.- Longitud.
- W.- Hemisferio (E: Este, W: Oeste).
- 000.00.- Velocidad en nudos.
- 000.0.- Rumbo en grados (0° a 360°).
- 111113.- Fecha UTC en la que se obtuvo la posición del GPS (DDMMAA).
- 08.0, E.- Variación magnética.
- D*27.- Checksum.

A continuación se muestra en la figura 36 el flujograma la programación realizada para obtener la cadena de datos GPRMC y su manipulación hasta obtener los parámetros de latitudinales y longitudinales.

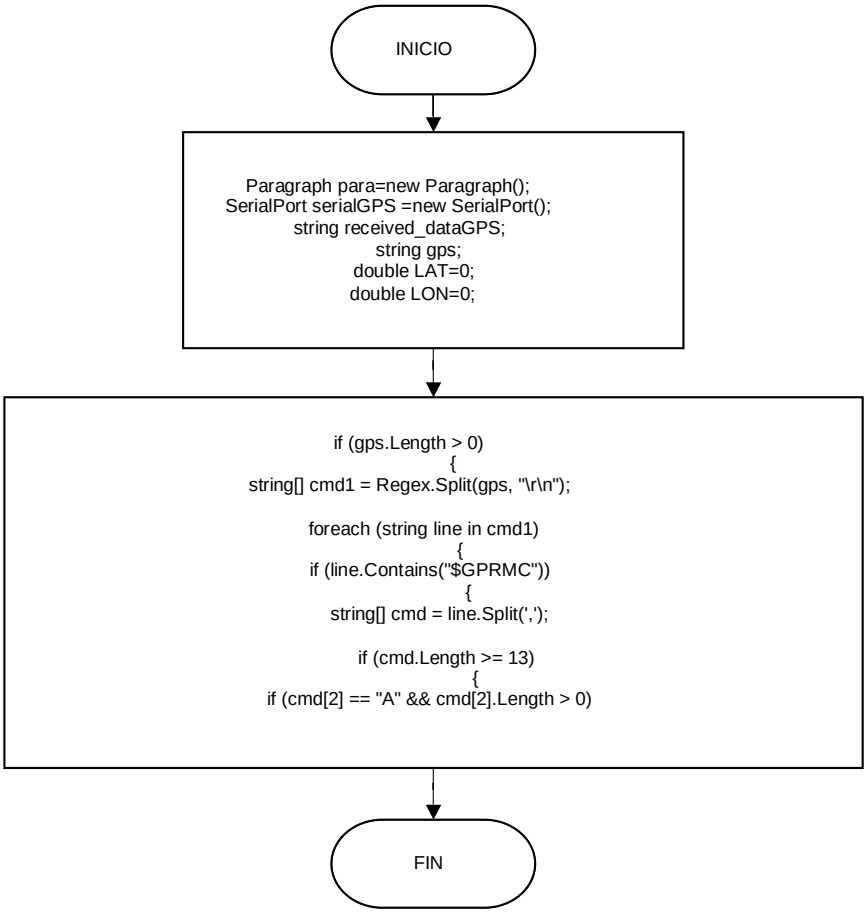


Figura 36 Adquisición de parámetros y flujo grama de comunicación GPS

4.3.5 Corrección de formato de coordenadas geográficas

Los parámetros proporcionados por el GPS de latitud y longitud son de mucha importancia para realizar el seguimiento en el mapa por lo que deben estar preparados para que las coordenadas se adapten sin inconveniente al sistema de seguimiento de Bing Maps. Se manipularon los datos obtenidos, adaptándolos a datos double de cuatro decimales, para luego transformarlos a datos sexagesimales, además las coordenadas proporcionadas por el GPS especifican puntos cardinales por lo cual este dato ha sido reemplazado por el signo negativo complementando la orientación cardinal junto al dato preparado, de esta manera el seguimiento se da con normalidad y correctamente posicionado al lugar donde el vehículo se encuentre, se puede

revisar el flujograma del acondicionamiento de las señales de latitud y longitud en las figuras a continuación:

Para la latitud se tiene el flujograma de la figura 37:

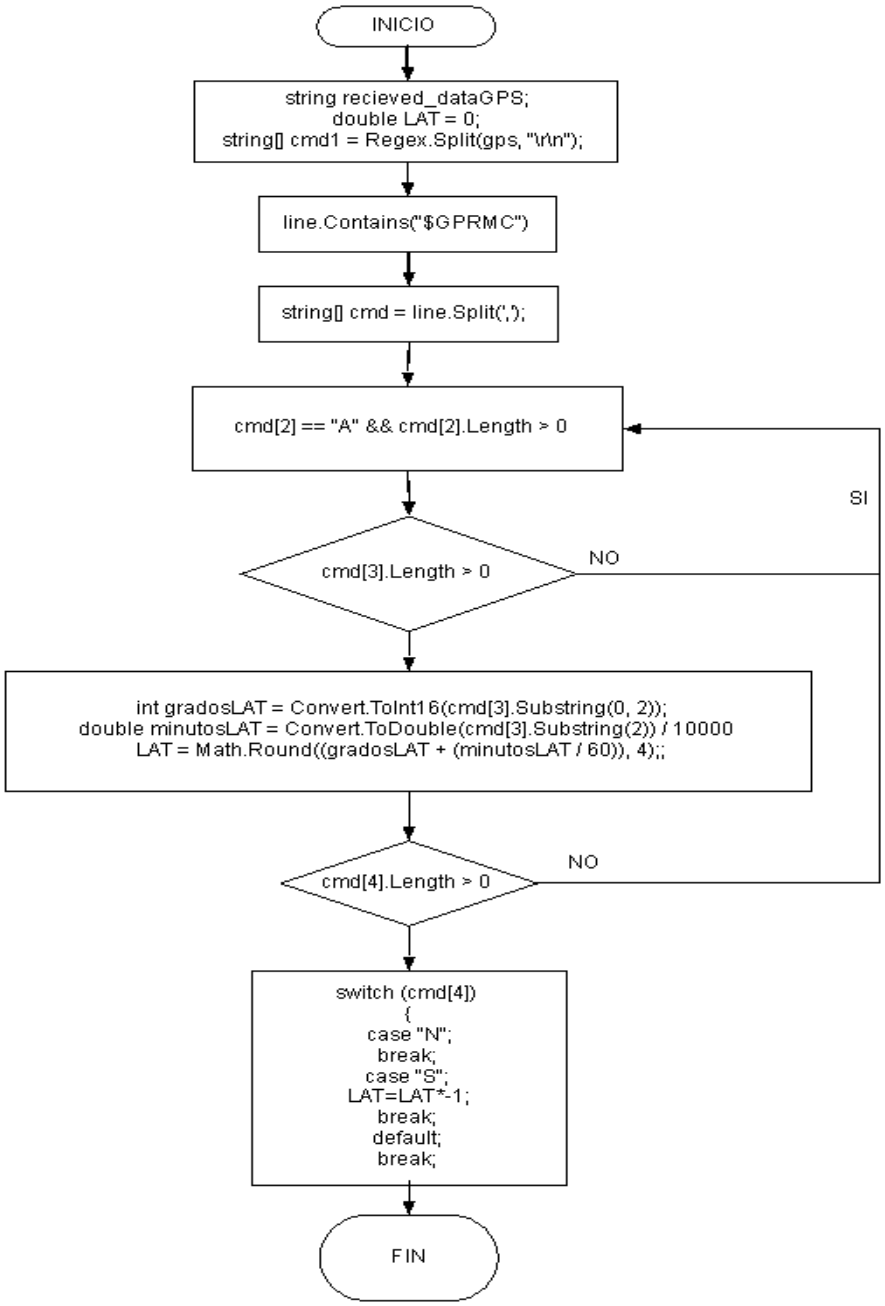


Figura 37 Flujograma completo de obtención de parámetros de latitud

Para la longitud se tiene el flujograma de la figura 38:

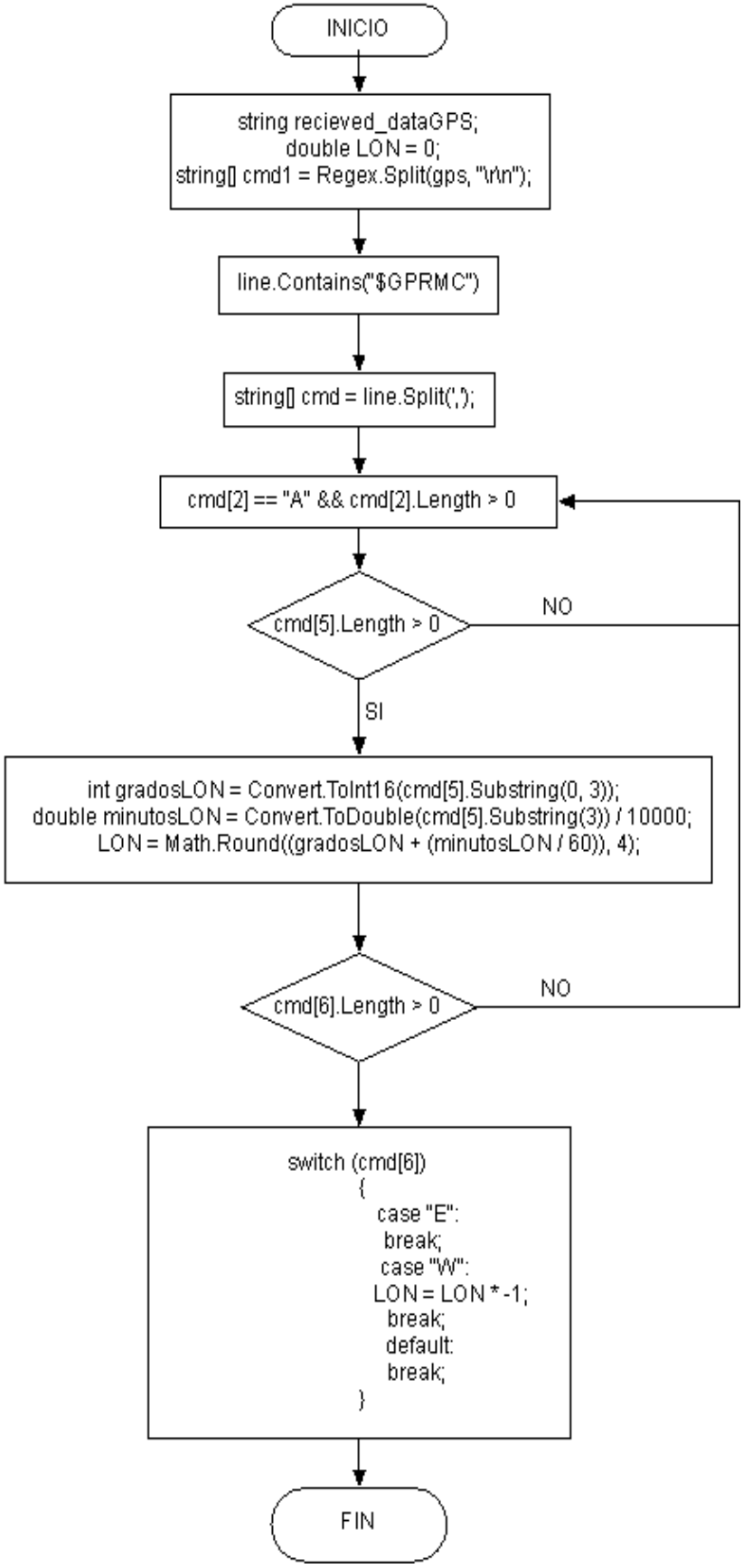


Figura 38 Flujograma completo de obtención de parámetros de longitud

4.3.6 Desarrollo del software de seguimiento satelital

Para realizar el sistema de seguimiento satelital se ha requerido del paquete de desarrollo de Bing Maps, este SDK contiene los archivos y librerías de desarrollo necesarias para vincular el sistema de seguimiento al sistema, se lo puede obtener en la red gratuitamente en el siguiente enlace:

https://www.microsoft.com/en-us/download/details.aspx?displaylang=en&id=27165

Se debe instalar el paquete de desarrollo y en el entorno de programación de C# y fue imprescindible incluir las siguientes librerías:

- using System.Windows.Threading;
- using Microsoft.Maps.MapControl.WPF;
- using Microsoft.Maps.MapControl.WPF.Design;

Estas librerías permiten al programador usar el mapa mundial proporcionado por Bing para ubicar o localizar direcciones, conocer lugares, posicionar datos de ubicación, entre otras opciones de localización y ubicación. Con este paquete de desarrollo se pretende vincular los datos obtenidos y manipulados del GPS para realizar el sistema de seguimiento.

De la adquisición de parámetros del GPS se manipularon los datos de latitud y longitud y se creó dos variables String llamadas LAT y LON que se encuentran óptimamente adecuados para ser vinculados al sistema de seguimiento. Existen otros parámetros que se deben configurar para la visualización del mapa, como es el ajuste de altura, punto de partida del tracking, modo mapa o modo real, zoom, tiempo de muestreo de trayectoria, y otros más de pequeña importancia. A continuación se indica el flujograma con el procedimiento marcado de puntos de referencia en el mapa, ver la figura 39:

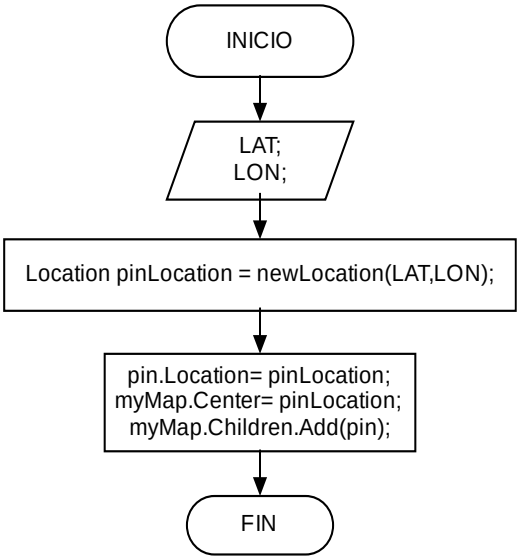


Figura 39 Flujograma sobre enlace de coordenadas GPS y Microsoft BingMaps

En la figura 40, se puede observar el sistema de seguimiento realizando un ejemplo de marcado de ubicación cada tres segundos.

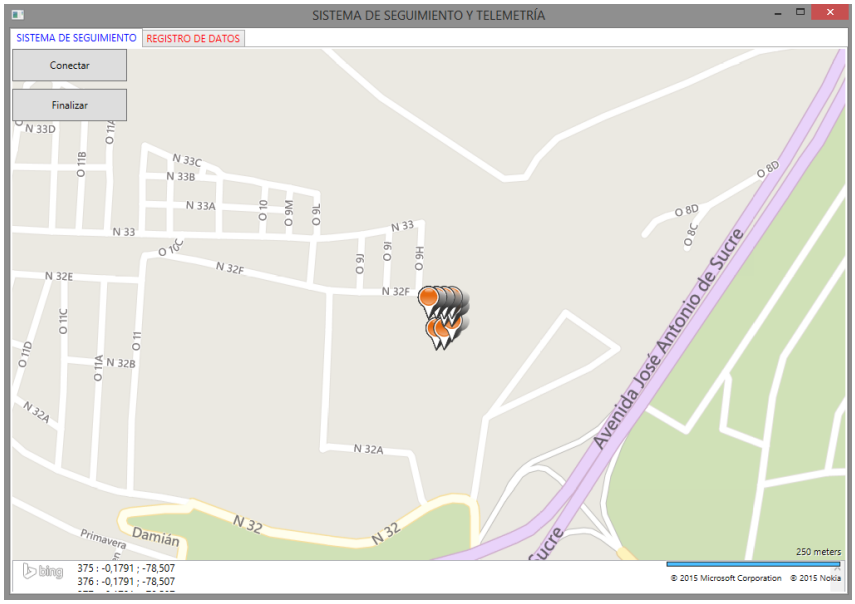


Figura 40 Sistema de seguimiento

A continuación se indica un flujograma de configuración del Dispatcher Timer o el marcado de los parámetros de ubicación tomando a un umbral de tres segundos, ver la figura 41:

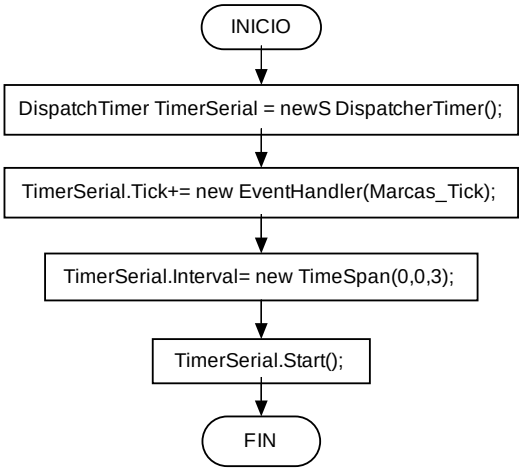


Figura 41 Flujograma del marcado de la ruta en un tiempo de tres segundos

4.4 GENERACIÓN DEL ARCHIVO DE REPORTE

Un archivo de extensión *.csv posee valores separados por coma que automáticamente se almacenan o se ubican en celdas ordenadas, estos archivos son de gran utilidad porque son de extensión multi formato, se pueden adaptar a una plantilla de Excel, a un bloc de notas o almacenarlos en una base de datos de cualquier programa si se desea, ver a continuación el flujograma de la figura 42:

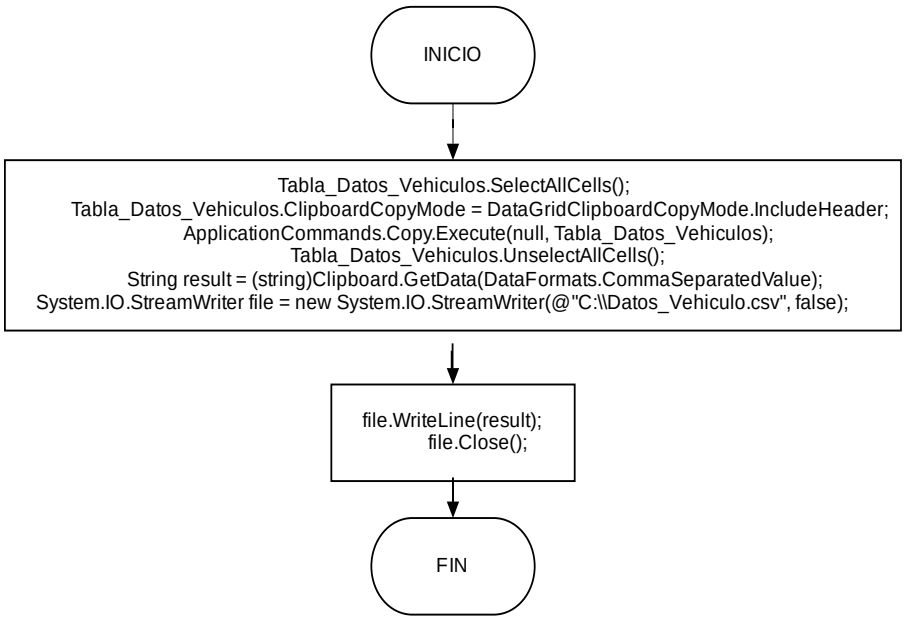


Figura 42 Flujograma correspondiente a la creación del archivo .csv

4.4.1 Vinculación de la interfaz visual con el correo electrónico y envío de dato adjunto

La conectividad a la red es muy importante para el proyecto porque es el motivo principal de la obtención de los parámetros indicados anteriormente, un administrador los puede conocer y tomar medidas inmediatas o planificar actividades como mantenimientos, reemplazo de partes, abastecimiento de servicios, monitoreo de la trayectoria tomada por el usuario, entre otros procedimientos.

Para el desarrollo de la conectividad con la red se utilizó las librerías que permiten al proyecto vincularse a la red, estas son:

- using System.Net.Mail;
- using System.Net;

Se procedió a crear y configurar las variables correspondientes a la configuración de correo electrónico. Para completar el servicio de correo como la variable nM que se refiere a nuevo mensaje y la variable sC hace referencia al cliente de correo SMTP (Protocolo de Transferencia de Correo Simple), además de configurar correctamente las propiedades de envío de correo como, emisor, receptor, receptor copia, título, asunto, cuerpo del correo, dato adjunto, y las credenciales del usuario, el cliente SMTP en este caso se ha tomado a gmail (smtp.gmail.com) y su puerto de conectividad es el 587, se puede trabajar con cualquier servidor de correo electrónico, simplemente agregando a la configuración el dominio bajo el que el proyecto trabaja, configurando el puerto de acceso y creando un correo personalizado para el vehículo, se puede apreciar en el flujograma de la figura 43 la configuración de los parámetros del envío de correo electrónico, cabe indicar que para realizar pruebas se ha creado un correo llamado pruebas OBD en el servicio de gmail.

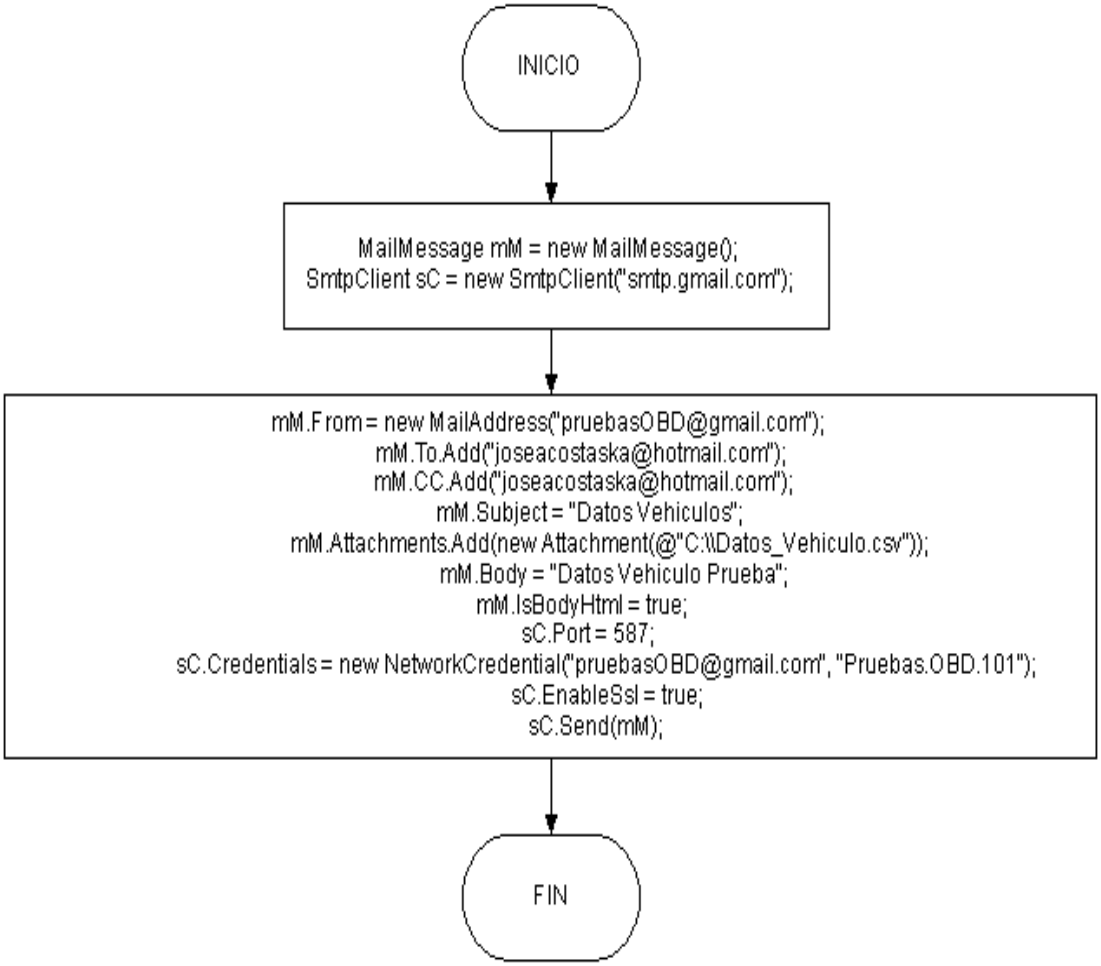


Figura 43 Flujograma de envío de correo electrónico con dato adjunto

4.5 ANÁLISIS DEL DESARROLLO DEL PROTOTIPO VIRTUAL Y PRUEBAS DE FUNCIONAMIENTO

Al haber desarrollado el software parte por parte se procedió a la vinculación de todos los códigos programación, corrigiendo posibles errores y reduciendo ciertos aspectos redundantes en la programación del software dando como resultado el prototipo virtual del sistema de seguimiento y adquisición de datos de ubicación y telemetría que para motivos de funcionamiento se ha programado una simulación de cadena de Strings para validar el almacenamiento como se puede observar en la figura 44. Se recomienda revisar el Anexo 5 que contiene la programación completa en el lenguaje C# sobre el desarrollo del software del prototipo virtual.

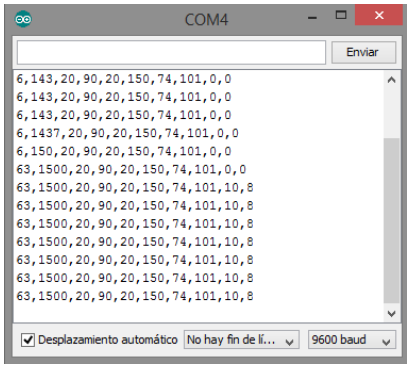


Figura 44 Generación de una cadena de datos String para pruebas del prototipo virtual

Se comprobó que el funcionamiento de nuestro prototipo virtual obtiene los datos generados desde el puerto serie del IDE de Arduino, los parámetros con los que cuenta el prototipo virtual son: hora, fecha, latitud, longitud, velocidad, rpm, acelerador, temperatura del refrigerante, tiempo de admisión, presión del combustible, nivel del combustible, presión barométrica, tiempo de recorrido y la distancia.

HORA	FECHA	LAT	LONG	VELOCIDAD	RPM	ACELERADOR	T. REFRIGERANTE	T. ADMISION	PRESION COMB	NIVEL COMB	P. BARO	T. RECORRIDO	DISTANCIA
12:52:39	04/30/1	-0,1231	-78,502	63	1500	20	90	20	150	74	101	10	8
12:52:42	04/30/1	-0,1231	-78,502	63	1500	20	90	20	150	74	101	10	8
12:52:46	04/30/1	-0,1231	-78,502	63	1500	20	90	20	150	74	101	10	8
12:52:48	04/30/1	-0,1231	-78,502	63	1500	20	90	20	150	74	101	10	8
12:52:51	04/30/1	-0,1231	-78,502	63	1500	20	90	20	150	74	101	10	8
12:52:55	04/30/1	-0,1231	-78,502	63	1500	20	90	20	150	74	101	10	8
12:52:58	04/30/1	-0,1231	-78,502	63	1500	20	90	20	150	74	101	10	8
12:53:01	04/30/1	-0,1231	-78,502	63	1500	20	90	20	150	74	101	10	8
12:53:04	04/30/1	-0,1231	-78,502	63	1500	20	90	20	150	74	101	10	8
12:53:07	04/30/1	-0,1231	-78,502	63	1500	20	90	20	150	74	101	10	8
12:53:10	04/30/1	-0,1231	-78,502	63	1500	20	90	20	150	74	101	10	8
12:53:13	04/30/1	-0,1231	-78,502	63	1500	20	90	20	150	74	101	10	8

Figura 45 Prueba de adquisición de datos de la cadena de Strings generadas

Para la comprobación del sistema de seguimiento se obtuvo los datos puntuales vinculados al sistema de seguimiento satelital de la cadena de Strings anteriormente generada, de esta forma se conoció que el sistema de seguimiento marca el lugar puntual correcto de ubicación satelital en el mapa como se ve en la figura 46.

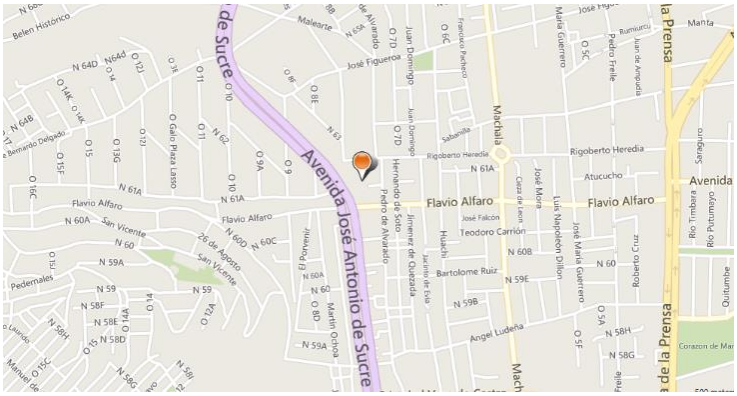


Figura 46 Ubicación satelital de las coordenadas obtenidas de la cadena de Strings

4.6 ANÁLISIS DEL DESARROLLO DEL PROTOTIPO FÍSICO

Se desarrolló el prototipo físico vinculando el escáner OBD, la placa Arduino MEGA, el GPS y la laptop donde reside el prototipo virtual desarrollado, el dispositivo USB y la placa de desarrollo se enlazan por medio de los puertos USB a la laptop y el escáner automotriz se conecta por comunicación serie hacia la placa de desarrollo de esta manera todo el prototipo físico se encuentra enlazado y listo para ser acoplado con el prototipo virtual, el prototipo físico se lo puede apreciar en la figura 47.

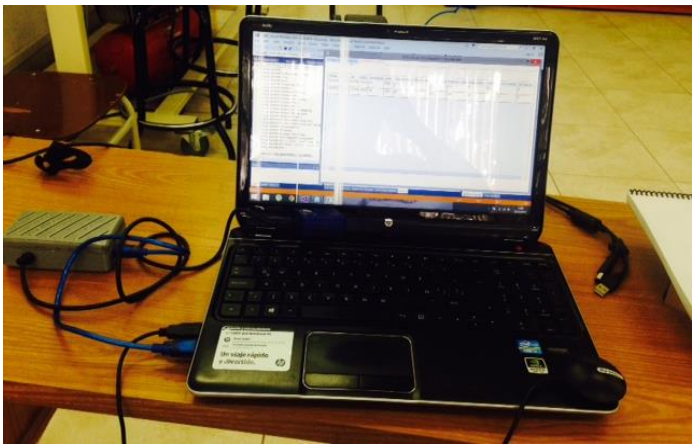


Figura 47 Prototipo físico del sistema de seguimiento satelital y obtención de datos de ubicación y telemetría

5 ANÁLISIS DE RESULTADOS

A continuación en este capítulo se detalló la implementación, pruebas de funcionamiento en algunos vehículos y el análisis de resultados obtenidos bajo los requerimientos del cliente, es importante conocer que el puerto del escáner automotriz es compatible con muchas marcas de vehículo pero la transferencia de información se activa siempre y cuando la configuración del protocolo de comunicación sea reconocido por el sistema electrónico del vehículo, esto se lo puede conocer revisando el manual de servicio del automóvil o buscando en la red por marca, año, modelo, ensamblaje, y otras características del vehículo.

Los protocolos soportados por el escáner automotriz son los que se muestran a continuación:

- CAN 500Kbps/11bit
- CAN 250Kbps/11bit
- CAN 500Kbps/29bit
- CAN 250Kbps/29bit
- ISO9141-2
- KWP2000 Fast
- KWP2000 5Kbps

No obstante los vehículos pueden ser a gasolina o a diésel, lo importante es que se comuniquen mediante un protocolo de los anteriormente indicados. Para conocer sobre modelos de vehículos que se comunican con el dispositivo OBD II se recomienda revisar el Anexo 1 donde se detalla información de vehículos compatibles con el escáner.

5.1 IMPLEMENTACIÓN VEHICULAR DEL DISPOSITIVO

Figura 48 Emulador automotriz OBD II de Freematics

Para la implementación del dispositivo y desarrollo de pruebas se utilizaron algunos vehículos para comprobar el funcionamiento del sistema de seguimiento y generación de reporte con los datos de ubicación y de telemetría y un dispositivo emulador vehicular que se muestra en la figura 48. Ara ver más vehículo en los cuales se hizo las pruebas de funcionamiento, revisar el Anexo 6.

5.1.1 Implementación en un vehículo Fabia Skoda

Figura 49 Fabia Skoda para implementación del sistema

La primera implementación se la realizó en un vehículo Fabia Skoda sedán del año 2001, ver la figura 49. Inicialmente se debe localizar el puerto para insertar el escáner automotriz que por lo general está ubicado de bajo del volante, su localización depende de la marca y modelo del vehículo, en nuestro caso, la ranura para conectar el escáner se encontraba localizado en una gaveta debajo del volante como se puede ver en la figura 50.

Figura 50 Puerto OBD dentro del vehículo

Primero se coloca el escáner automotriz en la ranura OBD del vehículo, como se muestra en la figura 51, si comienza a parpadear su led blanco interno quiere decir que se ha habilitado la comunicación con la placa de desarrollo.

Figura 51 Conexión escáner OBD en el puerto del primer vehículo

A continuación se conectó la antena GPS al equipo y se esperó hasta que el led rojo interno empezó a parpadear, lo que indica que el dispositivo permite obtener los parámetros de posición (latitud y longitud), se puede apreciar la antena GPS en el vehículo en la figura 52:

Figura 52 Conexión de GPS en el primer vehículo

Al haber conectado todos los dispositivos a la laptop se tiene todo el sistema instalado, como se muestra en la figura 53, se debe colocar el equipo en un sitio adecuado para evitar que existan molestias al conductor al momento de manejar.

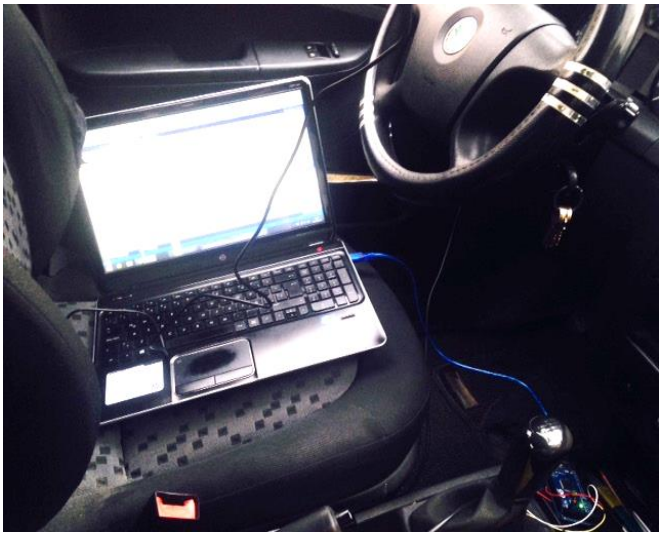


Figura 53 Sistema instalado en el primer vehículo

5.1.2 Implementación en un Chevrolet Aveo sedán

La segunda implementación se la realizó en un vehículo Chevrolet Aveo Sedán 2012 como se muestra en la figura 54.

Figura 54 Chevrolet Aveo para implementación del sistema

La ranura para conectar el escáner OBD se la encontró debajo del volante como se puede observar en la figura 55. El enlace empezó al emitir una luz azul de parpadeo, el led rojo indica que el dispositivo se encuentra conectado de una manera correcta. El GPS de la figura 56 se enlazó con la señal del satélite después de un tiempo aproximado de un minuto, el enlace se da a

todo momento pero las condiciones climáticas juegan un papel importante en el tiempo de enlace. El dispositivo se ha implementado de manera correcta.

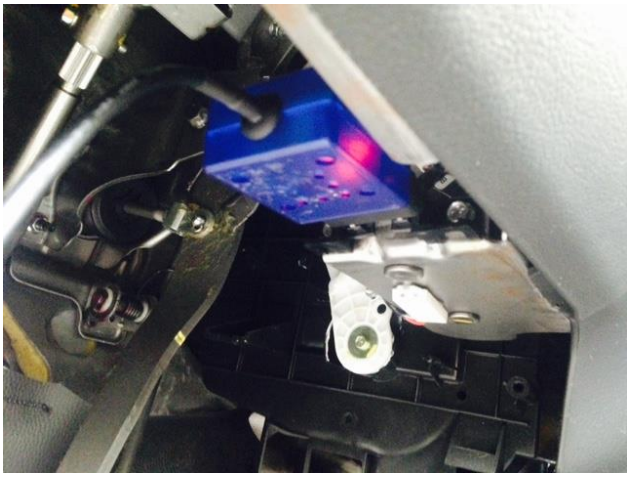


Figura 55 Conexión escáner OBD en el puerto del segundo vehículo

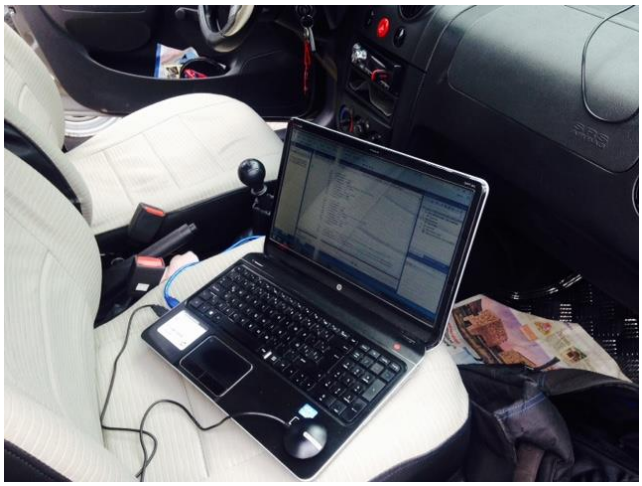


Figura 56 Sistema instalado en el segundo vehículo

5.1.3 Implementación en un Chevrolet Aveo sedán

Por último se implementó el dispositivo en un vehículo Hunday Getz como se puede observar en la figura 57. El GPS se enlazó rápidamente debido a que para esta prueba el cielo se encontraba despejado, el conector del escáner OBD se encontró bajo el volante pero a diferencia de los otros vehículos, estaba oculto a la vista del conductor como se puede apreciar en la figura 58. El enlace de datos fue inmediato.

Figura 57 Hunday Getz para implementación del sistema

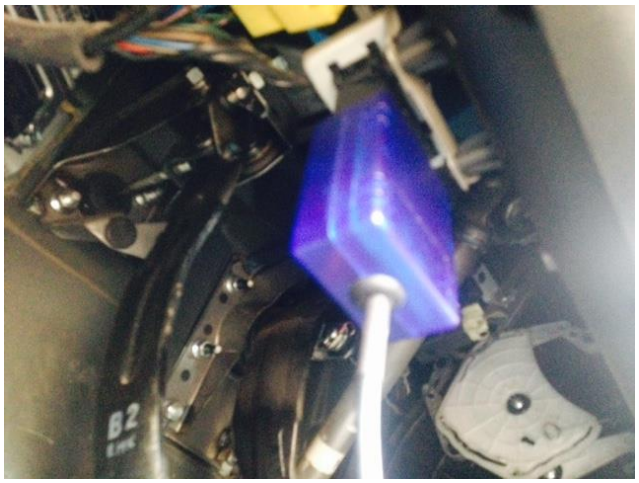


Figura 58 Conexión escáner OBD en el puerto del tercer vehículo

5.1.4 Implementación del dispositivo mediante el emulador automotriz

Para probar el escáner OBD II en el emulador, simplemente se ejecutó la aplicación del emulador, se conectó el escáner automotriz en el puerto del emulador como se aprecia en la figura 59, la conexión del GPS y de la placa de desarrollo es la misma que al instalar el dispositivo en el vehículo, se procede a esperar algunos segundos hasta que el dispositivo OBD II empiece a emitir datos de comunicación, al emitir los datos un led interno empezó a parpadear constantemente.

Figura 59 Conexión Emulador OBD con escáner automotriz

5.2 PRUEBAS DE FUNCIONAMIENTO

Después de haber implementado el dispositivo en tres vehículos diferentes además del emulador vehicular, se pretende dar seguimiento a los datos obtenidos por el dispositivo mediante la recepción y verificación de los datos proporcionados.

5.2.1 Pruebas de funcionamiento vehicular

Para iniciar las pruebas de funcionamiento vehicular primero se debe probar el funcionamiento del GPS, se abrió el software que proporciona la compañía donde se ha adquirido el dispositivo, se escogió el puerto de comunicación correspondiente y la velocidad de transferencia de bits de 4800, finalmente se pulsó en el botón Start GPS como se muestra en la figura 60, siempre y cuando las condiciones climáticas sean aceptables (cielo despejado) el enlace satelital se dará de forma inmediata, al no enlazarse el dispositivo permanecerá tratando de adquirir los datos, de igual manera se puede apreciar en la figura 60 que los satélites tornados de color azul son aquellos que proporcionan los datos de posicionamiento, al estar de color verde, el

dispositivo indica que son satélites que proporcionan datos de posición pero la señal no ha sido recibida y cuando se tornan de color rojo, indica que el enlace no es posible con esos satélites. El botón Cold Start acciona un reseteo en la adquisición de datos, que se lo puede utilizar para actualizar la adquisición de datos de posición del dispositivo.

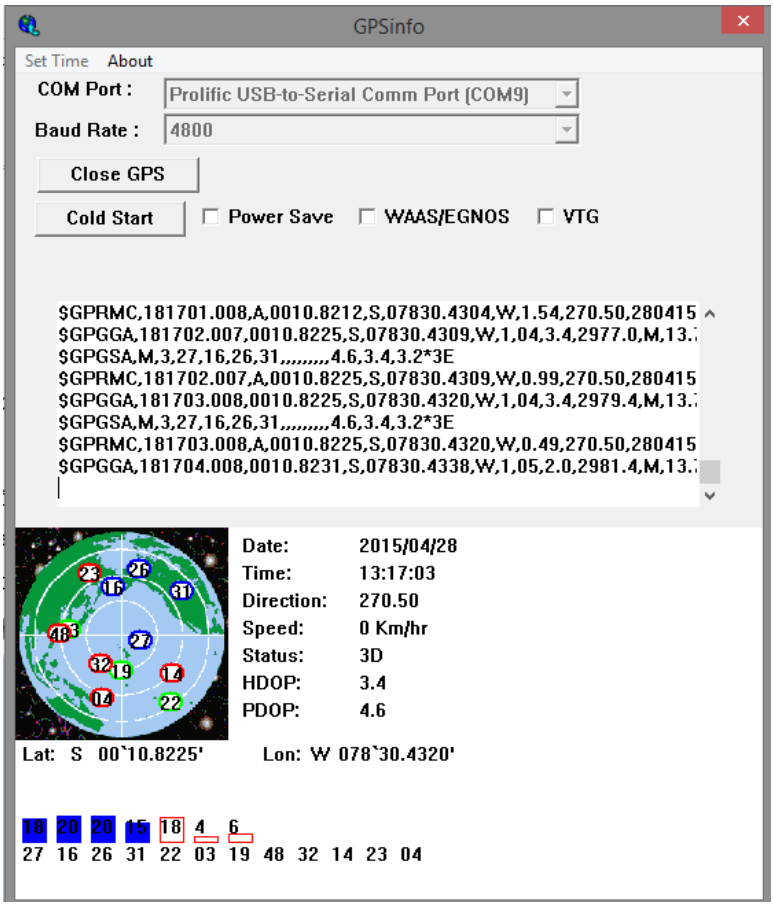


Figura 60 Pruebas funcionamiento de GPS

Al haber activado el software de seguimiento satelital, el sistema ubicará al automóvil en el instante que se encuentre para empezar con la emisión de datos de posición cada tres segundos que fue marcando el seguimiento, para realizar pruebas de funcionamiento, la trayectoria del vehículo fue en los alrededores de la Universidad Tecnológica Equinoccial en el campus Occidental, la prueba de seguimiento vehicular se lo puede apreciar en la figura 61.

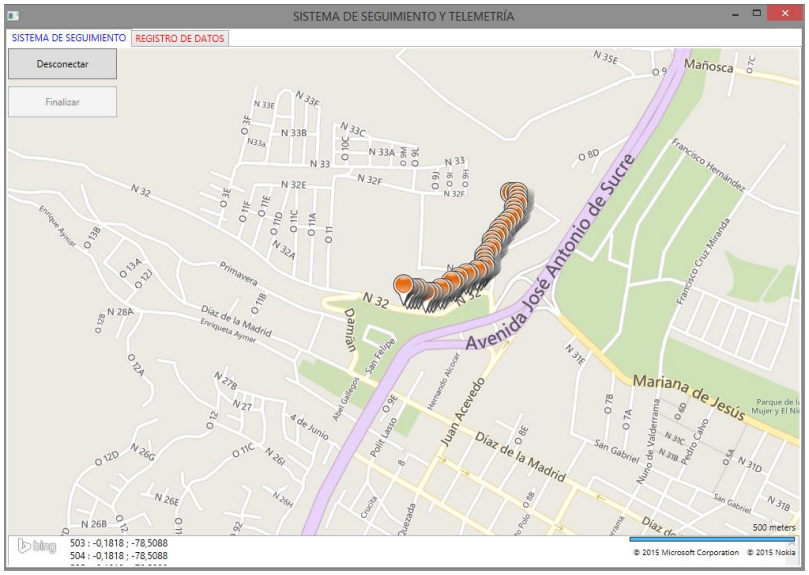
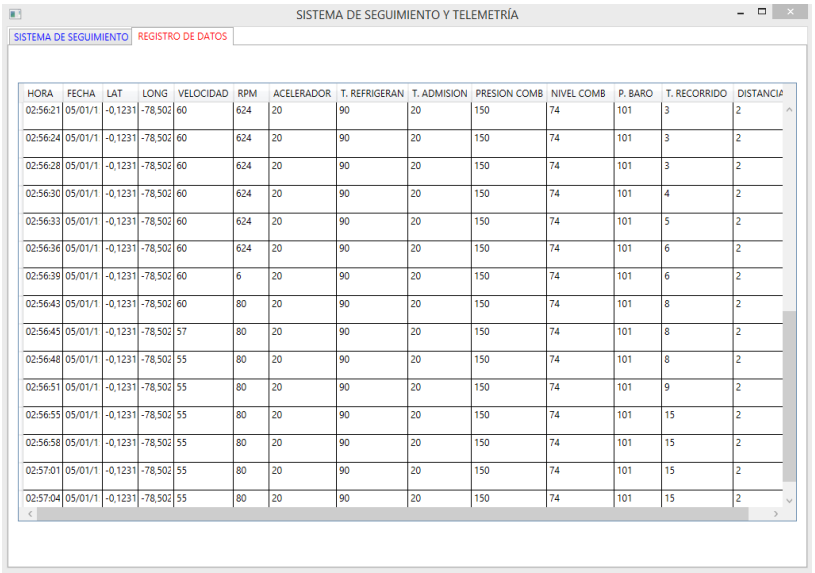


Figura 61 Seguimiento de pruebas en vehículo

Se pudo verificar que el seguimiento se dio de manera correcta y exacta a la posición del vehículo, la recepción de los datos de telemetría correspondiente a la prueba se la puede apreciar en la figura 62.



HORA	FECHA	LAT	LONG	VELOCIDAD	RPM	ACELERADOR	T. REFRIGERAN	T. ADMISION	PRESION COMB	NIVEL COMB	P. BARO	T. RECORRIDO	DISTANCIA
02:56:21	05/01/1	-0,1231	-78,502	60	624	20	90	20	150	74	101	3	2
02:56:24	05/01/1	-0,1231	-78,502	60	624	20	90	20	150	74	101	3	2
02:56:28	05/01/1	-0,1231	-78,502	60	624	20	90	20	150	74	101	3	2
02:56:30	05/01/1	-0,1231	-78,502	60	624	20	90	20	150	74	101	4	2
02:56:33	05/01/1	-0,1231	-78,502	60	624	20	90	20	150	74	101	5	2
02:56:36	05/01/1	-0,1231	-78,502	60	624	20	90	20	150	74	101	6	2
02:56:39	05/01/1	-0,1231	-78,502	60	6	20	90	20	150	74	101	6	2
02:56:43	05/01/1	-0,1231	-78,502	60	80	20	90	20	150	74	101	8	2
02:56:45	05/01/1	-0,1231	-78,502	57	80	20	90	20	150	74	101	8	2
02:56:48	05/01/1	-0,1231	-78,502	55	80	20	90	20	150	74	101	8	2
02:56:51	05/01/1	-0,1231	-78,502	55	80	20	90	20	150	74	101	9	2
02:56:55	05/01/1	-0,1231	-78,502	55	80	20	90	20	150	74	101	15	2
02:56:58	05/01/1	-0,1231	-78,502	55	80	20	90	20	150	74	101	15	2
02:57:01	05/01/1	-0,1231	-78,502	55	80	20	90	20	150	74	101	15	2
02:57:04	05/01/1	-0,1231	-78,502	55	80	20	90	20	150	74	101	15	2

Figura 62 Recepción de datos en vehículo

Como se puede observar los parámetros que se han adquirido durante estas pruebas de funcionamiento se las aprecia en la figura 62. Los variables elegidas por el cliente son las siguientes:

- **Hora:** Tiempo en el instante de la adquisición del dato.

- **Fecha:** Fecha en el instante de la adquisición del dato.
- **Latitud:** Coordenada geométrica correspondiente a la posición de latitud del vehículo en el instante.
- **Longitud:** Coordenada geométrica correspondiente a la posición de longitud del vehículo en el instante.
- **Velocidad:** Expresa la variación de posición del dispositivo en la unidad de tiempo.
- **RPM:** Expresa el número de rotaciones completas cada minuto realizadas por el motor del vehículo.
- **Acelerador:** Expresa la variación de la velocidad del vehículo en la unidad de tiempo.
- **Temperatura de refrigeración:** Es la variable que expresa la temperatura de refrigeración de componentes internos del motor del vehículo.
- **Tiempo de admisión:** Es el tiempo en el que el combustible ingresa al pistón para su posterior combustión.
- **Presión del combustible:** Indica la presión que existe en el combustible en el instante.
- **Nivel del combustible:** Indica el nivel del combustible en el instante.
- **Presión barométrica:** presión ejercida al dispositivo por la atmósfera en el instante.
- **Tiempo recorrido:** Tiempo de recorrido del sistema desde la ejecución del sistema.
- **Distancia:** Variable de distancia medido iniciado en la ejecución del sistema.

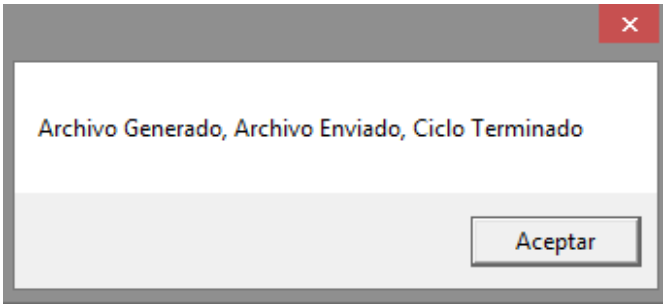


Figura 63 Confirmación de generación de archivo y envío del reporte

Al tener los datos de telemetría y de trayectoria listos se procede a finalizar el sistema generando automáticamente el reporte de datos y el envío de correo electrónico, al haber sido exitoso el proceso se tiene una ventana de confirmación de generación y envío del correo electrónico al administrador como se puede apreciar en la figura 63.

Para verificar el envío de los datos, se abrió el correo electrónico del receptor del reporte y se verificó que el correo con el reporte fue entregado como se muestra en la figura 64.

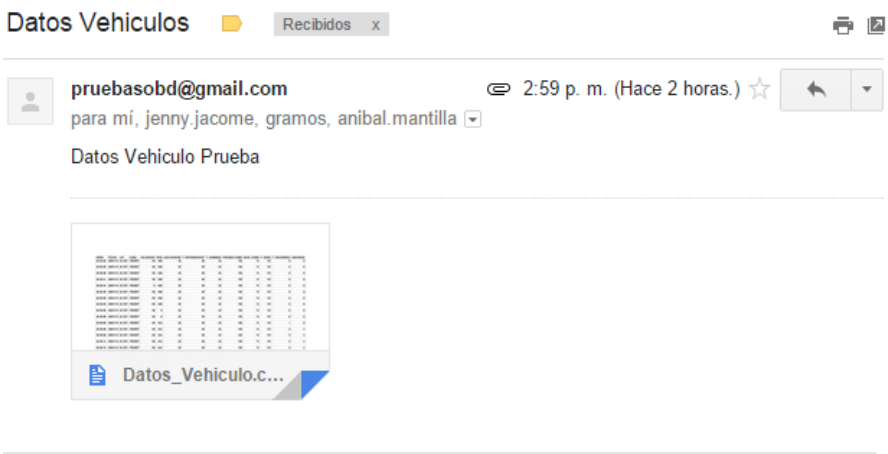


Figura 64 Recepción del reporte en el correo electrónico

Y al abrir el reporte se tiene el siguiente archivo que se lo observa en la figura 65.

02:56:21	05/01/15	-0,1231	-78,5027	60	624	20	90	20	150	74	101	3	2
02:56:24	05/01/15	-0,1231	-78,5027	60	624	20	90	20	150	74	101	3	2
02:56:28	05/01/15	-0,1231	-78,5027	60	624	20	90	20	150	74	101	3	2
02:56:30	05/01/15	-0,1231	-78,5027	60	624	20	90	20	150	74	101	4	2
02:56:33	05/01/15	-0,1231	-78,5027	60	624	20	90	20	150	74	101	5	2
02:56:36	05/01/15	-0,1231	-78,5027	60	624	20	90	20	150	74	101	6	2
02:56:39	05/01/15	-0,1231	-78,5027	60	6	20	90	20	150	74	101	6	2
02:56:43	05/01/15	-0,1231	-78,5027	60	80	20	90	20	150	74	101	8	2
02:56:45	05/01/15	-0,1231	-78,5027	57	80	20	90	20	150	74	101	8	2
02:56:48	05/01/15	-0,1231	-78,5027	55	80	20	90	20	150	74	101	8	2
02:56:51	05/01/15	-0,1231	-78,5027	55	80	20	90	20	150	74	101	9	2
02:56:55	05/01/15	-0,1231	-78,5027	55	80	20	90	20	150	74	101	15	2
02:56:58	05/01/15	-0,1231	-78,5027	55	80	20	90	20	150	74	101	15	2
02:57:01	05/01/15	-0,1231	-78,5027	55	80	20	90	20	150	74	101	15	2
02:57:04	05/01/15	-0,1231	-78,5027	55	80	20	90	20	150	74	101	15	2

Figura 65 Reporte de parámetros entregado

5.2.2 Pruebas de funcionamiento mediante el emulador automotriz

Para las pruebas de funcionamiento se implementó el dispositivo en un emulador automotriz de la figura 48 porque es equivalente a conectar el dispositivo en el puerto del escáner de cualquier vehículo, inclusive posee ventajas al tener la red inalámbrica al alcance del sistema para realizar pruebas de funcionamiento; se inicia las pruebas ejecutando la aplicación del escáner automotriz como se muestra en la figura 66:

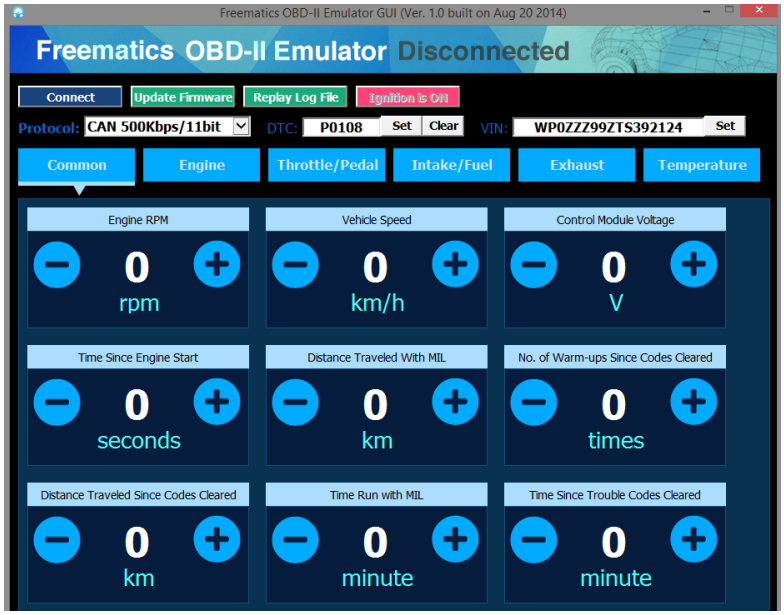


Figura 66 Aplicación del emulador OBD

Se puede observar que la aplicación se encuentra enlazada (Connected) en la figura 67. Para validar la adquisición de los parámetros se pretende ejecutar el monitor serial del entorno Arduino para comprobar la adquisición de estos parámetros desde la tarjeta de desarrollo como se puede apreciar en la figura 68.

Figura 67 Enlace del emulador activado

Figura 68 Pruebas de adquisición de datos desde el monitor serial de la placa de desarrollo

Antes de enlazar al emulador se puede notar que el monitor serial proporcionaba parámetros no válidos (0,0,0,-40,-40,0,0,0,0,0), pero al haberlo enlazado se puede apreciar que la adquisición de los datos han cambiado, se puede apreciar el kilometraje y las RPM que están en el emulador y se muestran en el monitor serial, observar la figura 36.

Se observó que toda la adquisición de datos se presentó sin ningún problema, por lo que se procederá a abrir la aplicación desarrollada como administrador para que tenga todos los permisos de funcionamiento, observar la figura 69:

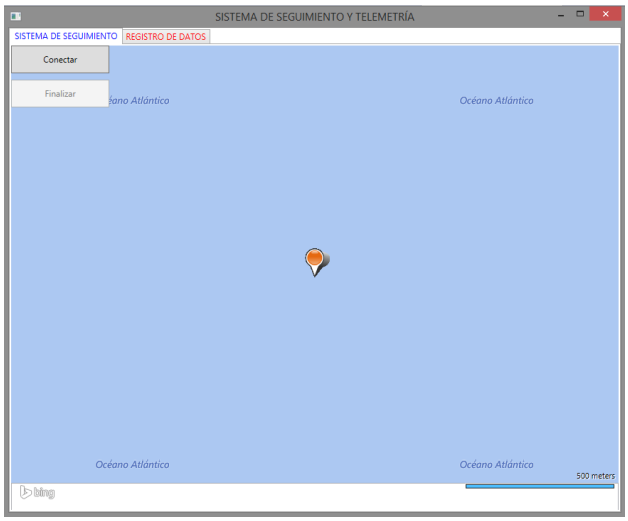


Figura 69 Inicio de aplicación

Al iniciar la aplicación, el sistema automáticamente se posicionó en el lugar en que el dispositivo GPS se encuentra en ese instante y empieza a marcar el seguimiento en el equipo como se observa a continuación en la figura 70:

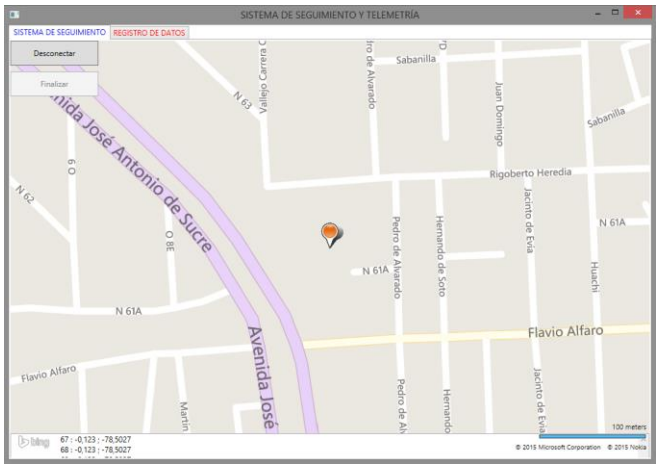


Figura 70 Inicio y posicionamiento del pin de seguimiento

Como se había programado, el sistema de seguimiento realiza el rastreo cada 3 segundos, por lo que se desplazó al equipo por el lugar para comprobar el rastreo de posiciones, se debe contar con una buena señal inalámbrica para evitar problemas de enlace, también se puede utilizar un modem wireless USB o compartir internet de plan de datos desde un dispositivo móvil, se puede comprobar el funcionamiento del seguimiento sin vehículo en la figura 71.

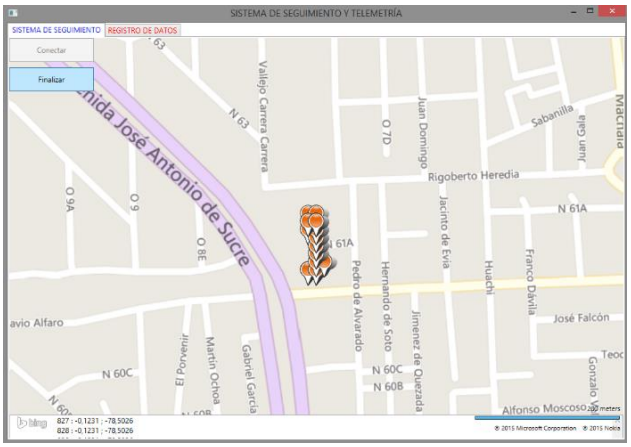


Figura 71 Muestreo del seguimiento satelital sin vehículo

Las marcas se mostraron cercanas porque para esta prueba no se contó con un vehículo, sino se utilizó el emulador automotriz y mientras el usuario camina, el sistema actualiza la posición del equipo por donde el usuario se encuentre. Se puede apreciar en la tabla de la figura 72 que los valores que se han almacenado son los proporcionados por la aplicación del emulador automotriz mientras el sistema estaba enlazado y en la figura 73 se aprecia la confirmación de la generación y envío del reporte al correo electrónico del administrador.

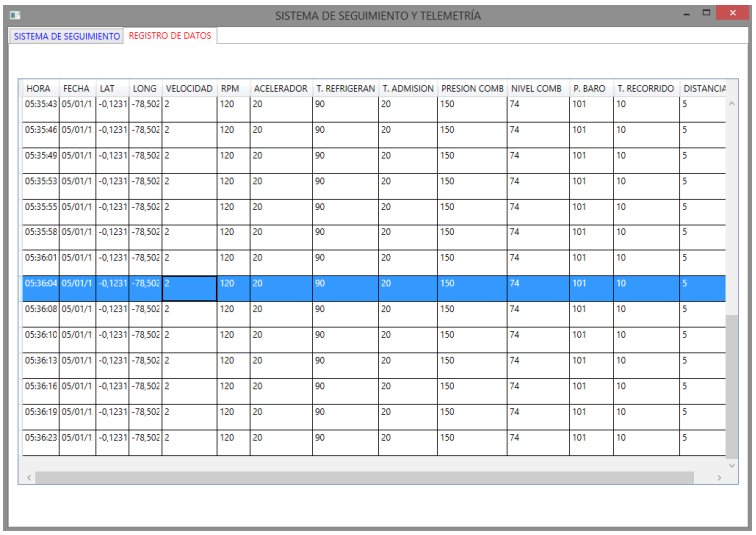


Figura 72 Reporte de datos de posición y telemetría en tabla

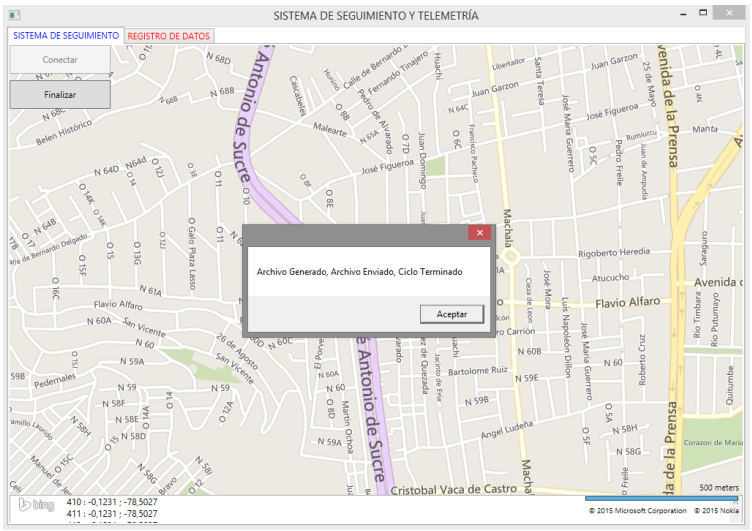


Figura 73 Generación y envío del reporte

Al pulsar el botón finalizar se almacenaron todos los parámetros registrados en un archivo de formato .csv y fueron enviados al administrador, un cuadro de mensaje se despliega para informar al usuario que se ha generado y se ha enviado el archivo con los parámetros de la tabla. Se puede comprobar la recepción del documento revisando el correo del administrador, en este caso se ha vinculado un correo especial para realizar pruebas como se observa en la figura 74.

Figura 74 Recepción de archivo con parámetros del sistema

Posteriormente para comprobar que los datos proporcionados son correctos se abrió el archivo adjunto para visualizar el documento proporcionado por el sistema, observar la figura 75:

HORA	FECHA	LAT	LONG	VELOCIDAD	RPM	ACELERADOR	T. REFRIGERANTE	T. ADMISION	PRESION COMB	NIVEL COMB	P. BARO	T. RECORRIDO	DISTANCIA
05:59:51	05/01/15	-0,1231	-78,5027	2	120	20	90	20	150	74	101	10	5
05:59:54	05/01/15	-0,1231	-78,5027	2	120	20	90	20	150	74	101	10	5
05:59:58	05/01/15	-0,1231	-78,5027	2	120	20	90	20	150	74	101	10	5
06:00:01	05/01/15	-0,1231	-78,5027	2	120	20	90	20	150	74	101	10	5
06:00:04	05/01/15	-0,1231	-78,5027	2	120	20	90	20	150	74	101	10	5
06:00:07	05/01/15	-0,1231	-78,5027	2	120	20	90	20	150	74	101	10	5
06:00:10	05/01/15	-0,1231	-78,5027	2	120	20	90	20	150	74	101	10	5
06:00:13	05/01/15	-0,1231	-78,5027	2	120	20	90	20	150	74	101	10	5
06:00:16	05/01/15	-0,1231	-78,5027	2	120	20	90	20	150	74	101	10	5
06:00:19	05/01/15	-0,1231	-78,5027	2	120	20	90	20	150	74	101	10	5
06:00:23	05/01/15	-0,1231	-78,5027	2	120	20	90	20	150	74	101	10	5
06:00:25	05/01/15	-0,1231	-78,5027	2	120	20	90	20	150	74	101	10	5
06:00:28	05/01/15	-0,1231	-78,5027	2	120	20	90	20	150	74	101	10	5
06:00:31	05/01/15	-0,1231	-78,5027	2	120	20	90	20	150	74	101	10	5
06:00:34	05/01/15	-0,1231	-78,5027	2	120	20	90	20	150	74	101	10	5
06:00:38	05/01/15	-0,1231	-78,5027	2	120	20	90	20	150	74	101	10	5
06:00:40	05/01/15	-0,1231	-78,5027	2	120	20	90	20	150	74	101	10	5
06:00:43	05/01/15	-0,1231	-78,5027	2	120	20	90	20	150	74	101	10	5
06:00:46	05/01/15	-0,1231	-78,5027	2	120	20	90	20	150	74	101	10	5
06:00:49	05/01/15	-0,1231	-78,5027	2	120	20	90	20	150	74	101	10	5
06:00:53	05/01/15	-0,1231	-78,5027	2	120	20	90	20	150	74	101	10	5
06:00:56	05/01/15	-0,1231	-78,5027	2	120	20	90	20	150	74	101	10	5
06:00:59	05/01/15	-0,1231	-78,5027	2	120	20	90	20	150	74	101	10	5

Figura 75 Archivo .csv abierto de datos proporcionados por el sistema

5.3 ANÁLISIS DE LOS RESULTADOS DE IMPLEMENTACIÓN Y PRUEBAS REALIZADAS

A continuación se presenta un análisis realizado posterior a la implementación del sistema en los vehículos de prueba y en el escáner automotriz, también sobre las pruebas de funcionamiento, para esto se analiza el funcionamiento del sistema bajo su conectividad, su enlace y la adquisición de datos.

5.3.1 Conectividad

De las conexiones realizadas anteriormente se pudo conocer que para asegurarnos de la recepción de datos, el puerto debe estar muy ajustado al escáner automotriz como se aprecia en la siguiente figura 76. Si la conexión está floja no se obtendrán datos de telemetría y es posible que la adquisición de datos esté en blanco.

Figura 76 Conexión ajustada entre el escáner OBD II y el puerto del vehículo

5.3.2 Enlace

En dos vehículos el equipo se enlazó correctamente, sin embargo en el tercer vehículo se presentaron problemas de enlace debido a que el equipo presentó una desconfiguración del puerto de comunicación de la placa de desarrollo, el

sistema no reconoció este puerto y no se pudo recibir los datos de telemetría, por lo que para que el equipo se enlace correctamente es importante realizar la conexión del sistema antes de encender el vehículo y desconectarlo cuando el vehículo se encuentre apagado, esto asegura el correcto enlace del sistema y evita la pérdida de recepción de datos de telemetría.

Si el dispositivo no emite los datos, el sistema solo funciona con el seguimiento satelital pero no posee un registro de datos.

5.3.3 Recepción de datos

El envío y recepción de datos se dio con normalidad en todas las pruebas de funcionamiento realizadas tanto con el vehículo como con el emulador, la velocidad del envío del reporte dependerá de la banda de red que proporcione el dispositivo de red inalámbrica para ser entregada, por lo general es inmediato y tarda de 30 a 40 segundos, si el reporte no ha sido enviado por problemas de red, el sistema presentará un mensaje de error de envío de registro.

6 CONCLUSIONES Y RECOMENDACIONES

6.1 CONCLUSIONES

- La adquisición correcta de datos de telemetría se establece de acuerdo a la selección adecuada del protocolo de comunicación que depende del modelo, año, marca y tipo de combustible de los automóviles.
- Gracias al escáner automotriz que se integró en los vehículos, se consiguió realizar un diagnóstico personalizado del vehículo y registrar información que después será necesaria para poder encontrar posibles averías en el vehículo con mayor facilidad y precisión.
- El sistema desarrollado permite a los administradores o a los conductores realizar un análisis sobre los datos del vehículo presentado en cada trayectoria realizada y conseguir solucionar algunos inconvenientes de funcionamiento del vehículo presentados con mayor facilidad al poder utilizar la información adquirida.
- El software desarrollado demuestra al usuario que se puede vincular algunos dispositivos a través de sus puertos de comunicación para la adquisición de datos en tiempo real.
- Es importante indicar que para el desarrollo del software se utilizó un software externo adicional que permite complementar con archivos necesarios adquiridos de la red para brindar un buen funcionamiento de la aplicación en el entorno donde sea aplicado.
- El sistema demostró que la adquisición de datos del escáner automotriz es para vehículos a gasolina como para los de diésel siempre y cuando los vehículos posean los protocolos de comunicación de datos de telemetría que el escáner automotriz posee.
- Los datos proporcionados por el sistema dependerán de los tipos de sensores que los vehículos integren, si el vehículo no presenta ciertos sensores, el dato correspondiente a ese sensor será un valor negativo para indicar que el vehículo no presta información sobre ese sensor.
- El escáner automotriz es de costo económico por lo que en caso de sufrir algún daño se lo puede sustituir con normalidad, además de no afectar a la configuración del proyecto.

- Se puede configurar al sistema para que entregue al cliente la totalidad de datos proporcionada por el escáner automotriz o personalizar la entrega de datos de acuerdo a su necesidad.

6.2 RECOMENDACIONES

- Al probar el GPS, su antena de transmisión debe estar directamente expuesta a la intemperie con la finalidad de que se obtengan datos de ubicación instantáneamente y sin retrasos por condiciones climáticas.
- Conectar el sistema antes de encender el vehículo para asegurar la transferencia de datos de telemetría y desconectarlo después de apagar el vehículo, siguiendo una secuencia correcta de enlace.
- Asegurar que el sistema tenga cobertura de red para poder visualizar el seguimiento vehicular en todo momento, la cobertura puede ser mediante el uso de un router USB wireless, compartir el plan de datos de un dispositivo movil o contratar un plan de datos para la laptop si el equipo lo permite.
- Asegurarse que al conectar el escáner automotriz en el puerto del vehículo quede muy ajustado con la finalidad de evitar problemas de envío de datos de telemetría al software.
- Revisar el manual de usuario del vehículo y comprobar si el vehículo cumple con uno de los protocolos de comunicación que posee el escáner automotriz para asegurar el enlace de datos.
- Tener un soporte adecuado para la laptop, la cual debería estar ubicada en una posición donde el conductor pueda revisar la ruta de seguimiento del vehículo de manera clara.
- Si se desea una mayor sensibilidad en la adquisición de datos de ubicación por parte del GPS, se recomienda adquirir un dispositivo de mejores características y de mayor sensibilidad en la comunicación.

GLOSARIO DE ASINCRÓNIMOS

ANSI American National Standards Institute, Instituto de estándares de Estados Unidos. Uno de los organismos de estandarización más importantes.

API Interfaz de Programación de Aplicaciones.

ATU Área de Trabajo de Usuario. Parte de la memoria que utilizan los procesos de usuario para almacenar los datos recibidos de una base de datos.

CLI (Common Language Infrastructure).

CLR Common Language Runtime.

DB Abreviatura de Data Base, base de datos.

DBA (Data Base Administrator) Nombre que recibe el administrador de la base de datos.

DBMS (Data Base Management System) Sistema gestor de bases de datos. El software encargado de administrar y producir bases de datos.

DCL (Data Control Language) Lenguaje de control de datos. Lenguaje que proporcionan las DBMS para controlar los usuarios de la base de datos.

DDL (Data Definition Language) Lenguaje de definición de datos. Lenguaje que proporcionan las DBMS para definir la base de datos.

DML (Data Modification Language) Lenguaje de modificación de datos. Lenguaje que proporcionan las DBMS para realizar operaciones de búsqueda y modificación de datos.

ERE Modelo entidad relación extendido.

GNSS Sistema Global de Navegación por Satélite

IDE Entorno de Desarrollo Integrado.

LINQ (Language Integrated Query) Lenguaje Query Integrado.

LI Lenguaje Intermedio.

SMTP Protocolo de Transferencia de Correo Simple.

SO Sistema Operativo.

SPARC (System Planing and Repairments Comitte) Comité de Planificación de Sistemas y Reparaciones.

UART (Universal Asincronical Receiver-Transmitter) Transmisor-Receptor Asíncrono Universal.

XAML (eXtensible Application Markup Language) Lenguaje Extensible de Formato para Aplicaciones.

X3 Sección de ANSI encargada de los estándares de ordenadores.

BIBLIOGRAFÍA

Arduino. (2015). *Arduino Mega*. Obtenido de Arduino: http://arduino.cc/en/pmwiki.php?n=Main/ArduinoBoardMega

Bolton, W. (2010). *Mecatrónica.* México: Alfaomega.

Bonet, E. (2002). *Universidad de Valencia.* Obtenido de El Lenguaje C: http://informatica.uv.es/estguia/ATD/apuntes/laboratorio/Lenguaje-C.pdf

Bryan, C. (2014). *Arduino makes electronics easier and we make Arduino projects elegant*. Obtenido de Arduino makes electronics easier and we make Arduino projects elegant: http://arduinodev.com/hardware/obd-kit

Concepción, M. (2010). *Estrategias de Sistemas OBD II.* U.S.A.: Mandy Concepción.

Eduardo Huerta, A. M. (2005). *GPS Posicionamiento Satelital. Primera Edición.* Argentina: UNR.

Gentilini, E. F. (2011). *Sistema teleoperado de un brazo robótico Mitsubishi RV-2AJ.* Costa Rica.

Jassi, N. (2015). *CAD/CAM/CAE Softwares*. Obtenido de CAD CAM Fundamentals: http://cadcamfunda.com/yahoo_site_admin/assets/images/t_flex.41161043_std.jpg

Katrib, M. (2009). *Windows Presentation Fundation 7.* Madrid, España: Luarna Ediciones, S.L.

MacDonald, M. (2012). *Pro WPF 4.5 in C#.* Washington, United States: 4 Ed., Apress.

Málaga, U. d. (2014). *Comunicaciones Asincrónicas UART's.* Obtenido de Departamento de Electrónica de la Universidad de Málaga: http://www.el.uma.es/marin/Practica4_UART.pdf

Mariano, D. (2013). *¿Qué es un Kit de desarrollo de Software (SDK)?* Obtenido de 4R Web Factory: http://www.4rsoluciones.com/que-es-un-kit-de-desarrollo-de-software-sdk/

Martínez, J. (10 de Abril de 2011). *JoseMaCo´s Blog.* Obtenido de Electrónica del Automóvil - OBD II: https://josemaco.files.wordpress.com/2011/04/sistema-obdii.pdf

Medina, G. I. (2000). *Algorítmica y Programación para Ingenieros. Primera Edición.* Barcelona - España: UPC Edicions.

Microsoft. (2015). *Introducción al Lenguaje C# y .NET Framework*. Obtenido de Microsoft Developer Network: http://msdn.microsoft.com/es-es/library/z1zx9t92.aspx

Narváez, P. (2008). *ASP.NET con C#.* Cornellá de Llobregat, Barcelona, España: EDICIONES ENI.

Rodríguez, L. (2007). *El Gran Libro del PC Interno: Programación de sistemas hardware a fondo.* Barcelona, España: MARCOMBO.

Sanz, J. F. (2009). *CAD.CAM: GRAFICOS, ANIMACION Y SIMULACION POR COMPUTADOR.* Madrid, España: S.A. EDICIONES PARANINFO.

Shafranovich. (Octubre de 2005). *Common Format and MIME Type for CSV Files*. Obtenido de IETF Tools: http://tools.ietf.org/html/rfc4180#page-2

SIEMENS. (2015). *CAD Diseño Asistido por Computadora.* Obtenido de Siemens PLC Automation: http://www.plm.automation.siemens.com/es_sa/plm/cad.shtml

Vasquez, J. (2008). *ANALISIS Y DISEÑO DE PIEZAS DE MAQUINAS CON CATIA V5: METODOS DE LOS ELEMENTOS FINITOS.* Ciudad de México, México: S.A. MARCOMBO.

ANEXOS

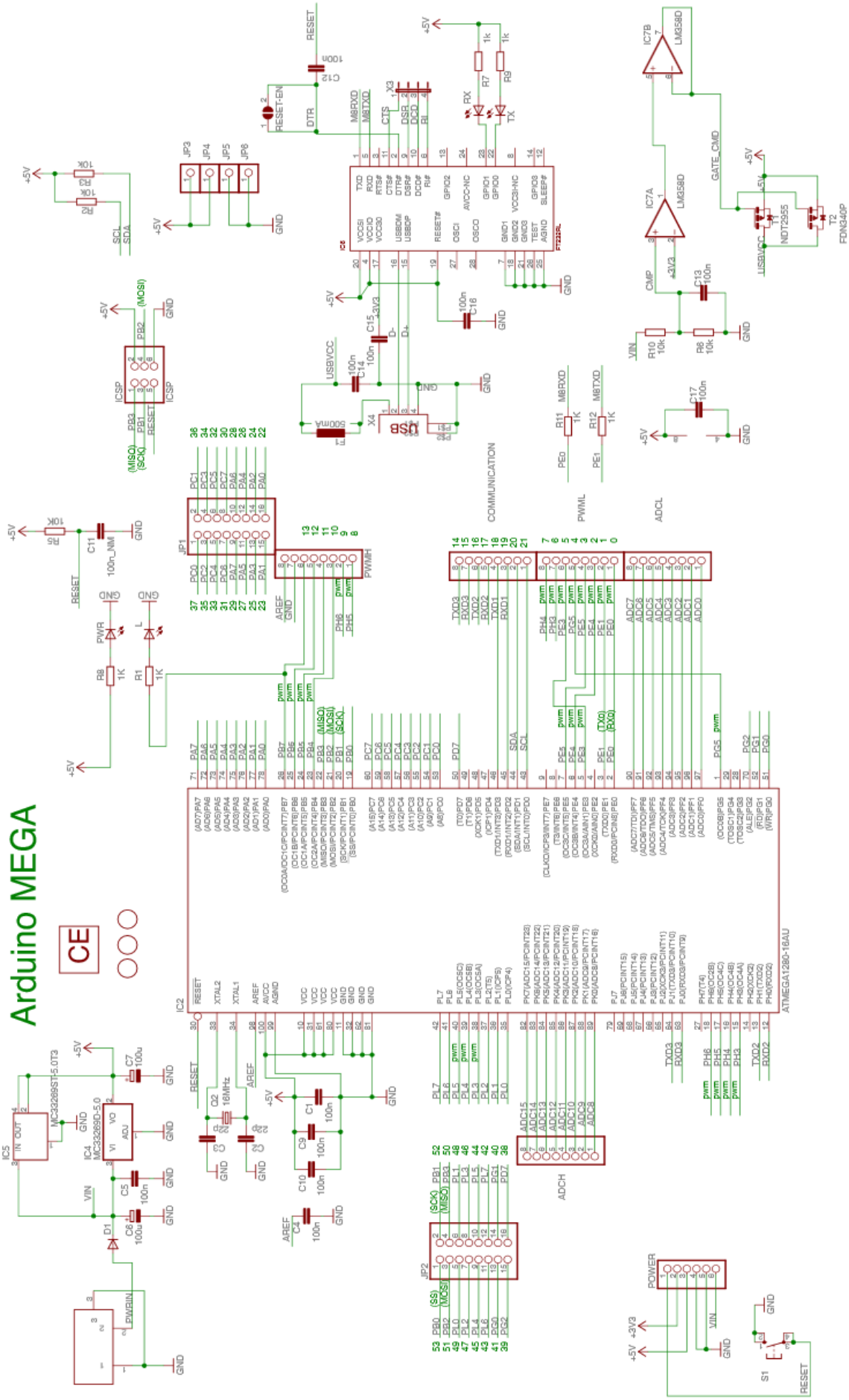
Anexo 1 Listado de compatibilidad del escáner OBD II de algunos vehículos

Model	Engine	Year (starting from)	OBD-2 Protocol
Buick Century		2002	VPW J1850
Buick Rivera		1998	VPW J1850
Buick Skylark		1996	VPW J1850
Cadillac Deville		2000	VPW J1850
Cadillac Eldorado		1998	VPW J1850
GMC Jimmy		1999	VPW J1850
GMC K2500		1997	VPW J1850
GMC Lumina		2001	VPW J1850
GMC Yukon		1998	VPW J1850
Chevrolet Astra	2.0 Flex Power, Gasoline (127 HP)	2004	ISO 14230-4, ISO 9141-2
Chevrolet Aveo	1.4, Gasoline (60HP)	2005	KWP FAST
	1.6, Gasoline (103HP)	2005	KWP FAST
	1.6, Gasoline (103HP)	2007	KWP FAST
	1.2, Gasoline (70HP)	2008	KWP FAST
	1.5, Gasoline (?HP)	2008	KWP FAST
	Gasoline (84 HP)	2009	ISO 14230-4
	1.2, Gasoline or LPG (82HP)	2009	KWP FAST
Chevrolet Blazer		1995	VPW J1850
Chevrolet Blazer LT	Gasoline (191 HP)	1997	VPW J1850
Chevrolet Camaro		1997	VPW J1850
		1997	ISO 14230-4, ISO 9141-2
		1998	VPW J1850
	3.8 V6, Gasoline (193HP)	1998	VPN
		1999	VPW J1850
	5.7 LS1, Gasoline (288HP)	1999	VPN
		2000	VPW J1850
	L36, Gasoline (191 HP)	2002	VPW J1850
	6.2 V8, Gasoline (405HP)	2012	CAN 11bit (500kb)

Model	Engine	Year (starting from)	OBD-2 Protocol
Chevrolet Caprice		1996	VPW J1850
Chevrolet Captiva	2.0 VCDI, Diesel (150HP)	2006	CAN 11bit (500kb)
	2.2 VCDI, Diesel (163HP)	2011	CAN 11bit (500kb)
Chevrolet Cavalier		1996	VPW J1850
		1998	VPW J1850
		1999	VPW J1850
Chevrolet Cobalt	Gasoline (145 HP)	2006	CAN
Chevrolet Corvette		2000	VPW J1850
	V8, Gasoline (400HP)	2007	CAN 11bit (500kb)
Chevrolet Corvette C5	5.7 V8, Gasoline (350HP)	2003	VPN
Chevrolet Corvette C6	6, Gasoline (404HP)	2005	ISO 9141
	LS7, Gasoline (512HP)	2007	CAN 11bit (500kb)
	6.2, Gasoline (436HP)	2008	CAN 11bit (500kb)
Chevrolet Cruze	2.0 VCDi, Diesel (150HP)	2010	CAN 11bit (500kb)
	2.0 VCDI, Diesel (163HP)	2011	CAN 11bit (500kb)
Chevrolet Epica	2.5, Gasoline (155HP)	2004	KWP FAST
	2.0 vdCi, Diesel (150HP)	2007	CAN 11bit (500kb)
Chevrolet Evanda	Gasoline (176 HP)	2005	
Chevrolet Express van	V8 5.3L, Gasoline (300HP)	2003	VPN
Chevrolet HHR	Gasoline (168 HP)	2008	
Chevrolet Impala		2000	VPW J1850
		2002	VPW J1850
Chevrolet Kalos	1.4 SE, Gasoline (93 HP)	2005	ISO 14230-4, ISO 9141-2
	1.4 16V, Gasoline (93 HP)	2006	ISO 14230-4
Chevrolet Lumina		1996	VPW J1850
Chevrolet Malibu		1997	
Chevrolet Malibu	3.1 V6, Gasoline (150HP)	2000	VPN
Chevrolet Matiz	1.0 SE LPG.i, Gasoline (64HP)	2006	KWP FAST
	Gasoline (66 HP)	2007	ISO 14230-4, ISO 9141-2

Model	Engine	Year (starting from)	OBD-2 Protocol
	0.8, Gasoline (65HP)	2009	KWP FAST
Chevrolet Nubira	1.8 BVA, Gasoline or LPG (122HP)	2006	KWP FAST
Chevrolet Optra	1.6, Gasoline (109HP)	2005	KWP FAST
Chevrolet Orlando	1.8, Gasoline (146HP)	2012	CAN 11bit (500kb)
	2.0 HDi, Diesel (163HP)	2012	CAN 11bit (500kb)
Chevrolet Rezzo	1.6 16v, Gasoline or LPG (107HP)	2005	ISO 9141
Chevrolet S10		2000	VPW J1850
Chevrolet Silverado		2002	VPW J1850
	6.5, Diesel (194HP)	1996	VPN
Chevrolet Spark	Gasoline (81 HP)	2008	
	1, Gasoline (68HP)	2011	KWP FAST
Chevrolet Tahoe		1996	VPW J1850
	5.7, Gasoline (265HP)	1998	VPN
	5.3, Gasoline (275HP)	2001	VPN
	5.3 V8, Gasoline (290HP)	2005	VPN
Chevrolet Trailblazer	Gasoline (270 HP)	2002	PWM J1850
	Gasoline (270 HP)	2002	
Chevrolet Trans Sport	Gasoline (184 HP)	2000	
	Gasoline (186 HP)	2003	
Chevrolet Transsport	Gasoline (184 HP)	1997	VPW J1850
Chevrolet Uplander	3.9, Gasoline (200HP)	2007	CAN 11bit (500kb)
	3.9, Gasoline (250HP)	2007	CAN 11bit (500kb)
Chevrolet Venture		1998	VPW J1850
	3.4, Gasoline (160HP)	2002	VPN

Anexo 2 Esquema electrónico de la placa Arduino MEGA 2650



Anexo 3 Programación del diseño de la Interfaz visual en XAML

```
<Window x:Class="SISTEMA_SEGUIMIENTO_TELEMETRÍA.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:MAPA="clr-namespace:Microsoft.Maps.MapControl.WPF;assembly=Microsoft.Maps.MapControl.WPF"
Title="SISTEMA DE SEGUIMIENTO Y TELEMETRÍA" Height="720" Width="1024">

<Grid>
        <TabControl SelectionChanged="TabControl_SelectionChanged">
           <TabItem>
               <TabItem.Header>
                   <StackPanel Orientation="Horizontal">
                       <TextBlock Text="SISTEMA DE SEGUIMIENTO" Foreground="Blue" />
                   </StackPanel>
        </TabItem.Header>
                 <MAPA:Map x:Name="myMap" CredentialsProvider="AnrEB8TwfV9IjlUv9PLRvLhzmevm9NhgQO-
                 F3PPYFCqg9567Dv0zEXDDkqhnv0vw" Center="-0.184459,-78.493334" ZoomLevel="16" Grid.Row="1">
                 <Button Content="Conectar" Name="Connect_btn" HorizontalAlignment="Left" Height="39"
                 VerticalAlignment="Top" Width="138" Click="Connect_btn_Click"/>
                 <RichTextBox Name="Commdata" Margin="0,615,0,0" VerticalScrollBarVisibility="Auto"
                 Height="66" VerticalAlignment="Top"/>
                 <Button Content="Finalizar" Name="Finalizar_btn" HorizontalAlignment="Left" Height="39"
                 VerticalAlignment="Top" Width="138" Margin="0,48,0,0" Click="Finalizar_btn_Click"/>
                 </MAPA:Map>
        </TabItem>
        <TabItem>
                 <TabItem.Header>
                          <StackPanel Orientation="Horizontal">
                                   <TextBlock Text="REGISTRO DE DATOS" Foreground="Red" />
                          </StackPanel>
                 </TabItem.Header>
                 <DataGrid AutoGenerateColumns="False" Name="Tabla_Datos_Vehiculos" CanUserAddRows="True"
                 ItemsSource="{Binding TestBinding}" Margin="10,45,10,55" >
                 <DataGrid.Columns>
                 <DataGridTextColumn Header="HORA" IsReadOnly="True" Binding="{Binding Path=_Hora}"
                 Width="50"></DataGridTextColumn>
                 <DataGridTextColumn Header="FECHA" IsReadOnly="True"  Binding="{Binding Path=_Fecha}"
                 Width="50"></DataGridTextColumn>
                 <DataGridTextColumn Header="LAT" IsReadOnly="True"  Binding="{Binding Path=_Lat}"
                 Width="45"></DataGridTextColumn>
                 <DataGridTextColumn Header="LONG" IsReadOnly="True"  Binding="{Binding Path=_Long}"
                 Width="45"></DataGridTextColumn>
                 <DataGridTextColumn Header="VELOCIDAD" IsReadOnly="True"  Binding="{Binding
                 Path=_Velocidad}" Width="75"></DataGridTextColumn>
                 <DataGridTextColumn Header="RPM" IsReadOnly="True"  Binding="{Binding Path=_RPM}"
                 Width="45"></DataGridTextColumn>
                 <DataGridTextColumn Header="ACELERADOR" IsReadOnly="True"  Binding="{Binding
                 Path=_Acelerador}" Width="85"></DataGridTextColumn>
                 <DataGridTextColumn Header="T. REFRIGERANTE" IsReadOnly="True"  Binding="{Binding
                 Path=_Trefri}" Width="90"></DataGridTextColumn>
                 <DataGridTextColumn Header="T. ADMISION" IsReadOnly="True"  Binding="{Binding Path=_Tadmi}"
                 Width="80"></DataGridTextColumn>
                 <DataGridTextColumn Header="PRESION COMB" IsReadOnly="True"  Binding="{Binding Path=_Pcomb}"
                 Width="95"></DataGridTextColumn>
```

```
                <DataGridTextColumn Header="NIVEL COMB" IsReadOnly="True"  Binding="{Binding Path=_Ncomb}"
                Width="85"></DataGridTextColumn>
                <DataGridTextColumn Header="P. BARO" IsReadOnly="True"  Binding="{Binding Path=_Pbaro}"
                Width="60"></DataGridTextColumn>
                <DataGridTextColumn Header="T. RECORRIDO" IsReadOnly="True"  Binding="{Binding Path=_Treco}"
                Width="90"></DataGridTextColumn>
                <DataGridTextColumn Header="DISTANCIA" IsReadOnly="True"  Binding="{Binding
                Path=_Distancia}" Width="90"></DataGridTextColumn>
                </DataGrid.Columns>
            </DataGrid>
        </TabItem>
    </TabControl>
</Grid>
</Window>
```

Anexo 4 Programación C# del software de seguimiento y adquisición de datos de posición y de telemetría

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using Microsoft.Maps.MapControl.WPF;
using Microsoft.Maps.MapControl.WPF.Design;
using System.Globalization;
using System.IO.Ports;
using System.Windows.Threading;
using System.Text.RegularExpressions;
using System.IO;            // File, FileStream, StreamWriter
using System.Net.Mail;      //libreria correo
using System.Net;

namespace OBD_NOVATRONIX_TAB_CONTROL
{
/// Lógica de interacción para MainWindow.xaml
class Vehiculo //creamos variables string para la recepción de todos los datos telemetria, fecha y ODB
    {
        public string _Hora { get; set; }
        public string _Fecha { get; set; }
        public string _Lat { get; set; }
        public string _Long { get; set; }
        public string _Velocidad { get; set; }
        public string _RPM { get; set; }
        public string _Acelerador { get; set; }
        public string _Trefri { get; set; }
        public string _Tadmi { get; set; }
        public string _Pcomb { get; set; }
        public string _Ncomb { get; set; }
        public string _Pbaro { get; set; }
        public string _Treco { get; set; }
        public string _Distancia { get; set; }

         public Vehiculo(string hora, string fecha, string lat, string longitud, string velocidad, string rpm,
         string acelerador, string trefri, string tadmi, string pcomb, string ncomb, string pbaro, string
         treco, string distancia)

         {
            this._Hora = hora;
            this._Fecha = fecha;
            this._Lat = lat;
            this._Long = longitud;
            this._Velocidad = velocidad;
            this._RPM = rpm;
            this._Acelerador = acelerador;
            this._Trefri = trefri;
            this._Tadmi = tadmi;
            this._Pcomb = pcomb;
            this._Ncomb = ncomb;
            this._Pbaro = pbaro;
            this._Treco = treco;
            this._Distancia = distancia;

        }
    }

    public partial class MainWindow : Window
```

```
{
    #region variables

    //Richtextbox
    FlowDocument mcFlowDoc = new FlowDocument();
    Paragraph para = new Paragraph();
    //Serial
    SerialPort serialGPS = new SerialPort();
    SerialPort serialOBD = new SerialPort();
    string recieved_dataGPS;
    string recieved_dataOBD;
    //GPS
    double LAT = 0;
    double LON = 0;
    //CONTROL
    int x = 0;

    #endregion

    public MainWindow()
    {
        InitializeComponent();
        Finalizar_btn.IsEnabled = false;
        #region Temporizadores
        DispatcherTimer TimerSerial = new DispatcherTimer();
        TimerSerial.Tick += new EventHandler(Marcas_Tick);  //Creamos evento de marcado
        TimerSerial.Interval = new TimeSpan(0, 0, 3);       // Adquisición de datos cada 3 seg
        TimerSerial.Start();
        #endregion
        Connect_btn.Content = "Conectar";
        myMap.Focus();                                      //uso teclado para mover y acercar
        myMap.Mode = new RoadMode();                        //TIPO DE MAPA CAMINO
    }

    #region COMUNICACION
    private void Connect_btn_Click(object sender, RoutedEventArgs e)
    {
        if (Connect_btn.Content == "Conectar")
        {
            try
            {
                //Configuración GPS
                serialGPS.PortName = "COM9";
                serialGPS.BaudRate = 4800;
                serialGPS.Handshake = System.IO.Ports.Handshake.None;
                serialGPS.Parity = Parity.None;
                serialGPS.DataBits = 8;
                serialGPS.StopBits = StopBits.One;
                serialGPS.ReadTimeout = 200;
                serialGPS.WriteTimeout = 50;
                serialGPS.Open();

                //Configuración OBD
                serialOBD.PortName = "COM4";
                serialOBD.BaudRate = 9600;
                serialOBD.Handshake = System.IO.Ports.Handshake.None;
                serialOBD.Parity = Parity.None;
                serialOBD.DataBits = 8;
                serialOBD.StopBits = StopBits.One;
                serialOBD.ReadTimeout = 200;
                serialOBD.WriteTimeout = 50;
                serialOBD.Open();

               //Armar el boton de estado para crear la función de llamada y datos recibidos del GPS
               Connect_btn.Content = "Desconectar";
               serialGPS.DataReceived += new System.IO.Ports.SerialDataReceivedEventHandler(RecieveGPS);
               serialOBD.DataReceived += new System.IO.Ports.SerialDataReceivedEventHandler(RecieveOBD);

            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.ToString());
            }
```

```
    }
    else
    {
        if (Connect_btn.Content == "Desconectar")
        {
            serialGPS.Close();
            serialOBD.Close();
            Connect_btn.Content = "Conectar";
            Finalizar_btn.IsEnabled = true;
            Connect_btn.IsEnabled = false;
        }
    }
}

#region Recieving GPS

private delegate void UpdateUiTextDelegate(string text);
private void RecieveGPS(object sender, System.IO.Ports.SerialDataReceivedEventArgs e)
{
    try
    {
        // Colecta los datos obtenidos y los almacena en un 'buffer' (string).
          recieved_dataGPS = serialGPS.ReadLine();
          Dispatcher.Invoke(DispatcherPriority.Send, new UpdateUiTextDelegate(WriteDataGPS),
          recieved_dataGPS);
    }
          catch (Exception ex)
    {
          serialGPS.Close();
    }
}
private void WriteDataGPS(string text)
{
    // Asigna del valor de un recieved_data para almacenarlo en el RichTextBox.

          var coordenadas = Posicion(text);
          para.Inlines.Add(";" + x.ToString() + " : " + coordenadas.Key.ToString() + " ; " +
          coordenadas.Value.ToString() + "\r\n");
          mcFlowDoc.Blocks.Add(para);
          Commdata.Document = mcFlowDoc;
          Commdata.ScrollToEnd();
          x = x + 1;
}

#endregion

#region Recieving OBD

//private delegate void UpdateUiTextDelegate(string text);
private void RecieveOBD(object sender, System.IO.Ports.SerialDataReceivedEventArgs e)
{
    try
    {
        // Colecta los caracteres recibidos a un 'buffer' (string).
          recieved_dataOBD = serialOBD.ReadLine();
          Dispatcher.Invoke(DispatcherPriority.Send, new UpdateUiTextDelegate(WriteDataOBD),
          recieved_dataOBD);
    }
    catch (Exception ex)
    {
          serialOBD.Close();
    }
}
private void WriteDataOBD(string text)
{
    // Assign the value of the recieved_data to the RichTextBox.

 string[] obd = text.Split(',');
 string h = DateTime.Now.ToString("hh:mm:ss tt");
 string f = DateTime.Now.ToString("MM/dd/yy");
 //var datos = new Vehiculo("50 km/h", "100km", "2 horas", "30 galones");
 var datos_veh = new Vehiculo(h, f, LAT.ToString(), LON.ToString(), obd[0], obd[1], obd[2], obd[3],
 obd[4], obd[5], obd[6], obd[7], obd[8], obd[9]);
          if (datos_veh != null)
```

```
    { Tabla_Datos_Vehiculos.Items.Add(datos_veh); }
    Tabla_Datos_Vehiculos.ScrollIntoView(datos_veh); //autoscroll
}

#endregion

#endregion
#region MAPA

#region GPS
public KeyValuePair<double, double> Posicion(string gps)
{


    if (gps.Length > 0)
    {
        string[] cmd1 = Regex.Split(gps, "\r\n");

        foreach (string line in cmd1)
        {
            if (line.Contains("$GPRMC"))
            {
                string[] cmd = line.Split(',');

                if (cmd.Length >= 13)
                {
                    if (cmd[2] == "A" && cmd[2].Length > 0)
                    {
                        #region LATITUD
                        if (cmd[3].Length > 0)
                        {
                            int gradosLAT = Convert.ToInt16(cmd[3].Substring(0, 2));
                            double minutosLAT = Convert.ToDouble(cmd[3].Substring(2)) / 10000;

                            LAT = Math.Round((gradosLAT + (minutosLAT / 60)), 4);
                        }
                        else
                        {
                            //LAT = 0;
                        }

                        if (cmd[4].Length > 0)
                        {
                            switch (cmd[4])
                            {
                                case "N":
                                    break;
                                case "S":
                                    LAT = LAT * -1;
                                    break;
                                default:
                                    break;
                            }
                        }
                        #endregion

                        #region LONGITUD
                        if (cmd[5].Length > 0)
                        {
                            int gradosLON = Convert.ToInt16(cmd[5].Substring(0, 3));
                            double minutosLON = Convert.ToDouble(cmd[5].Substring(3)) / 10000;

                            LON = Math.Round((gradosLON + (minutosLON / 60)), 4);
                        }
                        else
                        {
                            //LON = 0;
                        }

                        if (cmd[6].Length > 0)
                        {
                            switch (cmd[6])
                            {
                                case "E":
```

```
                                    break;
                                case "W":
                                    LON = LON * -1;
                                    break;
                                default:
                                    break;
                            }
                        }
                        #endregion

                    }
                }
            }
        }

    }
    else
    {
        LAT = 0;
        LON = 0;
    }

    return new KeyValuePair<double, double>(LAT, LON);
}
#endregion

#region MARCAS
private void Marcas_Tick(object sender, EventArgs e)
{
    Location pinLocation = new Location(LAT, LON);

    // The pushpin to add to the map.
    Pushpin pin = new Pushpin();
    pin.Location = pinLocation;
    myMap.Center = pinLocation;
    // Adds the pushpin to the map.
    myMap.Children.Add(pin);

}

#endregion

#endregion
#region EMAIL
public void sendEMailThroughHotMail()
{
    try
    {
        //Mail Message
        MailMessage mM = new MailMessage();
        //Mail Address
        mM.From = new MailAddress("pruebasobd@gmail.com");
        //receiver email id
        mM.To.Add("jz.akta@gmail.com");
        mM.CC.Add("jenny.jacome@ute.edu.ec");
        //subject of the email
        mM.Subject = "Datos Vehiculos";
        //deciding for the attachment
        mM.Attachments.Add(new Attachment(@"C:\\Datos_Vehiculo.csv"));
        //add the body of the email
        mM.Body = "Datos Vehiculo Prueba";
        mM.IsBodyHtml = true;
        //SMTP client
        SmtpClient sC = new SmtpClient("smtp.gmail.com");
        //port number for Hot mail
        sC.Port = 587;
        //credentials to login in to hotmail account
        sC.Credentials = new NetworkCredential("pruebasobd@gmail.com", "pruebas.OBD.101");
        //enabled SSL
        sC.EnableSsl = true;
        //Send an email
        sC.Send(mM);
    }//end of try block
    catch (Exception ex)
```

```
            {
                MessageBox.Show(ex.Message);
            }//end of catch
        }
        #endregion

        private void Finalizar_btn_Click(object sender, RoutedEventArgs e)
        {
            //guardar datos
            Tabla_Datos_Vehiculos.SelectAllCells();
            Tabla_Datos_Vehiculos.ClipboardCopyMode = DataGridClipboardCopyMode.IncludeHeader;
            ApplicationCommands.Copy.Execute(null, Tabla_Datos_Vehiculos);
            Tabla_Datos_Vehiculos.UnselectAllCells();
            String result = (string)Clipboard.GetData(DataFormats.CommaSeparatedValue);
            Clipboard.Clear();
            System.IO.StreamWriter file = new System.IO.StreamWriter(@"C:\\Datos_Vehiculo.csv", false);
            file.WriteLine(result);
            file.Close();
            //
            //pausa tarea//
            System.Threading.Thread.Sleep(1000);
            //
            //enviar email
            sendEMailThroughHotMail();

            MessageBox.Show("Archivo Generado, Archivo Enviado, Ciclo Terminado");

            Connect_btn.IsEnabled = true;
        }

        private void TabControl_SelectionChanged(object sender, SelectionChangedEventArgs e)
        {

        }//end of Email Method HotMail
    }
}
```

Anexo 5 Configuración de la librería OBD.h (Protocolos de comunicación OBD II)

```
#include <Arduino.h>



#define OBD_MODEL_UART 0
#define OBD_MODEL_I2C 1

#define OBD_TIMEOUT_SHORT 2000 /* ms */
#define OBD_TIMEOUT_LONG 7000 /* ms */
#define OBD_SERIAL_BAUDRATE 38400
#define OBD_RECV_BUF_SIZE 128

#ifndef OBDUART
#if defined(__AVR_ATmega328P__) || defined(__AVR_ATmega168P__)
//#define OBDUART Serial
#else
#define OBDUART Serial1
#endif
#endif

// Modo 1 PIDs
#define PID_ENGINE_LOAD 0x04
#define PID_COOLANT_TEMP 0x05
#define PID_SHORT_TERM_FUEL_TRIM_1 0x06
#define PID_LONG_TERM_FUEL_TRIM_1 0x07
#define PID_SHORT_TERM_FUEL_TRIM_2 0x08
#define PID_LONG_TERM_FUEL_TRIM_2 0x09
#define PID_FUEL_PRESSURE 0x0A
#define PID_INTAKE_MAP 0x0B
#define PID_RPM 0x0C
#define PID_SPEED 0x0D
#define PID_TIMING_ADVANCE 0x0E
#define PID_INTAKE_TEMP 0x0F
#define PID_MAF_FLOW 0x10
#define PID_THROTTLE 0x11
#define PID_AUX_INPUT 0x1E
#define PID_RUNTIME 0x1F
#define PID_DISTANCE_WITH_MIL 0x21
#define PID_COMMANDED_EGR 0x2C
#define PID_EGR_ERROR 0x2D
#define PID_COMMANDED_EVAPORATIVE_PURGE 0x2E
#define PID_FUEL_LEVEL 0x2F
#define PID_WARMS_UPS 0x30
#define PID_DISTANCE 0x31
#define PID_EVAP_SYS_VAPOR_PRESSURE 0x32
#define PID_BAROMETRIC 0x33
#define PID_CATALYST_TEMP_B1S1 0x3C
#define PID_CATALYST_TEMP_B2S1 0x3D
#define PID_CATALYST_TEMP_B1S2 0x3E
#define PID_CATALYST_TEMP_B2S2 0x3F
#define PID_CONTROL_MODULE_VOLTAGE 0x42
#define PID_ABSOLUTE_ENGINE_LOAD 0x43
#define PID_RELATIVE_THROTTLE_POS 0x45
#define PID_AMBIENT_TEMP 0x46
#define PID_ABSOLUTE_THROTTLE_POS_B 0x47
```

```
#define PID_ABSOLUTE_THROTTLE_POS_C 0x48
#define PID_ACC_PEDAL_POS_D 0x49
#define PID_ACC_PEDAL_POS_E 0x4A
#define PID_ACC_PEDAL_POS_F 0x4B
#define PID_COMMANDED_THROTTLE_ACTUATOR 0x4C
#define PID_TIME_WITH_MIL 0x4D
#define PID_TIME_SINCE_CODES_CLEARED 0x4E
#define PID_ETHANOL_FUEL 0x52
#define PID_FUEL_RAIL_PRESSURE 0x59
#define PID_HYBRID_BATTERY_PERCENTAGE 0x5B
#define PID_ENGINE_OIL_TEMP 0x5C
#define PID_FUEL_INJECTION_TIMING 0x5D
#define PID_ENGINE_FUEL_RATE 0x5E
#define PID_ENGINE_TORQUE_DEMANDED 0x61
#define PID_ENGINE_TORQUE_PERCENTAGE 0x62
#define PID_ENGINE_REF_TORQUE 0x63

typedef enum {
    PROTO_AUTO = 0,
    PROTO_ISO_9141_2 = 3,
    PROTO_KWP2000_5KBPS = 4,
    PROTO_KWP2000_FAST = 5,
    PROTO_CAN_11B_500K = 6,
    PROTO_CAN_29B_500K = 7,
    PROTO_CAN_29B_250K = 8,
    PROTO_CAN_11B_250K = 9,
} OBD_PROTOCOLS;

// Estados
typedef enum {
    OBD_DISCONNECTED = 0,
    OBD_CONNECTING = 1,
    OBD_CONNECTED = 2
} OBD_STATES;

uint16_t hex2uint16(const char *p);
uint8_t hex2uint8(const char *p);

class COBD
{
public:
       COBD():dataMode(1),errors(0),m_state(OBD_DISCONNECTED) {}
       virtual void begin();
       // inicia la conexión OBD II
       virtual bool init(OBD_PROTOCOLS protocol = PROTO_AUTO);
       // termina la conexión OBD II
       virtual void end();
       // Ingresamos el baud rate del serial
       virtual void setBaudRate(long baudrate);
       // Obtener la conexión de estado
       virtual OBD_STATES getState() { return m_state; }
       // Lectura específica del valor del OBD-II PID
       virtual bool read(byte pid, int& result);
       // Enlace del dispositivo
       virtual void sleep();
       // Despierta al dispositivo de un estado de hibernación
       virtual void wakeup();
```

```
        // Arma el estado del protocolo (estado Auto predeterminado)
        virtual bool setProtocol(OBD_PROTOCOLS h = PROTO_AUTO);
        // Quitar códigos erroneos
        virtual void clearDTC();
        // Obtiene los voltages de la batería (en 0.1V, e.g. 125 para 12.5V, trabaja sin
ECU)
        virtual int getVoltage();
        // Envío de problemas y especificaiones PID
        virtual void sendQuery(byte pid);
        // regresar la respuesta del PID
        virtual bool getResult(byte& pid, int& result);
        // Determinar si el PID es soportado
        virtual bool isValidPID(byte pid);
        // Dejar el modo PID corriente
        byte dataMode;
        // Presencia de errores
        byte errors;
        // bit map de PIDs doportados
        byte pidmap[4 * 4];
protected:
        virtual char* getResponse(byte& pid, char* buffer);
        virtual void dataIdleLoop() {}
        void recover();
        void debugOutput(const char* s);
        int normalizeData(byte pid, char* data);
        OBD_STATES m_state;
private:
        virtual byte receive(char* buffer = 0, int timeout = OBD_TIMEOUT_SHORT);
        virtual bool available();
        virtual char read();
        virtual void write(const char* s);
        virtual void write(char c);
        virtual uint8_t getPercentageValue(char* data)
        {
                return (uint16_t)hex2uint8(data) * 100 / 255;
        }
        virtual uint16_t getLargeValue(char* data)
        {
                return hex2uint16(data);
        }
        virtual uint8_t getSmallValue(char* data)
        {
                return hex2uint8(data);
        }
        virtual int16_t getTemperatureValue(char* data)
        {
                return (int)hex2uint8(data) - 40;
        }
};

#define I2C_ADDR 0x62

#define MAX_PAYLOAD_SIZE 32
#define MAX_PIDS 8

#define CMD_QUERY_STATUS 0x10
#define CMD_SEND_AT_COMMAND 0x11
```

```
#define CMD_APPLY_OBD_PIDS 0x12
#define CMD_LOAD_OBD_DATA 0x13
#define CMD_GPS_SETUP 0x20
#define CMD_GPS_QUERY 0x22

typedef struct {
    uint16_t age;
    uint16_t value;
} PID_INFO;

typedef struct {
    uint16_t time;
    uint8_t message;
    uint8_t data;
} COMMAND_BLOCK;

typedef struct {
    uint32_t time;
    uint32_t date;
    float lat;
    float lon;
    float speed;
    float alt;
    uint8_t sat;
    uint8_t state;
    uint16_t age;
    uint8_t reserved[4];
} GPS_DATA;

class COBDI2C : public COBD {
public:
    void begin();
       void end();
    bool init(OBD_PROTOCOLS protocol = PROTO_AUTO);
    bool read(byte pid, int& result);
    void write(const char* s);
    // Acceso asincronizado API
    void setPID(byte pid);
    void applyPIDs();
    void loadData();
    uint16_t getData(byte pid, int& result);
    // GPS API
    bool gpsQuery(GPS_DATA* gpsdata);
    void gpsSetup(uint32_t baudrate, const char* cmds = 0);
protected:
    bool sendCommand(byte cmd, uint8_t data = 0, byte* payload = 0, byte payloadBytes
= 0);
    byte receive(char* buffer, int timeout = OBD_TIMEOUT_SHORT);
    byte m_addr;
    PID_INFO obdInfo[MAX_PIDS];
    byte obdPid[MAX_PIDS];
};
```

Anexo 6 Pruebas de funcionamiento en otros vehículos

Figura a. Mitsubishi ASX

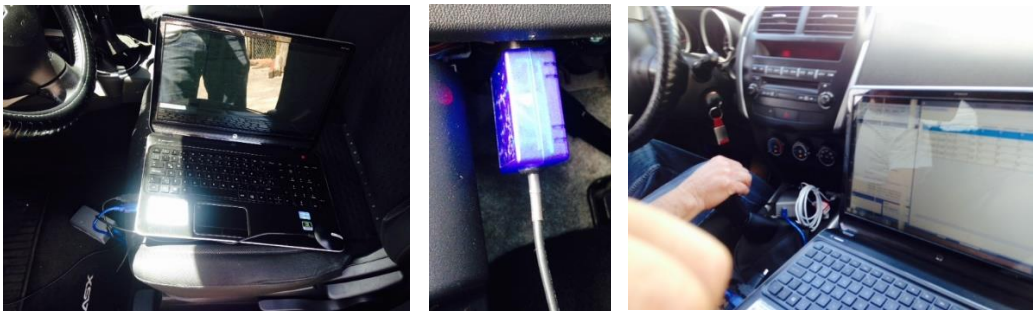


Figura b. Mitsubishi ASX 2

El video de las pruebas realizadas con este vehículo se lo puede observar desde el siguiente enlace: http://youtu.be/a5LzK8y9d2Y

Además de los siguientes vehículos:

Figura c. Chevrolet Spark y Daewoo Matiz