UNIVERSIDAD DE CARABOBO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA EN TELECOMUNICACIONES
DEPARTAMENTO DE SEÑALES Y SISTEMAS

DESARROLLO DE UN SISTEMA DE TELEMETRÍA PARA LOS PROTOTIPOS BAJA DE LA ORGANIZACIÓN SAE UC, EMULANDO UN INTÉRPRETE DE OBD A RS232

AARÓN RODRÍGUEZ
GABRIEL CASTRO

Bárbula, 15 de Julio del 2015

UNIVERSIDAD DE CARABOBO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA EN TELECOMUNICACIONES
DEPARTAMENTO DE SEÑALES Y SISTEMAS

DESARROLLO DE UN SISTEMA DE TELEMETRÍA PARA LOS PROTOTIPOS BAJA DE LA ORGANIZACIÓN SAE UC, EMULANDO UN INTÉRPRETE DE OBD A RS232

TRABAJO ESPECIAL DE GRADO PRESENTADO ANTE LA ILUSTRE UNIVERSIDAD DE CARABOBO PARA OPTAR AL TÍTULO DE INGENIERO DE TELECOMUNICACIONES

AARÓN RODRÍGUEZ
GABRIEL CASTRO

Bárbula, 15 de Julio del 2015

UNIVERSIDAD DE CARABOBO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA EN TELECOMUNICACIONES
DEPARTAMENTO DE SEÑALES Y SISTEMAS

AVAL DEL TUTOR

Quien suscribe EDUARDO GONZÁLEZ, titular de la cédula de identidad 11.149.218, en mi carácter de TUTOR del Trabajo Especial de Grado titulado:

DESARROLLO DE UN SISTEMA DE TELEMETRÍA PARA LOS PROTOTIPOS BAJA DE LA ORGANIZACIÓN SAE UC, EMULANDO UN INTÉRPRETE DE OBD A RS232

Y presentado por los bachilleres AARÓN RODRÍGUEZ, cédula de identidad 19.744.587, GABRIEL CASTRO, cédula de identidad 20.145.214, para optar al Título de Ingeniero de Telecomunicaciones, hago constar que dicho trabajo reúne los requisitos y méritos suficientes para ser sometido a la presentación pública y evaluación por parte del jurado examinador que se le designe

Firma
Prof. EDUARDO GONZÁLEZ
CI: 11.149.218
TUTOR

Bárbula, 15 de Julio del 2015

UNIVERSIDAD DE CARABOBO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA EN TELECOMUNICACIONES
DEPARTAMENTO DE SEÑALES Y SISTEMAS

CERTIFICADO DE APROBACIÓN

Los abajo firmantes miembros del jurado asignado para evaluar el trabajo especial de grado titulado "DESARROLLO DE UN SISTEMA DE TELEMETRÍA PARA LOS PROTOTIPOS BAJA DE LA ORGANIZACIÓN SAE UC, EMULANDO UN INTÉRPRETE DE OBD A RS232", realizado por los bachilleres AARÓN RODRÍGUEZ, cédula de identidad 19.744.587, GABRIEL CASTRO, cédula de identidad 20.145.214, hacemos constar que hemos revisado y aprobado dicho trabajo.

Firma
Prof. EDUARDO GONZÁLEZ
TUTOR

Firma
Prof. CARLOS MEJÍAS
JURADO

Firma
Prof. AHMAD OSMAN
JURADO

Bárbula, 15 de Julio del 2015

Dedicatoria

A mis padres Rebeca Díaz y Pedro Rodríguez; y a mis hermanos
Maitte Rodríguez, Liuxmila Rodríguez y Daniel Escalona
por apoyarme, amarme y ayudarme siempre.
A mi novia Gabriela Delgado por su cumpañía
y su apoyo incondicional, y a mis amigos.
A la escuela de telecomunicaciones de la UC y a la organización
SAE UC, que han sido como un hogar para mí.

AARÓN RODRÍGUEZ

A Dios por permitirme cerrar este ciclo y abrir otro.
A mis padres por ayudarme.
A mi novia por acompañarme siempre.

GABRIEL CASTRO

Agradecimientos

Agradecemos a dios por permitirnos realizar este proyecto.

A la Organización SAE UC por el apoyo brindado, y por permitir efectuar este trabajo especial de grado.

A nuestros familiares, profesores, compañeros y colaboradores que de alguna forma contribuyeron en el desarrollo de este proyecto.

A la escuela de telecomunicaciones por todos los conocomientos brindados y la experiencia gratificante de pertenecer a ella.

Índice general

Índice de figuras

Apéndices

Anexos

Indice de tablas

Apéndices

Acrónimos

GSM	**G**lobal **S**ystem for **M**obile comunications.
OBD	**O**n **B**oard **D**iagnostics.
RPM	**R**evolución **P**or **M**inuto.
SAE	**S**ocyety of **A**utomotive **E**ngineers.
UC	**U**niversidad de **C**arabobo.
USB	**U**niversidad **S**ímon **B**olívar.
ISO	**I**nternational **O**rganization for **S**tandardization.
ECU	**E**ngine **C**ontrol **U**nit.
LOS	**L**ine **O**f **S**ight.
ADC	**A**nalogic to **D**igital **C**onverter.
CAN	**C**ontrolled **A**rea **N**etwork.

DESARROLLO DE UN SISTEMA DE TELEMETRÍA PARA LOS PROTOTIPOS BAJA DE LA ORGANIZACIÓN SAE UC, EMULANDO UN INTÉRPRETE DE OBD A RS232

por

AARÓN RODRÍGUEZ y GABRIEL CASTRO

Presentado en el Departamento de Señales y Sistemas
de la Escuela de Ingeniería en Telecomunicaciones
el 15 de Julio del 2015 para optar al Título de
Ingeniero de Telecomunicaciones

RESUMEN

Desarrollo un sistema de telemetría aplicado en los prototipos monoplaza todo terreno fabricado por la Organización SAE UC Venezuela, emulando un ELM el cual estaría conectado a un sistema basado en el protocolo OBD para el formato e interpretación de los datos, este sistema puede enviar dichos datos mediante transmisión inalámbrica; con el fin de adquirir los datos pertinentes al desempeño del prototipo durante las pruebas y competiciones de interés. Los datos que se pueden adquirir son velocidad, aceleración, nivel de gasolina y RPM del motor.

Se procesan dichos datos y se envían de manera inalámbrica a una PC mediante un radio enlace de comunicaciones, para obtener en tiempo real dichas variables

y poder archivarlas de ser necesario. Además es tiene un módulo Bluetooth que admite la comunicación cercana para la adquisición de estos datos dentro del prototipo. Para dicho sistema se usaron sensores, microcontroladores y estándares de comunicaciones conocidos.

Palabras Claves: Telemetría, OBD, ELM

Tutor: EDUARDO GONZÁLEZ
Profesor del Departamento de Señales y Sistemas
Escuela de Telecomunicaciones. Facultad de Ingeniería

Capítulo I

Introducción

1.1. Motivación

La Organización SAE (Sociedad de Ingenieros Automotrices) UC Venezuela, es una organización sin fines de lucro y apolítica, que tiene como meta principal la representación de la Universidad de Carabobo en competencias colegiadas bajo SAE International, las cuales comprenden cuatro fases que son: diseño, construcción, competición y publicación de papers de un prototipo todo terreno [13].

El objetivo de la competición es probar el prototipo bajo una serie de condiciones dinámicas, las cuales ponen a prueba los distintos aspectos de diseño del vehículo. Cabe señalar, que se realiza una prueba llamada Endurance, que consiste en un circuito en el cual el prototipo es llevado al límite por un tiempo prolongado de 4 horas. Debido a esto, a medida que se desarrolla la competencia, es necesario conocer parámetros del prototipo en tiempo real, para mostrarlos al piloto y al equipo en los pits, los cuales podrían hacer un diagnóstico del problema que se presente y optimizar el tiempo de respuesta a este.

En las competencias, hay ocasiones en las que se presentan algunos inconvenientes, por ejemplo: es común que el prototipo haga paradas para recargar gasolina cuando no lo requiere; en ellas se pierde tiempo, el cual puede ser aprovechado.

Otro ejemplo es que, en algunas pruebas, se llega a depender de un automóvil comercial para poder realizarlas, tal es el caso de las pruebas de velocidad, en ellas el automóvil auxiliar debe alcanzar la velocidad del prototipo, ya que no cuenta con ningún sistema que realice mediciones ni registros directos de los parámetros del prototipo y mucho menos que se lo comunique al piloto e integrantes del equipo.

Esta organización ha participado en la categoría Baja SAE desde el año 2005 y ha competido 3 veces; en ellas hubo fases en las que no existen registros certeros y otras en las que no existe registro alguno del desempeño, lo que dificulta la tarea de optimizar el prototipo.

La falta de comunicación, visualización y registros de información de parámetros, son problemas que presentan otras organizaciones, trayendo como consecuencia el desarrollo de una serie de proyectos que intentan solventar algunos de los aspectos antes mencionados.

En 2009, Cárdenas y Márquez realizaron el trabajo de grado titulado «Desarrollo de un sistema de instrumentación para el proyecto Baja de la Organización SAE UC Venezuela» [14]. Según los autores se desarrolló un sistema de instrumentación para el proyecto Baja de la Organización SAE UC Venezuela con la finalidad de visualizar, en tiempo real, las siguientes variables: nivel de combustible, revoluciones del motor y velocidad del vehículo.

En este trabajo, se midieron algunas variables de interés mediante el uso de un microcontrolador que recibía y procesaba las señales otorgadas por los instrumentos de medición para luego mostrarse en el tablero, con el fin de optimizar el manejo del prototipo y su desempeño en las pruebas. Esta investigación tiene relación con el trabajo actual ya que se implementa la medición

En 2012, Marín realizó el trabajo de grado titulado «Diseño e implementación de un sistema embebido de telemetría para vehículos Baja SAE» [15]. Acá, se diseñó e implementó un sistema de adquisición de datos que puede ser instalado a bordo del vehículo Baja SAE USB y que permite monitorear, en tiempo real, las variables de interés así como guardar los datos para análisis futuros.

El proyecto antes descrito es de gran aporte en la propuesta actual ya que miden algunas de las variables en la etapa de prueba del prototipo, sirviendo como referencia a la hora de implementar sistemas de procesamiento y redes de comunicación de corto alcance.

En 2014, Ramos realizó el trabajo de grado titulado «Diseño de un sistema de monitoreo OBDII con comunicación GSM» [16]. En el cual se diseñó un sistema de comunicación mediante mensajes preestablecidos vía GSM para obtener un diagnóstico del sistema OBDII a la brevedad de las fallas.

Este último trabajo representa una referencia importante para la aplicación del protocolo, y la utilización de un sistema inalámbrico de comunicación.

Debido a la problemática planteada anteriormente, se desarrolla este sistema de telemetría que sirve para la evaluación del prototipo en las siguientes fases: retoques finales de construcción, pruebas pre-liminares, competición y rediseño.

Este sistema, permite dar información tanto al piloto como a los pits del estado en el que se encuentra el vehículo en tiempo real, permitiendo un ahorro de tiempo en las revisiones del prototipo en algunos casos.

Cabe señalar que, este desarrollo, puede ser utilizado también para el entrenamiento del piloto, este último podrá observar el desempeño en tiempo real del vehículo y así tener mejor control sobre los parámetros requeridos por la prueba que se ejecute en ese momento; además sirve como objeto de medición para la selección del piloto dentro de los integrantes de la organización al observar el rendimiento de los preseleccionados en las pruebas.

Este diseño sirve de referencia para sistemas de telemetría, sistemas de control de parámetros a distancia o para sistemas en los que se requiera dicha implementación así como para otros problemas de comunicación.

1.2. OBJETIVOS

1.2.1. Objetivo General

Desarrollar un sistema de telemetría de adquisición de datos para los prototipos Baja SAE UC

1.2.2. Objetivos Específicos

1.- Seleccionar el protocolo OBD y los protocolos de comunicación adecuados, para la transmisión de los datos.

2.- Seleccionar los dispositivos necesarios para la medición, el procesamiento y envío de los parámetros del prototipo.

3.- Realizar el diseño del sistema de telemetría, para la integración de todos los componentes y protocolos antes seleccionados.

4.- Implementar el diseño para la evaluación del desempeño del sistema.

1.3. ALCANCE

Se desarrollará un sistema de telemetría para el prototipo Baja SAE UC, con la finalidad de observar el desempeño de ciertos parámetros del vehículo en tiempo real, como la aceleración, velocidad, nivel de gasolina y revoluciones por minuto del motor, mediante la medición, transmisión y visualización de dichos parámetros. La visualización se podrá realizar desde cualquier PC que disponga del software compatible con los protocolos que se seleccionen por medio de conexión al emulador del ELM.

Capítulo II

Marco conceptual

2.1. Marco conceptual

2.1.1. Protocolos de comunicación OBD

Surge en un principio para controlar las emisiones de carbono que realizaban los vehículos automotrices comerciales, siendo desde la década de los 80 (en los EE.UU. y posteriormente en los otros países) un requerimiento indispensable para la fabricación de cualquier auto comercial. Este protocolo no se limita a la muestra de las emisiones de carbono, se puede implementar para diferentes parámetros, variables y características que se puedan tener en un vehículo.

Se han estandarizado más de un protocolo de comunicación OBD por distintos entes (principalmente SAE e ISO), creando así un conjunto de protocolos que varían en su forma de modulación, velocidad de transmisión y el voltaje máximo de la señal [16–18]. Normalmente se clasifican como OBDII para Estados Unidos, EOBD para Europa y JOBD para Japón.

2.1.2. Intérprete ELM

Los protocolos OBD no son entendibles directamente por las PC o dispositivos smart, los ELM son microcontroladores diseñados por la empresa ELM Electronics para interpretar estos protocolos y lograr una conexión mediante interfaz serial. Estos pueden detectar e interpretar automáticamente nueve tipos de protocolos OBD dependiendo del modelo de ELM, adicionalmente soportan comunicaciones de alta velocidad y un bajo consumo de potencia en modo de reposo.

Algunos son dispositivos configurables, adaptándolos a diferentes requerimientos mediante comandos AT. Los dispositivos más comunes de conexión para los sistemas que implementen algún ELM son USB y RS232 pero existen otros como tarjetas para PC, Dispositivos Ethernet, WiFi o Bluetooth [8].

Existen varios tipos de estos microcontroladores ELM entre ellos están ELM320, ELM322, ELM323, ELM325, ELM327 y ELM329; la diferencia entre ellos es el tipo de protocolo OBD que interpretan. Cabe destacar que, el ELM327, es el único dispositivo que tiene la capacidad de interpretar todos los protocolos OBD.

2.1.3. Módulos Xbee

Son dispositivos de conexión inalámbrica que emplean el protocolo Zigbee o Digimesh, basados en el estándar de comunicación para redes inalámbricas IEEE 802.15.4. Proveen dos formas de comunicación serial, modo AT y API. Los módulos pueden configurarse desde la PC con el software X-CTU o desde un microcontrolador y ser utilizados en redes punto a punto, punto a multipunto o en una red mesh. Existen adaptadores Xbee Explorer serial o Xbee Explorer USB, para una comunicación directa desde la PC.

2.1.3.1. Tipos

Existen dos series de módulos la serie 1 y la serie 2 o 2.5, estas series no son compatibles entre sí ya que poseen chipsets y protocolos diferentes.

La serie 1 está basada en el chipset Freescale y está pensado para ser utilizado en redes punto a punto y punto a multipunto. Los módulos de la serie 2 están basados en el chipset de Ember y están diseñados para ser utilizados en aplicaciones que requieren repetidores o una red mesh. Ambos módulos pueden ser utilizados en los modos AT y API [19].

2.1.4. XBee PRO 900HP

El módulo xbee utiliza una base de firmware de múltiples capas para ordenar el flujo de datos que transporta desde el host en la interfaz serial hasta la antena como punto final para los datos transferidos, dependiendo de la configuración de software y hardware que se elija [1].

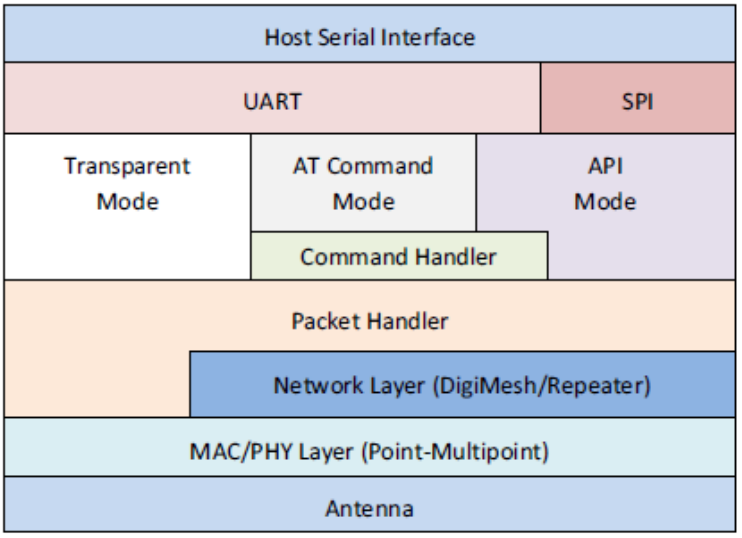


Figura 2.1: Diseño operacional básico del XBee [1].

En la figura 2.1 se muestra este diagrama de múltiples capas y se observa que existen tres modos de operación.

2.1.4.1. Modo comando AT

Este modo permite ingresar comandos AT al módulo para configurar o ajustar parámetros como el modo de operación, modo de sleep, dirección propia, entre

otros. Para poder ingresar estos comandos AT es necesario usar algún programa de emulación de terminal, como Hyperterminal en windows, el programa X-CTU de configuración de los módulos XBee que es gratuito o cualquier microcontrolador que maneje UART. El command Handler, es el código que procesa los comandos del modo comando AT.

Es importante mencionar que, todos los módulos XBee, tienen por defecto una configuración de velocidad en la interfaz serial de 9600 bps.

2.1.4.2. Modo transparente y modo API

En modo transparente, los datos que ingresen por el pin 3 (DATA in) son transmitidos vía RF y todo lo que ingresa como paquete RF es enviado al pin 2 (DATA out). Este modo transparente, generalmente, es usado en redes punto a punto y es el que viene por defecto en todos los módulos.

El modo API es más complejo, permite el uso de tramas con cabeceras que aseguran la entrega de los datos como en el protocolo TCP y métodos de detección de errores.

2.1.4.3. Comunicación serial

Los módulos XBee, se comunican con un host mediante el puerto serial a través de UART o SPI. La figura 2.2 representa como los dispositivos que tengan una interfaz UART se conectan con los pines del módulo RF. Los datos entran al módulo por el pin 3 (DIN) como una señal asíncrona.

Cada byte de datos consiste en un bit de inicio (bajo), 8 bits de datos (menos significativos primero) y un bit de parada (alto), como lo muestra el patrón de bits de la figura 2.3.

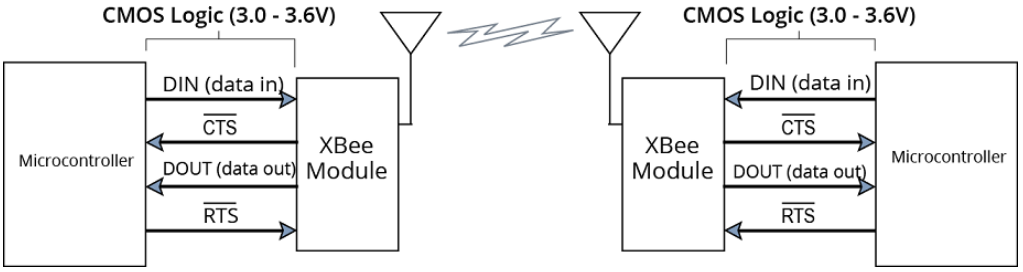


Figura 2.2: Diagrama de flujo de datos en un entorno UART [1].

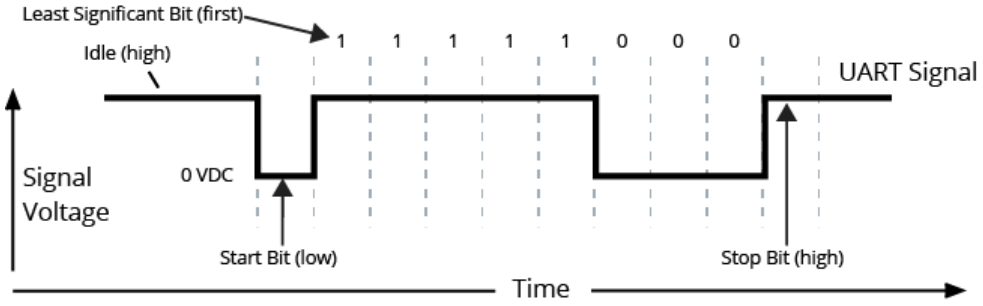


Figura 2.3: Paquete de datos UART que es transmitido al módulo RF [1].

2.1.4.4. Características del XBee PRO 900HP

La Tabla 2.1 muestra las características más resaltantes del módulo XBee PRO 900HP, estas pueden verse de forma más completa en datasheet del dispositivo [20].

2.1.5. Protocolo Digimesh

Protocolo desarrollado por Digi international, es una topología de punto a punto tipo malla, creado para soluciones de conectividad inalámbrica. Proporciona una arquitectura con funciones de red avanzada de fácil aplicación, además de proporcionar soporte de sleep para los routers con la finalidad de ahorrar batería y se puede implementar en gran cantidad de dispositivos creados por Digi [21].

Digimesh tiene un solo tipo de nodo, es decir, como una red homogénea, todos los nodos pueden enrutar datos entre sí, esto permite una configuración simple

Tabla 2.1: Características del módulo XBee PRO 900HP Programable.

Hardware	
Procesador	ADF7023 transceiver, Cortex-M3 EFM32G230 @28 MHz
Frecuencia	902 a 928 MHz, canal seleccionable desde el software
Antena	Wire, U.FL y RPSMA
Performance	
Velocidad de datos RF	10 kbps o 200 kbps
Rango Interno	10 kbps @ 610 m; 200 kbps @ 305 m
Rango Externo	10 kbps @ 15,5 km; 200 kbps @ 6,5km con dipolo de 2,1 dB
Potencia Tx	Hasta 24 dBm (250 mW) seleccionable
Sensibilidad	-101 dBm @200 kbps; -110 dBm @ 10 kbps
Rendimiento	
Interfaz de Datos	UART(3V), SPI
GPIO	15 I/O digitales, 4 entradas 10-bit ADC, 2 entradas PWM
Topologias de red	Digimesh, punto a punto, punto a multipunto, peer to peer

de la red y una flexibilidad para expandirla; además de una mayor fiabilidad en entornos donde los routers pueden aparecer o desaparecer por daño o interferencia, la topología se puede apreciar en la figura 2.4 [2].

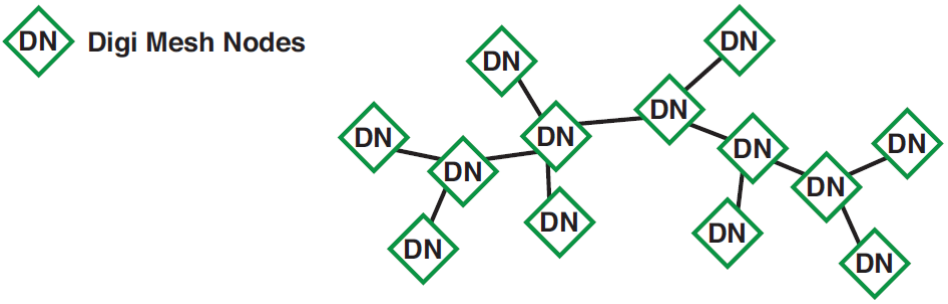


Figura 2.4: Diagrama de una red Mesh [2].

En cuanto a las características de la transmisión permite alcanzar distancias largas hasta 64 km, la taza de transmisión puede alcanzar hasta 256 Kbps dependiendo del dispositivo, el rendimiento aumenta para aplicaciones que requieren grandes bloques de datos, las frecuencias soportadas son 900 MHz para tazas de 10, 125, 150 y 200 Kbps y a 24 GHz para 256 Kbps, se usa el estándar de encriptación avanzada AES que permite la autentificación y encriptación de las comunicaciones, además de tolerancia a interferencia por medio de la modulación Frequency Hopping Spread Spectrum (FHSS).

2.1.6. Sensores

Son dispositivos que miden magnitudes físicas en una escala determinada mediante algún método de medición, teniendo como resultado una magnitud con respecto a una unidad de medición [22].

Normalmente, miden la unidad física respectiva y la convierten en una señal eléctrica, correspondiente a dicha magnitud la cual se procesa y, en algunos casos, se almacena. Hay una gran diversidad de sensores los cuales pueden medir longitud, peso, tiempo, corriente, temperatura, entre otros.

Debido a que son elementos que se encargan de un proceso de estimación bajo una unidad estándar, poseen un grado de incertidumbre o tolerancia como cualquier otro instrumento de medición, el cual tiene que ser tomado en cuenta para el cálculo de los errores. Los sensores se pueden clasificar en dos grupos con respecto a su salida, la cual puede ser analógica o digital. Normalmente, los sensores digitales tienen una salida que es representada en saltos de voltaje o mediante datos enviados en una trama, además poseen una salida más exacta y de mayor confiabilidad que los de salida analógica [23].

2.1.7. Revoluciones por minuto y motor Briggs&Stratton

Las revoluciones en este contexto representan las vueltas del eje final del motor. Este parámetro es de importancia ya que, a diferentes RPM se obtienen diferentes

valores de torque y potencia, los cuales son críticos para algunas de las pruebas que se presentan en este tipo de competiciones, como son la categoría de subida de pendiente y arrastre de peso muerto. El comportamiento de la relación entre las RPM y el torque del motor Briggs&Stratton se muestra en la figura 2.5.

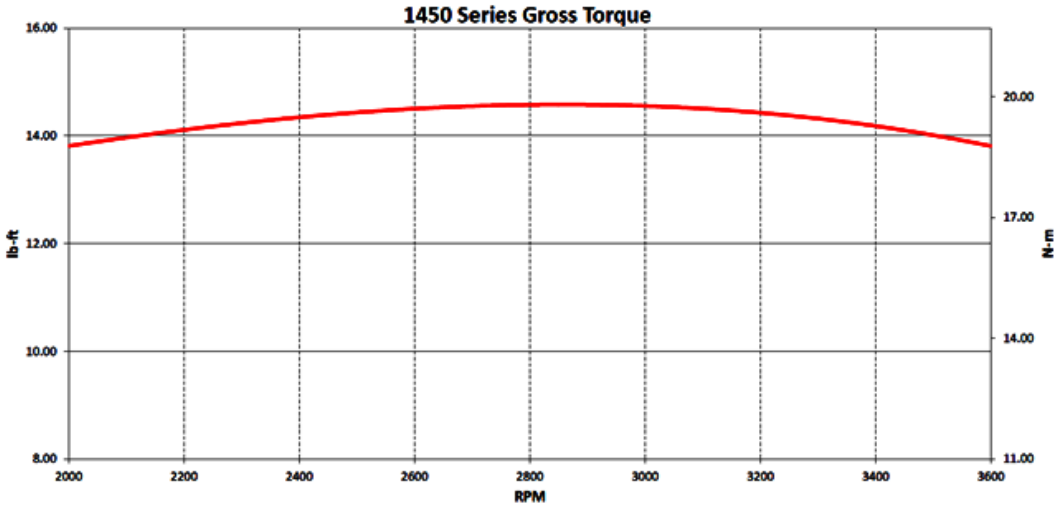


Figura 2.5: Torque versus RPM [3].

Teniendo conocimiento de las RPM se puede obtener un mejor desempeño del prototipo en algunas de las pruebas antes mencionadas, además podría resultar útil si se desarrolla un prototipo de cambio manuales o sincrónico en la organización, ya que en función de la cantidad de revoluciones se pueden ejecutar los cambios con mayor rendimiento en el desplazamiento del prototipo.

Los motores reglamentarios para dichas competencias, hasta la fecha, son los modelos 20S232 0036-F1, 205432 0536-E9, 205332 0536-E9 y 205332 0536-B1, de la marca Briggs&Stratton[24]. Tienen un mecanismo de arranque manual tipo podadora, en el cual se hala de una cuerda enrollada en la polea del cigüeñal para hacer girar este, logrando así que el cigüeñal salga de la inercia y al alcanzar cierta energía se convierte en un sistema estable, ya que el movimiento rotacional genera mediante magnetos una chispa, que produce la combustión y hace posible que se mantenga en marcha el motor.

Para detener dicho motor, es necesario cortocircuitar el voltaje generado de los magnetos, antes de que pase por el transformador que eleva este voltaje (aproximadamente a 20kV). El voltaje a cortocircuitar es entre 3 y 5V y el cortocircuito se hace entre el chasis del motor y un puerto que tiene disponible el motor diseñado para este fin.

2.1.8. Sistemas de adquisición de datos

En este contexto se hará referencia de adquisición de datos a todo el proceso involucrado en la adquisición de magnitudes físicas, medidas tanto de forma directa como indirecta y al procesamiento de los datos obtenidos de dicha medición, hasta que se consideren apropiados para su envío y muestra en las distintas interfaces gráficas.

Dichos sistemas deben poseer de manera intrínseca y prácticamente obligatoria, un sistema de conversión analógica/digital de gran precisión y resolución, una buena tasa de muestreo, debe considerarse la amplificación correcta. En fin, debe poseer etapas del sistema en la cuales la señal se adecue para su procesamiento con una mayor confiabilidad del mismo [25].

2.1.9. Pérdidas en un radio enlace

La propagación en los servicios móviles terrestres resulta afectada en diversos grados por la topografía, la vegetación, las estructuras artificiales, las constantes del terreno, la troposfera y la ionosfera, que generan una diferencia de la potencia en los terminales de la antena receptora en comparación con la potencia de salida en los terminales de la antena transmisora [26].

2.1.9.1. Perdidas básicas del espacio libre para un enlace punto a punto

Cuando se trata de un enlace punto a punto, como el de este caso, es preferible calcular la atenuación en el espacio libre entre antenas isótropas, denominadas también perdidas básicas de transmisión en el espacio libre con la siguiente ecuación [27]:

$$L_{bf} = 20\text{Log}\left(\frac{4\pi d}{\lambda}\right) \qquad \text{dB} \tag{2.1}$$

Donde:

L_{bf}: Pérdidas básicas de transmisión en el espacio libre [dB].

d: Distancia.

λ: Longitud de onda, expresada en las mismas unidades que la distancia.

La ecuación anterior también puede presentarse en función de la frecuencia:

$$L_{bf} = 32,4 + 20\text{Log}\,(f) + 20\text{Log}\,(d) \qquad \text{dB} \tag{2.2}$$

Donde:

f: Frecuencia [MHz].

d: Distancia [km].

La cual resulta más conveniente para el estudio de radioenlaces, ya que se trabaja en función de la frecuencia.

2.1.9.2. Pérdidas del sistema (de un enlace radioeléctrico)

La pérdida básica de un radio enlace respecto a las pérdidas relativas a la propagación en el espacio libre [28], se puede expresar del siguiente modo:

$$L_s = W_{tx} - W_{rx} = L_{bf} - G_t - G_r + L_m + L_{tc} + L_{rc} \qquad \text{dB} \qquad (2.3)$$

Donde:

W_{tx}: Potencia de salida en el transmisor [dB].

W_{rx}: Potencia de entrada en el receptor [dB].

L_s: Pérdida del radio enlace [dB].

G_t: Ganancia de la antena transmisora [dB].

G_r: Ganancia de la antena receptora [dB].

L_{bf}: Pérdidas por espacio libre [dB].

L_m: Pérdidas por absorción, difracción, entre otros [dB].

L_{tc}: Pérdidas en el circuito transmisor [dB].

L_{rc}: Pérdidas en el circuito receptor [dB].

Dicha fórmula contiene, de manera implícita, las pérdidas que se presentan por el entorno donde viaja la onda (Lm), estas pérdidas son las que se deben a la absorción por humedad, por lluvia, por multitrayecto, entre otros.

2.1.9.3. Elipsoides de Fresnel y zonas de Fresnel

Al estudiar los enlaces de ondas radioeléctricas entre dos puntos, se subdivide el espacio en una familia de elipsoides, llamados elipsoides de Fresnel, los cuales son usados para calcular los efectos de pérdidas por reflexión y difracción. Las primeras tres zonas son las que tienen la mayor parte del peso en la propagación [4].

El radio de un elipsoide de Fresnel de un punto entre el transmisor y el receptor viene dado por:

$$R_n = 550 \left[\frac{n d_1 d_2}{(d_1 + d_2) f} \right]^{1/2} \tag{2.4}$$

Donde:

f: es la frecuencia (MHz)

d_1 y d_2: son las distancias desde el transmisor y desde el receptor al punto en que se evalúa el elipsoide (km)

n: corresponde al número entero que caracteriza al elipsoide, siendo n=1 el primer elipsoide de Fresnel.

2.1.9.4. Modelo de pérdidas por difracción con un obstáculo filo de cuchillo

Existen trayectos de propagación, que comprenden un obstáculo o muchos obstáculos separados. Interesa calcular las pérdidas que estos producen, estos cálculos se hacen posibles idealizando las geometrías del obstáculo. En la realidad, los obstáculos tienen formas más complejas por lo que estas ecuaciones son una aproximación donde se obtienen valores certeros relativos a las pérdidas reales, teniéndose la modelación más sencilla de estos casos que es la atenuación por difracción de obstáculo en filo de cuchillo.

Estas se ocasionan cuando ocurre una obstrucción total o parcial (a partir de un 40 %) del radio del elipsoide principal de Fresnel, por un objeto idealizado como el filo de un cuchillo [4], tal como se muestra en la figura 2.6.

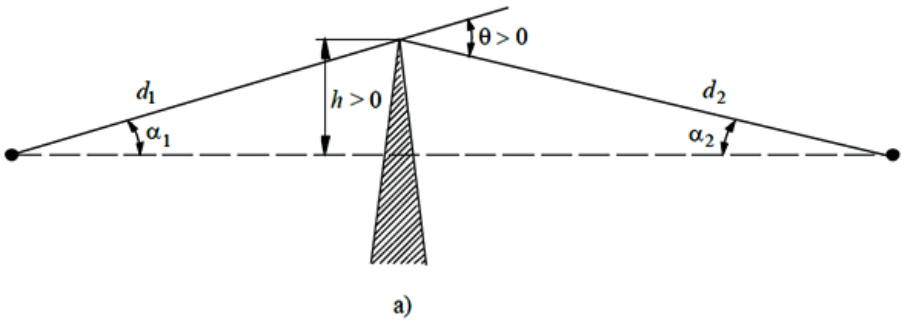


Figura 2.6: Ejemplo de modelo de filo de cuchillo [4].

En este caso, todos los parámetros geométricos del perfil radioeléctrico se agrupan en la siguiente ecuación:

$$\nu = h\sqrt{\frac{2}{\lambda}\left(\frac{1}{d_1}+\frac{1}{d_2}\right)} \tag{2.5}$$

Donde:

ν: parámetros geométricos [adim].

h: altura de la cima del obstáculo sobre la recta que une los dos extremos del trayecto. Si la cima queda por debajo de esa línea, h es negativa [m].

λ: longitud de onda [m].

d_1 y d_2: distancias desde los dos extremos del trayecto a la cima del obstáculo [m].

Luego, la atenuación por filo de cuchillo cuando ν es superior a -0.78 se expresa finalmente como:

$$J(\nu) = 6,9 + 20\,\mathrm{Log}(\sqrt{(\nu-0,1)^2+1}+\nu-0,1) \quad \mathrm{dB} \tag{2.6}$$

O también se puede obtener mediante gráfica 2.7, la cual sirve como una guía directa para determinar la pérdida dado un valor de ν.

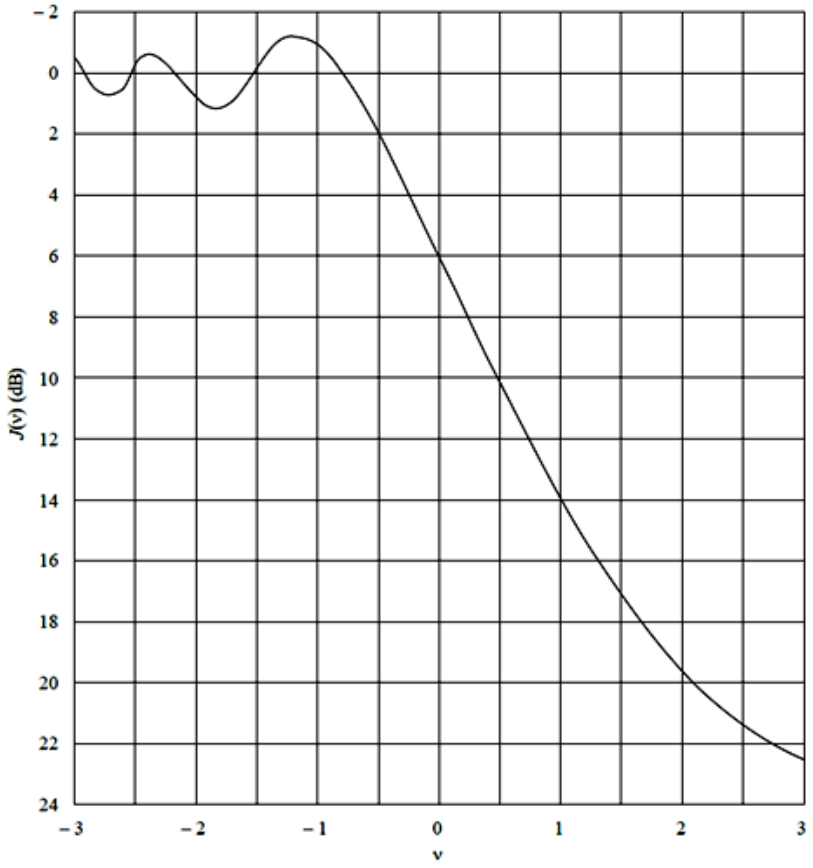


Figura 2.7: Pérdida por difracción en una arista filo de cuchillo [4].

2.1.9.5. Método computacional de predicción de pérdidas de radio transmisiones troposférica sobre terreno irregular (Longley-Rice)

Llamado también Modelo de Terreno Irregular (ITM por sus siglas en ingles), es un modelo aplicable para radio frecuencias por encima de los 20 MHz, este puede determinar un aproximado de las pérdidas mediante la información de los perfiles detallados del terreno o con los perfiles que son representativos de las características medias del terreno para un área dada. Para enlaces con línea de vista, el cálculo se realiza mediante la teoría de dos rayos (modelo de tierra plana) y un valor extrapolado de la atenuación por difracción. En el caso de enlaces que se extienden más allá del horizonte, el valor de referencia es mediante la atenuación por difracción o

por la atenuación por dispersión, el que sea menor entre estos [29].

Este método de predicción, fue diseñado para ser usado mediante una computadora digital, otorgando un estimado de la perdida de transmisión esperada dentro del rango de validez del método mostrado en la Tabla 2.2. Existen programas y aplicaciones online que usan este método como predicción de pérdidas en un radio enlaces, entre los programas se encuentran Radio Mobile y SPLAT!, que son los más conocidos.

Tabla 2.2: Rango de validez del modelo Longley-Rice

Parámetro	Rango
Frecuencia	20 - 40.000 MHz
. Altura de las antenas	0.5 - 3.000 m
Distancia	1 - 2.000 km
Refractividad de la superficie	250 - 400 N-unidades

2.1.10. Microcontrolador

Son dispositivos de procesamiento que se asemejan a una computadora aunque de un tamaño mucho menor, ya que constan de una memoria de programa, memoria RAM, puertos de entrada y salida, temporizadores, en algunos casos convertidores A/D y comparadores USART (Universal Synchronous/Asynchronous Receiver/Transmitter).

El microcontrolador es capaz de decidir todos los procesos de un circuito electrónico, basándose en las instrucciones de un programa de rutina específica para cumplir un propósito de control [30].

2.2. Definiciones básicas

Firmware: Conjunto de instrucciones de un programa guardado en una memoria, estas fijan la lógica de bajo nivel de algún dispositivo [31].

Isotropía: Característica de los cuerpos cuyas propiedades físicas no dependen de la dirección [31].

Capítulo III

Procedimiento metodológico.

Esta investigación se estructuró en tres partes principales, las cuales se ejecutaron en el siguiente orden.

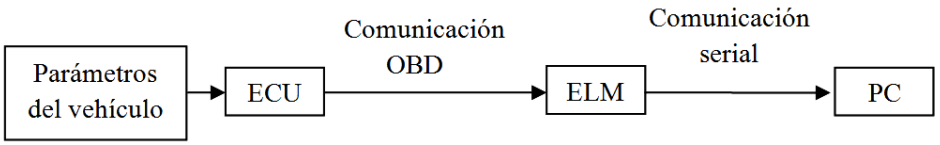
3.1. Selección de elementos que integran el sistema de telemetría.

El sistema de telemetría dispuso de dispositivos y protocolos necesarios para lograr los objetivos planteados. La selección de cada dispositivo dependió de la comparación entre distintos modelos que cumplieran la misma función. En cuanto a los protocolos, la selección se basó en la comparación de estándares. Estos se explican a continuación.

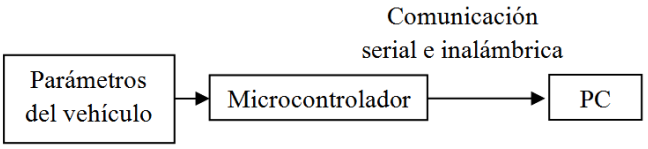
3.1.1. Selección del microcontrolador ELM que se emulará, el protocolo OBD y los protocolos de comunicación.

Se realizó un proceso de investigación y selección de los dispositivos a emular y protocolos a usar. Como base para esto, se consideraron las características de ellos que más se adaptaron a los requerimientos planteados en el proyecto, tomando en

cuenta que la implementación consiste en sustituir la ECU del automóvil, el microcontrolador ELM y la comunicación OBD por un microcontrolador que emule el comportamiento que tendrían estos, y que además recopile y envíe toda la información a la PC, como se muestra en la figura 3.1.



(a) Sistema convencional ECU-PC



(b) Sistema a implementar

Figura 3.1: Sistema convencional y sistema a implementar de adquisición de datos del vehículo.

En ambos sistemas, los datos enviados a la PC con respecto a los parámetros del vehículo deben mantener el mismo formato, dichos datos deben cumplir con los parámetros de la capa de aplicación y de direccionamiento de la capa de red del protocolo OBD seleccionado. Sin importar qué protocolo OBD se use, las especificaciones de la capa de aplicación son las mismas, mientras que para cada protocolo el direccionamiento cambia, por lo tanto la selección del protocolo dependerá de esto.

La finalidad de realizar este sistema, fue usar algún software basado en el protocolo OBD para la visualización de los parámetros, el envío de la información de forma inalámbrica, así como componentes adicionales y simplificar el sistema.

3.1.1.1. Selección del ELM y protocolo OBD a emular.

La selección del microcontrolador intérprete a emular dentro de los distintos modelos de la serie de ELM Electronics, se basó en una tabla comparativa, que contempla los distintos protocolos OBD de cada modelo que el ELM interpreta. La

emulación consta del funcionamiento básico, para ejecutar y mantener una conexión, respondiendo a las solicitudes que se le realicen en cuanto a la transmisión del estado de los parámetros.

En el diseño del sistema de telemetría no bastó sólo con emular el ELM, por lo que se tuvo que emular la respuesta a una comunicación OBD que tendría con la ECU del automóvil.

Luego de haber escogido el ELM, se seleccionó un protocolo OBD compatible con este. Así mismo se estudiaron las tramas de los distintos protocolos OBD, tomando en consideración el direccionamiento más sencillo para la emulación de estas, mediante códigos de programación.

3.1.1.2. Selección de los protocolos de transmisión inalámbrica.

La selección del protocolo para la transmisión inalámbrica remota, se basó en considerar la mínima atenuación calculada teóricamente en las frecuencias de bandas libres, para el radioenlace en estas bandas. A partir de esto, se optó por usar uno de estos protocolos. Se consideraron protocolos comerciales para la facilidad de adquisición de equipos y la versatilidad en el uso de los mismos.

Para el cálculo teórico de la atenuación, se consideró una obstrucción por filo de cuchillo de 15 m sobre el nivel de la línea de vista en el medio del radioenlace, caso que puede presentarse en dichas competencias. Teniendo una distancia entre las antenas de 2 km, se deja un rango de holgura, ya que en las competencias normalmente el radioenlace sería más corto y estaría más despejado.

En cuanto a la comunicación dentro del prototipo, se consideraron protocólos de uso comercial para dispositívos móviles, que fueran fácil de configurar y tuvieran una topología sencilla.

3.1.2. Selección de métodos y dispositivos de adquisición de los parámetros.

Durante esta etapa, se realizó una investigación de los proyectos previamente implementados en los prototipos BAJA, con la finalidad de analizar los métodos usados en la adquisición de parámetros, y en algunos casos, se consideraron otros métodos de medición. Posteriormente, se estudiaron las reglas de la competencia en cuanto a las modificaciones permitidas al prototipo, para escoger los dispositivos necesarios y realizar el diseño del sistema de adquisición de cada parámetro, cumpliendo estas reglas.

Nivel de gasolina:

Debido a que las reglas de la organización son estrictas en cuanto al tanque de combustible y la tapa, estos no pueden ser modificados, ni realizar algún tipo de abertura a los mismos, por lo que se consideró usar un método de medición remoto, mediante la medición de la presión del líquido en la salida del tanque o mediante la altura de un flotador que contenga algún dispositivo pasivo que pueda detectarse desde las afueras del tanque. Para determinar la altura del flotador se estudiaron dos métodos, la medición de la magnitud del campo magnético de un imán puesto en el flotador (efecto Hall) y el tiempo de rebote de una onda electromagnética en el elemento pasivo.

Luego de escoger el método más sencillo a implementarlo, se seleccionó un sensor que implementara este método, que cumpliera con los requerimientos de la medición y que fuera de fácil adquisición.

Revoluciones por minuto (RPM):

Se utilizó el método de adquisición del proyecto anterior, en el cual se midió este parámetro por medio de la señal de voltaje que genera las chispas en la bujía, en la que cada chispa representa una vuelta del eje del motor. Usando el puerto que

cortocircuita la bujía para detener el motor, se puede obtener la forma de onda de este voltaje, para luego analizarla y procesarla, determinando así el período de la señal, y por ende, el conteo de las revoluciones por minuto.

En primera instancia, hay que considerar la forma de onda para determinar cómo se calcularía el período de la señal, el cual corresponde a las chispas generadas en la bujía, y por ende a las RPM. Se sabe, por el trabajo de grado anterior [14], que esta señal tiene componentes negativas y positivas, además de ser muy rizada. Por lo que se necesita de un circuito que procese esta señal, y que tenga como salida una forma de onda más adecuada para la lectura y cumpla con los valores de entrada del microcontrolador.

Velocidad:

La forma escogida para obtener este parámetro, fue mediante la velocidad angular del eje final del prototipo, el cual está directamente conectado a las ruedas traseras. Se decidió usar un método, que pueda determinar el número de vueltas o desplazamientos angulares del disco de freno en función del tiempo, por medio de un dispositivo que detecte el paso de las partes de este, o el paso de algún material dispuesto en este disco, siendo posible calcular la velocidad angular de las ruedas, y sabiendo el tamaño de las mismas, se pudo conocer la velocidad del prototipo.

Luego de establecer el método de medición, se decidió adquirir el mismo sensor que se usó en la medición del nivel de gasolina.

Aceleración:

Este parámetro se obtendrá mediante el uso de un acelerómetro ya que se consideró el método de adquisición más sencillo. Se tomaron en cuenta varios modelos en una tabla comparativa escogiendo el dispositivo que tenga las mejores características entre potencia consumida, sensibilidad, fácil lectura mediante la plataforma Arduino y facilidad de adquisición en el mercado.

3.1.3. Módulos inalámbricos:

Luego de haber definido los protocolos de comunicación inalámbrica, se realizó una preselección de los módulos de comunicación que implementan dichos protocolos. Luego se consideraron varios modelos en una tabla comparativa y se estudiaron con respecto a su potencia, sensibilidad, velocidad de transmisión y costo; que cumpliera además con un margen de seguridad de 10dB en el radioenlace planteado al momento de la selección. Para el módulo de comunicación dentro del prototipo, el estudio se centró en la practicidad de configuración y de uso del mismo.

3.1.4. Microcontrolador:

Se realizó una tabla comparativa con distintas marcas de microcontroladores, se tomaron en cuenta los que tuvieran entradas tanto digitales como analógicas, una velocidad de procesamiento adecuada para que la actualización de datos sea lo más cercana posible a tiempo real, la compatibilidad de forma práctica con los módulos de comunicación escogidos, sencillez en la programación del mismo y un bajo costo. El microcontrolador escogido fue el Arduino, el cual posee su propia interfaz de programación.

Selección de recursos adicionales:

Además de los dispositivos primarios, se seleccionaron otros componentes necesarios para la interconexión y correcto funcionamiento de todo el sistema. Adicionalmente, se escogieron los software para la programación de rutinas, configuración, prueba de los dispositivos escogidos y simulaciones de los códigos desarrollados.

La selección de estos dispositivos se basó en considerar la manera más práctica para cada interconexión y el cumplimiento de requerimientos que se presentaron en cada caso. Con respecto a los software, algunos ya estaban definidos por la elección

del hardware (como el Arduino), y el resto se definieron en base a las necesidades que se presentaron, los cuales se muestran a continuación.

Radio Mobile:

Es un software gratuito diseñado por Roger Coudé, que utiliza datos digitales de la elevación del terreno (SRTM) para generar un perfil del trayecto entre el emisor y el receptor, que junto a características del entorno, información ambiental y estadística, usa el modelo de propagación de ondas de radio ITM (Irregular Terrain Model), basado en el algoritmo Longley-Rice para determinar el área de cobertura de sistemas de radiocomunicaciones, que trabajen con frecuencias comprendidas entre 20 MHz a 20 GHz. Este software se utilizó para las simulaciones del radioenlace.

Wireless Proto Shield:

Este dispositivo permitió que el Arduino se comunique vía inalámbrica, diseñado especialmente para dispositivos Xbee del fabricante Digi. Puede usarse para que el Xbee se comunique con el Arduino para la transmisión o la PC se comunique con el Arduino para la programación, este dispositivo se muestra en la figura 3.2.

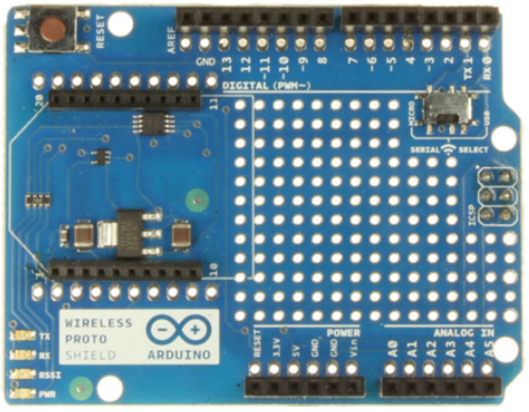


Figura 3.2: Dispositivo Wireless Proto Shield [5].

Se escogió este dispositivo debido a que es una solución práctica para conectar el Arduino con el Xbee sin mayores componentes adicionales, además posee una zona en la cual se usó como protoboard para interconectar los distintos componentes del montaje de los circuitos y conexiones adicionales

SaintSmart XBee USB Adapter for Arduino UNO MEGA R3:

Construido por la compañía SaintSmart, está diseñado especialmente para conectar de manera directa cualquier dispositivo Xbee de Digi a un enlace USB, con el cual se puede acceder a los pines seriales y de programación. Este dispositivo se muestra en la figura 3.3.

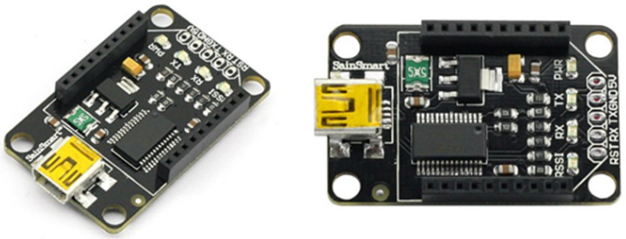


Figura 3.3: Dispositivo adaptador XBee a USB [6].

Por la practicidad de este *shield* para conectar un módulo Xbee con una PC, se escogió para la etapa receptora la cual estuvo conectada directamente a una PC vía USB, para la interpretación de datos en los pits.

Software Arduino 1.0.6:

Es un entorno escrito en *Java*, basado en *Processing* y *avr-gcc*. Con versiones para Windows, MAC OS X y Linux. Básicamente es el compilador que se usa para programar el Arduino mediante el código que se estructure por el usuario de acuerdo a sus requerimientos.

3.2. Diseño del sistema de telemetría:

Esta sección se dividió en 3 partes que constan en definir todo el proceso de adquisición de datos, la emulación del ELM en conjunto con el protocolo OBD y la simulación del radioenlace. Adicionalmente se desarrollarán todos los procedimientos y acciones necesarias para implementar el sistema de telemetría.

3.2.1. Descripción de los sistemas de adquisición de parámetros.

Para esta etapa se definieron diagramas de funcionamiento y las estructuras de programación como una rutina individual para cada parámetro, luego todo esto se integrará de forma global para la implementación.

Luego de realizar cada rutina, se simuló el comportamiento tanto de la señal de la variable a obtener, como del código realizado para corroborar el correcto funcionamiento y realizar las correcciones pertinentes antes de pasar a la etapa de implementación.

El proceso de simulación fue el siguiente para todas las subrutinas simuladas: luego de haber estructurado todo el código con el programa Arduino 1.0.5-r2 se verifica el *sketch*, mediante una opción de verificación la cual compila el código y verifica si hay errores de formato en las instrucciones, al verificarse se crea una carpeta en la ubicación `C:\Users\"Usuario"\AppData\Local\Temp\build##########.tmp` , la cual contiene el código compilado en formato hexadecimal "`nombre del skectch.ccp.hex`" y en formato `.elf`, "`nombre del skectch.ccp.elf`". Dichos archivos son utilizados para programar el Arduino virtual. Al localizarse dichos archivos, se pueden seleccionar como archivos de programación cuando se intentan cambiar las características del componente del Arduino digital.

Cabe destacar que para observar el valor de las variables que se utilizan en el código y para un proceso más sencillo de depuración, se debe usar el archivo con extensión `.elf`. Luego de que se han hecho todos estos pasos se procede con la simulación.

3.2.1.1. Obtención del nivel de gasolina.

Conociendo la salida del sensor, la cual es una salida analógica de voltaje que tiene una variación en función del nivel de líquido, se puede definir la estructura de la rutina que obtendrá el nivel de gasolina dentro del tanque de combustible.

En este caso, cada rango de voltaje leído representará un rango del nivel de líquido dentro del tanque, por lo que la función de la rutina principalmente será leer el voltaje para luego determinar el nivel de líquido correspondiente. La estructura seguída para realizar esta rutina es la mostrada en la figura 3.4.

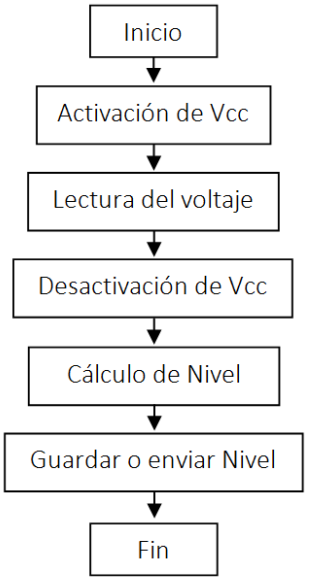


Figura 3.4: Flujograma de la rutina de programación.

Cabe destacar que en la implementación es necesario conocer la relación real entre el nivel de gasolina y la salida del sensor. Por lo que se debe realizar pruebas con diferentes niveles de líquido a la hora de implementar para obtener el valor de voltaje a la salida del sensor, que es necesario en la programación para determinar precisamente el nivel del líquido.

Luego de obtener la relación entre la salida del sensor y el nivel del líquido, se debe proceder a establecer la sensibilidad de la medición con intervalos que mejor

se adapten a la relación obtenida.

3.2.1.2. Obtención de velocidad.

Luego de haber escogido el método de adquisición, se definen la estructura y los cálculos utilizados en la rutina del programa. Dicho montaje, se vale del movimiento angular del disco de freno el cual genera un tren de pulsos para calcular la velocidad, las características de estos pulsos dependen de la velocidad angular del eje final y las distancias entre los imanes dispuestos en el disco de freno.

En primer lugar, se define el cálculo asociado a la rutina de programación, el cual determina la velocidad mediante la medición del tiempo entre pulsos, para luego aplicar la fórmula que se presenta en el análisis siguiente.

Midiendo el tiempo entre el inicio de dos pulsos consecutivos o el final de estos, se puede determinar la velocidad angular de las ruedas; y así la velocidad tangencial de estas, mediante una relación entre la velocidad angular del eje final y las dimensiones físicas de las ruedas se determina finalmente la velocidad del prototipo, es decir:

$$V_{tang} = \omega R \tag{3.1}$$

Donde:

V_{tang} es la velocidad tangencial [m/s].

ω es la velocidad angular [rad/s].

R es el radio de las ruedas [m].

Teniendo la distancia angular en radianes de los imanes que generan dos pulsos sucesivos, y siendo igual a θ_D se tiene que:

$$\omega = \frac{\theta_D}{T_P} \tag{3.2}$$

Donde:

T_P es el período o tiempo de paso de un imán con respecto al siguiente [s].

θ_D es la distancia angular entre dos imanes [rad].

La velocidad tangencial se obtiene sustituyendo la ecuación 3.2 en 3.1, quedando de la siguiente manera:

$$V_{tang} = \frac{\theta_D \, R}{T_P} \tag{3.3}$$

Se observa que la velocidad tangencial que se desea determinar depende del diámetro de las ruedas, la distancia angular entre imanes y el tiempo que se obtenga.

Se procedió calculando la velocidad del eje final dentro de un ciclo de programación, haciendo un sondeo cada cierto tiempo el cual depende de la estructura del programa y de la velocidad de procesamiento del microcontrolador, repitiendose hasta que se tenga el tiempo entre el inicio o el final de 2 pulsos consecutivos, o en su defecto que se consuma el tiempo máximo de ejecución del muestreo y muestre velocidad nula, "0".

El diagrama de bloque de dicha subrutina contemplando todo lo anterior queda como el de la figura 3.5.

Los tiempos de los cuales se hizo uso para determinar este parámetro, se obtuvieron tomando en cuenta cuanto tarda la rutina en ejecutar una iteración o mediante un contador propio del microcontrolador.

3.2.1.3. Obtención de las RPM.

Teniendo como referencia el trabajo de grado realizado anteriormente dentro de la organización, el cual muestra la forma de onda de las RPM para el motor usado en ese momento, y partiendo de la suposición de que la forma se asemeja en los distintos modelos de los motores permitidos en estas competencias, se considera

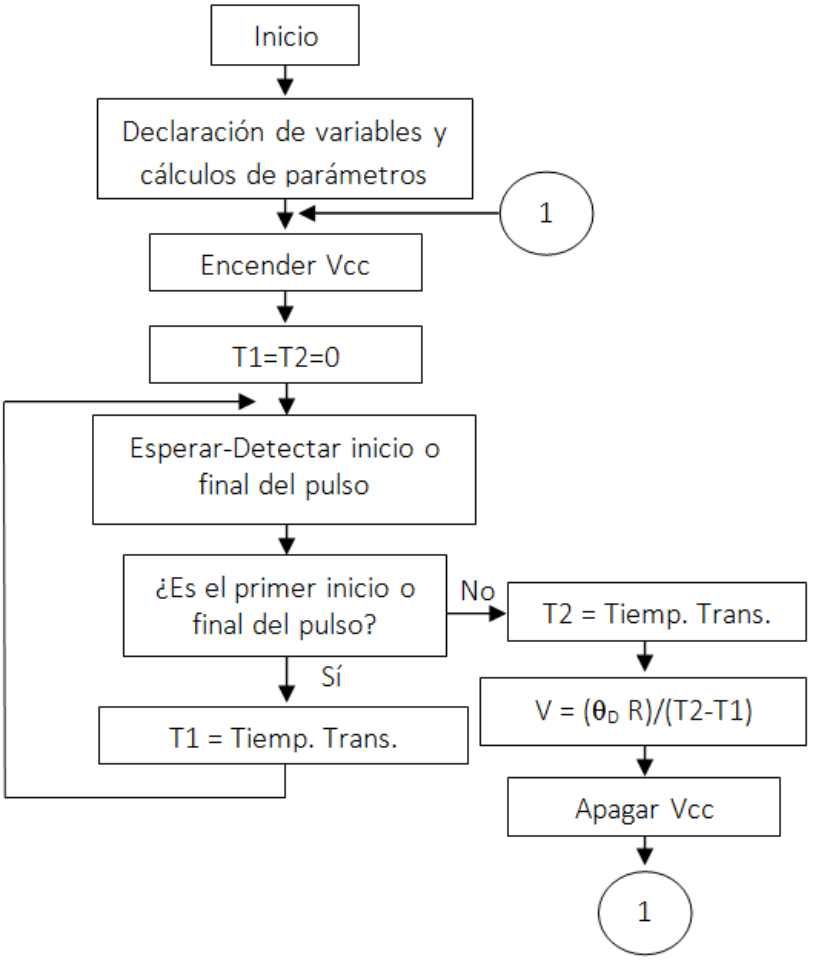


Figura 3.5: Flujograma de la rutina de programación.

utilizar una etapa que transforme la señal para su lectura. Dicha etapa consiste en un circuito que además de proteger el microcontrolador, muestra la señal en pulsos de niveles de voltaje TTL para su mejor lectura. El circuito se basa en un comparador de voltaje, que evalúa la señal de las RPM con una referencia escogida, y tiene como salida una serie de pulsos, que dependen de las características de la señal y de los parámetros del circuito. Este usa un integrado LM393N, en su configuración más sencilla como comparador.

De igual manera en la implementación, se obtuvo la forma de la señal para

corroborar que el comportamiento fuera el mismo, o en su defecto parecido al obtenido en el proyecto de grado anterior [14].

Luego de obtener la señal en pulsos, se puede implementar una rutina, la cual determine el período de la señal. Ya que se desea determinar las RPM a partir de 1000 revoluciones por minutos, esta cantidad es el límite inferior de los pulsos a medir. Dicha rutina debe cumplir con el flujograma de la figura 3.6.

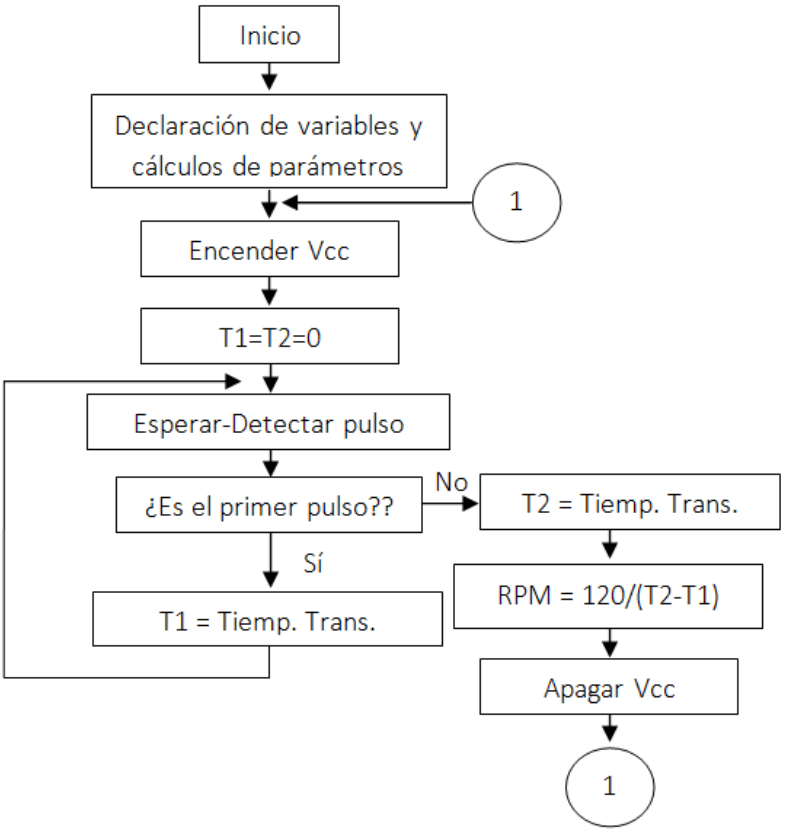


Figura 3.6: Flujograma de la rutina de programación.

Se determinará el tiempo entre los pulsos de forma distinta a la de la rutina anterior, pero su diagrama de flujo es prácticamente el mismo. La transformación que se debe hacer para obtener las RPM a partir del tiempo obtenido es la siguiente:

$$\mathrm{RPM} = 1\mathrm{min}\,\mathrm{C} = \frac{60\ \mathrm{s}}{\mathrm{T}} \tag{3.4}$$

Donde:

C es la frecuencia de los ciclos [Hz].

T es el período de cada ciclo [s].

3.2.1.4. Obtención de la Aceleración:

En este caso sólo es de importancia la aceleración en el eje X para tener la aceleración en sentido delantero del prototipo, por lo que debe tomar en cuenta que el sensor tenga sus ejes alineados con los de la figura 3.7, cuando se realice el montaje dentro del prototipo.

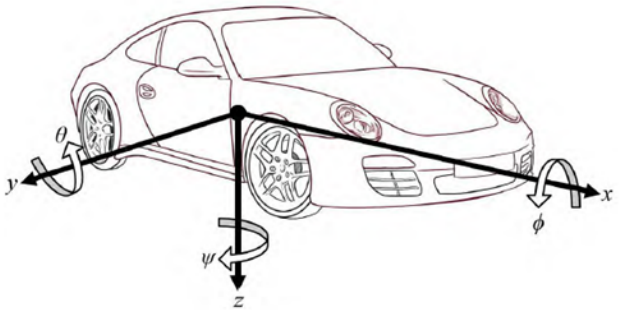


Figura 3.7: Sistema de ejes de coordenadas definido por SAE [7].

Considerando para la creación de la rutina, el valor de voltaje correspondiente para 0g como una constante y la sensibilidad obtenida del *datasheet* del microcontrolador, la rutina quedaría descrita por el siguiente diagrama de flujo que se presenta en la figura 3.8.

3.2.2. Emulación del ELM, del protocolo OBD y programación final del microcontrolador.

Para la simplificación del sistema y del código, se realizó el programa del microcontrolador con la configuración básica de un ELM mediante comandos AT, es decir, la emulación no consta de todas las características disponibles por un ELM

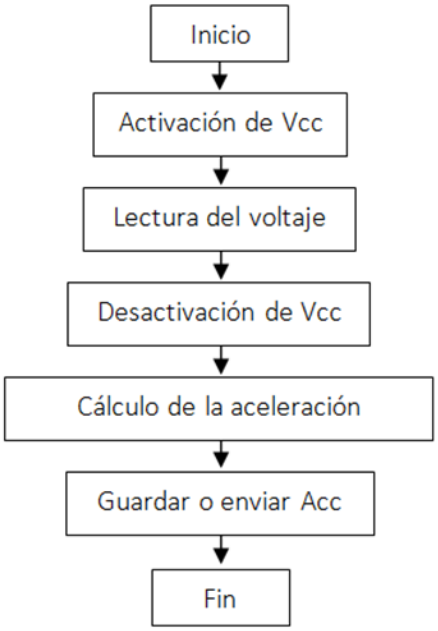


Figura 3.8: Flujograma de la rutina de programación.

verdadero. Algunos comandos quedaron excluidos por ser innecesarios para efectos prácticos de este trabajo de grado; como por ejemplo comandos que configuran la conexión ECU-ELM, ya que dicha conexión no existe, no tienen efecto alguno desde el punto de vista de la PC, usuario y los software probados no implementaban dichos comandos.

La programación final consta de un código que otorga el comportamiento del ELM y la comunicación OBD, este obtiene los parámetros deseados mediante funciones, que se basan en los diagramas explicados anteriormente.

3.2.2.1. Configuración básica necesaria del microcontrolador para emular el ELM.

Para determinar cuáles son los comandos AT que logran que determinado *software* inicie y mantenga una comunicación con el ELM se procedió a verificar por ensayo y error.

Se utilizaron varios *software* para interpretar el ELM mediante la PC, estos se conectaban a un puerto virtual, que a su vez estaba conectado a un programa para visualizar el puerto; la finalidad de esto fue monitorear de manera serial los comandos que otorgaría el software al ELM.

Siguiendo este procedimiento se utilizaron 2 programas, observando qué comandos enviaba cada programa intérprete por el puerto con destino al ELM, se usó el *datasheet* del ELM para saber el significado de dichos comandos, la respuesta a estos comandos que debería producir el ELM se emuló mediante códigos de programación. Repitiendo esto para los 2 programas intérpretes de ELM y para los distintos códigos enviados por los intérpretes; hasta lograr establecer y mantener una comunicación.

Al lograr establecer y mantener la comunicación con el Arduino virtual, se probó dicho código con varios programas intérpretes de los ELM, para corroborar que se pudieran obtener los parámetros mediante otro *software* y en caso contrario se hicieron las correcciones necesarias en el código, logrando la comunicación con el *software* en cuestión.

3.2.2.2. Emulación del protocolo OBD.

Los protocolos OBD cuentan con un modelo de capas que contiene capa física, capa de enlace de datos, capa de red y capa de aplicación. En este caso, solo se tomaron en cuenta las capas de red y de aplicación, ya que la naturaleza de la emulación permite obviar las otras capas, siendo de importancia solamente los datos que se presenten al final del sistema emulador y la interpretación de red que se le da a los datos en el emulador del ELM.

Las tramas de datos de los mensajes ODB se encuentran estructuradas de la misma manera para todos los protocolos, divididos en tres partes que son el encabezado o *header*, el modo y el PID (*Parameter ID*). Por lo que se tomó en cuenta estas partes, ya que sin estas no es posible la interpretación de los datos y por ende la correcta comunicación. Escogiendo el encabezado y el direccionamiento más sencillo de los distintos protocolos, para facilitar la programación del código.

3.2.3. Simulación del radioenlace.

El enlace radioeléctrico se verificó de forma teórica, previamente a la implementación para así realizar ajustes al sistema y controlar parámetros, como la altura necesaria de las antenas o la ganancia de estas.

Como la información en cuanto a competencias futuras es incierta, ya que no se sabe si siguen un formato en cuanto al diseño, la localidad y el terreno donde se podrían efectuar dichas competencias, se hizo un estudio radioeléctrico en base a 3 competencias anteriores, debido a que fueron de las que se consiguieron datos sobre la posición geográfica y dimensiones tanto de la pista como del aparcadero de los equipos participantes en estas competencias.

El radioenlace se formo entre el prototipo como emisor (Tx), este envía los parámetros correspondientes a las solicitudes de interés de la antena Rx, la cual estará dentro de un radio aproximado de 1,5 km de distancia como máximo. Para todas las simulaciones, se usaron como puntos fijos de estudio, los límites de ambas zonas que se encuentren más alejados entre sí, teniendo así el caso más desfavorable en cuanto atenuación por distancia entre los puntos del radioenlace.

El objetivo de las simulaciones realizadas fue determinar de forma teórica si la zona de cobertura del transmisor en la PC de diagnóstico abarca todo el recorrido del prototipo en la pista. El estudio se desarrolló bajo el *software Radio Mobile* que tomo en cuenta la potencia transmitida, sensibilidad de recepción, perdidas totales y las características del entorno geográfico para predecir el rendimiento del sistema de comunicación, el cual puede o no tener línea de vista, estudiando así el desempeño teórico que tendrían los módulos de comunicación y antenas seleccionadas, en los distintos entornos de estudio.

Las coordenadas en cuanto a latitud y longitud, del transmisor y receptor respectivamente, se obtuvieron de Google Earth. Estas coordenadas fueron usadas mediante el *software Radio Mobile* para obtener los datos del terreno y realizar la simulación.

3.3. Implementación del diseño.

3.3.1. Pruebas primarias.

Al finalizar el diseño del sistema en su totalidad, se realizaron pruebas primarias que no involucraron al prototipo para evaluar el desempeño junto a la correcta comunicación del sistema de telemetría y verificar que se obtuvieran los mismos resultados que en las simulaciones.

En principio, se probó el sistema de adquisición de datos de forma que enviara parámetros constantes para las mediciones de velocidad, RPM, aceleración y gasolina, para verificar el código de emulación y la correcta comunicación.

Luego se probó el sistema de comunicaciones dentro de un entorno suburbano, el objetivo era probar el desempeño dentro de este entorno, esto se logró mediante una prueba que dispone el software de los módulos inalámbricos, con el cual se obtiene la cantidad de paquetes perdidos respecto a la cantidad de enviados en puntos específicos de ese entorno, teniendo así una aproximación al rango de cobertura de los módulos.

3.3.2. Cableado e interconexión del sistema.

Para la interconexión y cableado, se trato que el sistema fuera fácilmente desmontable, además de que la interconexión entre el microcontrolador y los sensores fueran mediante conectores de uso comercial, los cuales harían más fácil la conexión y desconexión de algún elemento en específico, para desmontarse o cambiarse. El microcontrolador se colocó dentro de una caja, la cual está diseñada para contener a este con la tarjeta de expansión que le permite conectarse al módulo inalámbrico. Adicionalmente, dentro de la caja se colocó el circuito acondicionador de la señal de RPM, el acelerómetro y los conectores; estos conectores son el de la antena y el de los sensores.

Para la conectorización de los sensores al microcontrolador, se escogió usar un conector UTP ya que se necesitaban la misma cantidad de hilos de cobre que tiene

dicho conector. Mientras que para los conectores del cableado a los sensores de efecto Hall se requerían sólo 3 hilos, se decidió usar conectores RJ-11 para cada sensor. La conexión del cableado de las RPM se hizo directa al chasis de motor y al puerto de salida del voltaje de dicha señal, mediante los 2 hilos sobrantes del cable UTP. Se utilizó para el cableado a través del prototipo cable UTP diseñado para la intemperie, en un segmento del mismo se dividió para realizar las 3 conexiones correspondientes (RPM, velocidad y nivel de gasolina), cuidando de que todo el trayecto estuviera cubiero por su respectiva chaqueta y debidamente dispuesto en el prototipo.

El conector para la comunicación inalámbrica remota ya está definido mediante el módulo Xbee, por lo que se conecta la antena directamente a este módulo. Mientras que la conexión del módulo Bluetooth se realizó de manera directa a las entradas del Arduino.

3.3.3. Montaje y pruebas específicas.

Las pruebas y el proceso de correción de errores mostrados a continuación se realizaron en el prototipo 2013. Luego de la construcción del prototipo 2015, se realizó el mismo proceso de pruebas y montajes para este prototipo a excepción del sensor de gasolina y el sensor de velocidad con sus respectivos imanes, en el que la forma de montaje cambió, teniendo para esta implementación los resultados esperados para cada parámetro por parte del sistema.

Para el montaje de la gasolina en el prototipo 2013, se utilizó un cajetín telefónico, el cual contiene un puerto RJ-11 con el sensor debidamente colocado y conectado a dicho puerto. Este cajetín se dispuso en la parte inferior externa del tanque para detectar la proximidad del flotante mediante el sensor, dicho flotante fue diseñado con un material de goma semiespumosa, con una forma entre cónica y esférica asemejándose a un globo aerostático; el imán se pegó a este flotador con una pega de alta resistencia. El material del flotante y de la pega se podrían carcomer o

desgastar con la gasolina, por lo que se sumergió dicho flotante cuando al estar elaborado, bajo gasolina durante 2 días para comprobar que no despidiera impurezas que pudieran afectar en la combustión.

El tanque tiene 3 cilindros dentro del mismo, que van desde la parte superior hasta la parte inferior, alineados de tal manera que forman un triángulo equilátero con el eje de la boca de este tanque, por donde se surte la gasolina. La forma del flotante se escogió para que pueda subir y bajar con el líquido, siguiendo la línea central del tanque, guiado por los cilindros y que no pasara entre estos.

El procedimiento para obtener la relación del voltaje versus nivel de liquido, constó en realizar 3 pruebas en las que se adicionó agua en cantidades de 280ml, partiendo del tanque vacío hasta que se llenará, midiendo en cada adición de líquido el voltaje de salida del sensor, para obtener una correspondencia entre el nivel del líquido y voltaje de salida. Este resultado se graficó para obtener de forma de la variación. Luego mediante esta gráfica, se estableció la sensibilidad adecuada para la visualización del nivel del gasolina. Al tener la relación final se modificó el código, verificando su correcto funcionamiento y corroborando que se obtenienga el resultado esperado.

Luego al implementar en el prototipo 2015, sólo se cambia el método de montaje, siendo el cajetín telefónico muy grande ya que se tenía una limitación en cuanto al espacio disponible bajo el tanque, por lo que se decidió usar 2 laminas de plástico que contenían el sensor y sellados mediante una pega de alta resistencia. Obteniendo los mismos resultados que para el prototipo 2013 ya que los tanques eran idénticos.

El acelerómetro se dispuso en el *Shield* del Arduino, considerando que el eje "y" de este dispositivo estaría alineado con el eje "x" del prototipo, por lo que se adquiere el voltaje de salida correspondiente a dicho eje del dispositivo. Luego de implementar, se observó que existe un error de variación en el voltaje de salida del acelerómetro, el cual es de alta frecuencia y se debe tanto a la sensibilidad del acelerómetro como a la vibración del prototipo, por lo que se decidió eliminar este error mediante el código del Arduino.

Se observó además que el voltaje V_0 para 0g era distinto en diferentes pruebas que se realizaron, suponiendo que esta variación depende del voltaje de alimentación del Arduino, el cual puede variar de una prueba a otra, por lo que se decidió adquirir el voltaje V_0 al inicio del programa.

En el caso de las RPM, se procedió a realizar el circuito comparador, para luego obtener la forma de la señal de las RPM en la salida de este circuito con distintos valores de voltaje de referencia. Luego se graficaron para analizar y seleccionar la forma de onda que se considere más adecuada para su lectura. El circuito se fijó en una parte vacía de la tarjeta de expansión del Arduino. Luego de seleccionar el voltaje de referencia para obtener la forma de onda deseada, y disponer el circuito en un espacio adecuado, se realizaron las pruebas y correcciones pertinentes al programa que obtenía las RPM, ya que se obtenían errores en cuanto a la lectura de este parámetro.

En el montaje del sensor de velocidad, se procedió de forma similar al del nivel de gasolina, y se utilizó una caja de telefonía que contenía dentro el sensor de efecto Hall, dispuesta de tal manera que el sensor captare los imanes, que se colocaron a una distancia de 40° cada uno en el disco de freno.

Al momento de la implementación se obtuvieron errores en la obtención de este parámetro, por lo que se graficaron distintas señales que se obtenían del Arduino a distintas velocidades. Para observar como se comportaba la señal y verificar si correspondía a la que se esperaba, y realizar las correcciones correspondientes tanto en el código como en *hardware*. Para el prototipo 2015 los imanes se colocaron a una distancia de 90°, lo cual aumentó el tiempo de paso de los imanes, además, el montaje fue distinto ya que el espacio del que se disponía era reducido al igual que en la implementación del nivel de gasolina, por lo que se procedió usando una caja pequeña de plástico para contener el sensor que detectó los imanes, dicha caja se dispuso en la parte exterior de la cobertura de la transmisión cercana al disco de freno.

3.3.4. Pruebas finales.

Al haber corregido los errores mediante el código del Arduino y *hardware*, aplicando las correciones que se hayan considerado necesarias, se puso a prueba el sistema de telemetría, adquiriendo los parámetros en tiempo real y de manera remota en un ambiente abierto simulando una competencia, obteniéndose una aproximación al área de cobertura y el desempeño del sistema dentro de esta. Asi mismo se escogió que los 4 parámetros se obtuvieran al mismo tiempo, mostrándolos todos dentro de una misma gráfica, para observar si el envío de datos era fluido, ya que este sería el caso de mayor tráfico en la comunicación en la cual se podrían generar errores.

Después de probar todo el sistema, se procedió a verificar que los parámetros captados y enviados (de manera inalámbrica) tuvieran valores correctos, esto se hizo comparando los valores de RPM y velocidad obtenidos del sistema con otro dispositivo externo para corroborar que las mediciones fueran certeras. En cuanto a la gasolina y a la aceleración, no se procedió de esta manera, debido a que la lectura del nivel de la gasolina ya estaba previamente definda por la cantidad de líquido que existe dentro del tanque y la aceleración fue obtenida a través de un instrumento de medición, con lo cual la certeza de este instrumento escapa de las manos del presente proyecto de grado.

Por último, se efectuó un análisis y se enuncian las conclusiones y consideraciones sobre la implementación del proyecto, para que se tomen en cuenta en el uso de este sistema y otros sistemas de comunicación y telemetría.

Capítulo IV

Análisis, interpretación y presentación de los resultados

4.1. Selección del microcontrolador ELM que se emulará, el protocolo OBD y los protocolos de comunicación.

4.1.1. Selección del ELM a emular.

Dentro de los ELM mostrados en la tabla 4.1, el integrado ELM327 puede soportar todos los protocolos OBD de relevancia para este estudio (ya que hay otros protocolos para vehículos de carga), debido a esto se escogió emular dicho integrado ya que fue práctico a la hora de seleccionar el protocolo OBD.

Tabla 4.1: Protocolos aceptados por los diferentes modelos de ELM.

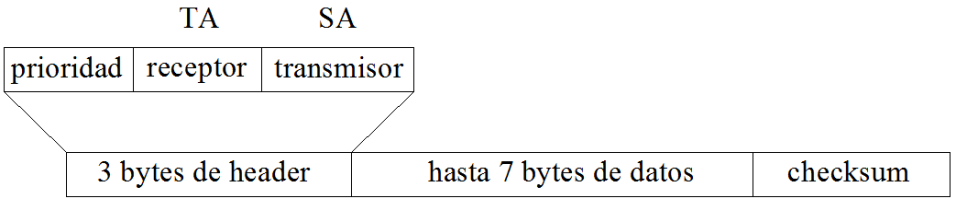
	ELM320	ELM322	ELM323	ELM327	ELM329
SAE J1850-PWM	X			X	
SAE J1850-VPW		X		X	
ISO 9141-2			X	X	
ISO 14230-4 (lento)		X		X	
ISO 14230-4 (rápido)		X		X	
ISO 15765-4 (CAN)				X	X
SAE J2411 (SWCAN)				X	X
SAE J1939 (250 kbps)				X	X
SAE J1939 (500 kbps)				X	X

Es decir que cualquier protocólo OBD que se hubiera seleccionado podría emularse junto el comportamiento de este microcontrolador.

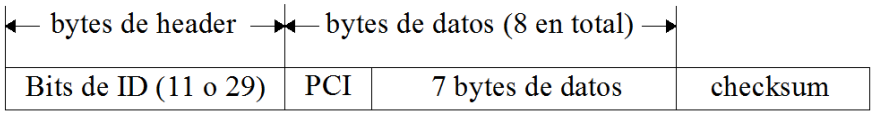
4.1.2. Selección del protocolo OBD a emular.

Los protocolos OBD se dividen en 2 tipos con respecto al direccionamiento, los protocolos OBD y los protocolos CAN OBD. Los protocolos OBD incluyen los protocolos ISO 9141-2, ISO 14230-4 y SAE J1850; entre estos protocolos hay diferencias en la capa física y en la capa de enlace de datos, pero el direccionamiento mantiene el mismo formato cambiando sólo el contenido de los distintos campos de este formato para cada protocolo. Mientras que para el protocolo CAN OBD (ISO 15765-4) el direccionamiento es distinto y puede variar dependiendo si el header o encabezado es de 11 o 29 bits, como se puede apreciar en la figura 4.1.

En ambos casos, se necesita un direccionamiento ya que puede suceder que en un automóvil se tenga más de una ECU. Existen dos tipos de direccionamiento para estos casos con respecto al protocolo CAN OBD, el direccionamiento físico y el direccionamiento funcional. Este último, es usado como broadcast o difusión, si se observa desde el punto de vista de una red TCP/IP. Esto se debe a que en algunas



(a) Mensaje OBD



(b) Mensaje OBD CAN

Figura 4.1: Estructura de los distintos mensajes OBD [8].

ocasiones la herramienta de prueba (en este caso el ELM) no sabe a qué ECU direccionar dicho mensaje (en el caso de que esté en proceso de reconocimiento de la red de ECUs), o necesita enviar el mismo mensajes a todas las ECUs.

En el header de un mensaje OBD se tienen 3 parámetros que pueden cambiar dependiendo del tipo de solicitud o respuesta que se haga, los cuales son la prioridad del mensaje, el TA (Target Address) y el SA (Source Address).

Para un mensaje CAN OBD se tiene sólo un header, que para 11 bits sólo podría representar uno de los siguientes casos: la dirección de donde proviene el mensaje de una respuesta de una ECU hacia el ELM, la dirección física hacia cual ECU va dirigido el mensaje desde el ELM o una dirección funcional (mensaje dirigido a todas las ECUs). En el caso de 29 bits se presenta un direccionamiento parecido al de un mensaje OBD, es decir se tiene un TA y un SA, adicionalmente se tienen otros bits de los cuales unos tienen un valor constante y los restantes toman un valor dependiendo si es direccionamiento físico o direccionamiento funcional.

El mensaje CAN OBD de 11 bits es el más sencillo ya que si se simula una sola ECU, el mensaje visto en la PC tendría siempre el mismo header y sería más corto que el de 29 bits; a diferencia de un mensaje OBD en el cual podría cambiar la prioridad del mensaje. Por esto se escogió el protocolo CAN (ISO 15765-4) de 11 bits a simular junto con el ELM.

Además en la imagen 4.1 para el mensaje CAN OBD se observa un parámetro dentro del campo de datos que se denomina PCI, esto no es más que la cantidad de los bytes de datos a enviar en un mensaje. Es importante destacar esto ya que en la respuesta del ELM a la PC el header permaneció igual, sólo cambió el campo PCI y los datos a enviar.

En ambos casos el checksum no es visible para la PC ya que el ELM realiza todos los cálculos pertinentes a este, por lo tanto no los muestra y carece de interés en este proyecto.

4.1.3. Selección de los protocolos de transmisión inalámbrica.

Los distintos protocolos tomados en cuenta son los mostrados en la tabla 4.2.

Tabla 4.2: Protocolos inalámbricos tomados en consideración.

Protocolo	Wi-Fi	Bluetooth	ZigBee/DigiMesh
Frecuencia	2.4/5 GHz	2.4 GHz	902-928 MHz
Velocidad de transmisión	54 Mbps	1 Mbps	10 o 250 kbps
Alcance Indoor	70 m	-	-
Alcance Outdoor	250 m	-	2,5 km (con LOS)
Potencia de salida (máx.)	Varía con el país	100 mW	100 mW
Modulación	2GFSK con FHSS	GFSK	O-QPSK y BPSK con DSSS (ZigBee) o FHSS (DigiMesh)
Tipo de red	WLAN	WPAN	LR-WPANs

Para la comunicación remota se tuvo que seleccionar un protocolo que sea apropiado para el ambiente donde se realizan estas competencias, que normalmente es un área rural o sub-urbana, en la que el enlace puede o no tener LOS. Por ende para la selección del protocolo se tomó como criterio la pérdida de la potencia, que se calculó teóricamente para un radioenlace con una obstrucción de filo de cuchillo, obteniendo cual tiene menor atenuación ante los objetos que obstruyan la línea de vista.

Por lo que se evaluaron los rangos de frecuencias descritos en la tabla 4.2, considerando una obstrucción por filo de cuchillo de 15 m sobre el nivel de la línea de vista en el medio del radioenlace. Con una distancia entre las antenas para este caso de 2 km se deja un rango de holgura para la colocación de estas y sus alturas, suponiendo que estas antenas son isotrópicas.

Usando las ecuaciones 2.3 y 2.6, siendo $d1 = d2 = 1$ km, se obtuvo el resultado mostrado en la tabla 4.3.

Tabla 4.3: Atenuación de un radienlace 2 km debido a la difracción de una arista por filo de cuchillo de 15 m por encima de la LOS.

Frecuencia	Atenuación a 2 km
915 MHz	115,17 dB
2,4 GHz	127,54 dB
5 GHz	137,04 dB

Como se aprecia en la tabla, la frecuencia de 915 MHz es la que tiene menor atenuación en cuanto a la difracción de objetos y pérdidas de espacio libre, por esto y porque su atenuación debido a la lluvia es prácticamente despreciable respecto a las otras ya que la longitud de onda de 33 cm, se decidió el uso de un sistema de comunicaciones que use alguno de los protocolos Zigbee o DigiMesh.

En cuanto a la comunicación dentro del prototipo, se podrían haber usado el protocolo Bluetooth o el Wi-Fi. Para usar el protocolo Wi-Fi. se necesitaría que uno de los 2 dispositivos (el sistema de telemetría o el dispositivo móvil que se conectaría a este), funcione como coordinador de red y que emule algunas funciones de red o por lo menos DHCP. Debido a que lo anterior complicaría el desarrollo del sistema de telemetría, se decidío usar el protocolo Bluetooth ya que al establecerse la comunicación se tendría una comunicación serial transparente.

4.2. Selección de métodos, dispositivos de adquisición de los parámetros y componentes necesarios.

4.2.1. Dispositivo para la obtención del nivel de gasolina:

Usando un sensor de intensidad de campo magnético junto con un flotador magnético se puede determinar la proximidad de dicho flotante al fondo del tanque de gasolina y por ende el nivel del líquido. En la figura 4.2, se muestra como sería esta configuración.

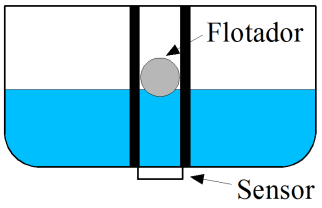


Figura 4.2: Esquema del montaje del sensor y el flotador en el tanque.

Se puede usar la misma estructura del tanque, el cual dispone de tres tubos en el centro del tanque dispuestos de manera que forman los vértices de un triangulo equilátero. Estos tubos sirvieron como guía para que el flotador no se mueva por la superficie del líquido, manteniendolo en la línea vertical sobre el sensor y logrando una medición más exacta.

El sensor escogido para esta aplicación es el modelo A1301 de Allegro MycroSystems, el cual es un sensor de efecto Hall y posee una salida analógica que varía con respecto a la intensidad de campo magnético que detecte. Las características principales se muestran en la tabla 4.4.

Dicho sensor no se comparó con ningún otro, ya que sus características eran adecuadas para la aplicación y era el único sensor de su tipo encontrado en el mercado local.

Tabla 4.4: Características del sensor de efecto Hall.

Modelo	A1301
Voltaje de operación	4,5 - 6 V
Corriente máxima de entrada	11 mA
Tiempo de encendido	3 - 5 µs
Voltaje de salida (B = 0 G)	2,4 - 2,6 V
Sensibilidad	2 - 3 mV/G
Ancho de banda de salida	20 kHz

4.2.2. Dispositivo para la obtención de las revoluciones por minuto (RPM):

La forma de onda del voltaje a la salida del motor se muestra en la imagen 4.3 que se obtuvo haciendo uso de la herramienta softcope de Matlab.

Figura 4.3: Forma de la señal del voltaje generado por el motor hacia la bujía.

Esta señal se debe procesar previamente, ya que los microcontroladores trabajan con voltajes normalmente TTL o entradas analógicas compatibles sólo con voltaje positivo. Por esto se escogió implementar un comparador de voltaje, para procesar dicha señal evitando algún posible daño en el microcontrolador y acondicionando la señal para su mejor lectura.

Usando un comparador de voltaje se puede eliminar la componente negativa, además de obtener pulsos cuadrados correspondientes a las crestas positívas que

superen el voltaje de referencia. El circuito que se usó es el de la figura 4.4.

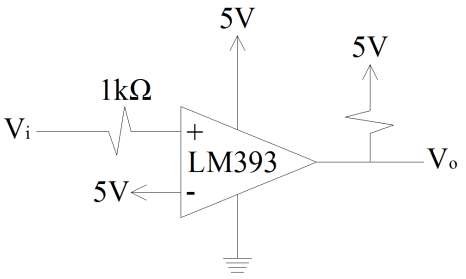


Figura 4.4: Esquema circuital del comparador de voltaje.

Se decidió usar el integrado LM393N debido a su fácil configuración y por la facilidad de su adquisición. Todos los valores de alimentación de este circuito se obtendrán desde el microcontrolador.

4.2.3. Dispositivo para la obtención de la velocidad:

Al igual que en la obtención del nivel de gasolina, se utilizó un sensor de efecto Hall del mismo modelo, para la detección de los imanes que estén posicionados en el disco de freno. Los imanes utilizados serán retazos de una banda magnética de aproximadamente 0,5 cm de espesor, usada para fines artesanales. Estos se fijaron en el disco de freno cada 90 grados en el prototipo 2015 y cada 45° en el prototipo 2013, de esta forma midiendo el tiempo entre la detección de un imán y el siguiente se determina la velocidad del prototipo.

4.2.4. Dispositivo para la obtención de la aceleración:

Debido a que se decidió usar un acelerómetro para determinar la aceleración, se compararon 3 de estos los cuales se consideran los más apropiados para la implementación, estos se muestran en la tabla 4.5.

Por motivos de simplicidad se escogió el ADXL335, ya que al poseer una salida analógica la lectura de dicho parámetro sería más rápida que la que se implementaría si la salida fuera serial como en el caso del MPU-6050; además que este sensor

Tabla 4.5: Comparación de los acelerómetros ADXL335, MPU-6050 y MMA7361LC.

Modelo	ADXL335 (Analog Devices)	MPU-6050 (InveSense)	MMA7361LC (Freescale Semiconductors)
Voltaje de operación	2-3,6 V	2,375-3,46 V	2,2-3,6 V
Rango de medición	±3 g y ±3,6 g	±2 g, ±4 g, ±8 g y ±16 g	±1,5 g y ±6 g
Factor de escala de sensibilidad	300 mV/g	16,384; 8,192; 4,096 y 2,048 LSB/g	800 mV/g para 1,5 g y 206 mV/g para 6 g
Ejes de medición	3	3	3
Corriente normal	320 µA	500 µA	400 µA

no necesita programarse para comenzar a operar. Se debe tomar en cuenta (para este caso) la sensibilidad con respecto a la aceleración a la hora de definir cuál sería la resolución de conversión A/D. Este acelerómetro se muestra en la figura 4.5.

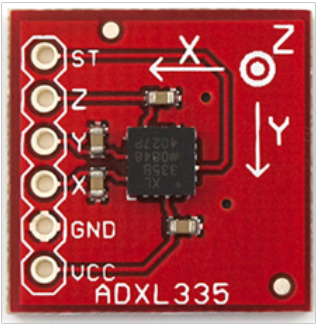


Figura 4.5: Acelerómetro usado en el proyecto (ADXL335) [9].

4.2.5. Modulos inalámbricos:

Para la implementación de la comunicación inalámbrica remota mediante alguno de los protocolos ZigBee o Digimesh, se escogieron 5 modelos XBee del fabricante Digi. La selección de estos 5 modelos dependió directamente de los protocolos de comunicación antes definidos que poseen características como frecuencia de 900

MHz, velocidad de 9,6 Kbps y la potencia aproximada de transmisión que cumpla con los requerimientos del radioenlace. Dicha potencia aproximada es la resultante al aplicar las perdidas básicas del espacio libre adicionando una obstrucción por filo de cuchillo de 15 m sobre la línea de vista y a la mitad del radioenlace, ya que no se garantiza línea de vista (LOS) en la competición, todo calculado a una distancia del radioenlace de 2 km aunque normalmente en las competencias la distancia sería menor.

Teniendo en cuenta todos los elementos del radioenlace, las pérdidas básicas del espacio libre a 2 km y una atenuación por obstrucción de filo de cuchillo de un objeto de 15 m de altura, situado a la mitad del radioenlace, se obtuvo la tabla 4.6, la cual muestra las pérdidas o ganancia de cada elemento considerado.

Tabla 4.6: Presupuesto de potencia para el radioenlace a 900 MHz y distancia de 2 km.

Perdidas básicas del espacio libre (L_{bf})	97,50 dB
Perdida por obstrucción filo de cuchillo (L_m)	17,25 dB
Perdida de la línea de 1 m en Tx (L_{tc})	0,5 dB
Perdida de la línea de 3 m en Rx (L_{rc})	1,5 dB
Ganancia de la antena transmisora (G_t)	-3 dB
Ganancia de la antena receptora (G_r)	-3 dB
Perdidas del radioenlace (L_s)	110,75 dB

Tomando en cuenta los valores antes descritos, se ubtuvo el resumen de la ecuación 2.3 como se muestra a continuación, que será aplicada a cada módulo de comunicación para la comparación del presupuesto de potencia.

$$W_{rec}(\mathrm{dBm}) = W_{ent}(\mathrm{dBm}) - 110,75 \text{ dB}$$

La tabla 4.7 muestra las características de interés de los distintos módulos XBee y el balance de potencia aplicado con la ecuación anterior.

Se consideró que para tener una transmisión sin cortes en el enlace es necesario elegir un módulo que permita tener un margen de seguridad de 10 dB en el presupuesto de potencia para garantizar la confiabilidad del enlace[32]. De tal forma que

Tabla 4.7: Características de los módulos XBee y balance de potencia aplicado.

Modelo	Tasa de transmisión	Frecuencia	Potencia de Tx	Sensibilidad de Rx	Balance de potencia	Margen de seguridad	Costo
XBee-PRO 868	230 kbps	868 MHz	25 dBm	-112 dBm	-86,75 dBm	25,25 dB	45 $
XBee-PRO 900HP	9,6 y 200 kbps	902-928 MHz	24 dBm	-110 dBm y -101 dBm	-87,75 dBm	22,25 dB y 13,25 dB	39 $
XBee-PRO XSC(3SB)	9,6 o 20 kbps	902-928 MHz	24 dBm	-109 dBm o -107 dBm	-87,75 dBm	21,25 dB o 19,25 dB	42 $
XBee-PRO XSC(S3)	9,6 kbps	902-928 MHz	20 dBm	-106 dBm	-91,75 dBm	14,25 dB	42 $
XBee-PRO 900	156 Kbps	902-928 MHz	17 dBm	-100 dBm	-94,75 dBm	5,25 dB	54 $

quedó descartado el módulo XBee PRO 900 por no cumplir este requerimiento.

El resto de los dispositivos de la tabla 4.7 puedieron ser usados en la implementación ya que el margen de seguridad está por encima de los 10dB. Tomando en cuenta el costo de los módulos se definió el XBee PRO 900HP como la mejor opción con la finalidad de ahorrar costos y unido a eso deja flexibilidad en cuanto a la tasa de transmisión y aumento de la distancia del enlace. El Xbee PRO 900HP se muestra en la figura 4.6 .

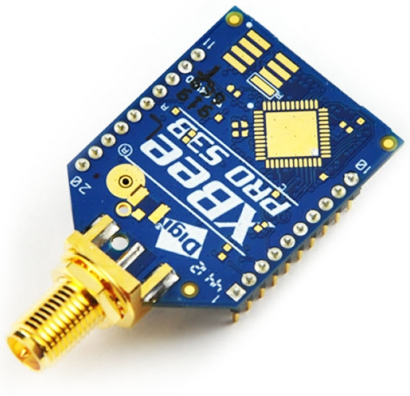


Figura 4.6: Xbee PRO 900HP [10].

Es importante mencionar los siguientes aspectos: la fuente de entrada tiene límites entre 2,4 a 3,6 VDC por lo que puede ser implementado en cualquier interfaz USB, sus pequeñas dimensiones (3.29 cm x 2.44 cm x 0.546 cm) hacen fácil la implementación en muchos escenarios, estas se especifican en la figura 4.7; su peso esta comprendido entre 5 – 8 gr, soporta temperaturas entre -40°C a 85°C lo que lo hace robusto a la implementación en distintos escenarios y permite conectar diferentes tipos de antenas wire, U.FL y RSMA. Su frecuencias está en la banda de 902-928 MHz, la cual está definida por la Unión Internacional de Telecomunicaciones como banda libre ISM (Industrial, Scientific and Medical) en la que no se necesita licencia.

Otra característica importante es que el protocolo de transmisión usa la técnica de espectro esparcido por salto de frecuencia (FHSS), que le permite ser menos susceptible a interferencias y ruido.

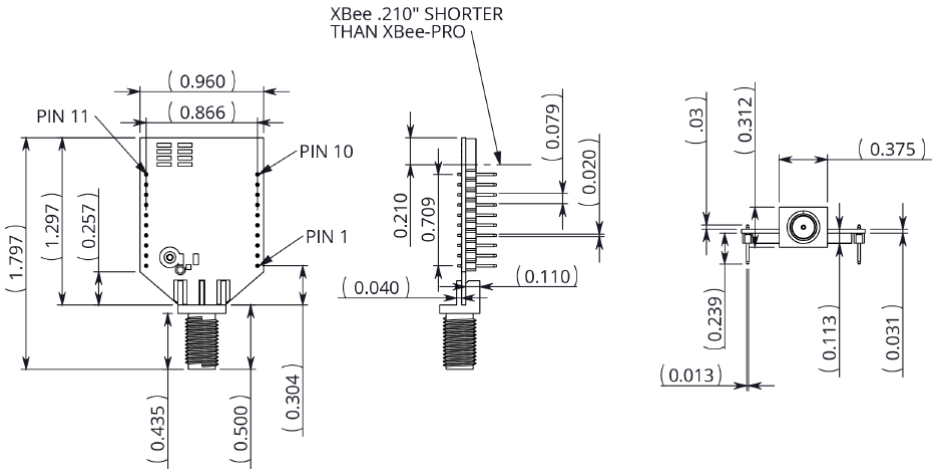


Figura 4.7: Dimensiones del XBee PRO 900HP [1].

Con respecto a los módulos Bluetooth sólo se compararon 2 modelos de la misma serie, ya que se consideró que cualquiera de estos cumpliría con los requerimientos para la implementación. Los módulos considerados fueron el HC-05 y el HC-06, la principal diferencia entre estos es que el HC-05 puede operar como maestro y como esclavo, mientras que el HC-06 solo opera como esclavo. En la implementación el dispositivo móvil sería el maestro ya que iniciría la comunicación, debido a esto no se require que el módulo trabaje como maestro, y además que la configuración de este modo en el módulo HC-05 necesitaría una conexión extra en el microcontrolador, por lo que se selecciona el módulo HC-06 para la implementación. Dicho módulo se presenta en la figura 4.8.

Las características de interés de este módulo se presentan en la tabla 4.8.

4.2.6. Microcontrolador:

En la tabla 4.9 se presentan las características más resaltantes de los microcontroladores que más se ajustaron a la implementación de este proyecto.

Ya que el proyecto requiere de entradas de señales analógicas para la recolección de algunas de las variables de interés se descartó en primera instancia la Raspberry

Figura 4.8: Modulo HC-06 [11].

Tabla 4.8: Características del módulo HC-06.

Voltaje de operación	3,1-4,2 V
Corriente de operación: En apareamiento En comunicación	 30-40 mA 8 mA
Temperatura de trabbajo	-25°C - +75°C 0,5 dB
Sensibilidad	-80 dBm
Potencia de salida	3 dBm
Velocidad por defecto	9600 Baudios

Pi b+. Como se seleccionaron los módulos Xbee para la comunicación,se tomó en cuenta la interconexión entre uno de estos con el microcontrolador. Para el caso de la STM32 Discovery se tendría que haber hecho una conexión manual o un circuito que conecte dicho Xbee con este microcontrolador; en cambio existen módulos de expansión Arduino que hacen posible la conexión directa de un módulo Xbee con el Arduino UNO. Además, el Arduino UNO tiene el doble de velocidad de procesamiento que el STM32 Discovery.

Tabla 4.9: Cuadro comparativo de los microcontroladores considerados.

Nombre del micro	Rasperry Pi b+	Arduino UNO	STM32 Discovery
I/O	27 GPIO	20	QFP64 I/Os
Puertos digitales	27DPIO	14	48
Entradas analógicas	0	6	16
Velocidad del reloj	700 MHz	16 MHz	8 MHZ
Salidas	HDMI, RCA, Audio Jack, SD Card, USB2.0	USB, UART Audio Jack	Mini USB
Voltaje de alimentación	5 V Micro USB	7-12 V o 5 V via USB	3,3 o 5 V
Voltaje de salida	3,3 V máx.	3,3-5 V	3-5 V
RAM	512 MB (SDRAM)	2 KB	8 KB
Flash	-	32 Kb	128 Kb
USB	4	1	1
Ethernet	Si	NO	NO
Costo	37,79 $	12,95 $	28,89 $

Por lo anterior expuesto, se decidió usar el modelo Arduino UNO y adicionalmente entre los tres dispositivos puestos bajo estudio fue el que resulto con menor costo. En la figura 4.9 se muestra la imagen del Arduino a usar.

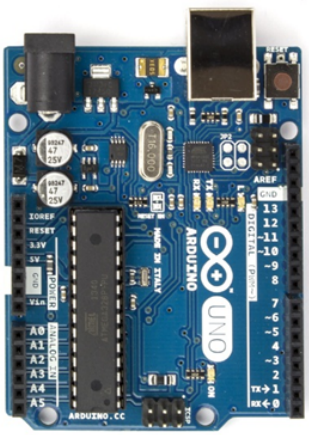


Figura 4.9: Arduino visto desde la parte superior [12].

Se podría pensar que la cantidad de memoria de este dispositivo fue una desventaja, puesto que es la más baja entre los que se estudiaron, pero no se requirío de una memoria RAM de gran tamaño para la implementación de este proyecto ya que los datos se enviaron en tiempo real y si el almacenamiento de este proceso se hizo en la PC.

Lista final de componentes principales:

En resumen la siguiente tabla contempla todos los dispositivos físicos que se utilizaron para hacer posible la implementación:

Tabla 4.10: Lista de componentes.

Componente	Cantidad	Costo
Amplificador Op LM393	2	1,52 $
Sassin Electrics LM18-3005NA	1	22,62 $
ADXL335 (Acelerómetro)	1	14,95 $
Arduino UNO	1	15,95 $
XBEE PRO DigiMesh S3B 900HP RP-SMA	2	53,69 $
Arduino Wireless Proto Shield	1	23,97 $
SainSmart XBee USB Adapter for Arduino UNO MEGA R3	1	12,74 $
GSM 800MHz 850MHz 900MHz Omni-directional Antenna	2	7,99 $
Box for Arduino	1	10,30 $
Cable coaxial tipo RG 58/U dieléc. PE Sólido (1m)	4	0,60 $
Conector SMA Reverso Macho Recto	2	0,22 $
Conector SMA Reverso Hembra	2	0,50 $
Total		230.78 $

4.3. Diseño del sistema de telemetría.

4.3.1. Descripción de funcionamiento de los sistema de adquisición de parámetros.

En esta sección se muestra la estructura con respecto a la programación de las rutinas de adquisición de los parámetros. Estos códigos no contemplan la comunicación hacia el XBee en la etapa de salida, dado que en dicha comunicación se realizó primero el programa de emulación del ELM, para luego integrar todas las rutinas individuales. Esas subrutinas se simularon mediante el programa Proteus ISIS 7 Professional, mediante una librería de Arduino que simula el funcionamiento del microchip ATmega328.

4.3.1.1. Monitoreo del nivel de gasolina.

Para la programación se definieron 9 rangos de detección, en los cuales el microcontrolador podrá definir cual es el nivel de líquido en porcentaje con respecto a la capacidad máxima del tanque, ya que este es el formato en el cual trabajan los protocolos OBD. El rango del nivel de gasolina va desde 10 % hasta 90 % en saltos de 10 %. El código para la adquisición de este parámetro se presenta en el apéndice **??**

Para este caso no fue necesario simular dicho código, dado a la simplicidad del código y a que la confiabilidad de los datos obtenidos dependió de la calibración en la etapa de implementación.

4.3.1.2. Monitoreo de la velocidad.

En la obtención de la velocidad instantánea del eje final, el programa muestra dicha velocidad y termina su rutina, destacando que para velocidades mayores se tiene un tiempo de paso menor, es decir la rutina tarda menos en determinar la velocidad y por ende se tiene una visualización más rápida.

Es necesario que se haya considerado la velocidad tangencial mínima de las ruedas, debido a que esta determina el tiempo en el cual se aplica la subrutina. Se fijó una velocidad mínima o una ventana de muestreo para que el microcontrolador sea capaz de terminar la rutina antes de continuar las siguientes, manteniendo un flujo de datos que el usuario en la PC observe en tiempo real y evitando que caiga en un ciclo infinito en caso de que el carro esté detenido.

En la declaración de variables del programa se pueden introducir los valores de la ventana de muesteo, al igual que el diámetro de las ruedas traseras y otros parámetros que puedan variar dependiendo del prototipo al que se aplique este sistema de telemetría. Es decir que la rutina es escalable y versátil a la hora de cambiar las características del disco de freno, las de las ruedas y la velocidad de visualización que se desee.

Se tomó en cuenta el valor de la ventana máxima, ya que es más práctico en este caso cambiar la estructura de tiempo de la subrutina que trabajar en función de la velocidad mínima, que al final llevaría un cálculo asociado al tiempo de la ventana, debido a que se trabaja en función del tiempo; además, otorgando un tiempo de ventana máximo se puede determinar y manejar de mejor forma la visualización por parte del usuario.

El montaje de los imanes en el disco de freno se presenta en la figura 4.10.

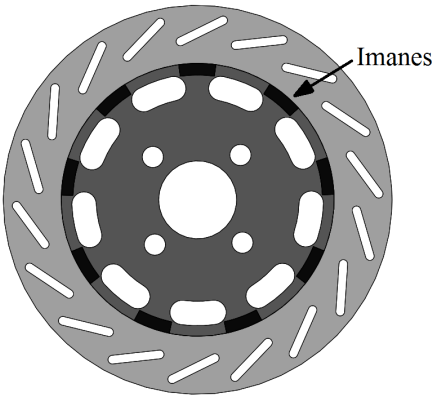


Figura 4.10: Montaje de los imanes en el disco de freno.

Se tomó en cuenta que en el montaje, la separación entre los imanes fuera constante entre todos estos para que en el cálculo no existan errores por esta separación.

En el apendice **??** se muestra el código, se tomó en cuenta hacer la mayoría de los procesos matemáticos fuera del ciclo iterativo, haciendo que los ciclos sean mas rápidos además de simplificar el código.

En el código se tomaron en cuenta los cambios de estado en los flancos ascendentes, por lo que se debe tener conocimiento si el voltaje que se lee es el de un valle para capturar el flanco ascendente, y además saber si es el primer o segundo valle.

La instrucción millis(), devuelve el tiempo que ha transcurrido en milisegundos desde que se encendió el Arduino, dicho contador se reinicia aproximadamente cada 70 minutos por lo que para este caso es útil ya que los errores debido a este contador serían como máximo 1 cada 70 min, además que sería poco probable que suceda el error.

En las constantes del código anterior, se usaron las dimensiones físicas del prototipo BAJA 2013.

Estos prototipos alcanzan una velocidad máxima de 70 km/h aproximadamente, por lo que esta velocidad junto con la velocidad mínima serán las velocidades de estudio para la simulación. El montaje de simulación en el software Proteus fue el de la figura 4.11.

Este esquema se simuló para señales de períodos de tiempo de paso de 100 ms y 10,6699 ms que son los correspondientes a la señal que se obtendría en la velocidad mínima y en la máxima de los prototipos, respectivamente. Los resultados a la velocidad mínima en varias iteraciones continuas, fueron los de la figura 4.12.

Se observa que para esta velocidad no hay error en la medición y se obtiene una velocidad de 7 km/h. En cambio para la velocidad máxima de 70 km/h se tiene el resultado de la figura 4.13.

En el cual se obtuvo un error de ± 1 km/h, que para este caso es aceptable, ya que normalmente los prototipos no alcanzan por tiempo prolongado la velocidad

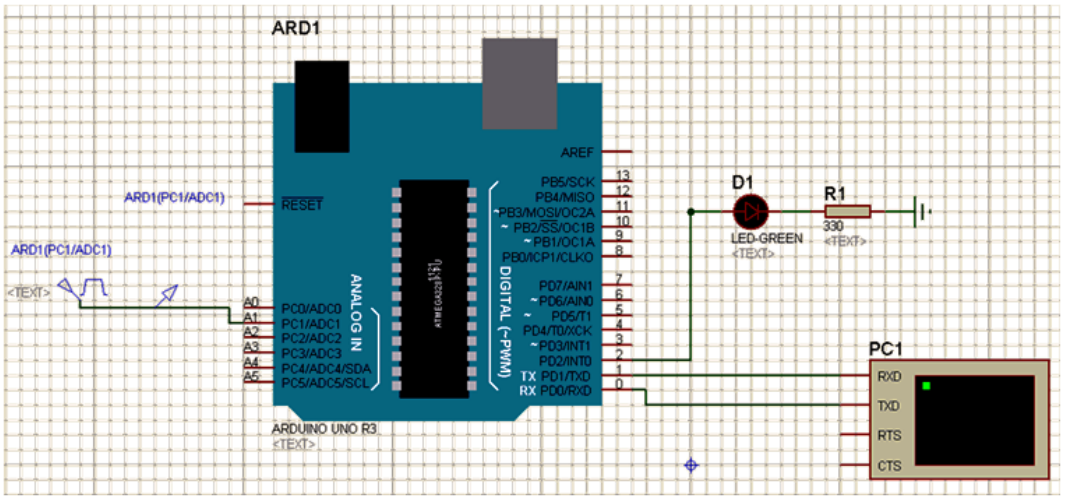


Figura 4.11: Esquema de simulación del sistema de obtención de la velocidad.

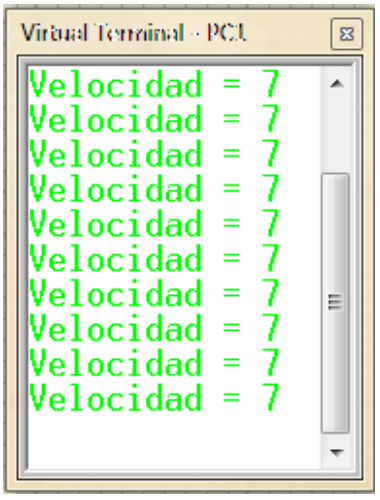


Figura 4.12: Resultado de la simulación del sistema de obtención de velocidad para 100 ms.

máxima debido a las características de la pista. Este error se debe a que la rutina introduce un tiempo adicional debido a la velocidad de muestreo, que a medida que se aumenta la velocidad del prototipo, aumenta el error ya que el tiempo entre muestras se hace comparable con respecto a los pulsos y valles.

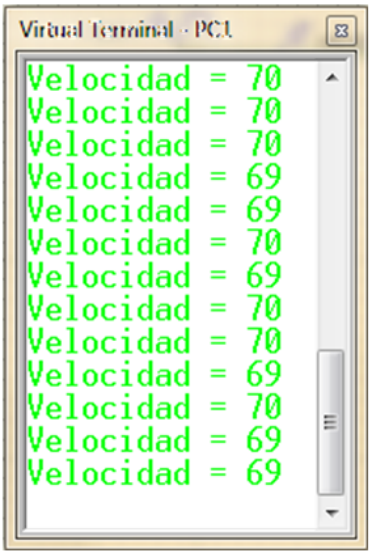


Figura 4.13: Resultado de la simulación del sistema de obtención de velocidad para 10,6699 ms.

4.3.1.3. Monitoreo de las RPM.

Teniendo la señal en un tren de pulsos, se diseñó una rutina que captura el inicio de 2 pulsos consecutivos y mediante estos tiempos determinar el período de las RPM; considerando además que cada revolución genera una cierta cantidad de pulsos de los cuales sólo debe detectarse el primero. Se puede ver de la figura 4.3, que ese tren de pulsos de una misma señal tiene un tiempo de vida de un poco menos de 10 ms, por lo que al detectar el primer pulso se deben ignorar los siguientes hasta que se detecte el siguiente tren de pulsos.

Se diseñó la rutina que determina el período de las RPM mediante los contadores propios del microcontrolador, capturando de estos el tiempo del inicio de cada pulso detectado e implementando un retardo de 12 ms para ignorar los pulsos generados de ese tren de pulsos, dicho código se muestra en el apendice **??**

En el código, se definió un límite de tiempo para que el microcontrolador, al igual que en la adquisición de velocidad, de manera que entre en un ciclo infinito y tenga una visualización adecuada.

Esta subrutina es parecida a la de la velocidad, debido a que se desea una medición mínima de 700 RPM, la medición máxima está definida por la velocidad de muestreo que depende de la estructura de la rutina y de la velocidad de procesamiento del microcontrolador. Esta medición máxima tiene un valor muy grande como para ser considerada.

Este código se simuló mediante el montaje de la figura 4.14, el cual presentaba un pulso de entrada de 3 ms de ancho y un período que para este caso es variable ya que representan las RPM que se deseen determinar.

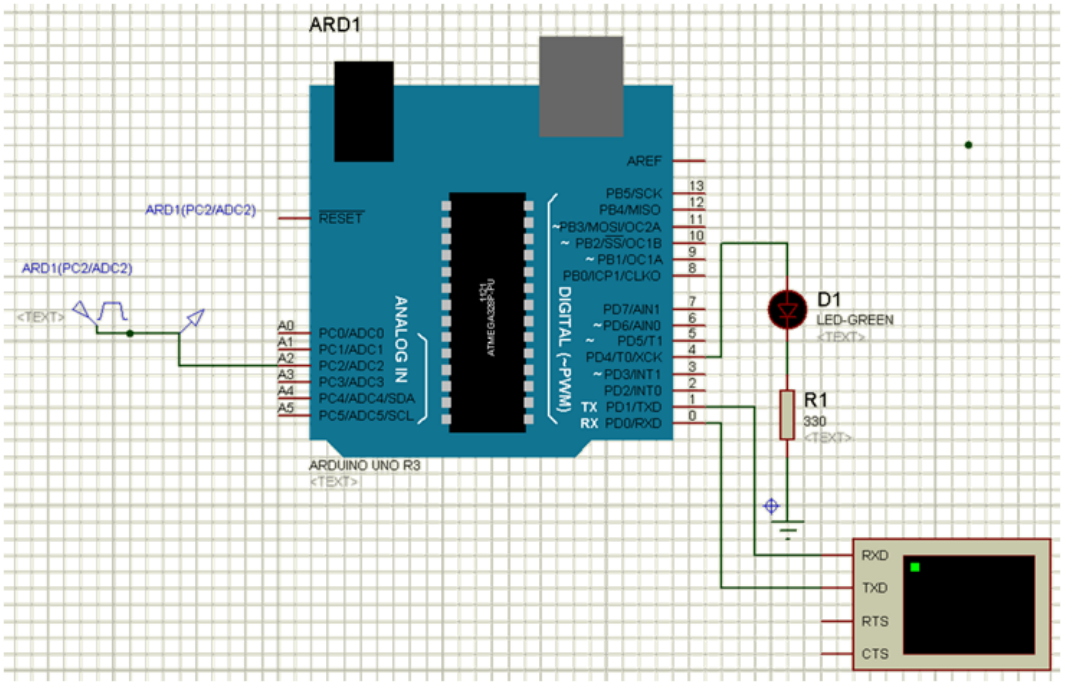


Figura 4.14: Esquema de simulación del sistema de obtención de las RPM.

En este montaje se simularon pulsos con diferentes períodos para observar si se adquieren los valores correspondientes y el error asociado a estos. La tabla 4.11 muestra los resultados de las distintas mediciones que representan unas 30 iteraciones.

Este error es aceptable debido a que generalmente el rango de resolución de las RPM en instrumentos típicos de medición es del orden de las centenas, y al obtener

Tabla 4.11: Resultados de la simulación del sistema de obtenciónd de RPM.

Valor en prueba (RPM)	Valor promedio obtenido (RPM)	Error (RPM)
700	0	0
702	701,5	-0,5
1500	1499,5	-0,5
2800	2799,5	-0,5
3600	3599,5	-0,5

un rango de error de orden de unidades lo hace suficientemente preciso para esta aplicación.

4.3.1.4. Monitoreo de la aceleración.

Debido a que mediante el uso del acelerómetro se puede tener una medición en cuanto a la dirección de la gravedad, se podría hacer un cálculo que determine la inclinación del prototipo con respecto a sus distintos ejes, si es de interés para un tipo de estudio determinado.

En el diseño de la rutina se tuvo una resolución de 1024 valores en el rango de medición por parte del ADC, mediante simples reglas de 3 o de equivalencias y conociendo los valores estáticos del acelerómetro se pueden obtener las siguientes relaciones:

$$\mathtt{Volt} = \frac{(5\mathrm{V}) * \mathtt{Val}}{1024} \tag{4.1}$$

$$\mathtt{Acc} = \frac{[\mathtt{Volt} - \mathtt{Volt}(0\mathrm{g})]}{\mathtt{Sens}} \tag{4.2}$$

Donde:

$\mathtt{Val}$ es el valor leído por el ADC del Arduino [adim].

$\mathtt{Volt}$ es el voltaje correspondiente al valor leído por el Arduino [V].

$Volt(0g)$ es el voltaje del eje X del acelerómetro a 0g [V].

$Sens$ es la sensibilidad del eje X del acelerómetro, que es igual a $Volt(1g)/1g$ [V/g]

En la última relación se expresa la forma de obtener la aceleración en el eje X con respecto al voltaje en 0g y la sensibilidad de dicho eje.

Es importante mencionar que al momento de la implementación, se tuvo que tener en cuenta que el eje X del instrumento de medición estuviera alineado con el eje del automóvil del cual se quiere saber dicha aceleración, de lo contrario la lectura era incorrecta, ya que al no estar alineado se introduce un error que corresponde a la medición de la gravedad (siendo 1 g = 9.80665 m/s^2), y la unica forma de excluirla es tomando en cuenta que el eje de medición debe estar perpendicular al vector gravedad.

Utilizando el algoritmo de la sección 3.2.1.4 como guía, se ubtuvo la aceleración del prototipo y obtener el código final, que se muestra en el apendice **??**

Este presenta una estructura corta con respecto a las anteriores y mucho más rápida también.

Dicho código se simuló en un montaje en Proteus, el cual tiene en la entrada analógica un voltaje fijo que representa 1g, es decir 0,332 V por encima del voltaje de salida para cuando existe 0g. Dicho montaje se muestra en la figura 4.15.

Al configurar el montaje con todos los datos correspondientes antes mencionados, el código de programación del Arduino y la interfaz serial con una tasa de baudios correcta se tiene el resultado de la figura 4.16.

Se observa que el resultado obtenido es el esperado, dicho resultado debe ser cambiado a formato hexadecimal a la hora de transmitir estos datos emulando al ELM327.

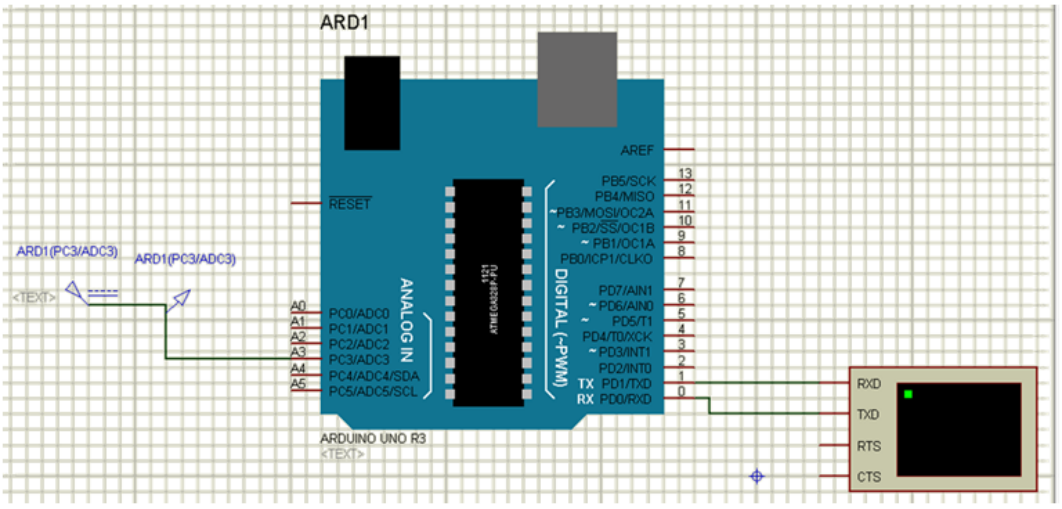


Figura 4.15: Esquema de simulación del sistema de obtención de la Aceleración.

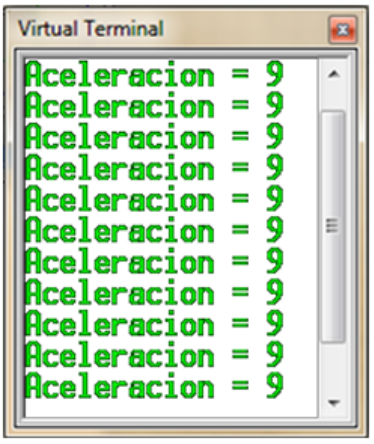


Figura 4.16: Resultado de la simulación del sistema de obtención de la aceleración para 1g.

4.3.2. Emulación del ELM327 y del protocolo OBD.

La emulación consta de un ELM327, el cual estaría conectado a una sola ECU (la ECU 1 del automóvil en este caso), esto tiene que ver con el identificador en el *header* del mensaje de respuesta de esta ECU, el cual siempre será el mismo.

El envío de la trama de datos OBD consiste en una serie de Bytes interpretados en pares de caracteres hexadecimales, a modo ilustrativo en los ejemplos mostrados

se separarán los Bytes por espacios.

4.3.2.1. Emulación del ELM327:

Como se dijo anteriormente, el microcontrolador ELM327 se comunica con las PCs mediante comandos AT, dichos comandos son los que establecen, mantienen y configuran la comunicación entre el microcontrolador y la PC (adicionales a estos están los comandos OBD que se explicarán en la sección siguiente). Entonces, se define en este punto la emulacion del ELM327 como la configuración del Arduino que puede establecer, mantener y configurar la comunicación de manera básica. Para algunos comandos la PC no nota diferencia o cambio alguno en la comunicación cuando se ejecutan ciertos comandos AT, ya que estos sólo configuran la conexión ELM-ECU que es distinta a la comunicación ELM-PC. Además no se emularon todos los comandos ya que para los *software* probados no existe la presencia del resto de estos comandos lo que los hace innecesarios para este caso.

Cabe destacar que para los comandos AT en los que no se presente cambio alguno visto desde la PC para efectos de la emulación y cálculo de parámetros. Se configuró para enviar la respuesta OK para que el software interprete que el cambio que solicitaba se hizo, aunque dicho cambio no presente ningún efecto en la rutina del Arduino o en la interpretación de los datos en la PC. Cualquier letra enviada desde y para el ELM327 no distingue entre mayúsculas y minúsculas.

Los comandos obtenidos, luego de verificar cuales comandos emitía cada *software*, son los descritos en el apéndice A.

La emulación de estos comandos logra que los distintos *software* se comuniquen con el Arduino, de manera que el *software* crea que se comunica con un ELM327 mediante una sesión básica, ya que el conjunto de comandos es limitado con respecto al real, pero es suficiente para lograr la comunicación.

Ya teniendo los comandos descritos para la configuración básica, el código para la emulación de un ELM327 es la primera parte del código mostrado en el apéndice B. Dicho código sirve para iniciar una sesión con algunos de los software de visualización diseñados para usarse con un ELM327.

4.3.2.2. Emulación del protocolo OBD:

En la emulación de la ECU mediante el protocolo CAN OBD de 11 bits (ISO 15765-4), se debe especificar el header, considerando como se dijo antes que existe una sola ECU en la red CAN. Las tramas OBD enviadas por el Arduino (excepto las de solicitudes de multiples PIDs), son iguales siguiendo la estructura de una sola trama, ya que existen 4 tipos de tramas que son: una sola trama, primera trama, trama consecutiva y control de flujo; dichas estructuras siguen el formato de la figura 4.17. Ya se dijo que el *checksum* no es visible para la PC, por lo que no es necesario considerarlo ya que queda fuera de la comunicación PC-ELM327 (en este caso PC-Arduino), y es parte de la comunicación ELM327-ECU.

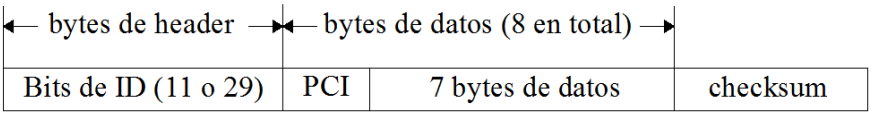


Figura 4.17: Mensaje OBD CAN [8].

Todos los otros elementos dentro de la estructura del mensaje CAN OBD deben ser considerados. Los 11 bits de *header* son el identificador CAN que para cada elemento dentro de la red CAN cambia, éstos no son mas que las direcciones físicas que se les dan a los distintos dispositivos dentro de una red CAN, ya que como se simuló 1 sola ECU, se tiene un identificador único el cual no cambia. Los distintos identificadores que existen se muestran en la tabla 4.12.

Se escogió que la ECU (Arduino) en este caso fuera el #1, tomándose el identificador del *header* 7E8. Los demás fueron ignorados ya que no se necesitan emular otras ECUs y al no existir la comunicación ECU-ELM se ignora el *header* de la herramienta de prueba.

Tabla 4.12: Identificadores CAN OBD de 11 bits.

Identificador CAN	Descripción
7DF	Identificador CAN para un mensaje de solicitud con direccionamiento funcional enviado por el equipo de prueba externo (equipo de diagnóstico).
7E0	Identificador CAN para una solicitud física desde la herramienta de prueba externa hacia la ECU #1.
7E8	Identificador CAN para una respuesta física desde la ECU #1 hacia la herramienta de prueba externa.
7E1	Identificador CAN para una solicitud física desde la herramienta de prueba externa hacia la ECU #2.
7E9	Identificador CAN para una respuesta física desde la ECU #2 hacia la herramienta de prueba externa.
7E2	Identificador CAN para una solicitud física desde la herramienta de prueba externa hacia la ECU #3.
7EA	Identificador CAN para una respuesta física desde la ECU #3 hacia la herramienta de prueba externa.
7E3	Identificador CAN para una solicitud física desde la herramienta de prueba externa hacia la ECU #4.
7EB	Identificador CAN para una respuesta física desde la ECU #4 hacia la herramienta de prueba externa.
7E4	Identificador CAN para una solicitud física desde la herramienta de prueba externa hacia la ECU #5.
7EC	Identificador CAN para una respuesta física desde la ECU #5 hacia la herramienta de prueba externa.
7E5	Identificador CAN para una solicitud física desde la herramienta de prueba externa hacia la ECU #6.
7ED	Identificador CAN para una respuesta física desde la ECU #6 hacia la herramienta de prueba externa.
7E6	Identificador CAN para una solicitud física desde la herramienta de prueba externa hacia la ECU #7.
7EE	Identificador CAN para una respuesta física desde la ECU #7 hacia la herramienta de prueba externa.
7E7	Identificador CAN para una solicitud física desde la herramienta de prueba externa hacia la ECU #8.
7EF	Identificador CAN para una respuesta física desde la ECU #8 hacia la herramienta de prueba externa.

El campo PCI es un Byte representado en un par de caracteres hexadecimales, que indican cuantos Bytes existen en el mensaje dentro del campo Byte de datos, excluyendo en la numeración este mismo segmento (PCI). Dado que se pueden mandar como máximo 7 Bytes de datos, este campo (PCI) tiene un valor máximo de 07, excepto para el caso de tramas consecutivas del cual se habla más adelante.

Dentro del campo de Bytes de datos se encuentran los modos y los PIDs, que hacen posible la comunicación y el envío de los parámetros.

Los modos de operación están descritos en el estándar SAE J1979, cada uno estos modos presentan información distinta en cuanto al diagnóstico del automóvil, entre ellas la visualización real de los sensores en el automóvil, la verificación de códigos de diagnostico (DTCs, *Diagnostic Trouble Codes*) y la información sobre el vehículo. Estos modos van del 01 al 0A y no es un requerimiento que los automóviles soporten todos los modos.

Los diferentes modos son los siguientes:

Modo 01: Muestra información actual de los sensores.

Modo 02: Muestra datos de imagen congelada.

Modo 03: Muestra códigos de diagnostico almacenados.

Modo 04: Borrar códigos de diagnostico y valores almacenados.

Modo 05: Resultados de las pruebas, monitoreo del sensor de oxígeno (no sólo CAN).

Modo 06: Resultados de pruebas, otros componentes/monitoreo del sistema (resultados de las pruebas, el monitoreo del sensor de oxígeno para CAN solamente).

Modo 07: Mostrar los códigos de diagnóstico pendientes (detectados durante el ciclo de conducción actual o el último).

Modo 08: Control de operación de componentes/sistemas abordo.

Modo 09: Solicitud de información del vehículo.

Modo 0A: Códigos de diagnóstico Permanentes (DTCs).

El modo 01 es el único modo de interés para la emulación, por lo tanto, fue el único que se estudió, ya que los datos que se enviaron mediante el sistema de telemetría son en tiempo real y no se almacenan.

Los PIDs son los identificadores de los parámetros que se desean obtener. Dentro del modo 01 se encuentran una serie de PIDs, que se muestran en el apéndice C. Por ejemplo, suponiendo que el software desea conocer cuales PIDs soporta el modo 01, se le envía al Arduino la solicitud 01 00, dicha respuesta debe tener el formato que se muestra en la tabla 4.13.

Por lo que la respuesta para el comando 01 00, en el caso mostrado en la tabla sería:

41 00 BE 1F A8 13

Donde el 41 representa la respuesta al modo 01, es decir una respuesta al comando 02 00 comenzaría con el par de caracteres hexadecimales 42, una respuesta al modo 03 00 comenzaría con 43, y así sucesivamente. El Byte 00 representa el número del PID específico por el cual se pregunta y los demás Bytes son la respuesta en sí del PID que fue solicitado, que para el caso del PID 00, representa el cunjunto de PIDs que van desde el 01 hasta el 20 (en hexadecimal), en los cuales cada uno representa una variable o un parámetro del vehículo, el cual puede ser soportado o no. La respuesta a esta solicitud contiene cuales PIDs del conjunto son soportados por el vehículo. En cambio para una solicitud de un parámetro en específico, se obtiene como respuesta el valor de dicho parámetro. Es decir, una respuesta para el comando 01 0C que serían las RPM del vehículo, podría ser 41 0C 0F A0, en donde los parámetros 0F A0 representan 1000 RPM, mientras que los parametros 41 y 0C representan la respuesta al modo 1 y la respuesta al PID 0C (RPM), respectivamente.

Cabe destacar que al momento de inicializar la comunicación se inicializan todos los PIDs que son soportados por el vehículo y el software los registra para su

Tabla 4.13: Ejemplo del formato de los PIDs.

Hexadecimal	B				E				1				F				A				8				1				3			
Binario	1	0	1	1	1	1	1	0	0	0	0	1	1	1	1	1	1	0	1	0	1	0	0	0	0	0	0	1	0	0	1	1
¿Soportado?	Sí	No	Sí	Sí	Sí	Sí	Sí	No	No	No	No	Sí	Sí	Sí	Sí	Sí	Sí	No	Sí	No	Sí	No	No	No	No	No	No	Sí	No	No	Sí	Sí
Número de PID	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	20

lectura. De no ser soportado alguno de estos PIDs, el software al momento de inicializar detecta los que son soportados y toma en cuenta únicamente a éstos, es decir que no habrán mensajes de solicitud de PIDs de alguno de los sensores o parámetros que no sea soportado.

Debido a que el protocolo está diseñado para una gran cantidad de sensores y diagnóstico de parámetros, existen varios conjuntos de PIDs, no sólo el 01 00 sino también los conjuntos 01 20, 01 40, 01 60, 01 80, 01 A0 y 01 C0. Éstos se muestran en detalle en el apéndice C.

Para este trabajo de grado se tendrá que en la inicialización, al *software* que se use para comunicarse con el emulador se le deben proporcionar los conjuntos de PIDs 01 00, 01 20 y 01 40 (ya que los parámetros deseados para la implementación se encuentran en estos PIDs). Las respuestas correspondientes a estas solicitudes serán los PIDs soportados por el sistema de telemetría, dichas respuestas deben cumplir con el formato de la tabla 4.13, teniéndose que la respuesta para el comando 01 00 es:

41 00 00 18 00 01

Se observan que los PIDs 0C, 0D y 20 del modo 1 son aceptados por el vehículo, dichos PIDs representan respectivamente las RPM, la velocidad y que la ECU acepta uno o más PIDs del rango 21 al 40. Debido a que el PID 20 está activo en dicho sistema, el *software* en su intento de conocer todos los sensores o parámetros de los que dispone el vehículo, envía el comando 01 20 al cual se le responde de la siguiente manera:

41 20 00 02 00 01

Lo que da a conocer que los otros PIDs soportados son 2F y 40, que representan el nivel del combustible de entrada y que alguno de los PIDs del 41 al 60 está activo. Para este último, al igual que la solicitud 01 20, se realiza una solicitud 01 40 para obtener información sobre los PIDs aceptados en ese rango, cuya respuesta es:

41 40 00 00 00 40

Lo cual implica que el único PID aceptado de este rango es 5A que es la posición relativa del pedal del acelerador. Esta última variable se puede usar como la aceleración neta mediante un simple cambio de unidades y siempre que el usuario de la PC tenga conocimiento de esto. Al no estar activo el PID 60 dentro de la respuesta, el *software* de manera directa entiende que estos son los únicos parámetros del modo 01 del vehículo que puede obtener. En resumen los únicos PIDs soportados por este sistema son: 0C, 0D, 2F y 5A.

La presencia del campo *header* en la respuesta la configura el software al momento de la inicialización. Al introducir el *header* y el campo PCI se tiene que para la respuesta del PID 00 del modo 01, quedaría de la siguiente forma:

7E8 06 41 00 00 18 00 01

Ya que la cantidad de bytes es de 6 el campo PCI toma el valor 06, de tener otra cantidad el campo PCI tomaría ese valor.

Para la solicitud de multiples PIDs se pueden obtener respuestas de 2 tipos, la primera es una respuesta sencilla de una sola trama que depende de la cantidad de parámetros que se soliciten dentro del comando OBD que ejecute el *software*. La cantidad máxima de parámetros (PIDs) que pueden solicitarse dentro de esta implementación para que la respuesta permanezca como una de una sola trama son 2. En este caso la solicitud de multiples PIDs para 2 parámetros tiene el mismo formato antes visto, pero incluyendo ambos PIDs en la solicitud sin importar el orden, siendo de las siguientes maneras:

01	2F	0C
Modo	PID 1	PID 2
	o	
01	0C	2F
Modo	PID 1	PID 2

Para multiples PIDs se puede solicitar cualquier PID establecido en la inicialización. La respuesta a esta solicitud es de una sola trama, en la cual se debe seguir el orden en el que se solicitan. Incluyéndose el PID 1 seguido de su valor y luego el PID 2 seguido de su valor, como se muestra a continuación:

Solicitud

01	2F	0C
Modo	PID 1	PID 2

Respuesta sin headers

41	2F	3F	0C	17	B8
Respuesta del modo 1	PID 1	Valor del PID 1	PID 2	Valor del PID2	

Respuesta con headers activos

7E8	06	41	2F	3F	0C	17	B8
Header	PCI	Respuesta del modo 1	PID 1	Valor del PID 1	PID 2	Valor del PID2	

Para el caso de 3 o más PIDs que se soliciten la respuesta cambia a una de primera trama seguida de respuestas de trama consecutiva. Como en este sistema sólo hay 4 parámetros se estudiará este caso, en el cual se tiene una respuesta de primera trama seguida por una sola respuesta de trama consecutiva. La solicitud conserva el mismo formato que para 2 PIDs, solo que se incluyen los otros 2, recordando que no importa el orden, es decir que una solicitud podría ser:

01	2F	0D	0C	5A
Modo	PID 1	PID 2	PID 3	PID 4

La respuesta en el caso que los *headers* estén desactivados, lo primero que se envía es la cantidad de Bytes en la respuesta (PCI) seguido de un salto de línea, posteriormente para cada trama, sea la primera o la consecutiva, se envía el identificador de línea seguido por los Bytes de respuesta, con una retorno de acarreo

por cada línea. El campo de respuesta al modo que se solicita va incluido en la respuesta de primera trama. Es decir que una respuesta a la solicitud anterior podría ser:

```
00A
0:  41  2F  3F  0D  3F  0C
1:  17  B8  5A  21  00  00  00
```

Nótese que si se termina un mensaje y la respuesta a un parámetro no ha terminado se continúa en la siguiente línea. Las líneas deben estar identificadas ya que en redes CAN reales puede que lleguen los mensajes con un orden distinto. Los bytes adicionales "00"son ignorados por el *software*, aquí se usan de manera ilustrativa aunque cabe destacar que los ELM327 envían estos Bytes. Además el campo PCI se representa con 3 dígitos hexadecimales al principio como se muestra.

Para la misma respuesta pero con los *header* activos se le añade un Byte al *header* que especifica el tipo de mensaje de esa línea, ya sea de primera trama (10) o de trama consecutiva (21,22,23,...), y el campo PCI va luego del *header* en la respuesta de la primera trama y esta vez se representa con 2 dígitos. La respuesta con los *headers* activos sería la siguiente:

```
7E8  10  0A  41  2F  3F  0D  3F  0C
7E8  21  17  B8  5A  21  00  00  00
```

Para este trabajo sólo se aplica la respuestas con *headers* ya que los *software* recomendados usan sólo comunicación con *header*.

El código que emula todo lo descrito en esta sección para este sistema de telemetría es la segunda parte del mostrado en el apéndice B. Éste es la continuación de la programación básica para emular el ELM327, dicho código al complementarlo con los códigos para la obtención de cada parámetro muestra todos los parámetros adquiridos por el sistema de telemetría. Las variables en este caso se enviaron de manera constante, pero el código final tiene un llamado a las rutinas de obtención de parámetro, para cada uno en específico. Dicha comunicación se simuló mediante el montaje de la figura 4.18.

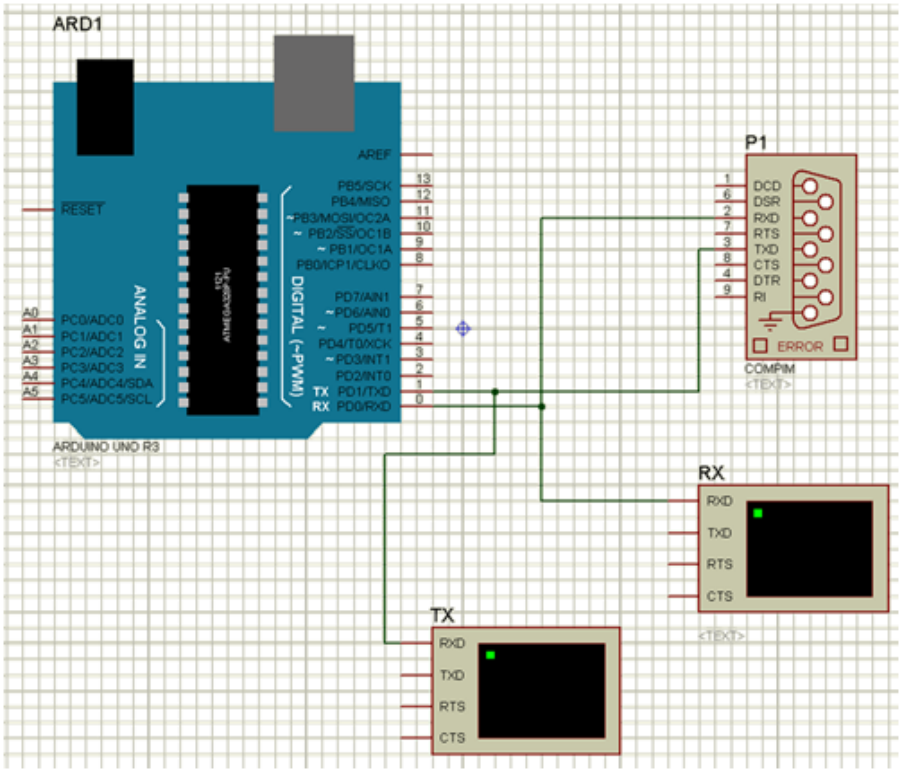


Figura 4.18: Esquema de simulación para el código emulador del ELM327 y OBD.

El puerto P1 simuló un puerto RS232, que podía estar asociado a otro puerto virtual o a uno físico. Los terminales virtuales TX y RX que sirvieron para monitorear la comunicación entre el Arduino y lo que se conecte a dicho puerto. La comunicación para la simulación se logró haciendo uso de un programa que genere puertos virtuales en la PC y los asocie. Por lo que el Arduino virtual se conectó a un puerto y el programa OBD para probar el Arduino se conectó en el otro puerto asociado. Para la simulación del ELM327 se obtuvo como resultado en la inicialización con el programa ODB Auto Doctor el comportamiento de la figura 4.19.

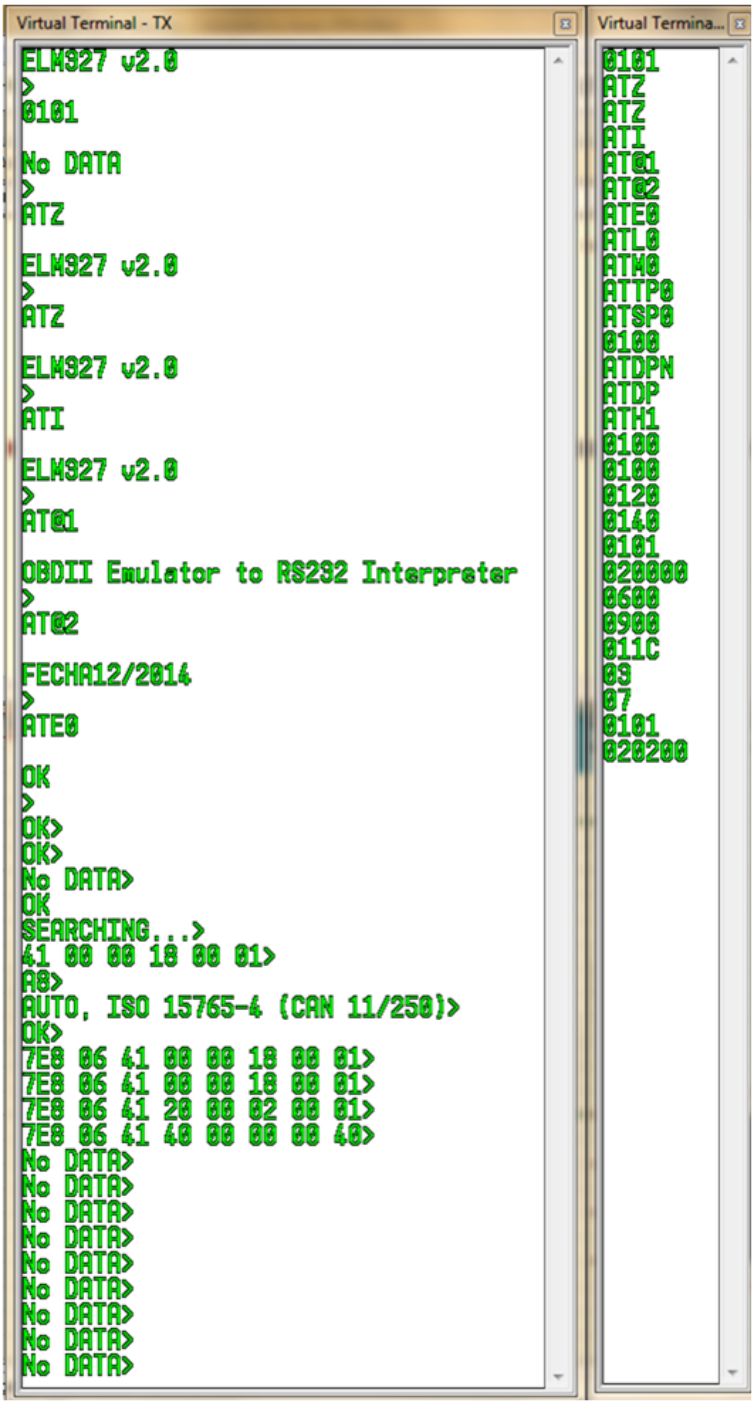


Figura 4.19: Resultado de la simulación de la inicialización de ELM y OBD mediante el *software* OBD Auto Doctor.

Este resultado es el que se obtuvo para una sesión básica del software conectado

con el Arduino virtual y después corroborando con el Arduino UNO físico. Luego de realizar esta inicialización se pueden adquirir los parámetros, que en dicha inicialización se presenten. La tabla 4.14 muestra los programas con los que se probó dicho código y especifica con cual se pudo lograr la conexión y con cual no.

Tabla 4.14: Softwares probados para la simulación.

Software	Conectividad
Multiecu Scan	No
Easy OBDII	Sí
OBD 2007	Sí
Digimoto 4.30	Sí
wOBD	Sí
ProScan 5.9	Sí
ForScan	No
OBD Auto Doctor	Sí
Dyno-Scan	No
EOBD-Facile	Sí
EngineCheck LE	Sí

Como se observa en la tabla la emulación logró iniciar sesión con aproximadamente el 72 % de los *software* probados, en el caso de Dyno-Scan al parecer la velocidad que utiliza el programa es desconocida ya que se probó con las velocidades de transmisión (tasa de baudios) de 1200, 2400, 4800, 9600, 19200, 38400 y 57600 baudios; a ninguna velocidad se pudo distinguir mediante los terminales virtuales algún comando entre los caracteres recibidos. El *software* Multiecu Scan fue capaz de detectar el emulador a la hora de hacer una prueba de escaneo de puerto (que es una opción que posee el mismo *software* para determinar si hay alguna herramienta de prueba conectada al puerto), pero en el momento de realizar la conexión no logró completar la inicialización de la comunicación. Por su parte el *software* ForScan enviaba los comandos de manera correcta al emulador, pero al momento de recibir la respuesta del sistema emulador no recibía ninguna, esto podría deberse al tiempo de espera que se estipuló en el código del Arduino para que determine cuando se terminó de recibir la data mediante la interface serial desde que recibió el primer Byte.

4.3.2.3. Softwares recomendados para la visualización:

De los *software* probados, se escogieron los que tuvieran una interfaz gráfica mas completa, además de que la visualización de variables fuera la mejor posible y que las gráficas obtenidas de los parámetros se puedan almacenar; dichos *software* resultaron ser el OBD Auto Doctor y el EOBD-Facile. Estos cumplen con lo antes mencionado, aunque no son totalmente gratuitos y debe cancelarse una licencia para poder usar todas las características de los *software*. Estos son los únicos de los estudiados que grafican los datos que se obtienen, el OBD Auto Doctor grafica un solo parámetro a la vez mientras que el EOBD-Facile puede graficar hasta 4 parámetros en una misma gráfica, lo cual sería útil para observar todos los parámetros que se plantean en este sistema de telemetría propuesto.

Los autores entregarán una licencia para el Software EOBD-Fácile a la organización SAE UC. Cabe destacar que la visualización de los 4 parámetros se hace mediante una solicitud de múltiples PIDs.

4.3.2.4. Comunicación mediante los Xbees

Debido a que la comunicación mediante los Xbees es una comunicación serial transparente al estar configurados de punto a punto, es decir lo que se envía desde un Xbee se recibe sin ser modificado en el Xbee receptor, la comunicación mediante los Xbees simula una conexión serial directa como si el Arduino estuviera conectado directamente con la PC; además de que el mensaje al ser enviado no necesita ningún tipo de direccionamiento o encabezado.

Es decir que el código de programación ya mostrado no necesita modificación alguna, además del envío serial directo por el puerto predeterminado del Arduino (puestos 0 y 1), para el envío de los datos mediante el Xbee y se trabaja como si se tuviera una comunicación serial.

4.3.3. Configuración de los Xbees.

La transmisión de los datos se realizó en modo transparente o modo comando AT, ya que el software intérprete o de diagnóstico requiere una comunicación serial transparente para establecer la conexión con el Arduino UNO.

Para conectar los módulos a la PC y configurarlos, se debe tener instalado el software gratuito XCTU. Adicionalmente fue necesario instalar el controlador del Explorer USB serial, de no realizarse la descarga automática en Windows se pueden descargar de la página oficial de Digi.

Para la configuración se debe proceder de la siguiente forma: Se debe iniciar XCTU y aparecerá la ventana del software, en esta se tienen dos opciones, agregar un módulo de radio especificando los ajustes del puerto o indicarle al software que descubra automáticamente los módulos de radio conectados a la máquina. En este caso, se selecciona la opción de descubrir de manera automática como se muestra en la figura 4.20, en la que saldrá la ventana «*Add a radio module*», en esta ventana se deben especificar los parámetros del puerto, por lo que se debe saber que los módulos XBee tienen una tasa de transmisión serial de 9600 baudios preestablecida de fábrica, de no ser así ocurrirán errores en los que se les pide al usuario reiniciar los módulos cortocircuitando el pin *Reset* con tierra.

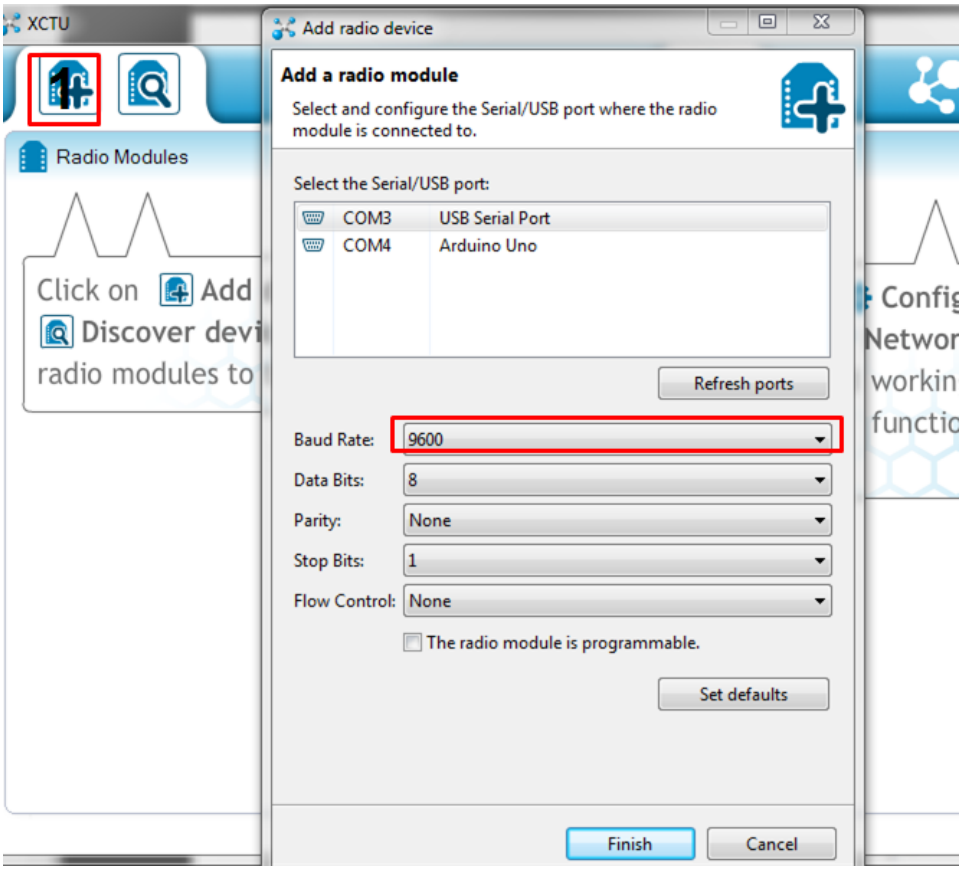


Figura 4.20: Ventana principal del software X-CTU.

Al estar conectados a los dos módulos, es importante configurarlos de tal forma que estén en la misma dirección de red (network ID) de lo contrario estos no podrán establecer comunicación entre ellos, en cuanto al modo de transmisión no es necesario cambiarlo, ya que por defecto está en el modo AT que es necesario para la transmisión de los parámetros. Haciendo clic sobre el módulo se obtuvo toda configuración actual, se modificó la network ID, para que sea la misma en ambos. En la figura 4.21 se puede observar lo antes expuesto. Además se puede agregar encriptación, modificar el nivel de potencia de la señal, aplicar los ajustes necesarios y finalmente escribir los datos sobre el módulo.

Adicionalmente, se pueden cambiar las direcciones de destino por la MAC del módulo al que se quiere enviar, con esto se asegura que la comunicación sea punto a punto, incluso si los módulos entran a una red que tenga la misma dirección de red

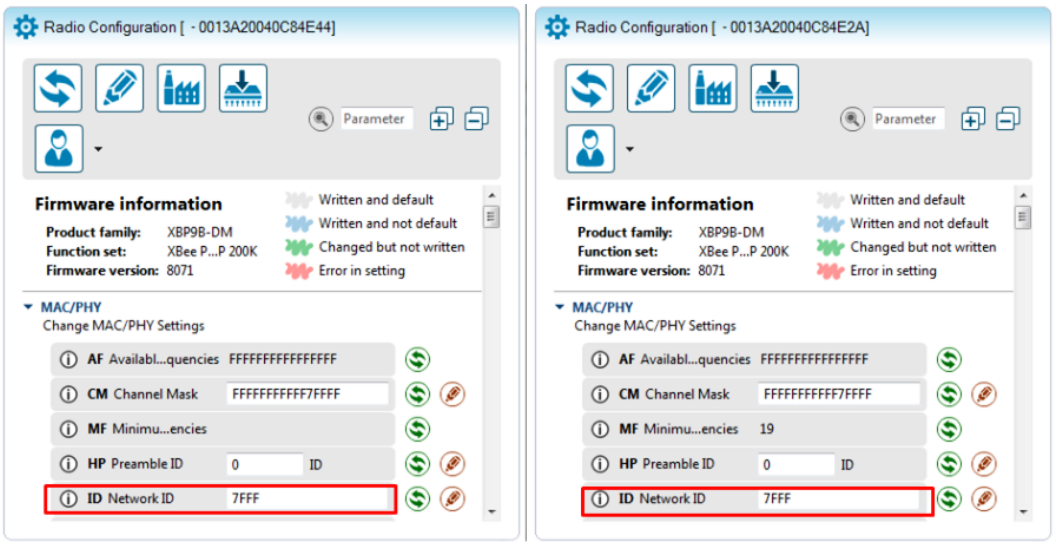


Figura 4.21: Establecimiento del mismo número de network ID para ambos módulos.

antes configurada. Esta configuración de dirección de destino se puede observar en la figura 4.22.

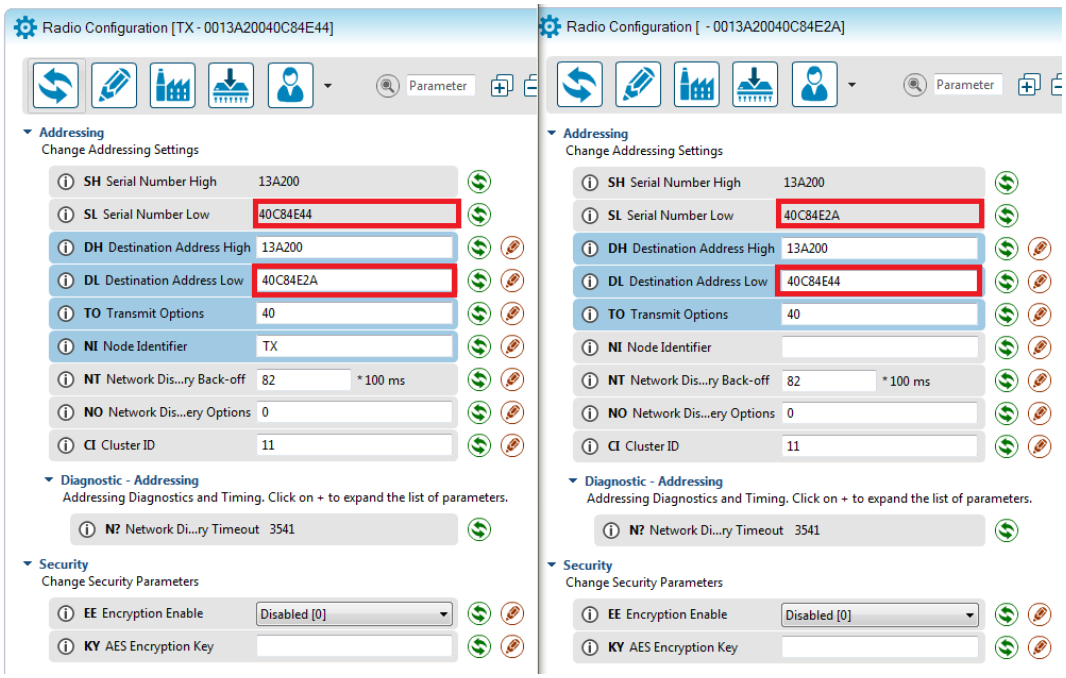


Figura 4.22: Configuración de una dirección única de envío.

Para las pruebas se debe hacer clic en los recuadros rojos que aparecen en la

figura 4.23, en el orden de su enumeración, luego de esto ya se ha abierto la comunicación con cada módulo de radio y así comenzar a transmitir en RF. Para probar que los módulos se comuniquen entre sí, se envió un mensaje desde un módulo a otro. En la figura 4.23 se muestran las 2 interfaces, en donde la escritura azul y roja son el paquete transmitido y recibido, respectivamente.

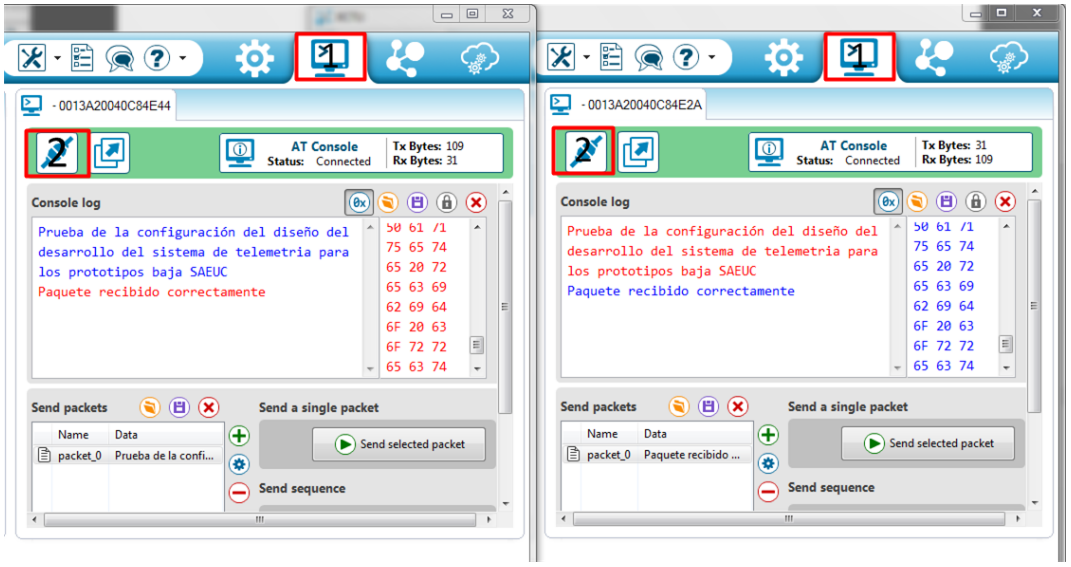


Figura 4.23: Visualización de paquete enviado en azul y recibido en rojo de los módulos respectivos.

Luego de haber hecho esta configuración, se pueden comunicar estos módulos, sólo el uno con el otro y siendo incapaces de comunicarse con otro módulo, asi esté dentro de la misma red. Adicionalmente, se debe señalar que al energizar los módulos, estos necesitan un comando antes de que entren en el modo de comando AT, sin esto no puede lograrse la comunicación. Al energizar los módulos se debe esperar unos segundos (al rededor de 5 segundos), luego se debe enviar un salto de línea para que el módulo muestre las distintas opciones de un menú siguiente:

B-Bypass
F-Update App
T-Timeout
V-BL Ver.
A-App Ver.
R-Reset

La opción "B-Bypass.[es] la que hace que el módulo trabaje en modo serial transparente, por lo que se debe enviar el caracter "B", mediante la comunicación serial. Debido a esto en la inicialización del programa (en la sección del comando void **setup**()), lo primero que se envía al Xbee es un salto de línea y el caracter "B", antes de seguir con la programación y el ciclo iterativo.

4.3.4. Configuración del HC-06.

La configuración de este módulo Bluetooth se hizo por medio de una comunicación serial entre el Arduino y dicho módulo. Ya que la comunicación entre el Arduino y el Xbee hará uso de los puertos 0 y 1 del Arduino, se definieron otros 2 puertos adicionales (Rx y Tx para el HC-06), esto se hizo mediante el uso de la librería SoftwareSerial de Arduino y definiendo estos puertos luego de hacer el llamado de la librería al comienzo del programa. Es decir que las primeras líneas del código final fueron las siguientes:

```
#include <SoftwareSerial.h>
#define RxD 8
#define TxD 9
SoftwareSerial Bluetooth(RxD, TxD);
```

Luego de haber configurado estas entradas como el puerto de conexión con el módulo Bluetooth, se usa este al igual que el puerto serial que ya viene configurado en el Arduino por defecto (puertos 0 y 1), pero en vez de escribir la palabra "Serial" en los comandos se debe escribir como se nombró a ese puerto en la configuración, es decir "Bluetooth". Cabe destacar que este puerto se inicializa al igual que el puerto que viene por defecto en el Arduino.

Al haber definido el puerto serial <<Bluetooth>> y al estar terminada la conexión cableada entre el Arduino y el HC-06, se procedió a configurar el módulo Bluetooth el cual acepta 5 comandos, que son los mostrados en la tabla siguiente.

Las velocidades en las cuales se pueden configurar el módulo HC-06 son las mostradas en la tabla 4.16.

Tabla 4.15: Características y Resultados de la simulación

Comando	Descripción	Respuesta
AT	Se utiliza para corroborar que se puede acceder al modo AT	OK
AT+VERSION	Muestra la versión del Firmware	OKLinvorV1.8
AT+BAUD4	Cambia la velocidad del puerto serial del módulo, donde 4 corresponde a 9600 Baudios (Tabla 4.16)	OK9600
AT+PIN1234	Cambiar la contraseña, donde 1234 representa la nueva contraseña	OK1234
AT+NAMESAE UC	Cambia el nombre a mostrar del dispositivo, donde SAE UC es el nombre nuevo	OKsetname

Tabla 4.16: Características y Resultados de la simulación

Comando	Velocidad	
AT+BAUD1	1200	
AT+BAUD2	2400	
AT+BAUD3	4800	
AT+BAUD4	9600	(Por defecto)
AT+BAUD5	19200	
AT+BAUD6	38400	
AT+BAUD7	57600	
AT+BAUD8	115200	
AT+BAUD9	230400	
AT+BAUDA	460800	
AT+BAUDB	921600	
AT+BAUDC	1382400	

De estos comandos, el único usado fue AT+NAMESAE UC, que cambió el nombre a mostrar del dispositivo por "SAE UC". La configuración se hizo mediante el envío del comando anteriormente dicho al puerto Bluetooth mediante una rutina del Arduino, esta configuración debe hacerse sin que el módulo Bluetooth esté apareado a otro dispositivo.

4.3.5. Programación final del Arduino.

Para la programación final del Arduino se incorporaron los códigos de las rutinas para cada parámetro en el código que emula el protocolo OBD, ya que este es el que responde a la solicitud de dichos parámetros. La comunicación está incorporada dentro de este código.

Al tratar de unificar todo lo correspondiente a códigos de programación dichos hasta ahora, se presentaron inconvenientes en cuanto al funcionamiento del Arduino. Primero se tenía un código que establecía la conexión OBD, adquiría los parámetros y los enviaba individualmente de manera correcta mediante los Xbees, teniendo un correcto funcionamiento.

Luego al incorporar la comunicación Bluetooth y el envío de multiples parámetros como respuesta a una solicitud de multiples PIDs, el Arduino se comportaba de una manera incorrecta, en la comunicación serial enviaba caractéres que nada tenían que ver con la comunicación ni con el código. Dicho código fue revisado por los autores para su verificación, para corroborar errores de los cuales no se halló ninguno.

Esta eventualidad se había presentado antes en la emulación del ELM, en un principio se tenía un código que emulaba una gran cantidad comandos del microcontrolador ELM con el cual el Arduino se comporataba de manera inesperada, teniendo una respuesta en la comunicación serial que no concordaba con el código. Los autores durante el desarrollo de este código, se dieron cuenta que lo extenso del código o la gran cantidad de instrucciones <<if>> tenía algo que ver con esto, debido a que cada vez que se añadía una cierta cantidad de comandos a emular se corría el código en el Arduino virtual para corroborar el comportamiento de dichos comandos. Se observó que existía un punto en el cual se añadía una instrucción <<if>> adicional y el Arduino se comportaba de manera distinta a la estipulada por el código, por lo que en ese momento se escogió sólo simular los códigos necesarios para para establecer y mantener la comunicación.

Al incidir de nuevo en este escenario, se probó disminuir el tamaño del código final, por lo que se hicieron 2 pruebas, una sin la comunicación Bluetooth y otra sin la respuesta a múltiples PIDs. En ambos casos el Arduino establecía y mantenía la conexión, además de enviar los parámetros correctamente.

Ya que por parte de los autores no se pudo tener un solo código que incorpore todo lo visto hasta ahora y debido que se consideran importantes estos dos aspectos (la comunicación Bluetooth y la respuesta a varios parámetros), se presentan 2 códigos, uno que puede comunicarse mediante Bluetooth a un dispositivo móvil y enviar sólo un parámetro a través de los Xbees, y el segundo que puede comunicarse sólo por Xbees pero con la capacidad de dar respuesta a una solicitud de múltiples PIDs.

Dichos códigos se muestran en el apéndice D. Y queda a criterio del usuario usar el código que mejor convenga en cierto momento.

4.3.6. Simulaciones del enlace de radioeléctrico en pistas anteriores.

Existen 2 zonas de interés para este sistema en las competencias, la primera es el aparcadero en donde se le hacen las reparaciones necesarias al prototipo durante la competencia y donde se encuentra la mayoría del equipo técnico, la segunda zona es la pista donde se realiza el Endurance. Dado que el equipo técnico requiere de un diagnostico oportuno, se considera que la PC que lee los parámetros debe estar lo más cercano al aparcadero mientras que el prototipo esté en la pista. Esto no fue posible en todas las simulaciones, ya que el enlace al no tener línea de vista y al ser muy largo, se tiene una gran atenuación con la cual no se podría tener una comunicación, por lo que para estos casos se varió la posición de la PC dentro del aparcadero o se consideró usarla en el lugar donde estaría posicionado el público.

Las figuras que no se consideren de mayor relevancia se mostrarán en el anexo A, se hará mención a ellas aunque no son del todo concluyentes y representan los primeros estudios realizados para cada pista en los que se obtuvo como resultado que no existiría la comunicación.

Estudio de la pista 1

Esta competencia tuvo lugar en 5000 Warren Rd. Burlington, Wisconsin, EE.UU. en el 2012. En la imagen 4.24 se aprecia la forma del entorno donde se desarrolló, también se observan 2 representaciones dentro del aparcadero del módulo receptor y la representación dentro de la pista del prototipo BAJA como transmisor.

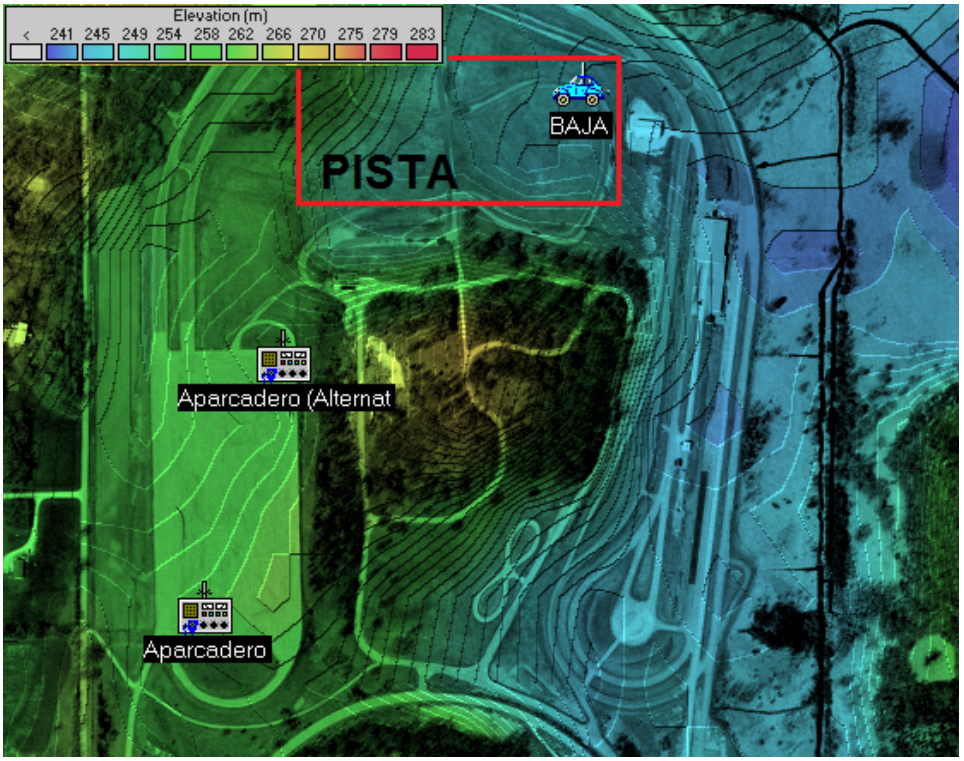


Figura 4.24: Ubicación de los módulos en el mapa de simulación del radioenlace en la pista 1.

Para la primera simulación que correspondía al enlace con mayor distancia entre el aparcadero y la pista, se obtuvo que la cobertura dentro de la pista era baja y para el radioenlace simulado se tenía una pérdida de 133,9 dB; con la cual no existiría comunicación. A modo de referencia, la figura a.1 representaría la cobertura del receptor en el aparcadero y la figura a.2 muestra el resultado de las características del radioenlace simulado.

En la segunda simulación se obtuvo que la cobertura posibilitaba la comunicación y en el radioenlace se obtuvo una pérdida de 91,1 dB; por lo que existiría comunicación. La zona de cobertuda obtenida es la mostrada en la figura 4.25, en la cual se puede observar que en casi todos lo puntos de la pista existe la misma atenuación en el enlace.

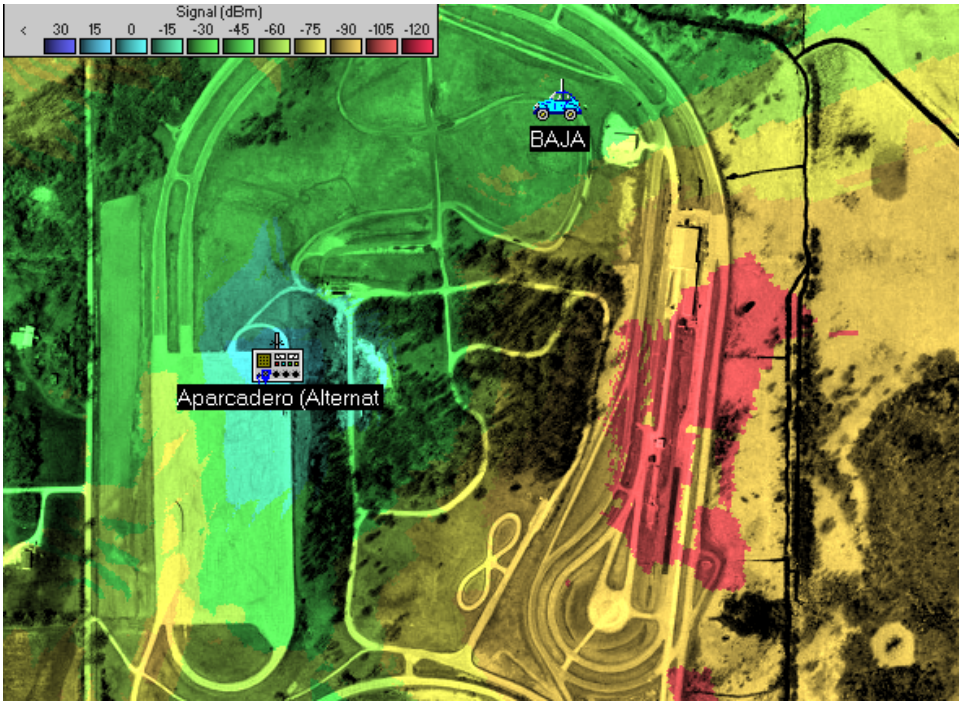


Figura 4.25: Mapa de cobertura del receptor (PC) de la segunda simulación para la pista 1.

Para el radioenlace de esta simulación se tuvieron las características resumidas en la tabla 4.17, estos datos se obtuvieron del programa y la imagen de dichos resultados se muestra en la figura a.3.

Tabla 4.17: Características y resultados de la segunda simulación para la pista 1.

Parámetro	Tx	Rx
Latitud	42,63403°	42,63155°
Longitud	-88,30966°	-88,31307°
Elevación del terreno	249 m	262 m
Altura de antena	1 m	3 m
Azimut	225,34°	45,34°
Inclinación	2,324°	-2,327°
Radiopropagación		
Ganancia de la antena Tx: 3 dB	Perdidas de línea Tx: 1 dB	
Ganancia de la antena Rx: 3 dB	Perdidas de línea Rx: 3 dB	
Potencia de Tx: 24 dBm	Sensibilidad de Rx: -110 dBm	
Frecuencia: 915 MHz	Distancia: 0,39 km	
Perdida del espacio libre: 83,5 dB	Perdidas de Obstrucción: 2,4 dB	
Perdidas urbanas: 0 dB	Perdidas por vegetación: 1,0 dB	
Perdidas estadísticas: 4,2 dB	Pérdidas totales: 91,1 dB	
Potencia en recepción: -63,1 dBm	Margen de recepción: 46,9 dB	

Los resultados de la simulación del radioenlace muestran que la comunicación entre el prototipo y el equipo en los pits podría realizarse, en vista de que se obtuvo un margen de 46,9 dB de diferencia entre la señal recibida y la sensibilidad del receptor.

Estudio de la pista 2

Esta competencia tuvo lugar en Research Rd. Pittsburg, Kansas 66762 EE.UU en 2014. Se puede observar en la imagen 4.26 la disposición de los módulos transmisor y receptor para la simulación.

Figura 4.26: Ubicación de los módulos en el mapa de simulación del radioenlace en la pista 2.

El procedimiento fue el mismo que en el escenario anterior, obteniéndose para la primera simulación de este entorno que la cobertura permitía la comunicación dentro de la pista. Esta cobertura se muestra en la figura 4.27.

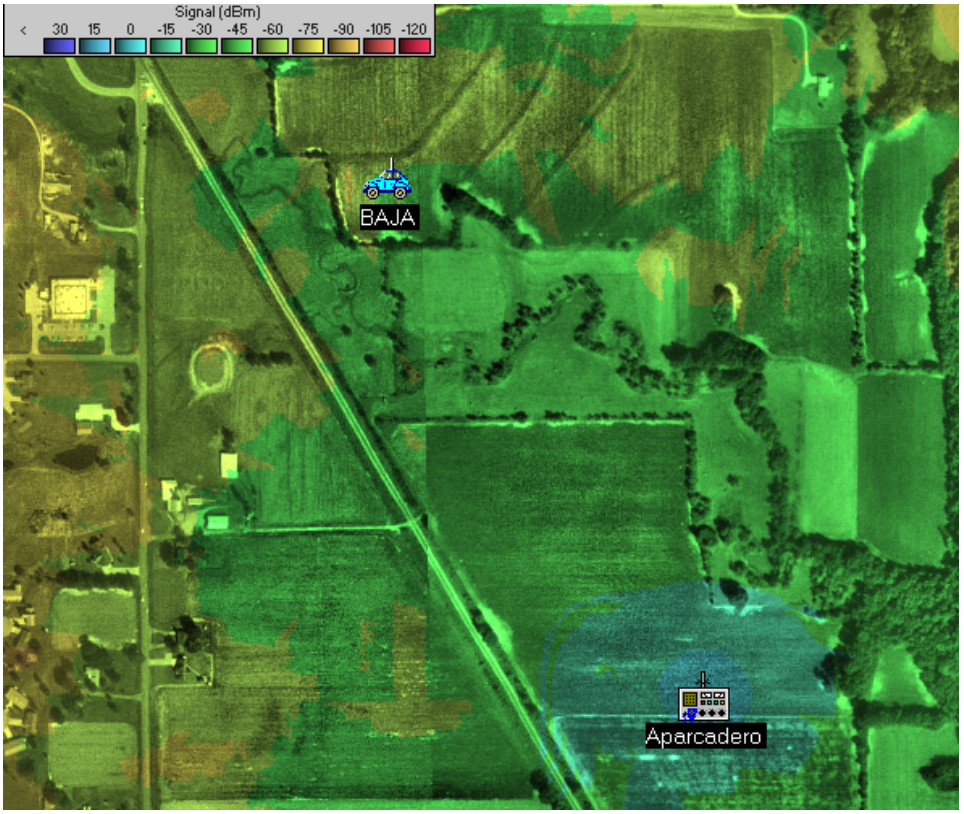


Figura 4.27: Mapa de cobertura del receptor (PC) de la simulación para la pista 2.

Como se observa debido a las características del terreno y del radioenlace, podría existir la comunicación dentro de cualquier tramo de la pista. En la tabla 4.18 se resume los resultados obtenídos por el software, los cuales se presentan en la figura a.4.

Tabla 4.18: Características y resultados de la simulación para la pista 2.

Especificación	Tx	Rx
Latitud	37,38933°	37,38477°
Longitud	-94,68418°	-94,67939°
Elevación del terreno	277 m	274 m
Altura de antena	1 m	3 m
Azimut	140,18°	320,18°
Inclinación	0,053°	-0,059°
Radiopropagación		
Ganancia antena Tx: 3 dB	Perdidas de línea Tx: 1 dB	
Ganancia antena Rx: 3 dB	Perdidas de línea Rx: 3 dB	
Potencia Tx: 24 dBm	Sensibilidad Rx: -110 dBm	
Frecuencia: 915 MHz	Distancia: 0,66 km	
Perdida del espacio libre: 88,0 dB	Perdidas de Obstrucción: 8,6 dB	
Perdidas urbanas: 0 dB	Perdidas por vegetación: 1,0 dB	
Perdidas estadísticas: 4,2 dB	Pérdidas totales: 101,8 dB	
Potencia en recepción: -73,8 dBm	Margen de recepción: 36,2 dB	

En la simulación se obtuvo un margen de recepción de 36,2 dB; con lo cual se garantiza teóricamente que la comunicación dentro de este entorno sería satisfactoria para el radioenlace y para la pista ya que la cobertura es aceptable en toda la pista. Todo esto a pesar que la distancia del radioenlace es mayor que para el caso anterior aunque existe un mayor despeje.

Estudio de la pista 3

Esta compotencia se desarrolló en la Universidad de Texas, El Paso, 500 W University Ave. EE.UU., en la imagen 4.28 se muestra la disposición de los módulos en el mapa. Para esta se simularon 3 posiciones posibles para el receptor (PC), este es el entorno más extenso en cuanto a área de los estudiados, además que es también la de mayor relieve. Debido a esto en las 2 primeras simulaciones no se obtuvo margen de recepción que aseguraría la comunicación.

Para las dos primeras simulaciones la cobertura estaba limitada tanto por el entorno como por lo extenso del radioenlace, teniéndose las gráficas de coberturas a.5 y a.7, en las cuales se aprecia que la cobertura para estos casos es prácticamente

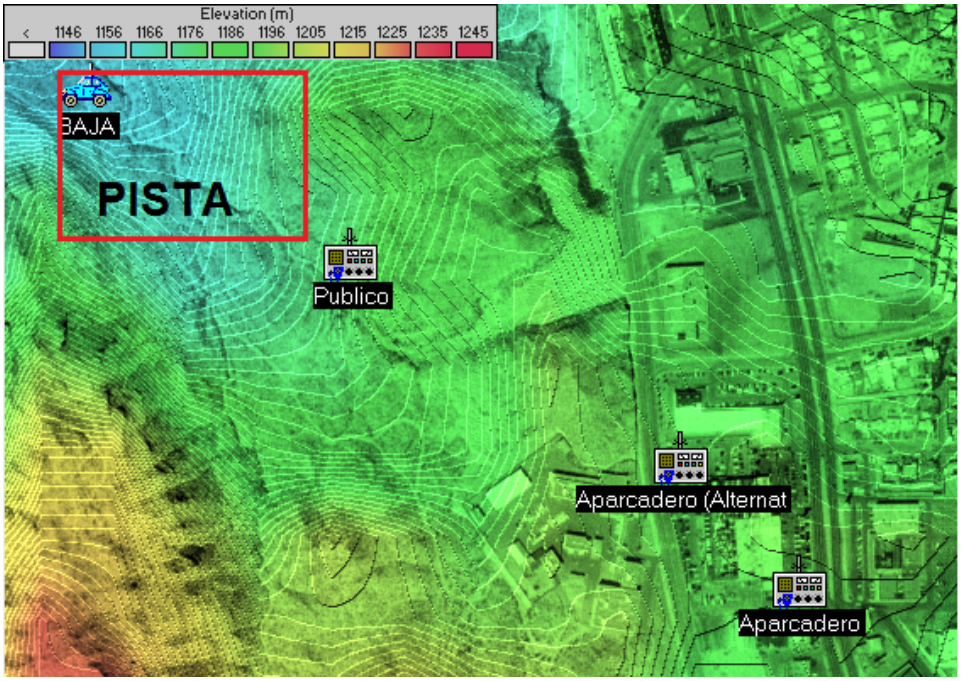


Figura 4.28: Ubicación de los módulos en el mapa de simulación del radioenlace en la pista 3.

nula, en estos 2 casos la posición designada para el receptor (PC) fue dentro del aparcadero. En cuanto a la potencia de los enlaces simulados se tiene para el primero una recepción de -116,8 dBm con lo cual no se tendría una comunicación. Mientras que para la segunda simulación se tiene una recepción de -106,0 dBm, con lo cual se podría tener comunicación según la hoja de datos del fabricante, pero los autores debido a la experiencia que adquirieron mediante el uso de los módulos de comunicación, recomiendan que exista un margen de por lo menos 30 dB con respecto a la sensibilidad para garantizar la comunicación.

Para la tercera simulación se tomó como posición de la PC el área de espectadores de la competencia, ya que esta área estaba más cerca y despejada que desde el aparcadero, la cobertura desde dicho punto se muestra en la figura 4.29.

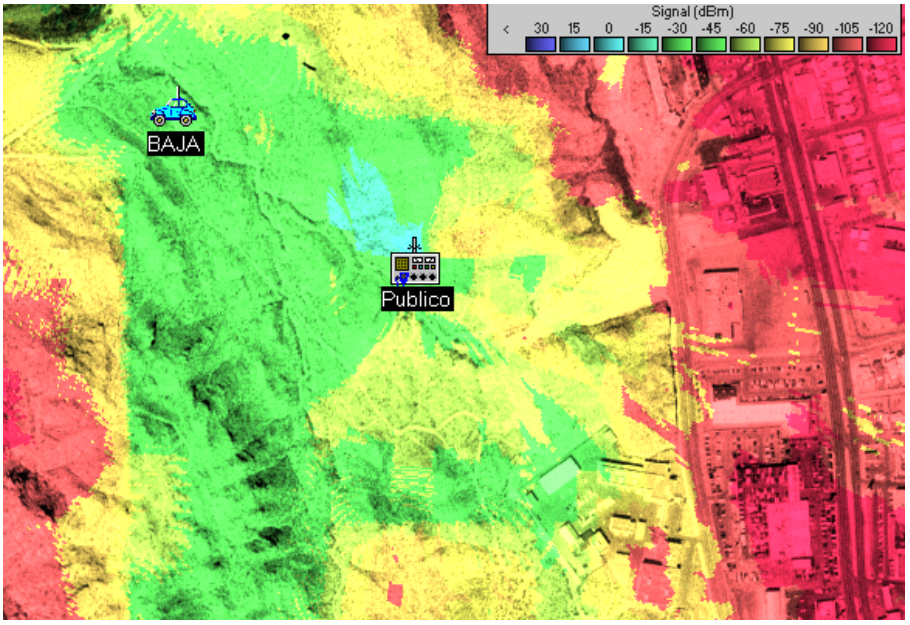


Figura 4.29: Mapa de cobertura del receptor (PC) de la tercera simulación para la pista 3.

En la imagen anterior se muestra la cobertura del sistema en la pista, observando que existiría comunicación en cualquier parte de la pista para este caso. Las características del radioenlace se muestran resumidas en la tabla 4.19, en donde se observa que para la tercera simulación se obtiene un margen de recepción con respecto a la sensibilidad de 43,1 dB.

Tabla 4.19: Características y resultados de la tercera simulación para la pista 3.

Especificación	Tx	Rx
Latitud	31,78518°	31,78381°
Longitud	-106,5136°	-106,5107°
Elevación del terreno	1.169 m	1.187 m
Altura de antena	1 m	3 m
Azimut	118,92°	298,93°
Inclinación	4,100°	-4,102°
Radiopropagación		
Ganancia antena Tx: 3 dB	Perdidas de línea Tx: 1 dB	
Ganancia antena Rx: 3 dB	Perdidas de línea Rx: 3 dB	
Potencia Tx: 24 dBm	Sensibilidad Rx: -110 dBm	
Frecuencia: 915 MHz	Distancia: 0,31 km	
Perdida del espacio libre: 81,6 dB	Perdidas de Obstrucción: 6,5 dB	
Perdidas urbanas: 0 dB	Perdidas por vegetación: 1 dB	
Perdidas estadísticas: 5,9 dB	Pérdidas totales: 94,9 dB	
Potencia en recepción: -66,9 dBm	Margen de recepción: 43,1 dB	

El resultado obtenido en los estudios anteriores demuestran teóricamente que el radioenlace es fiable, ya que en el peor de los escenarios planteados se obtuvo un margen de recepción de 36,2 dB, considerando algunas modificaciones en la posición del receptor. Por lo que se garantiza la transmisión de los parámetros de interés dentro de los escenarios planteados y en los que asemejen a estos, no obstante los módulos permitirían la comunicación a más distancia y que aún sea fiable la comunicación al aumentar las perdidas, esto depende también del entorno en donde se aplique, aunque se recomienda tener un margen de recepción de al menos unos 30 dB.

4.4. Implementación del diseño.

4.4.1. Pruebas primarias.

Arduino y comunicación mediante los Xbees.

El objetivo de esta prueba fue corroborar el correcto funcionamiento del sistema mediante la comunicación inalámbrica y la correcta interpretación mediante el *softare* intéprete de los datos a enviar los cuales serán constantes. Con respecto a la inicialización mediante el *software* EOBD-Facile, esta fue satisfactoria cumpliendo las características definidas en la parte de diseño, la inicialización correcta por parte del software y del Arduino se muestra en la figura 4.30.

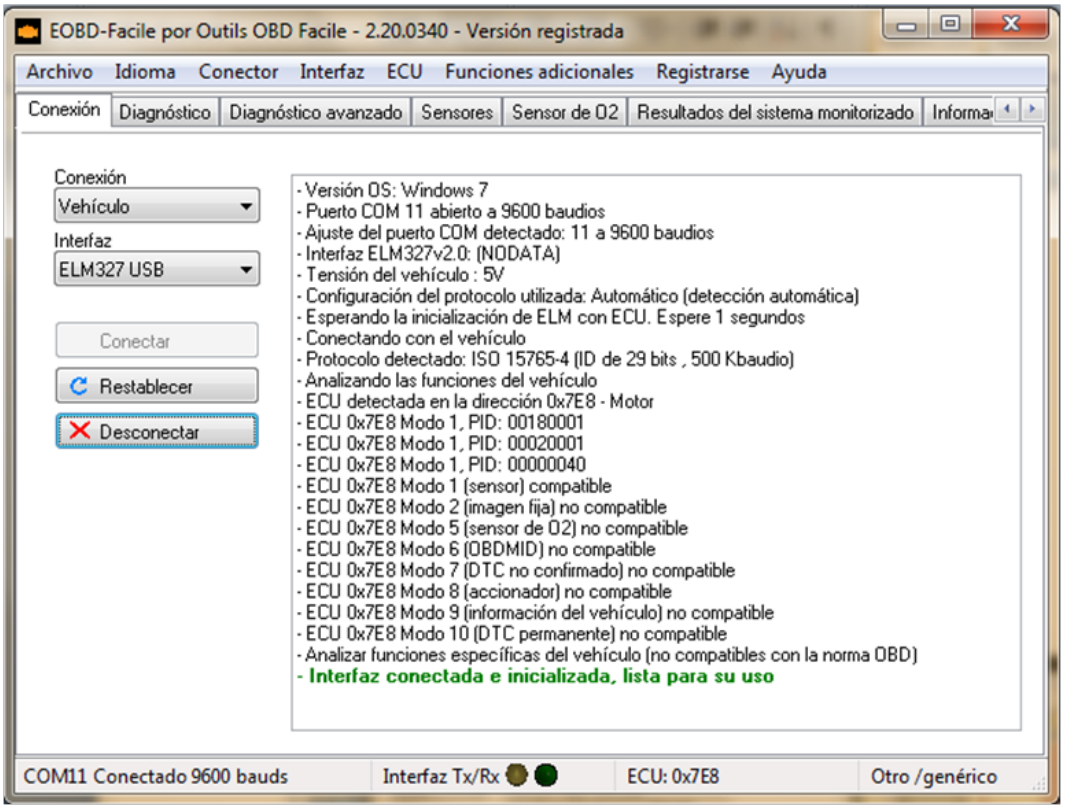


Figura 4.30: Resultado de la inicialización del Arduino con el programa EOBD-Facile.

Para la visualización de los parámetros se decidió usar los valores constantes de 1000 RPM, 60 km/h, 50 % del nivel de combustible y 75 % de la posición relativa del pedal del acelerador; que eso representa en formato hexadecimal los bytes 0F A0, 3C, 7F y BF, respectivamente respetando el formato de los PIDs mostrados en la tabla del apéndice C. Obteniendo para este caso los valores mostrados en la figura 4.31.

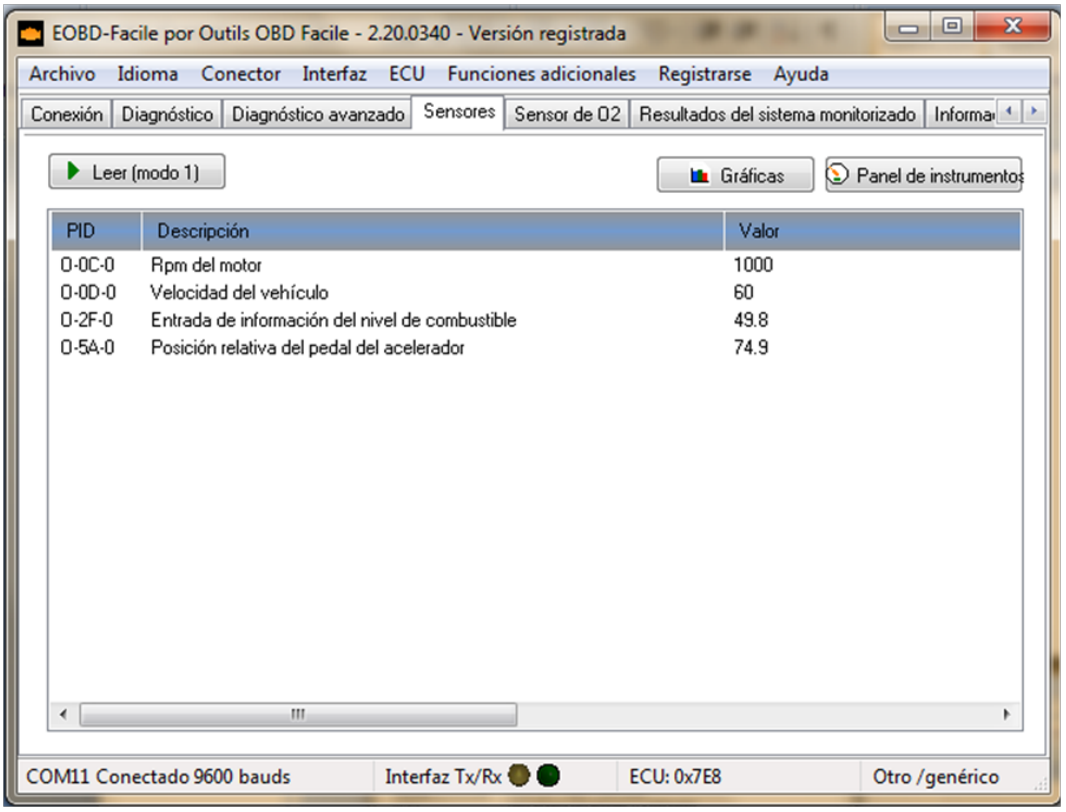
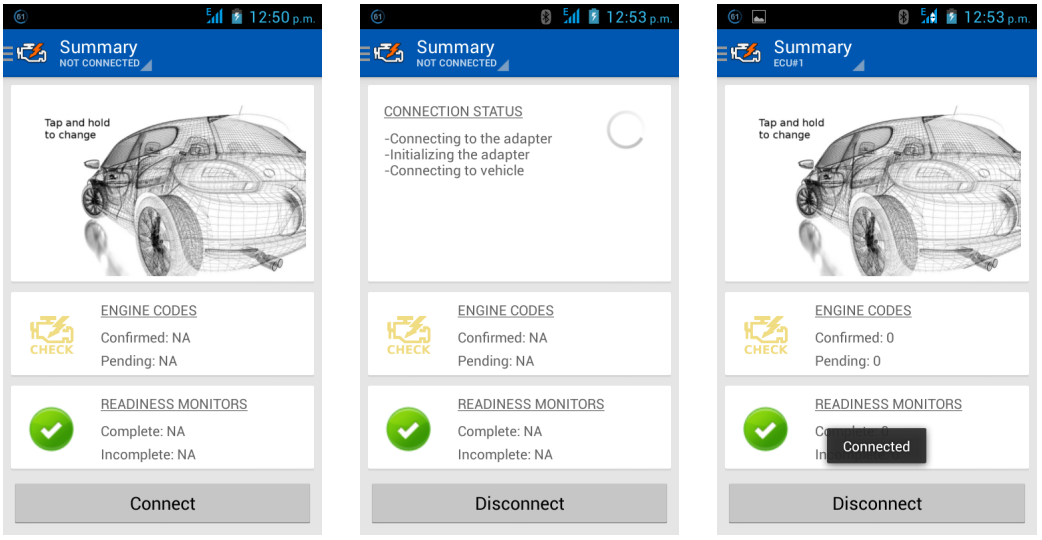


Figura 4.31: Resultado de la lectura de los parámetros fijos.

Mediante estos resultados, se pudo comprobar la correcta inicialización y el correcto envío de los parámetros por parte del Arduino, por lo que si se obtiene una medición errónea en alguno de los parámetros se debería a la forma de obtención del mismo y no por un problema de interpretación.

Comunicación mediante Bluetooth.

Para la comunicación Bluetooth se procedió a comprobar mediante un teléfono móvil la correcta inicialización e interpretación de los parámetros, se hizo uso de la aplicación OBD Auto Doctor Lite 2.4.10 el cual puede ser descargado desde la Play Store. Al verificar en la configuración que el método de conexión seleccionado es el Bluetooth, se procede luego a aparear el móvil con el módulo Bluetooth llamado SAE UC, despues se tiene que seleccionar la opción de conectar (*Connect*) de la ventana principal, como se muestra en la figura 4.32a.



(a) Sin conectar. **(b)** Conectando. **(c)** Conectado.

Figura 4.32: Proceso de conexión del móvil y el Arduino mediante Bluetooth.

Al comenzar la conexión, el móvil se comunicó con el módulo Bluetooth intercambiando los comandos que establecen y configuran la comunicación, tal como se inicializa la comunicación desde la PC. En la ventana principal se muestra el estado del inicio de la sesión de comunicación, como se muestra en la figura 4.32b. Luego de establecerse la conexión se muestra un mensaje momentáneo que dice <<*Connected*>> en la pantalla principal, indicando que la comunicación se encuentra establecida (figura 4.32c).

Los detalles de la comunicación que son suministrados al móvil durante la inicialización de la comunicación se muestran en las imágenes de la figura 4.33, esta

información es la incluida dentro del código de programación y describen los distintos elementos que se emulan y la identificación de red que se le otorga a la ECU.

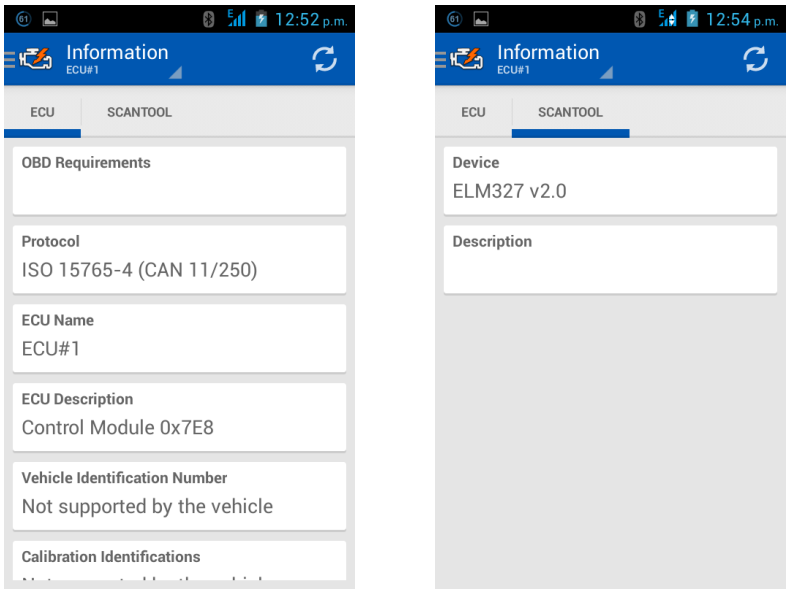


Figura 4.33: Características recibidas por el móvil de los distintos elementos emulados.

En la figura 4.34 se muestran los parámetros enviados y recibidos por el móvil durante la inicialización de la comunicación. Esto es útil para la verificación de los comandos utilizados por el programa, con lo cual se puede determinar si un comando dentro del código no es solicitado por el programa o si necesario usar más comandos dentro de la emulación. Además las últimas líneas mostradas corresponden a la lectura de parámetros, que se hicieron antes de solicitar el archivo de registro del programa, para verificar el correcto envío e interpretación de los parámetros.

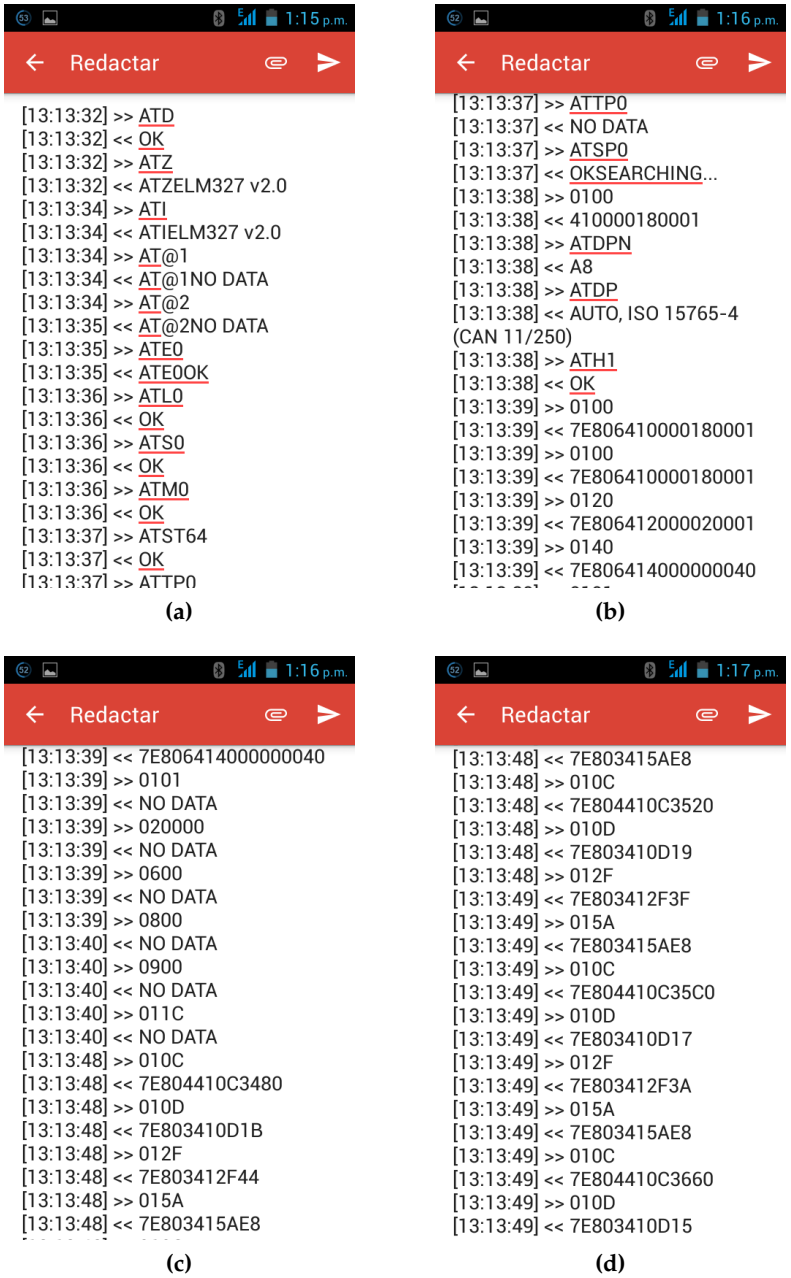
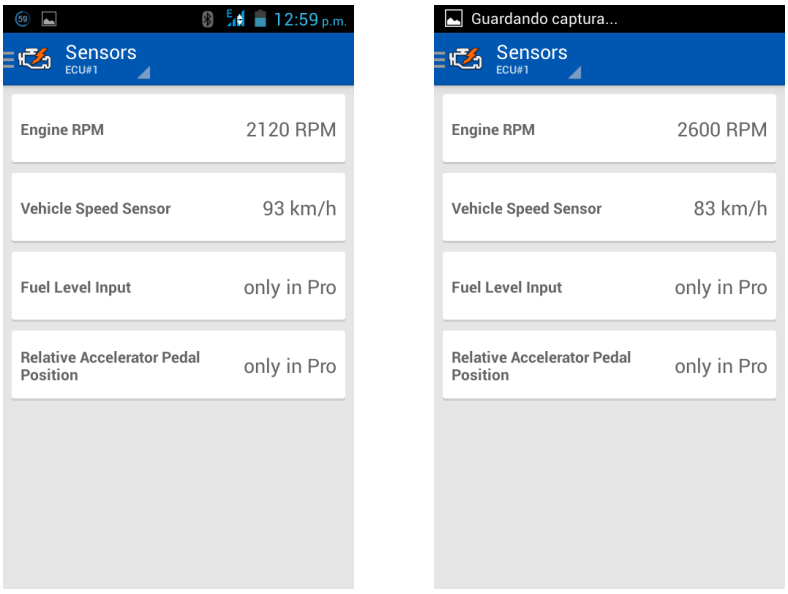


Figura 4.34: Detalles de los comandos para inicializar el programa y para la lecura de los parámetros.

La recepción de los parámetros pueden observarse de dos maneras. Se puede escoger observar los datos de manera grupal, en la que se muestra para cada uno el valor numérico del parámetro que se desea medir, como se muestra en la figura

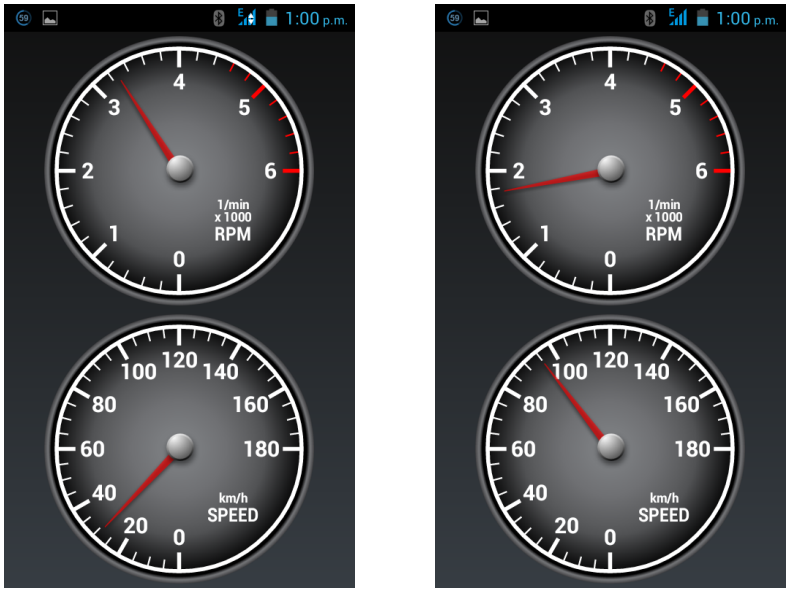
4.35. Para este caso se muestran sólo la velocidad y las RPM ya que esta versión del software era una gratuita, por lo que no admitía los otros parámetros involucrados en la simulación.



(a) **(b)**

Figura 4.35: Lectura de todos los parámetros mediante un valor numérico.

También se puede escoger la forma de visualización como el tablero de un automóvil como se muestra en la figura 4.36, donde la visualización es mediante un velocímetro y un tacometro, siendo ambos instrumentos analógicos de agujas.



(a) (b)

Figura 4.36: Lectura de la velocidad y las RPM mediante un velocímetro y un tacómetro analógico.

Esta visualización es la que se recomienda al momento de leer los parámetros dentro del prototipo mientras este se desplaza, ya que es la más visual y la que más se adapta a la visualización típica de un automóvil.

Prueba de comunicación remota.

Esta prueba tuvo lugar en una zona urbana en la que en ningún momento existió línea de vista entre Tx y RX. Entre el transmisor y el receptor habían varias estructuras (viviendas), la finalidad de esto fue probar el desempeño de los módulos seleccionados en un ambiente urbano. La imagen 4.37 muestra la posición de los módulos en el lugar de prueba y se puede observar la gran cantidad de edificios entre el transmisor y el receptor.

Cabe destacar que el módulo fijo Rx se dejó en la parte superior de una edificación de 2 pisos, es decir a unos 6 metros, mientras que el módulo Tx se trasladaba en un automóvil y su antena estaba entre 1 y 2 metros a nivel del suelo.

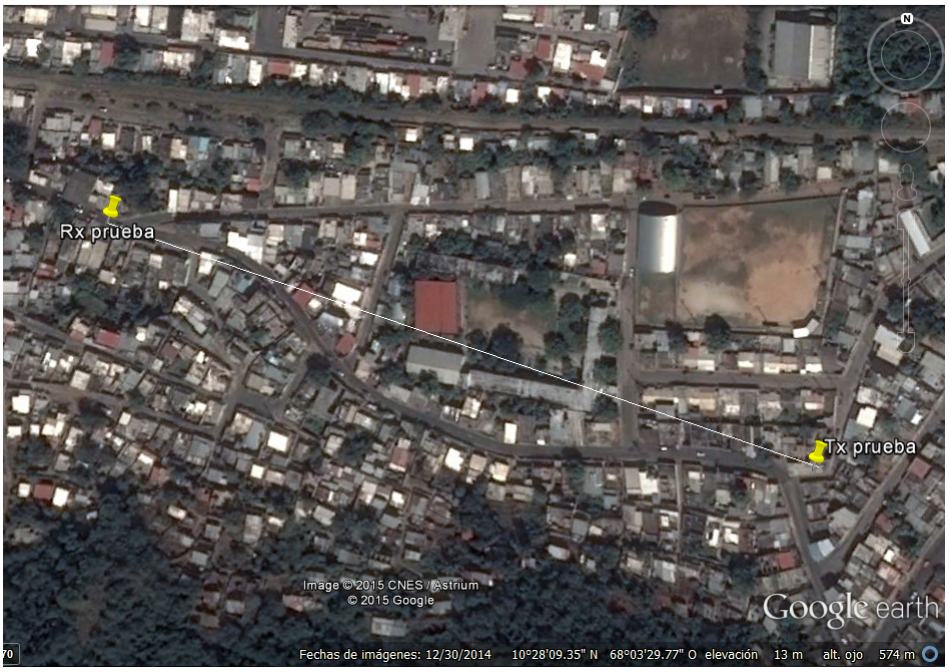


Figura 4.37: Posición de Tx y Rx en el lugar de prueba de la comunicación

La figura 4.38 muestra una prueba realizada con el software XCTU, dicha prueba está enfocada en obtener la calidad del radioenlace, mediante contadores de paquetes enviados, recibidos y perdidos, y la potencia de recepción en el módulo en el que se efectúa la prueba.

Se observa que se obtuvo una cantidad de potencia recibida de -76 dBm a esta distancia de 432 metros y el porcentaje de paquetes recibidos que es de 62 %, lo que es un buen resultado ya que tomamos en cuenta que el sitio de competición es una zona rural o sub-rural, además de que entre el transmisor y el receptor no existirán tantas obstrucciones como las que existieron en esta prueba.

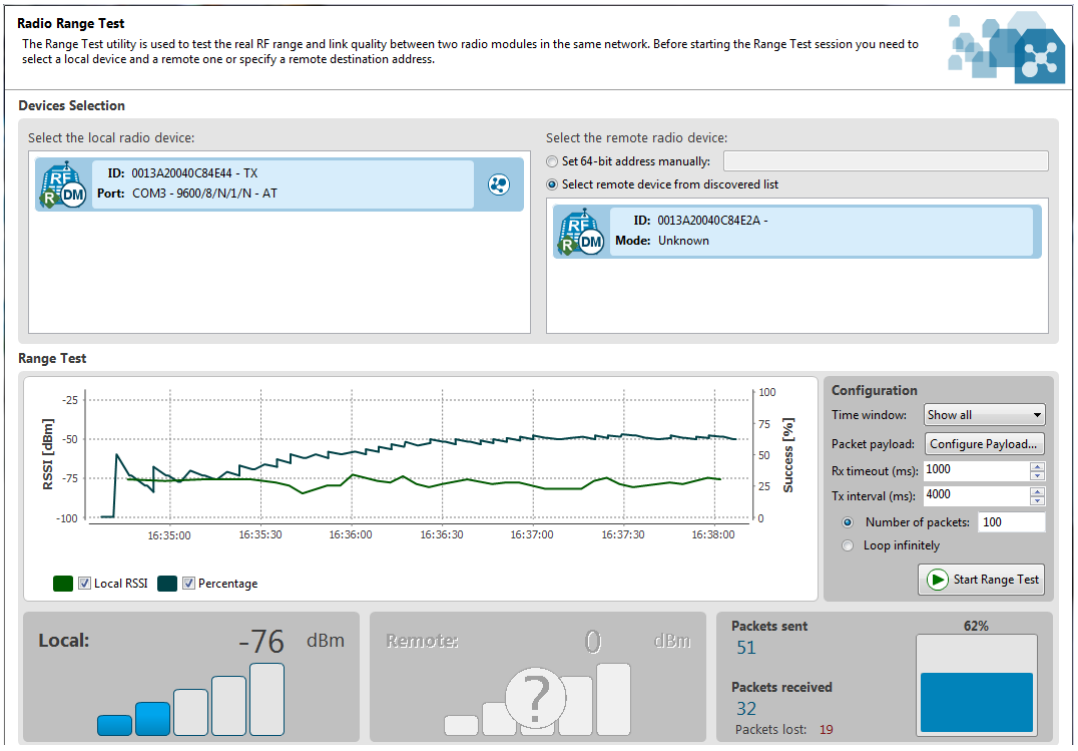


Figura 4.38: Resultados de la prueba de comunicación inalambrica estática a 432 metros.

Obtención de la relación para la medición del nivel de gasolina.

Luego de haber elaborado el flotante magnético, se realizaron pruebas con el sensor de efecto Hall colocado en la parte externa de la base del tanque de gasolina y el flotante dentro del tanque guiado por las partes internas de este que se usaron como guía.

En cada aumento de nivel del líquido se tomaron los datos pertinentes, para lograr definir la relación de voltaje de salida del sensor con respecto el nivel del líquido, obteniéndose los valores promedios mostrados en la tabla 4.20.

Para mayor visualización se graficaron estos datos con el programa Graph, para luego determinar cual es el valor correspondiente del voltaje según el nivel de líquido. La gráfica es la mostrada en la figura 4.39.

Tabla 4.20: Valores promedio obtenidos de las 2 pruebas del nivel de líquido del tanque de gasolina.

No. de prueba	Nivel de líquido (cc)	Nivel de líquido (%)	Voltaje (V)
1	0	0	3,620
2	280	7,69	3,620
3	560	15,38	3,620
4	840	23,07	3,620
5	1120	30,76,52	3,620
6	1400	38,45	2,700
7	1680	46,14	2,580
8	1960	53,83	2,532
9	2240	61,52	2,515
10	2520	69,21	2,505
11	2800	76,90	2,500
12	3080	84,59	2,495
13	3360	92,28	2,49
14	3640	99,97	2,49

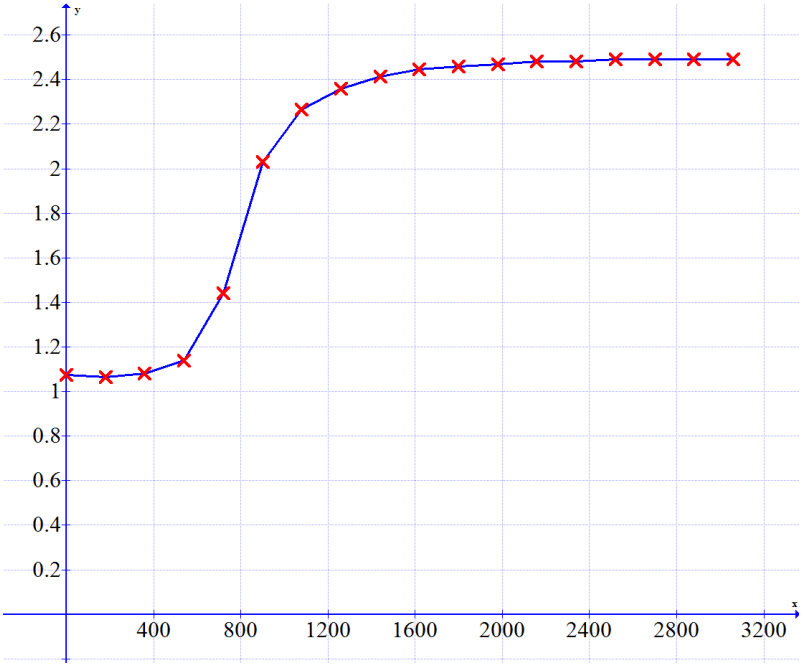


Figura 4.39: Gráfica de los valores promedios de nivel de líquido en cc (eje x) y el voltaje de salida del sensor (eje y).

El programa realiza una aproximación que es simple pero aceptable para la aplicación. Mientras que se realizaba esta prueba, se observó que el flotador comenzaba a desplazarse verticalmente a partir de de aproximadamente 30 % del líquido, debido a que el flotador al estar el tanque totalmente vacío reposaba en el fondo, es decir que el sistema no puede proveer información del nivel de gasolina por debajo del 30 %.

Al estudiar los valores tanto de la gráfica como de la tabla, se observó que la resolución o sensibilidad no puede ser del orden del 1 % o 2 % debido a que la variación no es lineal y la forma de onda obtenida es compleja como para expresarla matemáticamente. Por lo que se decidió usar una resolución en saltos iguales a la resolución obtenida, en los cuales el sistema podría expresar la variación entre 92 y 31 %, debido a que tampoco es capaz de detectar cambios sustanciales entre los niveles de líquido de 100 y 92 %.

Los rangos de valores escogidos para determinar el nivel de líquido con respecto al voltaje son los mostrados en la tabla 4.21.

Tabla 4.21: Valores para implementar en el programa para la obtención del nivel de gasolina.

Nivel de líquido (%)	Voltaje (V)
92	<2,49
84	2,49 - 2,495
76	2,495 - 2,500
69	2,500 - 2,505
61	2,505 - 2,515
53	2,515 - 2,532
46	2,532 - 2,580
38	2,580 - 2,700
31	2,700 - 3,625
0	>3,625

Además en la implementación, se notó que al cambiar el lado de lectura del sensor se tendría otro voltaje debido a la polaridad leída por el sensor, este cambio podría hacerse por reinstalación, sustitución por uno nuevo, entre otros, incluso se podría obtener este cambio de voltaje al cambiar el imán o diseñar otro flotador y

no considerar la polaridad. Por lo que se debe tomar en cuenta esto y realizar una calibración para cada implementación, ya que pueden haber variaciones en cuanto a la forma del flotador y una colocación distinta del imán o del sonsor. Además se añadió una rutina de retardo, mediante una sumatoria y un promedio para que en la lectura la fluctuación no sea tan significante, ya que el prototipo puede adquirir distintas inclinaciones y la vibración del mismo afectaría directamente la medición de la gasolina.

En la implementación también se cambió el valor de 30 % por 0 % a solicitud del director técnico de la organización, ya que de esta manera el piloto sabría que tiene que recargar combustible obligatoriamente.

Obtención de la relación para la medición de la velocidad.

Se sabe que el disco de freno tiene 4 partes metálicas igualmente espaciadas con respecto al radio del eje de dicho disco, donde se colocaron los imanes, es decir que a una velocidad constante se obtendrían pulsos periódicos de este sistema. Además se sabe que hay cuatro secciones de partes metalicas, por esto se deduce que la distancia angular entre los inicios de 2 pulsos consecutivos es:

$$\theta_D = \frac{2\pi}{4} = \frac{360^o}{4} = 90^o$$

Este valor fue introducido en la función de la obtención de la velocidad del programa principal, para lograr determinar los pasos de los imánes.

Todo esto es para el prototipo 2015, pero debido a que en el momento de hacer la prueba no estaba construido este prototipo se puso a prueba este sistema en el prototipo 2013, que tiene 9 secciones de metal en el disco de freno, es decir un θ_D de 40^o. De estas pruebas se obtuvo el voltaje para el paso de los 9 imanes por el sensor, dicha forma de onda se muestra en la figura 4.40.

La forma de obtener esta gráfica fue mediante el mismo Arduino, el cual leía tanto el voltaje y su contador propio de μs, almacenaba estos valores en un vector

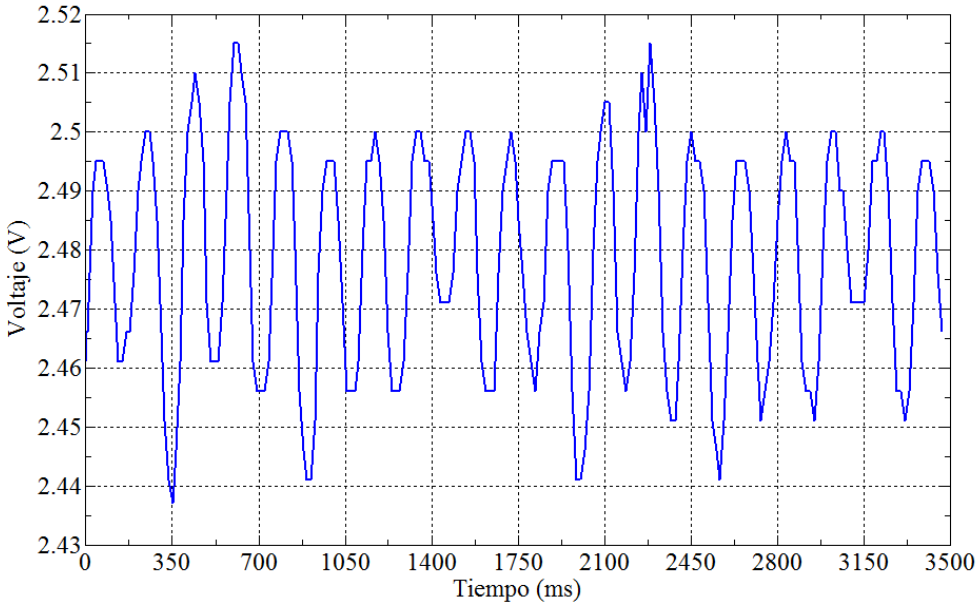


Figura 4.40: Gráfica de paso de los imanes en el disco de freno del prototipo 2013 por el sensor de efecto Hall.

para luego imprimirlos serialmente y ser almacenados manualmente por el usuario que lo manejaba en ese momento. Luego dichos datos fueron graficados en Matlab.

En la gráfica se puede observar la peridiocidad de los pulsos, en donde cada valle (debido a la polaridad de los imanes) representa el paso de un imán. Se puede observar que para algunos valles y algunas crestas la amplitud de estos varían, por lo que se tomó en cuenta el valor máximo de los valles y el valor mínimo de las crestas, al momento de determinar en la rutina de velocidad si la lectura actual representa un valle o un pulso, y así no se ignore ninguna cresta o valle dentro de la captación de estos.

Al tratar de obtener la velocidad de esta manera se obtenía un error, el cual tenía como resultado, variaciones significativas en la lectura de la velocidad. Para observar mejor la onda de la gráfica anterior, se aumentó la velocidad de muestreo de la gráfica para obtener una mejor resolución del paso de alguno de los imanes por el sensor. De esta manera se obtendría una gráfica que se aproxima más al

voltaje leído por el Arduino en el proceso de adquisición de la velocidad, dicha gráfica es la de la figura 4.41.

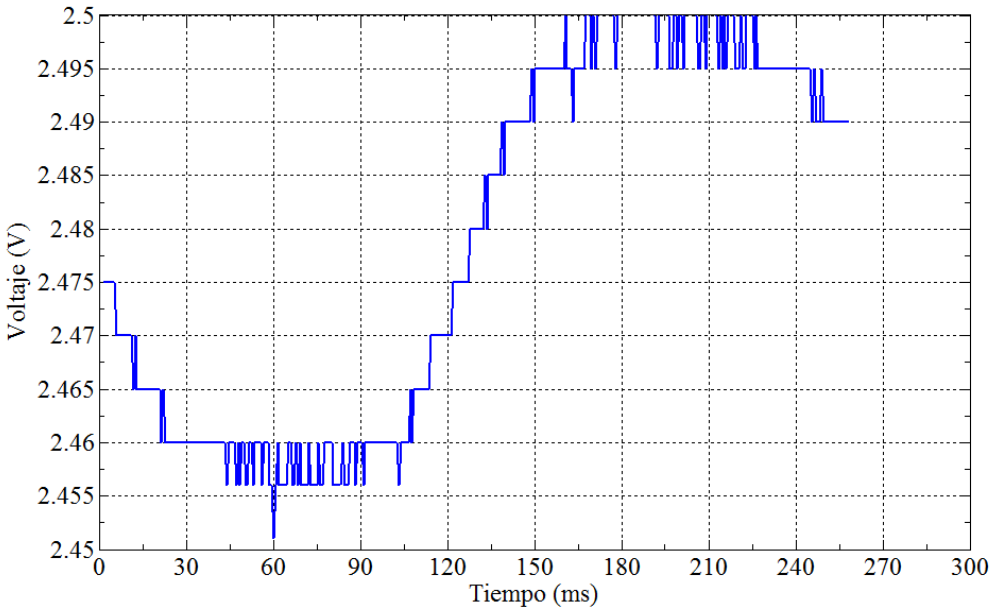


Figura 4.41: Gráfica de paso de los imanes con mayor frecuencia de muestreo.

De la imagen se observa que debido a que el microcontrolador pondera cada nivel de voltaje con un valor binario, por lo que puede existir una gran fluctuación en ciertos casos por una variación del voltaje menor a la sensibilidad. Recordando que el Arduino lee el voltaje de manera discreta teniendo un valor binario como resultado de esta lectura, el cual es a su vez transformado de nuevo en voltaje dentro de la misma rutina realizada por los autores, mediante una regla de tres.

Se observó que la variación normalmente era de ± 1 dentro del valor binario, por lo que decidió realizar una sección dentro del código de obtención de velocidad el cual ignorara estos cambios para excluir estas variaciones de la lectura.

Luego de implementar nuevamente habiendo efectuado los cambios descritos, se obtiene para la velocidad el comportamiento de la figura 4.42.

Se observa que en ciertos momentos cuando la velocidad debería ser constante o mantener una variación mínima, la lectura obtenida cambia su valor de manera

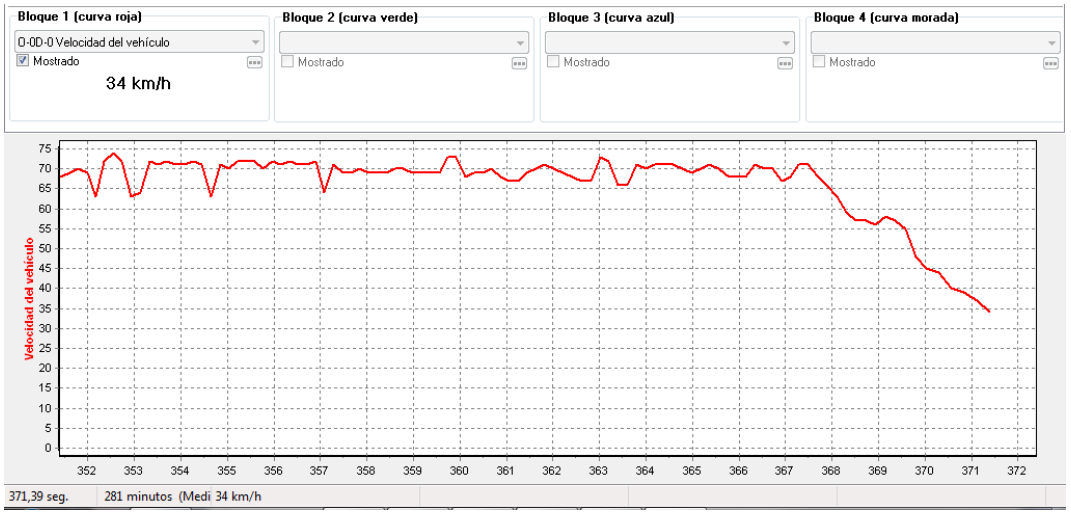


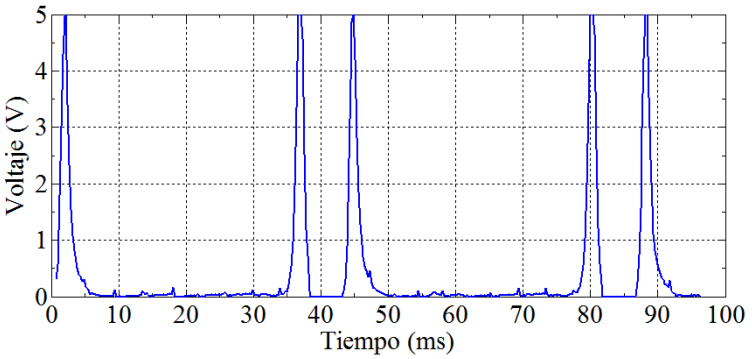
Figura 4.42: Velocidad obtenida para la prueba del sistema de adquisición de velocidad.

inesperada aunque aceptable para la aplicación de este sistema. Esto podría deberse a que esta prueba fue estática (el prototitpo tenía las ruedas del eje trasero suspendidas), pudiendo presentarse variaciones por las pérdidas que existen en la transmision del prototipo.

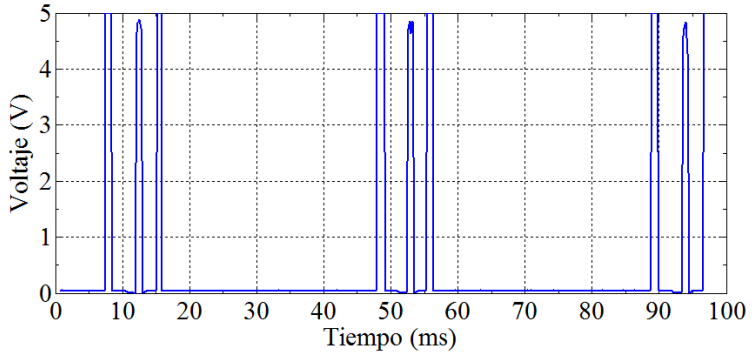
Obtención de la relación para la medición de las RPM.

En primer lugar se escogió el nivel de voltaje que debe usarse en el circuito comparador que acondiciona la señal para su lectura, debido a que el Arduino tiene 2 salidas de voltaje, una de 3,3 V y una de 5 V; estos fueron los voltajes tomados en cuenta para la comparación del voltaje de la señal de las RPM. Adicionalmente para el estudio se conectó directamente la señal a la entrada de lectura del Arduino para observar como era la lectura directa. Los resultados obtennidos de las distintas señales se muestran en la figura 4.43.

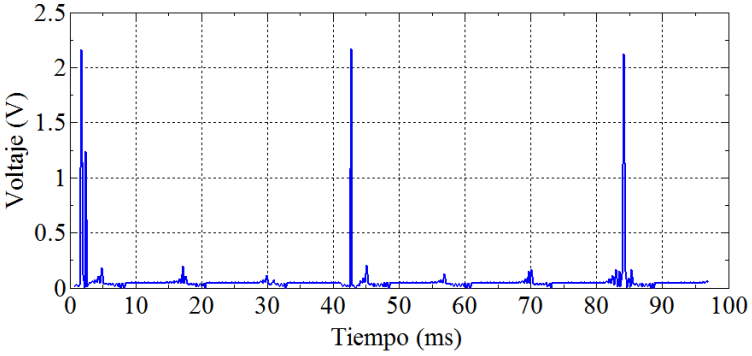
En la gráfica de la conexión directa al Arduino se observó que se obtienen 2 pulsos positivos en cada período, mientras que la parte negativa y por encima de 5 V queda ignorada debido al ADC del Arduino. Estos pulsos podrían ser captados por la rutina de obtención de las RPM, pero se decide usar el circuito comparador



(a) Conexión directa con el Arduino.



(b) Voltaje de referencia de 3,3 V.



(c) Voltaje de referencia de 5 V.

Figura 4.43: Gráficas de las RPM en conexión directa y con 2 valores distintos de voltaje en el circuito comparador.

para asegurar que el voltaje de entrada del Arduino siempre estaría entre 0 y 5 V, protegiendo al Arduino y protegiendo cualquier voltaje alto, tanto positivo como negativo que se pudiera presentar. Además se evidencia un rizado entre pulsos con

magnitud al rededor de 0 V, debido a esto no se puede aplicar como voltaje de comparación 0 V en el circuito acondicionador, ya que se obtendrían multiples pulsos que no serían representativos a la señal que se desea adquirir. El objetivo de la obtención de esta gráfica fue de carácter ilustrativo y para conocer el comportamiento de la señal, al menos dentro del rango positivo de voltaje aceptado por el ADC del Arduino.

En el caso de la gráfica para la referencia de 5 V, se observó en primer lugar que los pulsos no alcanzan un nivel adecuado de voltaje TTL, que es el objetivo para que el código sea sencillo. Además de esto, los pulsos no están bien definidos ya que la respuesta del integrado LM393N, no es lo suficientemente rápida en comparación con el tiempo que permanece la señal de las RPM por enciama de los 5 V de referencia.

Para la gráfica con el circuito comparador usando el voltaje de referencia de 3,3 V; se tienen 3 pulsos bien definidos dentro de los valores discretos TTL. Aunque se obtuvieron 3 pulsos, esto no es de gran importacia ya que el primer pulso es el que se detecta y los demas son ignorados mediante el comando delay(), para luego detectar el primer pulso del siguiente tren de pulsos. Por esto se tomó como voltaje de referencia para la obtención de las RPM el voltaje de 3,3 V.

4.4.2. Ejecución del montaje.

Montaje del sensor para la medición del nivel de gasolina.

El montaje del sensor de efecto Hall para la medición del nivel de combustible del prototipo resultó como se observa en la figura 4.44, donde se aprecia la parte baja del tanque de combustible y el sensor dispuesto en el centro del triángulo equilátero rojo que se dibujó para fines explicativos. Este triángulo es formado en la parte interna del tanque por los tres cilindros que dan soporte al mismo y que además dieron libertad al flotante de moverse sólo en sentido vertical.

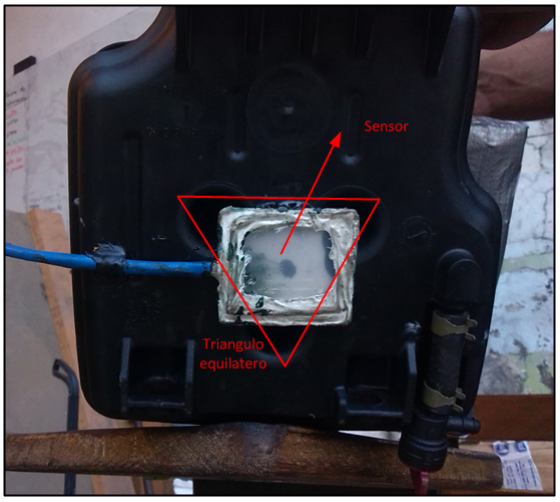


Figura 4.44: Montaje del sensor de efecto Hall para la medición del nivel de combustible.

Montaje del sensor para la medición de la velocidad.

La posición del sensor de efecto Hall en la caja del prototipo y de los imanes en el disco de freno para la medición de la velocidad del prototipo se muestran en la figura 4.45, es importante mencionar que fueron colocados capacitores de desacoplamiento en ambos sensores de efecto Hall para evitar alguna inducción electromagnética externa o del motor del prototipo en el cableado del sistema, ya que se captaron picos inesperados a la hora de la implementación, por los cuales también se realizó una rutina dentro de la programación que ignoraba estos picos.

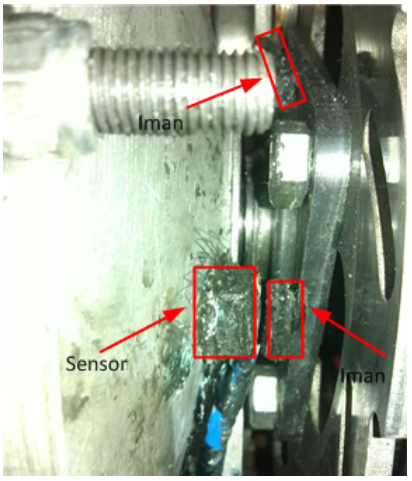


Figura 4.45: Montaje del sensor de efecto Hall para medir la velocidad.

Montaje final del sistema.

El montaje final del sistema se muestra en la figura 4.46, se pueden observar los recuadros identificados con los números 1, 2 y 3 que corresponden con el circuito acondicionador para la captación de la señal de las RPM en el que los pines del motor están conectados directamente, el módulo de transmisión Xbee y el sensor de aceleración, respectivamente. Adicionalmente se puede observar la interconexión de los pines del cableado UTP con la placa del sistema; estos se especifican en el apéndice E.

Es importante mencionar que el módulo de comunicación bluetooth está dispuesto entre la placa Wireless Photo Shield y la placa del Arduino UNO, ya que no se observa en la ilustración.

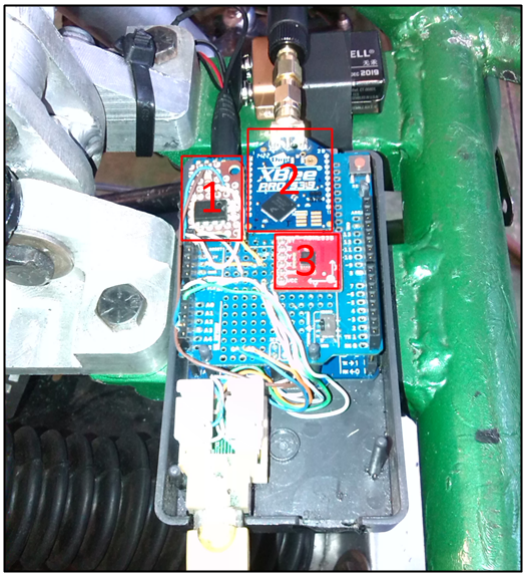


Figura 4.46: Montaje final de todo el sistema en el prototipo

4.4.3. Pruebas finales.

Esta sección contempla las pruebas para corroborar que la adquisición de parametros es correcta y un estudio de cobertura hecho en la facultad de ingeniería de la UC.

Los parámetros en los que se va a verificar la veracidad de los datos obtenidos son las RPM y la velocidad, ya que la aceleración se obtiene mediante un instrumento de medición adquirido y el nivel de combustible se obtiene de la rutina de comandos transformando directamente el valor de voltaje leído en un nivel de combustible que fue estudiado anteriormente mediante pruebas.

Prueba de la velocidad.

La tabla 4.22 muestra la comparación entre los resultados obtenidos del sistema de telemetría y un velocímetro con una aplicación de un *smartphone* que calcula la velocidad por medio de GPS, para comprobar que estos son correctos. La comparación se realizó midiendo la velocidad máxima alcanzada por el prototipo ya que el velocímetro guarda la máxima velocidad a la que es sometido mientras que en el

sistema se obtiene una gráfica en tiempo real, el procedimiento se basó en tomar el dato almacenado en el velocímetro y el valor máximo registrado en la gráfica repitiendo esto a distintas velocidades para realizar un estudio sobre la variación entre las dos mediciones.

Tabla 4.22: Resultados obtenidos del sistema de telemetría comparados con los del velocímetro digital.

Prueba	Medición de velocidad con sistema de telemetría km/h	Medición de velocidad con tacómetro km/h	Error absoluto km/h
1	10	12	2
2	18	19	1
3	24	24	0
4	27	30	3
5	31	32	1
6	36	38	2
7	39	40	1
8	42	42	0
9	45	46	1
10	50	51	1

Resultando una media en las mediciones del sistema de 32.5 km/h mientras que en el velocímetro fue de 33.1km/h, con un error absoluto promedio de 0.6 km/h. Se observa además que existe un error absoluto máximo de 3 km/h, lo que hace que el método de la adquisición de la velocidad resulte adecuado para esta implementación.

Prueba de las RPM.

Se realizaron pruebas para corroborar el correcto funcionamiento, comparando el resultado obtenido de este sistema con la medición de un tacómetro digital existente en la organización. Dicho tacómetro tiene una sensibilidad de 60 RPM mientras que el sistema de telemetría tiene una sensibilidad de 1 RPM. Los resultados del sistema de telemetría que se muestran en la tabla 4.23, variaban en cada actualización de datos para una RPM <<fija>> en el motor en un orden de ± 20 RPM, por lo que se escogió una medición aleatoria de este sistema para cada medición constante del tacómetro digital. Se cree que esto ocurre debido a que el sistema

de telemetría es muy preciso y dentro del motor existen pérdidas que hacen que el mismo no tenga un valor constante en RPM.

Tabla 4.23: Resultados obtenidos del sistema de telemetría comparados con los del tacómetro digital.

Número de prueba	Medición de tacómetro (RPM)	Medición de sistema de telemetría (RPM)	Error absoluto
1	1500	1562	62
2	1560	1568	8
3	1740	1769	29
4	1680	1714	34
5	1860	1905	45
6	1980	1998	18
7	2040	2064	24
8	2100	2150	50
9	2160	2204	44
10	2220	2245	25
11	2280	2333	53
12	2340	2383	43
13	2400	2437	37
14	2520	2600	80
15	2640	2610	30
16	2700	2740	40
17	2880	2917	37
18	3000	2999	1
19	2940	2974	34

Resultando una media en la medición del sistema de 2272 RPM mientras que en el tacómetro digital de 2238 RPM, con un error absoluto promedio de 34 RPM y un error absoluto máximo de 80 RPM.

Se obtuvo que este sistema tiene una mayor resolución que el tacómetro digital, lo cual puede ser útil para ciertos estudios del comportamiento del motor en diversas pruebas. En la práctica también se observó que la obteción de las RPM por parte del sistema de telemetría era instantánea, mientras que en el tacómetro se observó cierto retardo, por lo que el sistema de telemetría tiene un mejor desempeño y una mayor confiabilidad con respecto al tacómetro digital.

Prueba de la cobertura.

Esta prueba se desarrolló en la facultad de ingeniería, el enlace constó de la antena transmisora situada en las afueras de la escuela de telecomunicaciones, mientras que la antena receptora en conjunto con el sistema de telemetría se trasladó a través de 4 tramos, recorriendo cada uno de estos de manera separada para el estudio. La altura de la antena transmisora fue de unos 3 metros sobre el suelo, mientras que la altura de la antena receptora fue de 1 metro. Los tramos recorridos de la antena receptora y la posición de la antena transmisora se muestran en la figura 4.47.

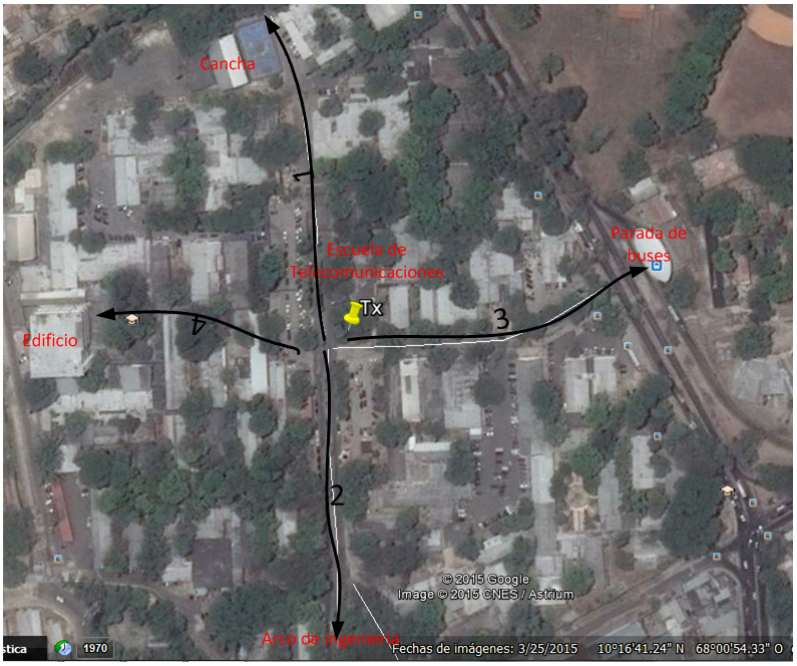


Figura 4.47: Prueba de cobertura en la facultad de ingeniería

Para cada cierta distancia se tomó nota de la posición de la antena receptora y de la potencia recibida en la transmisora con el software X-CTU, para obtener un comportamiento aproximado de la cobertura del radioenlace en este ambiente suburbano, que representó uno de los peores casos a los que se sometería el sistema de telemetria. Usando como herramienta el contador de paquetes perdidos del mismo software, se determinó cuando la comunicación del radioenlace comenzaba a fallar y se tomó ese punto como punto final del tramo.

La movilidad del dispositivo la dieron los autores de manera peatonal debido a que el prototipo, a la hora de hacer esta prueba no se encontraba disponible ya que había partido para la competencia del año en curso.

En el anexo B se presenta el plano donde se especifíca con mayor detalle la posición de cada estación con su respectivo nivel de potencia, para cada recorrido. En la tabla 4.24 se muestra el nivel de potencia recibido en cada estación para los 4 tramos de estudio.

Tabla 4.24: Potencia para cada estación en los cuatro recoridos de estudio.

Tramo 1								
Distancia entre las antenas (m)	30	61	84	106	125	153	187	200
Potencia (dBm)	-40	-42	-55	-59	-49	-69	-75	-63
Tramo 2								
Distancia entre las antenas (m)	43	96	154	177	218			
Potencia (dBm)	-41	-63	-60	-61	-70			
Tramo 3								
Distancia entre las antenas (m)	46	101	144	162				
Potencia (dBm)	-40	-62	-66	-70				
Tramo 4								
Distancia entre las antenas (m)	48	74	117	143	191			
Potencia (dBm)	-41	-52	-68	-74	-74			

Es importante destacar que las obstrucciones que se presentarse en el radioenlace de los tramos 1 y 2 eran debido a vegetación y automóviles que se encontraban estacionados entre el enlace, y que en la mayoría de estos tramos existío línea de vista óptica. Mientras que para los tramos 3 y 4, la mayor obstrucción que existió fue debido a edificaciones. De los resultados se observa que la cobertura del sistema con línea de vista tiene como límite entre 180 y 200 m; mientras que en el caso de obstrucción por edificaciones la cobertura es hasta unos 130 y 150 m. Recordando que esta prueba se realizó en un ambiente sub-urbano y que su desempeño en un entorno rural (como el que se presenta en las competencias), debería se mayor y por lo tanto tener una mayor cobertura.

Capítulo V

Conclusiones y recomendaciones

5.1. Conclusiones

Se emuló el ELM327 para simplificar el sistema, evitando etapas de circuitos y dispositivos adicionales. No bastó con emular sólo el ELM, sino que se tuvo que emular también la comunicación OBD.

Al usar el protocolo Digimesh para la transmisión inalámbrica remota, se obtiene una mayor zona de cobertura del radioenlace. Además esta banda de frecuencia está definida por la ITU como banda libre por lo que no se requieren licencias para la transmisión. En cuanto a la comunicación dentro del prototipo se implementó el protocolo Bluetooth por su practicidad en este sistema.

La selección del acelerometro ADXL335 para la obtención de la gasolina, simplificó la adquisicón de este parámetro, ya que el nivel de voltaje de salida del acelerómetro era proporcional a la aceleración del prototipo.

Se obtuvo un buen desepeño de los sensores de efecto Hall en la implementación, ya que los sensores tienen una velocidad de respuesta alta y además tienen un rango amplio de detección de campo magnético.

Se seleccionó el módulo inalámbrico Xbee PRO 900HP, debido a que era uno de los más potentes y el de menor precio de los módulos comparados. Su salida

RP-SMA facilitó la elección de antenas que fueran prácticas a la hora de la implemenación.

La interconexión con el transmisor Xbee a través de un modulo de expansión Arduino facilitó la implementación e integración de todo el sistema.

El procedimiento de tanteo para la emulación del protocolo OBD y el microcontrolador ELM, fue satisfactorio a la hora de determinar cuales eran los comantos AT necesarios para la comunicación entre la PC y el sistema de telemetría. Logrando establecer la comunicación con 9 de los 11 software probados.

Se diseñaron 2 programas finales con distintos modos de visualización, ya que al intentar hacer sólo un programa el Arduino adquiría un comportamiento errado, lo cual fue compensado por la separación de los modos de visualización. Se cree que la cantidad de bloques *if"* tiene que ver con el problema.

Se realizó una prueba de comunicación remota donde se pusieron a prueba los modulos Xbee y así obtener el alcance de los mismos, de esto se obtuvo un alcance de 432 metros con línea de vista y una potencia de recepción a la máxima distancia de -76 dBm con un porcentaje de paquetes recibidos de 62 %.

Ocurrieron errores en la medición de la velocidad, debido a que los niveles de voltaje arrojados por el sensor de efecto Hall no eran iguales para todos los imanes. Por otro lado la medición de las RPM también arrojó valores erróneos, debido a que la señal leída tenia variaciones en los niveles de voltaje al pasar el tiempo. En estos casos se solventaron dichos problemas mediante programación y condensadores de desacople.

Por ultimo, se realizaron las pruebas finales donde se verifico la correcta lectura de los parámetros conparandolos con otros dispositivos de medición externos, para corroborar la exactitud de los datos obtenidos por el sistema; obteniendo que el sistema era lo suficientemente preciso para esta implementación. En cuanto a la cobertura se obtuvo una distancia máxima de 200 metros sin línea de vista, con lo que debería ser posible la comunicación en las competencias que se implemente este sistema, ya que en estas normalmente existe línea de vista.

5.2. Recomendaciones

- Implementar métodos más rápidos de adquisición de la velocidad y de las RPM, para una visualización con mayor resolución en función del tiempo.
- Considerar la aplicación de LabVIEW para la interpretación de los datos, esto permitiría incluir parámetros que no son aceptados por los protocolos OBD como posición geográfica y también podría resumir en gran medida el código para aplicar la completa comunicación.
- Resumir el código final para unificar la comunicación Bluetooth y el envío de los 4 parámetros a la vez.
- Utilizar una antena sectorial de mayor ganancia en el receptor para tener un mayor alcance y considerar aumentar la altura de esta.
- Para los próximos prototipos diseñar el montaje y seleccionar los elementos adicionales antes de la construcción del prototipo, para que en el momento del montaje no se malgaste tiempo ni se presenten inconvenientes mayores.
- Posicionar la antena del prototipo con una mejor posición que favorezca la cobertura para la comunicación.
- Garantizar un margen de 30 dB con respecto a la sensibilidad en recepción para asegurar la comunicación.
- Utilizar para las conexiones y conectores a lo largo del prototipo estándares de la industria automotiz.

Apéndice A

Comandos AT para la emulación básica del ELM327:

AT0, AT1 y AT2, control de tiempo adaptativo: Estos comandos estipulan el tiempo de espera de respuesta del automóvil al microcontrolador, dichos tiempos pueden variar por eso algunos softwares lo controlan para que el tiempo del microcontrolador ELM sea acorde con el tiempo de respuesta esperado por el software. Para efectos la emulación cualquiera de los 3 no tiene ningún efecto en la rutina del Arduino.

BI, omitir la secuencia de inicialización: Evita la inicialización entre el ELM327 y la ECU, esta inicialización normalmente es para determinar el protocolo. No tiene ningún efecto para la rutina del Arduino.

CAF0 y CAF1, desactivar o activar el auto formato CAN: Sirve para activar o desactivar el formato en el cual se envían y reciben mensajes. Por ejemplo para el caso de CAF1 (que es el que se presentó para los softwares), se activa el auto formato para los mensajes enviados y recibidos, es decir el ELM se encarga de darle el formato CAN a los mensajes OBD, agregándole los headers y los bytes PCI. Para el caso de recepción el ELM ignora estos elementos y muestra sólo la trama útil. En caso contrario (CAF0) no agrega estos bytes y deben ser introducidos por el

software o por la aplicación, en la recepción se muestran los headers y los bytes PCI. No tiene ningún efecto para la rutina del Arduino.

CFC0 y CFC1, desactivar o activar el control de flujo CAN: si esta activado el protocolo CAN espera que al mandar un primer mensaje se le responda con un mensaje de control de flujo, si esta desactivado pasa lo contrario. No tiene ningún efecto para la rutina del Arduino.

CRA hhh, ajusta la dirección de Rx CAN con hhh: La dirección hhh es la dirección física de la ECU del automóvil a la cual se desea conectar el ELM, las direcciones físicas de las ECUs son 7E0, 7E1, 7E2. . . , 7R7 que representan la ECU 1, la ECU 2, la ECU 3. . . , la ECU8 y la respuesta de estas ECUs son 7E8, 7E9, 7EA. . . 7EF, respectivamente. Normalmente los softwares utilizan este comando para configurar la dirección como una dirección funcional, es decir que el mensaje es enviado a todas las ECUs, siendo la dirección funcional igual a 7DF. Cabe destacar que estas direcciones son los identificadores CAN en el header. Como se emulará una sola ECU y la dirección utilizadas por los softwares es una dirección funcional (un broadcast haciendo la analogía a redes TCP/IP), la respuesta del ELM hacia la PC para un comando OBD siempre mantendrá el identificador del header, que para este caso sería 7E8.

D, ajusta todo a sus valores por defecto: Se borra toda la información y configuración que se haya almacenado. Reinicia los valores originales de las variables al principio del código.

DP, muestra el protocolo actual: Muestra un mensaje con alguna de las descripciones de la tabla A.1.

Si está seleccionado para buscar automáticamente (comando ATSP0) cualquier protocolo el mensaje tendría la siguiente forma: “AUTO, ISO 15765-4 (CAN 11/250)>” el cual será para este caso el protocolo que se usó.

DPN, muestra el número del protocolo: muestra el número del protocolo el cual está conectado al ELM, dicho número corresponde al de la tabla anterior. Si el comando ATSP0 fue usado el comando ATDPN tendrá la siguiente respuesta: “AN”

Tabla A.1: Numeración de los protocolos para el ELM327.

Protocolo	Descripción
0	Automático
1	SAE J1850 PWM (41,6 kbaudios)
2	SAE J1850 VPW (10,4 kbaudios)
3	ISO 9141-2 (5 baudios de inicialización)
4	ISO 14230-4 KWP (5 baudios de inicialización)
5	ISO 14230-4 KWP (Inicialización rápida)
6	ISO 15765-4 CAN (11 bit ID, 500 kbaudios)
7	ISO 15765-4 CAN (29 bit ID, 500 kbaudios)
8	ISO 15765-4 CAN (11 bit ID, 250 kbaudios)
9	ISO 15765-4 CAN (29 bit ID, 250 kbaudios)
A	SAE J1939 CAN (29 bit ID, 250 kbaudios)
B	Usuario1 CAN (11 bit ID, 125 kbaudios)
C	Usuario2 CAN (11 bit ID, 50 kbaudios)

donde N es el número del protocolo.

E0 y E1, desactiva o activa el eco: la configuración inicial del ELM provee a toda respuesta de un eco del comando que se insertó para generar una respuesta, por ejemplo una entrada del comando ATDPN para el protocolo numero 5 tendría la siguiente respuesta:

>ATDPN
ATDPN
A5
>

Esta opción principalmente es para determinar si los códigos fueron introducidos correctamente por el usuario, en dado caso de que fuera un manejo por alguna persona desde un programa de emulación de terminal.

H0 y H1, desactiva o activa los headers: esta activación o desactivación de headers se hace solo en las respuestas del automóvil, esto es principalmente para observar cuantas y cuales ECU son las que responden. Además se presentan los bytes de PCI que especifican la longitud del mensaje.

I, identifíquese: este comando hace que el ELM se identifique con su ID, esto es para que el software sepa y muestre al usuario con cual modelo ELM está trabajando.

L0 y L1, desactiva o activa los saltos de línea: al inicializar el ELM al final de cada respuesta produce un caracter de salto de línea luego del carácter de retorno de acarreo. Esto es útil cuando una persona trabaja con un programa emulación de terminal, de lo contrario cuando se comunica con un programa, al enviar un salto de línea como es más información la comunicación se hace un poco más lenta.

M0 y M1, desactiva o activa la memoria: en el ELM hay una memoria no volátil la cual se puede activar o desactivar para almacenar el último protocolo usado. No tiene ningún efecto para la rutina del Arduino ya que el protocolo no cambia.

RV, leer el voltaje de entrada: Lee el voltaje de entrada del ELM327 en el pin 2 y provee una respuesta de dicho voltaje. Para este caso se usará 5V para informar que ese es el voltaje en el cual se trabajará en el sistema para la adquisición de datos, exceptuando la alimentación del Arduino.

S0 y S1, desactiva o activa el espacio entre los comandos de respuesta: Para la respuesta del ELM327 se puede presentar una respuesta con espacios, en el caso de los comandos OBD entre la máscara, el byte PCI y la trama normalmente se encuentran separado mediante un espaciamiento, y dentro de la misma trama entre cada byte también existe un espaciamiento, el cual puede ser desactivado o activado mediante estos comandos, por ejemplo:

Con caracteres de espacio activados:

>7E8 06 41 00 80 18 00 11

Con caracteres de espacio desactivados:

>7E806410080180011

SH xyz, SH xx yy zz o SH ww xx yy zz, configura el header en formato xyz, xx yy zz o ww xx yy zz: ajusta el header para la comunicación con las diferentes ECU, el formato del header depende del protocolo con el cual se esté trabajando. Para este caso se ignorará cualquier header ya que para efectos de la emulación de la comunicación ELM-OBD no es necesario, pero se mostrará un mensaje de OK como si se hubiera configurado el header.

SP h, selección de protocolo numero h: Este comando ajusta el protocolo al cual el ELM tratará de conectarse, en dado de que el número sea 0 busca todos los protocolos posibles.

ST hh, ajuste de tiempo de espera a hh: Luego de que el ELM327 le envía una solicitud al automóvil espera un tiempo determinado antes de determinar que no hay respuesta y mostrar "NO DATA", dichos tiempos varían en aumentos de 4ms. Para este caso debido a que la emulación es conjunta y no existe comunicación como tal entre el ELM y la ECU, se muestra el mensaje OK y no tiene efecto alguno en el comportamiento del Arduino.

WS y Z, comandos de reinicio: A efectos de la programación del Arduino los 2 comandos trabajan de la misma forma, ya que causan un reinicio de todo el ELM y todas las características preestablecidas vuelven a su estado original como si se apagara y se encendiera de nuevo el ELM. La diferencia entre estos dos comandos aplicados a un ELM es que el comando ATZ reinicia también la luz de encendido y es un poco más lento que el comando ATWS.

@1, muestra la descripción del dispositivo: Muestra la descripción del dispositivo que normalmente en los ELM es "OBDII to RS232 Interpreter", pero ésta en algunos casos dependiendo de la interfaz gráfica puede ser cambiada.

@2, muestra la identificación del dispositivo: esta identificación de 12 caracteres puede ser guardada por un usuario mediante el comando @3 y es útil para fechas de producción, números seriales, códigos de producción, entre otros. Si no posee una identificación de dispositivo guardada muestra el carácter de error ("?").

@3 ccccccccccc, graba el identificador: Este comando se puede usar una sola vez en los ELM y luego que se use no puede ser removida o cambiada la información del identificador. En este caso no se usa este comando ya que si quiere cambiarse el identificador se puede cambiar desde la base del código del programa.

Apéndice B

Código de la emulación básica del ELM327 y el protocólo OBD:

Primera parte: Emulación de comunicación del ELM327.

```
String A,salto,header,spc,PCI;
int i;
int Band,echo,TOut,HEAD;
void setup(){
  TOut=25;
  echo=0;
  HEAD=0;
  salto="\r\n";
  header="";
  spc="";
  PCI="";
  Serial.begin(9600);
  Serial.print("ELM327v2,0"+salto+" > ");
  Serial.setTimeout(TOut);
}
void loop(){
```

```
Band=0;
A="";
if (Serial.available() >0) { //lectura del byte de entrada:
  A = Serial.readString();
  Serial.println();
  if (echo==1){
    Serial.println(A);
 }
  A.trim();
  A.replace("","");
  if (A.substring(0,4)=="ATAT"){ //Control adaptativo de tiempo
    Band=1;
    Serial.print("OK"+salto+" > ");
  }
 if (A.substring(0,5)=="ATCAF") //Elimina o activa el autoformateo CAN,
    Band=1; //es decir los mensajes se imprimen tal cual
    Serial.print("OK"+salto+" > "); //como los recibe el ELM. Ningun efecto
  }
  if (A.substring(0,5)==.ATCFC") //Elimina los mensajes de flow control del
    Band=1; //protolo CAN. Ningun efecto
    Serial.print("OK"+salto+" > ");

  if (A.substring(0,5)=="ATCRA") //Ajusta la mascara CAN
    Band=1;
    Serial.print("OK"+salto+" > ");
  }
  if (A.equalsIgnoreCase("ATD")) //Restaura todos los valores de fábrica
    echo=1;
    HEAD=0;
    salto="\r\n";
    header="";
    spc="";
```

```
    PCI="";
    Serial.print("OK"+salto+" > ");
  }
  if (A.equalsIgnoreCase("ATDP")){ //Muestra la descripcion el protocolo
    Band=1; // presente
    Serial.print("AUTO, ISO15765 − 4(CAN11/250)"+salto+" > ");
  }
  if (A.equalsIgnoreCase("ATDPN")){ //Muestra el numero del protocolo
    Band=1; // presente
    Serial.print("A8"+salto+(" > ");
  }
  if (A.equalsIgnoreCase("ATE0")) {//Desactiva el eco de los comandos
    Band=1; //(retransmitir cada comando recibido)
    echo=0;
    Serial.print("OK"+salto+" > ");
  }
  if (A.equalsIgnoreCase("ATE1")){ //Activa el eco de los comandos
    Band=1;
    echo=1;
    Serial.print("OK"+salto+" > ");
  }
  if (A.equalsIgnoreCase("ATH0")){ //Desactiva los headers de la respuesta del
    header=""; //vehiculo
    HEAD=0;
    Band=1;
    Serial.print("OK"+salto+(" > ");
  }
  if (A.equalsIgnoreCase("ATH1")){ //Activa los headers de la respuesta del
    header="7E8"+spc; //vehiculo
    HEAD=1;
    Band=1;
    Serial.print("OK"+salto+(" > ");
```

```
}
if (A.equalsIgnoreCase("ATI")){ //Hace que el chip se identifique a si mismo
  Band=1;
  Serial.print("ELM327v2,0"+salto+(" > ");
}
if (A.equalsIgnoreCase("ATL0")){ //Desactiva el salto de línea luego de enviar
  salto="\r"; //una respuesta
  Band=1;
  Serial.print("OK"+salto+(" > ");
}
if (A.equalsIgnoreCase("ATL1")){ //Activa el salto de línea luego de enviar
  salto="\r\n"; //una respuesta
  Band=1;
  Serial.print("OK"+salto+(" > ");
}
if (A.substring(0,3)=="ATM"){ //Desactiva o activa la memoria no volatil
  Band=1; //de los protocolos
  Serial.print("OK"+salto+(" > ");
}
if (A.equalsIgnoreCase("ATRV")){ //Muestra el voltaje de entrada del ELM
  Band=1;
  Serial.print("5V"+salto+(" > ");
}
if (A.equalsIgnoreCase("ATS0")){ //Elimina los espacios entre los caracteres
  Band=1; //de las respuestas
  spc="";
  Serial.print("OK"+salto+(" > ");
}
if (A.equalsIgnoreCase("ATS1")){ //Añade los espacios entre los caracteres
  Band=1;       //de las respuestas
  spc="";
  Serial.print("OK"+salto+" > ");
```

```
    }
    if (A.substring(0,4)=="ATSH"){ //Establece el header a usar en CAN
      Band=1; //Pero para este caso como se simula una
      Serial.print("OK"+salto+" > ");
//sola ECU el header de respuesta
                                  //permanece igual
    }
    if (A.substring(0,4)=="ATSP"){ //Selecciona la lectura del protocolo
      Band=1; //introducido, como se simula un solo
      Serial.print("OK\r\nSEARCHING..."+salto+" > "); //protocolo, se hace
    } //caso omiso al numero y se mantiene el protocolo
    if (A.substring(0,4)=="ATST"){ //Selecciona el tiempo de espera del ELM
      A=A.substring(0,6); //antes de determinar que no hay respuesta
      Band=1; //del vehiculo
      Serial.print("OK"+salto+" > ");
    }
    if (A.equalsIgnoreCase("ATWS")){ //Realiza un reseteo mas rapido que el ATZ
      echo=1;
      header="";
      HEAD=0;
      salto="\r\n";
      Band=1;
      Serial.print("ELM327v2,0"+salto+" > ");
    }
    if (A.equalsIgnoreCase("ATZ")){ //Realiza un reseteo al igual que ocurriria
      echo=1; //si se apaga y se prende el ELM
      header="";
      HEAD=0;
      salto="\r\n";
      Band=1;
      Serial.print("ELM327v2,0"+salto+" > ");
    }
```

```
if (A.equalsIgnoreCase("AT@1")){ //Muestra la descripcion del dispositivo
  Band=1;
  Serial.print("OBDIIEmulatortoRS232Interpreter"+salto+" > ");
}
if (A.equalsIgnoreCase("AT@2")){ //Muestra una descripcion guardada de 12
  Band=1; //caracteres
  Serial.print("FECHA12/2014"+salto+" > ");
}
```

Segunda parte: emulación de comandos OBD.

```
if (A.equalsIgnoreCase("0100")){ //Solicitud del PID 00 el cual muestra
  Band=1;              //todos los PIDs admitidos hasta el PID 20
  if (HEAD==1){
    PCI="06"+ spc ;
  }else{
    PCI="";
  }
  Serial.print(header+PCI+"41"+ spc +"00"+ spc +"00"+ spc +"18"+ spc +"00"+
  spc +"01"+salto+" > ");
}
if (A.equalsIgnoreCase("0120")){ //Solicitud del PID 20 el cual muestra
  Band=1; //todos los PIDs admitidos hasta el PID 40
  if (HEAD==1){
    PCI="06"+ spc ;
  }else{
    PCI="";
  }
  Serial.print(header+PCI+"41"+ spc +"20"+ spc +"00"+ spc +"02"+ spc +"00"+
  spc +"01"+salto+" > ");
}
if (A.equalsIgnoreCase("0140")){ //Solicitud del PID 20 el cual muestra
  Band=1; //todos los PIDs admitidos hasta el PID 60
  if (HEAD==1){
    PCI="06"+ spc ;
  }else{
    PCI="";
  }
  Serial.print(header+PCI+"41"+ spc +"40"+ spc +"00"+ spc +"00"+ spc +"00"+
  spc +"40"+salto+" > ");
}
```

```
if (A.equalsIgnoreCase("010C")){ //Solicitud de las RPM
  Band=1;
  if (HEAD==1){
    PCI="04"+ spc ;
  }else{
    PCI="";
  }
  Serial.print(header+PCI+"41"+ spc +"0C"+ spc +"01"+ spc +"A0"+salto+
  " > ");
}
if (A.equalsIgnoreCase("010D")){ //Solicitud de la velocidad
   Band=1;
   if (HEAD==1){
     PCI="03"+ spc ;
   }else{
     PCI="";
   }
   Serial.print(header+PCI+"41"+ spc +"0D"+ spc +"A0"+salto+" > ");
}
if (A.equalsIgnoreCase("012F")){ //Solicitud del nivel de gasolina
   Band=1;
   if (HEAD==1){
     PCI="03"+ spc ;
   }else{
     PCI="";
   }
   Serial.print(header+PCI+"41"+ spc +"2F"+ spc +"A0"+salto+" > ");
}
if (A.equalsIgnoreCase("015A")){ //Solicitud de la aceleración
   Band=1;
   if (HEAD==1){
     PCI="03"+ spc ;
```

```
        }else{
          PCI="";
        }
        Serial.print(header+PCI+"41"+ spc +"5A"+ spc +"A0"+salto+" > ");
      }
      if ((Ent.substring(0,2)=="01")&&(Band==0)){ //Solicitud de varios parámetros
        Ent.replace("01","");
        if (Ent.length()>2){
          Band=1;
          multiple(Ent,spc,header,salto,A,B,GasSalida,AccSalida,VelV);
        }
      }
      if (Band==0){
        Serial.print("NoDATA"+salto+" > ");
      }
    }
}

//Función "multiple"que imprime los datos en caso de solicitar varios PIDs.

void multiple(String Ent, String spc, String header, String salto, int A, int B, int Gas-
Salida, int AccSalida, int VelV){
  int NroBytesRecibidos;
  int NroBytesEnviados;
  int Ref=0;
  int Bandera=0;
  NroBytesRecibidos=Ent.length()/2;
  NroBytesEnviados=1;
  for (int cont=0; cont<Ent.length(); cont=cont+2){
    if (Ent.subString (cont,cont+2)==("0C")){
      NroBytesEnviados=NroBytesEnviados+3;
```

```
    }else{
      NroBytesEnviados=NroBytesEnviados+2;
    }
  }
  if (NroBytesRecibidos<=2){
    Serial.print(header+spc+"0");
    Serial.print(NroBytesEnviados,HEX);
  }else{
    Serial.print(header+spc+"10"+spc+"0");
    if (NroBytesEnviados<10){
      Serial.print(NroBytesEnviados,HEX);
    }else{
      Serial.print("A");
    }
  }
  Serial.print(spc+"41"+spc);
  NroBytesEnviados=0;
  while(Ref<NroBytesRecibidos){
    if (Ent.subString (Ref*2,Ref*2+2)==("5A")){/////Aceleracion
      if (NroBytesEnviados<5){
        Serial.print("5A"+spc);
        NroBytesEnviados++;
      }else {
        if (Bandera==0){
          Bandera=1;
          Serial.print(salto+header+spc+"21"+spc);
        }
        Serial.print("5A"+spc);
        NroBytesEnviados++;
      }
      if (NroBytesEnviados<5){
        if (AccSalida<16){
```

```
            Serial.print("0");
            Serial.print(AccSalida,HEX);
            Serial.print(spc);
          }else{
            Serial.print(AccSalida,HEX);
            Serial.print(spc);
          }
          NroBytesEnviados++;
        }else{
        if (Bandera==0){
          Bandera=1;
          Serial.print(salto+header+spc+"21"+spc);
        }
        if (AccSalida<16){
          Serial.print("0");
          Serial.print(AccSalida,HEX);
          Serial.print(spc);
        }else{
          Serial.print(AccSalida,HEX);
          Serial.print(spc);
        }
        NroBytesEnviados++;
        }
      }
      if (Ent.subString (Ref*2,Ref*2+2)==("2F")){/////Nivel de gasolina
        if (NroBytesEnviados<5){
          Serial.print("2F"+spc);
          NroBytesEnviados++;
        }else {
          if (Bandera==0){
            Bandera=1;
            Serial.print(salto+header+spc+"21"+spc);
```

```
        }
        Serial.print("2F"+spc);
        NroBytesEnviados++;
      }
      if (NroBytesEnviados<5){
        if (GasSalida<16){
          Serial.print("0");
          Serial.print(GasSalida,HEX);
          Serial.print(spc);
        }else{
          Serial.print(GasSalida,HEX);
          Serial.print(spc);
        }
        NroBytesEnviados++;
      }else{
        if (Bandera==0){
          Bandera=1;
          Serial.print(salto+header+spc+"21"+spc);
        }
        if (GasSalida<16){
          Serial.print("0");
          Serial.print(GasSalida,HEX);
          Serial.print(spc);
        }else{
          Serial.print(GasSalida,HEX);
          Serial.print(spc);
        }
        NroBytesEnviados++;
      }
    }
    if (Ent.subString (Ref*2,Ref*2+2)==("0C")){/////RPM
      if (NroBytesEnviados<5){
```

```
    Serial.print("0C"+spc);
    NroBytesEnviados++;
  }else {          if (Bandera==0){          Bandera=1;
      Serial.print(salto+header+spc+"21"+spc);
    }
    Serial.print("0C"+spc);
    NroBytesEnviados++;
  }
  if (NroBytesEnviados<5){
    if (A<16){
      Serial.print("0");
      Serial.print(A,HEX);
      Serial.print(spc);
    }else{
      Serial.print(A,HEX);
      Serial.print(spc);
    }
    NroBytesEnviados++;
  }else{
    if (Bandera==0){
      Bandera=1;
      Serial.print(salto+header+spc+"21"+spc);
    }
    if (A<16){
      Serial.print("0");
      Serial.print(A,HEX);
      Serial.print(spc);
    }else{
      Serial.print(A,HEX);
      Serial.print(spc);
    }
    NroBytesEnviados++;
```

```
    }
    if (NroBytesEnviados<5){
      if (B<16){
        Serial.print("0");
        Serial.print(B,HEX);
        Serial.print(spc);
      }else{
        Serial.print(B,HEX);
        Serial.print(spc);
      }
      NroBytesEnviados++;
    }else{
      if (Bandera==0){
        Bandera=1;
        Serial.print(salto+header+spc+"21"+spc);
      }
      if (B<16){
        Serial.print("0");
        Serial.print(B,HEX);
        Serial.print(spc);
      }else{
        Serial.print(B,HEX);
        Serial.print(spc);
      }
      NroBytesEnviados++;
    }
  }
  if (Ent.subString (Ref*2,Ref*2+2)==("0D")){/////Velocidad
    if (NroBytesEnviados<5){
      Serial.print("0D"+spc);
      NroBytesEnviados++;
    }else {
```

```
    if (Bandera==0){
      Bandera=1;
      Serial.print(salto+header+spc+"21"+spc);
    }
    Serial.print("0D"+spc);
    NroBytesEnviados++;
  }
  if (NroBytesEnviados<5){
    if (VelV<16){
      Serial.print("0");
      Serial.print(VelV,HEX);
      Serial.print(spc);
    }else{
      Serial.print(VelV,HEX);
      Serial.print(spc);
    }
    NroBytesEnviados++;
  }else{
    if (Bandera==0){
      Bandera=1;
      Serial.print(salto+header+spc+"21"+spc);
    }
    if (VelV<16){
      Serial.print("0");
      Serial.print(VelV,HEX);
      Serial.print(spc);
    }else{
      Serial.print(VelV,HEX);
      Serial.print(spc);
    }
    NroBytesEnviados++;
  }
```

```
    }
    Ref++;
  }
  Ref=NroBytesEnviados;
  Serial.print(salto" > ");
  if (Ref<=6){
    while(Ref<5){
      Serial.print("00"+spc);
      Ref++;
    }
    Serial.print(salto" > ");
  }else{
    while(Ref<12){
      Serial.print("00"+spc);
      Ref++;
    }
  Serial.print(salto" > ");
  }
}
```

Tercera parte: Códigos para la obtención de cada parámetro.

Nivel de gasolina

```
// Constantes (entradas y salidas)
const int GasVcc = 2; // Pin de alimentación del sensor de presion
const int GasLect = A0;; // Pin lector del voltaje analógico
// Variables:
int GasNivel = 0; //Variable de la presión obtenida de la entrada analógica
int GasSalida = 0; //Variable de la salida final correspondiente a la altura
//del tanque en %
void setup(){
```

```
  //Lectura del parametro nivel de gasolina
  Serial.begin(9600); //Inicio de la comunicacion Serial
  //Se inicializa la variable GasVcc como salida:
  pinMode(GasVcc, OUTPUT);
  //Se inicializa la variable de lectura como entrada:
  pinMode(GasLect, INPUT);
  VelR = VelR/100.0; //Se selecciona como voltaje de referencia el voltaje por de-
fecto (5V)
  analogReference(DEFAULT);
}
void loop(){
  delay(200); //Se selecciona un ratardo para mejor visualizacion
  //Lectura del voltaje de entrada
  //Activamos la alimentación Vcc
  digitalWrite(GasVcc, HIGH);
  Serial.print("Nivel = "); //Se comienza a mostrar el resultado
  //Se pone un tiempo de espera por el posbile transitorio del sensor
  delay(5);
  //Se lee el parámetro
  GasNivel = analogRead(GasLect);
  //Se apaga la alimentación para ahorrar energía
  digitalWrite(GasVcc, LOW);
  //Se definen los rangos de porcentajes respecto al voltaje
  if ((GasNivel <= 512)&&(GasNivel >510)){
    GasSalida = 90;
  }else if ((GasNivel <= 510)&&(GasNivel >509)){
    GasSalida = 80;
  }else if ((GasNivel <= 509)&&(GasNivel >507)){
    GasSalida = 70;
  }else if ((GasNivel <= 507)&&(GasNivel >504)){
    GasSalida = 60;
  }else if ((GasNivel <= 504)&&(GasNivel >497)){
```

```
    GasSalida = 50;
  }else if ((GasNivel <= 497)&&(GasNivel >479)){
    GasSalida = 40;
  }else if ((GasNivel <= 479)&&(GasNivel >420)){
    GasSalida = 30;
  }else if ((GasNivel <= 420)&&(GasNivel >258)){
    GasSalida = 20;
  }else if (GasNivel <= 258){
    GasSalida = 10;
  }
  //Se muestra el resultado
  Serial.print(GasSalida);
  Serial.println("%");
  //Para el envio al Xbee se debe mostrar el valor en hexadecimal
}
```

Obtención de la Velocidad

```
// Constantes (entradas y salidas)
const float VelTheta = 0.6981317; // Periodo angular en radianes de de pulso a pulso
const int VelVcc = 3; // Pin alimentador al circuito
const int VelLect = A1; // Pin lector de la entrada
// Variables:
float VelR = 29.718; // Radio de las ruedas del prototipo (cm)
unsigned long VelTv = 100; // Tiempo de ventana (ms y sin decimales)
int VelLimInf = 512; //Valores de referencia para comparar con la entrada
int VelLimSup = 500; //corresponde al voltaje con resolucion de 1024=5V
int VelValLect; //Valor que se lee de la entrada analógica
int VelBand = 0; //Bandera usada en el proceso
unsigned long VelT1; //Tiempos usados para determinar de la velocidad
unsigned long VelT2;
```

```
unsigned long VelAux; //Variable auxiliar para el calculo
int VelV = 0; //Velocidad final (salida)
void setup(){
  Serial.begin(9600); //Inicio de la comunicacion Serial
  //Se inicializa la variable VelVcc como salida:
  pinMode(VelVcc, OUTPUT);
  //Se inicializa la variable de lectura como entrada:
  pinMode(VelLect, INPUT);
  VelR = VelR/100.0; // Se transforma R, de cm a m
  VelTv = VelTv*1000; //Se transforma el ancho de la ventana a a microsegundos
}
void loop(){    //Lectura del parametro velocidad
  //Se selecciona un retardo para mejor visualizacion
  delay(200);    //Se activa la alimentación Vcc
  digitalWrite(VelVcc, HIGH);
  //Se reinician las variables que se necesiten calcular
  VelBand=VelV=VelT1=VelT2=0;
  //Como referencia de la ventana de muestra se toma el tiempo
  //del contador de microsegundos,que se presenta al instante
  //de comenzar a muestrear
  VelAux=micros();
  //Comienzo del ciclo de captura de tiempos
  while((micros()-VelAux)<VelTv){
    //Para cada ciclo se toma el valor actual del voltaje
    VelValLect = analogRead(VelLect);
    //Se verifica si es un valle
    if (VelValLect<=VelLimInf){
      //Si es el primer pulso se identifica en la bandera
      if (VelBand == 0){
        VelBand = 1;
      }
      //Si es el segundo pulso se identifica en la bandera
```

```
      if ((VelBand == 1)&&(VelT1 != 0)){
        VelBand = 2;
      }
    //Se verifica si es una cresta
    else if (VelValLect>=VelLimSup){
      //Si sale de un valle a una cresta se toma el primer tiempo
      if ((VelBand == 1)&&(VelT1 == 0)){
        VelT1 = micros();
        VelAux = micros();
    //Se cambia el valor de referencia de la
        //ventana para obtener una mejor medicion a velocidades bajas
      }
      //Si sale del 2do valle a la 2da cresta se toma el segundo tiempo
      if (VelBand == 2){
        VelT2 = micros();
        VelAux=0; //Se cambia el valor de referencia de la ventana para
        //obligar que el ciclo termine
      }
    }
  }
  //Se apaga la alimentación para ahorrar energía
  digitalWrite(VelVcc, LOW);
  //Se comienza a mostrar el resultado
  Serial.print("Velocidad = ");
  //Calculo de la velocidad
  if (VelT2 != 0){
    VelV = int(VelR*VelTheta*3600000.0/(VelT2-VelT1));
  }
     Serial.println(VelV); //Se muestra el resultado
  //Para el envio al Xbee se debe mostrar el valor en hexadecimal
}
```

Obtención de las RPM.

```
// Constantes (entradas y salidas)
const int RPMVcc = 4; //Pin de alimentación del circuito acondicionador
const int RPMLect = A2; //Pin lector del voltaje analógico
int RPMmin = 700; //Revoluciones minimas a medir
// Variables:
unsigned long RPMTv; //Tiempo de ventana
unsigned long RPMT1 = 0; //Tiempos para el calculo de las RPM
unsigned long RPMT2 = 0;
int RPMBand1 = 0;
int RPMBand2 = 0;
unsigned long RPMAux = 0;
unsigned long RPMK = 0;
int RPM = 0; //Variable de salida de RPM
void setup() {
  Serial.begin(9600); //Inicio de la comunicacion Serial
  //Se inicializa la variable RPMVcc como salida:
  pinMode(RPMVcc, OUTPUT);
  //Se inicializa la variable de lectura como entrada:
  pinMode(RPMLect, OUTPUT);
  //Tiempo de ventana equivalente en ms, Ej: a 800 RPM o
  //400 pulsos por segundo
  RPMTv=120000/RPMmin;
}
void loop(){
  //Lectura del parámetro RPM
  delay(150); //Se selecciona un ratardo para mejor visualizacion
  //Activamos la alimentación Vcc
  digitalWrite(RPMVcc, HIGH);
  Serial.print("RPM = "); //Se comienza a mostrar el resultado
  //Se reinician las variables que se necesiten calcular
```

```
RPMBand1=RPMBand2=RPM=0;
//Como inicio de la ventana de muestra se toma el tiempo en micros que
//Se utiliza otra variable como ventana cambiante en us
RPMK = RPMTv*1000;
//se presenta en al instante de comenzar a muestrear
RPMAux=micros();
//Se comienza el ciclo de muestreo
while ((micros()-RPMAux)<RPMK){
  switch (digitalRead(RPMLect)){
    case HIGH:{
      if (RPMBand1==0){ //Cuando es un primer pulso se toma el tiempo
        RPMT1=micros(); //de dicho pulso
        RPMAux=micros();
        RPMBand1=1;
        break;
      }
      if (RPMBand2==1){ //Cuando es un segundo pulso, se toma dicho
        RPMT2=micros(); //tiempo y ya se puede obtener el periodo de
        RPMK=0; //la señal
        break;
      }
      default:{
        break;
      }
    }
    case LOW:{ //Cuando hay una ausencia de pulso se registra
      if (RPMBand1==1){ //que ya termino el primer pulso
        RPMBand2=1;
      }
      break;
    }
  }
```

```
  }
  //Se apaga la alimentación para ahorrar energía
  digitalWrite(RPMVcc, LOW);
  //Se calcula y muestra las RPM final
  RPM = int(120000000/(RPMT2-RPMT1));
  Serial.println(RPM);
  //Para el envio al Xbee se debe mostrar el valor en hexadecimal
}
```

Obtención de la Aceleración

```
// Constantes (entradas y salidas)
const int AccVcc = 5; // Pin de alimentación del aceleremoetro
const int AccLect = A3; // Pin lector del voltaje analógico
// Variables:
float Acc = 0; //Variable de la entrada analógica obtenida y que
//se utilizara para calcular la aceleracion
int AccSalida = 0; //Valor final de salida de la aceleracion
void setup() {
  Serial.begin(9600); //Inicio de la comunicacion Serial
  //Se inicializa la variable AccVcc como salida:
  pinMode(AccVcc, OUTPUT);
  //Se inicializa la variable de lectura como entrada:
  pinMode(AccLect, INPUT);
  //Se selecciona como voltaje de referencia el voltaje por defecto (5V)
  analogReference(DEFAULT);
}
void loop(){
  //Lectura del parametro aceleracion
  delay(200); //Se selecciona un ratardo para mejor visualizacion
  //Lectura del voltaje de entrada
  //Activamos la alimentación Vcc
```

```
  digitalWrite(AccVcc, HIGH);
  Serial.print("Aceleracion = "); //Se comienza a mostrar el resultado
  //Se pone un tiempo de espera por el transitorio del sensor
  delay(3);
  //Se lee el parámetro
  Acc = (float)analogRead(AccLect);
  //Se apaga la alimentación para ahorrar energía
  digitalWrite(AccVcc, LOW);
  //Se transforma la variable de valor binario al valor del voltaje y se
  //hacen las operaciones correspondientes, para obtener la velocidad
  Acc = (0.144229*Acc-49.32567);
  //Se toma la parte entera del resultado y se muestra en pantalla
  AccSalida = int(Acc);
  Serial.println(AccSalida);
  //Para el envio al Xbee se debe mostrar el valor en hexadecimal
}
```

Apéndice C

PIDs del modo 01

Tabla C.1: PIDs del modo 01 (00-12).

PID (hex)	Data bytes returned	Description	Units	Formula
00	4	PIDs supported [01 - 20]		Bit encoded [A7..D0] == [PID $01..PID $20]
01	4	Monitor status since DTCs cleared. (Includes malfunction indicator lamp (MIL) status and number of DTCs.)		Bit encoded.
02	2	Freeze DTC		
03	2	Fuel system status		Bit encoded.
04	1	Calculated engine load value	%	A*100/255
05	1	Engine coolant temperature	°C	A-40
06	1	Short term fuel % trim—Bank 1	%	(A-128) * 100/128
07	1	Long term fuel % trim—Bank 1	%	(A-128) * 100/128
08	1	Short term fuel % trim—Bank 2	%	(A-128) * 100/128
09	1	Long term fuel % trim—Bank 2	%	(A-128) * 100/128
0A	1	Fuel pressure	kPa (gauge)	A*3
0B	1	Intake manifold absolute pressure	kPa (absolute)	A
0C	2	Engine RPM	rpm	((A*256)+B)/4
0D	1	Vehicle speed	km/h	A
0E	1	Timing advance	° relative to #1 cylinder	(A-128)/2 (A-128)/2
0F	1	Intake air temperature	°C	A-40
10	2	MAF air flow rate	grams/sec	((A*256)+B) / 100
11	1	Throttle position	%	A*100/255
12	1	Commanded secondary air status		Bit encoded.

Tabla C.2: PIDs del modo 01 (13-1C).

13	1	Oxygen sensors present		[A0..A3] == Bank 1, Sensors 1-4. [A4..A7] == Bank 2...
14	2	Bank 1, Sensor 1: Oxygen sensor voltage Short term fuel trim	Volts %	A/200 (B-128) * 100/128 (if B==$FF, sensor is not used in trim calc)
15	2	Bank 1, Sensor 2: Oxygen sensor voltage Short term fuel trim	Volts %	A/200 (B-128) * 100/128 (if B==$FF, sensor is not used in trim calc)
16	2	Bank 1, Sensor 3: Oxygen sensor voltage Short term fuel trim	Volts %	A/200 (B-128) * 100/128 (if B==$FF, sensor is not used in trim calc)
17	2	Bank 1, Sensor 4: Oxygen sensor voltage Short term fuel trim	Volts %	A/200 (B-128) * 100/128 (if B==$FF, sensor is not used in trim calc)
18	2	Bank 2, Sensor 1: Oxygen sensor voltage Short term fuel trim	Volts %	A/200 (B-128) * 100/128 (if B==$FF, sensor is not used in trim calc)
19	2	Bank 2, Sensor 2: Oxygen sensor voltage Short term fuel trim	Volts %	A/200 (B-128) * 100/128 (if B==$FF, sensor is not used in trim calc)
1A	2	Bank 2, Sensor 3: Oxygen sensor voltage Short term fuel trim	Volts %	A/200 (B-128) * 100/128 (if B==$FF, sensor is not used in trim calc)
1B	2	Bank 2, Sensor 4: Oxygen sensor voltage Short term fuel trim	Volts %	A/200 (B-128) * 100/128 (if B==$FF, sensor is not used in trim calc)
1C	1	OBD standards this vehicle conforms to		Bit encoded.

Tabla C.3: PIDs del modo 01 (1D-28).

1D	1	Oxygen sensors present		Similar to PID 13, but [A0..A7] == [B1S1, B1S2, B2S1, B2S2, B3S1, B3S2, B4S1, B4S2]
1E	1	Auxiliary input status		A0 == Power Take Off (PTO) status (1 == active) [A1..A7] not used
1F	2	Run time since engine start	seconds	(A*256)+B
20	4	PIDs supported [21 - 40]		Bit encoded [A7..D0] == [PID $21..PID $40]
21	2	Distance traveled with malfunction indicator lamp (MIL) on	km	(A*256)+B
22	2	Fuel Rail Pressure (relative to manifold vacuum)	kPa	((A*256)+B) * 0.079
23	2	Fuel Rail Pressure (diesel, or gasoline direct inject)	kPa (gauge)	((A*256)+B) * 10
24	4	O2S1_WR_lambda(1): Equivalence Ratio Voltage	N/A V	((A*256)+B)*2/65535 or ((A*256)+B)/32768 ((C*256)+D)*8/65535 or ((C*256)+D)/8192
25	4	O2S2_WR_lambda(1): Equivalence Ratio Voltage	N/A V	((A*256)+B)*2/65535 ((C*256)+D)*8/65535
26	4	O2S3_WR_lambda(1): Equivalence Ratio Voltage	N/A V	((A*256)+B)*2/65535 ((C*256)+D)*8/65535
27	4	O2S4_WR_lambda(1): Equivalence Ratio Voltage	N/A V	((A*256)+B)*2/65535 ((C*256)+D)*8/65535
28	4	O2S5_WR_lambda(1): Equivalence Ratio Voltage	N/A V	((A*256)+B)*2/65535 ((C*256)+D)*8/65535

Tabla C.4: PIDs del modo 01 (29-36).

29	4	O2S6_WR_lambda(1): Equivalence Ratio Voltage	 N/A V	 ((A*256)+B)*2/65535 ((C*256)+D)*8/65535
2A	4	O2S7_WR_lambda(1): Equivalence Ratio Voltage	 N/A V	 ((A*256)+B)*2/65535 ((C*256)+D)*8/65535
2B	4	O2S8_WR_lambda(1): Equivalence Ratio Voltage	 N/A V	 ((A*256)+B)*2/65535 ((C*256)+D)*8/65535
2C	1	Commanded EGR	%	A*100/255
2D	1	EGR Error	%	(A-128) * 100/128
2E	1	Commanded evaporative purge	%	A*100/255
2F	1	Fuel Level Input	%	A*100/255
30	1	# of warm-ups since codes cleared	N/A	A
31	2	Distance traveled since codes cleared	km	(A*256)+B
32	2	Evap. System Vapor Pressure	Pa	((A*256)+B)/4 (A and B are two's complement signed)
				complement signed)
33	1	Barometric pressure	kPa (Absolute)	A
34	4	O2S1_WR_lambda(1): Equivalence Ratio Current	 N/A mA	 ((A*256)+B)/32,768 ((C*256)+D)/256 - 128
35	4	O2S2_WR_lambda(1): Equivalence Ratio Current	 N/A mA	 ((A*256)+B)/32,768 ((C*256)+D)/256 - 128
36	4	O2S3_WR_lambda(1): Equivalence Ratio Current	 N/A mA	 ((A*256)+B)/32,768 ((C*256)+D)/256 - 128

Tabla C.5: PIDs del modo 01 (37-47).

37	4	O2S4_WR_lambda(1): Equivalence Ratio Current	 N/A mA	 ((A*256)+B)/32,768 ((C*256)+D)/256 - 128
38	4	O2S5_WR_lambda(1): Equivalence Ratio Current	 N/A mA	 ((A*256)+B)/32,768 ((C*256)+D)/256 - 128
39	4	O2S6_WR_lambda(1): Equivalence Ratio Current	 N/A mA	 ((A*256)+B)/32,768 ((C*256)+D)/256 - 128
3A	4	O2S7_WR_lambda(1): Equivalence Ratio Current	 N/A mA	 ((A*256)+B)/32,768 ((C*256)+D)/256 - 128
3B	4	O2S8_WR_lambda(1): Equivalence Ratio Current	 N/A mA	 ((A*256)+B)/32,768 ((C*256)+D)/256 - 128
3C	2	Catalyst Temperature Bank 1, Sensor 1	°C	((A*256)+B)/10 - 40
3D	2	Catalyst Temperature Bank 2, Sensor 1	°C	((A*256)+B)/10 - 40
3E	2	Catalyst Temperature Bank 1, Sensor 2	°C	((A*256)+B)/10 - 40
3F	2	Catalyst Temperature Bank 2, Sensor 2	°C	((A*256)+B)/10 - 40
40	4	PIDs supported [41 - 60]		Bit encoded [A7..D0] == [PID $41..PID $60]
41	4	Monitor status this drive cycle		Bit encoded.
42	2	Control module voltage	V	((A*256)+B)/1000
43	2	Absolute load value	%	((A*256)+B)*100/255
44	2	Fuel/Air commanded equivalence ratio	N/A	((A*256)+B)/32768
45	1	Relative throttle position	%	A*100/255
46	1	Ambient air temperature	°C	A-40
47	1	Absolute throttle position B	%	A*100/255

Tabla C.6: PIDs del modo 01 (48-5B).

48	1	Absolute throttle position C	%	A*100/255
49	1	Accelerator pedal position D	%	A*100/255
4A	1	Accelerator pedal position E	%	A*100/255
4B	1	Accelerator pedal position F	%	A*100/255
4C	1	Commanded throttle actuator	%	A*100/255
4D	2	Time run with MIL on	minutes	(A*256)+B
4E	2	Time since trouble codes cleared	minutes	(A*256)+B
4F	4	Maximum value for equivalence ratio, oxygen sensor voltage, oxygen sensor current, and intake manifold absolute pressure	, V, mA, kPa	A, B, C, D*10
50	4	Maximum value for air flow rate from mass air flow sensor	g/s	A*10, B, C, and D are reserved for future use
51	1	Fuel Type		From fuel type table
52	1	Ethanol fuel %	%	A*100/255
53	2	Absolute Evap system Vapor Pressure	kPa	((A*256)+B)/200
54	2	Evap system vapor pressure	Pa	((A*256)+B)-32767
55	2	Short term secondary oxygen sensor trim bank 1 and bank 3	%	(A-128)*100/128 (B-128)*100/128
56	2	Long term secondary oxygen sensor trim bank 1 and bank 3	%	(A-128)*100/128 (B-128)*100/128
57	2	Short term secondary oxygen sensor trim bank 2 and bank 4	%	(A-128)*100/128 (B-128)*100/128
58	2	Long term secondary oxygen sensor trim bank 2 and bank 4	%	(A-128)*100/128 (B-128)*100/128
59	2	Fuel rail pressure (absolute)	kPa	((A*256)+B) * 10
5A	1	Relative accelerator pedal position	%	A*100/255
5B	1	Hybrid battery pack remaining life	%	A*100/255

Tabla C.7: PIDs del modo 01 (5C-72).

5C	1	Engine oil temperature	°C	A - 40
5D	2	Fuel injection timing	°	(((A*256)+B)-26,880)/128
5E	2	Engine fuel rate	L/h	((A*256)+B)*0.05
5F	1	Emission requirements to which vehicle is designed		Bit Encoded
60	4	PIDs supported [61 - 80]		Bit encoded [A7..D0] == [PID $61..PID $80]
61	1	Driver's demand engine - percent torque	%	A-125
62	1	Actual engine - percent torque	%	A-125
63	2	Engine reference torque	Nm	A*256+B
64	5	Engine percent torque data	%	A-125 Idle B-125 Engine point 1 C-125 Engine point 2 D-125 Engine point 3 E-125 Engine point 4
65	2	Auxiliary input / output supported		Bit Encoded
66	5	Mass air flow sensor		
67	3	Engine coolant temperature		
68	7	Intake air temperature sensor		
69	7	Commanded EGR and EGR Error		
6A	5	Commanded Diesel intake air flow control and relative intake air flow position		
6B	5	Exhaust gas recirculation temperature		
6C	5	Commanded throttle actuator control and relative throttle position		
6D	6	Fuel pressure control system		
6E	5	Injection pressure control system		
6F	3	Turbocharger compressor inlet pressure		
70	9	Boost pressure control		
71	5	Variable Geometry turbo (VGT) control		
72	5	Wastegate control		

Tabla C.8: PIDs del modo 01 (73-C4).

73	5	Exhaust pressure		
74	5	Turbocharger RPM		
75	7	Turbocharger temperature		
76	7	Turbocharger temperature		
77	5	Charge air cooler temperature (CACT)		
78	9	Exhaust Gas temperature (EGT) Bank 1		Special PID.
79	9	Exhaust Gas temperature (EGT) Bank 2		Special PID.
7A	7	Diesel particulate filter (DPF)		
7B	7	Diesel particulate filter (DPF)		
7C	9	Diesel Particulate filter (DPF) temperature		
7D	1	NOx NTE control area status		
7E	1	PM NTE control area status		
7F	13	Engine run time		
80	4	PIDs supported [81 - A0]		Bit encoded [A7..D0] == [PID 81..PIDA0]
81	21	Engine run time for Auxiliary Emissions Control Device(AECD)		
82	21	Engine run time for Auxiliary Emissions Control Device(AECD)		
83	5	NOx sensor		
84		Manifold surface temperature		
85		NOx reagent system		
86		Particulate matter (PM) sensor		
87		Intake manifold absolute pressure		
A0	4	PIDs supported [A1 - C0]		Bit encoded [A7..D0] == [PID A1..PIDC0]
C0	4	PIDs supported [C1 - E0]		Bit encoded [A7..D0] == [PID C1..PIDE0]
C3	?	?	?	Returns numerous data, including Drive Condition ID and Engine Speed*
C4	?	?	?	B5 is Engine Idle Request B6 is Engine Stop Request*

Apéndice D

Programación final del Arduino

Primer código: Comunicación remota (4 parámetros).

```
////////////////////////////////////////
Funcion de las RPM ////////////////////////
////////////////////////////////////////
int FuncRPM(int RPMLect){
////////////// Constantes y variables de RPM/////////////////
unsigned long RPMT1 = 0; //Tiempos para el calculo de las RPM
unsigned long RPMT2 = 0;
int  RPMBand1;
int  RPMBand2;
unsigned long RPMAux;
unsigned long RPMK;
int  RPM;
int  RPMmin = 1000;
unsigned long RPMTv;
  RPMBand1=RPMBand2=RPM=0;
  RPMTv=60000/RPMmin;
  RPMK = RPMTv*1000;
  if (digitalRead(RPMLect)==HIGH){
```

```
    delay(12);
  }
  RPMAux=micros();
  while ((micros()-RPMAux)<RPMK){
    switch (digitalRead(RPMLect)){
      case HIGH:{
        delayMicroseconds(100);
        if (digitalRead(RPMLect)==HIGH){
          if (RPMBand1==0){ //Cuando es un primer pulso se toma el tiempo
            RPMT1=micros(); //de dicho pulso
            RPMAux=micros();
            RPMBand1=1;
            delay(12);
            break;
          }
          if (RPMBand2==1){ //Cuando es un segundo pulso, se toma dicho
            RPMT2=micros(); //tiempo y ya se puede obtener el periodo de
            RPMK=0; //la señal


            break;
          }
        }
        break;
      }
      case LOW:{
        if (RPMBand1==1){
          RPMBand2=1;
        }
        break;
      }
    }
  }
```

```
  if (RPMT2!=0){
    RPM = int(60000000/(RPMT2-RPMT1));
  }
  return RPM;
}
//////////////////////////////////////
Funcion de la velocidad ////////////////////
//////////////////////////////////////
int FuncVel(int VelLect){
unsigned long VelTv = 100;
int VelLimInf = 507;
int VelLimSup = 508; /*Valores de referencia para comparar con la entrada
corresponde al voltaje con resolucion de 1024=5V*/

int VelBand = 0;
int VelA;
unsigned long VelT1;
unsigned long VelT2;
unsigned long VelAux;
int VelV;
const float VelTheta = 0.6981317; //Periodo angular en radianes de de pulso a pul-
so
float VelR = 29.718; // Radio de las ruedas del prototipo (cm)
int VelValLect; //Valor que se lee de la entrada analógica
int K, cont;
int N = 15;
  VelTv = VelTv*1000;
  VelR = VelR/100.0;
  VelBand=VelV=VelT1=VelT2=K=0;
  VelAux=micros();
  K = analogRead(VelLect);
  while ((micros()-VelAux)<VelTv){
```

```
    VelValLect = cont = 0;
    VelValLect = analogRead(VelLect);
    if ((VelValLect>=K-1)&&(VelValLect<=K+1)){
      if (VelValLect<VelLimInf){
        VelBand = 1;
        if ((VelBand == 1)&&(VelT1 != 0)){
          VelBand = 2;
        }
      }else if (VelValLect>VelLimSup){
        if ((VelBand == 1)&&(VelT1 == 0)){
          VelT1 = micros();
          VelAux = micros();
          delayMicroseconds(8300);
        }
        if (VelBand == 2){
          VelT2 = micros();
          VelAux=0;
        }
      }
    }
    K = analogRead(VelLect);
  }
  if (VelT2 != 0){
    VelV = int(VelR*VelTheta*3600000.0/(VelT2-VelT1));
  }
  return VelV;
}
////////////////////////////////////////
Funcion de la Aceleracion ////////////////////
////////////////////////////////////////
int FuncAcc(int AccLect){
float Acc; //Variable de la entrada analógica obtenida y que
```

```
int AccLeida = 0; //Aceleracion leida del ADC
float AccSens = 0.3465; //(1.704-1.353)=0.351... (2.046-1.704)=0.342
float AccV0 = 1.699;
int  AccSalida; //Valor final de salida de la aceleracion
  AccLeida = analogRead(AccLect);
  Acc = (((5.0*AccLeida-1024.0*AccV0)*9.80665)/(1024.0*AccSens));
  if (Acc<=0){
    Acc=Acc*-1;
  }
  AccSalida = int(Acc*255/100);
  return AccSalida;
}
////////////////////////////////////////
Funcion de la Gasolina ////////////////////
////////////////////////////////////////
int FuncGas(int GasLect){
int GasNivel = 0;
int GasSalida = 0;
  delay(5);
  GasNivel = analogRead(GasLect);
  if (GasNivel <= 510){
    GasSalida = 92;
  }else if ((GasNivel >510)&&(GasNivel <= 511)){
    GasSalida = 84;
  }else if ((GasNivel >511)&&(GasNivel <= 512)){
    GasSalida = 76;
  }else if ((GasNivel >512)&&(GasNivel <= 513)){
    GasSalida = 69;
  }else if ((GasNivel >513)&&(GasNivel <= 515)){
    GasSalida = 61;
  }else if ((GasNivel >515)&&(GasNivel <= 518)){
    GasSalida = 53;
```

```
  }else if ((GasNivel >= 518)&&(GasNivel <= 528)){
    GasSalida = 46;
  }else if ((GasNivel >= 528)&&(GasNivel <= 553)){
    GasSalida = 38;
  }else if ((GasNivel >= 553)&&(GasNivel <= 742)){
    GasSalida = 30;
  }else if (GasNivel >742){
    GasSalida = 0;
  }
  return GasSalida;
}
//////////////////////////////////////
Comienzo del programa ////////////////////
//////////////////////////////////////
String Ent,salto,header,spc,PCI,Salida1,Salida2;
int Band,echo,TOut,HEAD,A,B,NroBytesEnviados,NroBytesRecibidos,Ref;
unsigned long contador1,contador2;
////////// Constantes y variables de RPM/////////////
const int RPMLect = A4; //Pin lector del voltaje analógico
int  RPM; //Variable de salida de RPM
////////// Constantes y variables de velocidad //////////
const int VelLect = A1;
int VelV = 0;
////////// Constantes y variables de la gasolina ///////////
const int GasLect = A0; // Pin lector del voltaje analógico
int  GasValor;
int GasCont = 0;
int GasSalida; //Variable de la salida final correspondiente a la altura
//del tanque en %
////////// Constantes y variables de la aceleracion ////////
const int AccLect = A3; // Pin lector del voltaje analógico
int  AccSalida = 0; //Valor final de salida de la aceleracion
```

```
void setup(){
  contador1=contador2=0;
  TOut=25;
  echo=1;
  HEAD=0;
  salto="\r\n";
  header="";
  spc="";
  PCI="";
  pinMode(RPMLect, INPUT);
  pinMode(VelLect, INPUT);
  pinMode(GasLect, INPUT);
  pinMode(AccLect, INPUT);
  analogReference(DEFAULT);
  Serial.begin(9600);
  delay(2000);
  Serial.print(salto);
  delay(1000);
  Serial.print("B"+salto);
  delay(1000);
  Serial.print("ELM327v2,0"+salto+" > ");
  Serial.setTimeout(TOut);
}
void loop(){
  Band=NroBytesRecibidos=NroBytesEnviados=0;
  Ent="";
  if (Serial.available() >0) {
  // lectura del byte de entrada:
    Ent = Serial.readString();
    Serial.println();
    if (echo==1){
      Serial.println(Ent);
```

```
    }
    Ent.trim();
    Ent.replace("","");
    if (Ent.substring(0,4)=="ATAT"){          //Control adaptativo de tiempo
      Band=1;
      Serial.print("OK"+salto+" > ");
    }
    if (Ent.substring(0,5)=="ATCAF"){ //Elimina o activa el autoformateo CAN,
es
      Band=1;
//decir los mensajes se imprimen tal cual
      Serial.print("OK"+salto+" > "); //como los recibe el ELM. Ningun efecto
    }
    if (Ent.substring(0,5)=="ATCFC"){ //Elimina los mensajes de fLOW control
del
      Band=1; //protolo CAN. Ningun efecto
      Serial.print("OK"+salto+" > ");
    }
    if (Ent.substring(0,5)=="ATCRA"){ //Ajusta la mascara CAN
      Band=1;
      Serial.print("OK"+salto+" > ");
    }
    if (Ent.equalsIgnorecase("ATD")){ //Restaura todos los valores de fábrica
      echo=1;
      HEAD=0;
      salto="\r\n";
      header="";
      spc="";
      PCI="";
      Serial.print("OK"+salto+" > ");
    }
```

```
if (Ent.equalsIgnorecase("ATDP")){ //Muestra la descripcion el protocolo pre-
sente
  Band=1;
  Serial.print("AUTO, ISO15765 − 4(CAN11/250)"+salto+" > ");
}
if (Ent.equalsIgnorecase("ATDPN")){ //Muestra el numero del protocolo pre-
sente
  Band=1;
  Serial.print("A8"+salto+" > ");
}
if (Ent.equalsIgnorecase("ATE0")){ //Desactiva el eco de los comandos
  Band=1; //(retransmitir cada comando recibido)
  echo=0;
  Serial.print("OK"+salto+" > ");
}
if (Ent.equalsIgnorecase("ATE1")){//Activa el eco de los comandos
  Band=1;
  echo=1;
  Serial.print("OK"+salto+" > ");
}
if (Ent.equalsIgnorecase("ATH0")){ //Desactiva los headers de la respuesta del
  header=""; //vehiculo
  HEAD=0;
  Band=1;
  Serial.print("OK"+salto+" > ");
}
if (Ent.equalsIgnorecase("ATH1")){ //Activa los headers de la respuesta del
  header="7E8"+spc; //vehiculo
  HEAD=1;
  Band=1;
  Serial.print("OK"+salto+" > ");
}
```

```
if (Ent.equalsIgnorecase("ATI")){ //Hace que el chip se identifique a si mismo
  Band=1;
  Serial.print("ELM327v2,0"+salto+" > ");
}
 if (Ent.equalsIgnorecase("ATL0")){ ////Desactiva el salto de línea luego de enviar
  salto="\r"; //una respuesta
  Band=1;
  Serial.print("OK"+salto+" > ");
}
if (Ent.equalsIgnorecase("ATL1")){ //Activa el salto de línea luego de enviar
  salto="\r\n"; //una respuesta
  Band=1;
  Serial.print("OK"+salto+" > ");
}
if (Ent.substring(0,3)=="ATM"){ //Desactiva o activa la memoria no volatil
  Band=1; //de los protocolos
  Serial.print("OK"+salto+" > ");
}
if (Ent.equalsIgnorecase("ATRV")){ //Muestra el voltaje de entrada del ELM
  Band=1;
  Serial.print("5V"+salto+" > ");
}
if (Ent.equalsIgnorecase("ATS0")){ //Elimina los espacios entre los caracteres
  Band=1; //de las respuestas
  spc="";
  Serial.print("OK"+salto+" > ");
}
if (Ent.equalsIgnorecase("ATS1")){ //Añade los espacios entre los caracteres
  Band=1; //de las respuestas
  spc="";
  Serial.print("OK"+salto+" > ");
```

```
    }
    if (Ent.substring(0,4)=="ATSH"){
//Establece el header a usar en CAN        Band=1; //Pero para este caso como se
simula una
      Serial.print("OK"+salto+" > ");
//sola ECU el header de respuesta          //permanece igual
    }
      if (Ent.substring(0,4)=="ATSP"){ //Selecciona la lectura del protocolo
      Band=1; //introducido, como se simula un solo
      Serial.print("OK\r\nSEARCHING..."+salto+" > ");
//protocolo, se hace caso      } //omiso al numero y se mantiene el protocolo
    if (Ent.substring(0,4)=="ATST"){ //Selecciona el tiempo de espera del ELM
      Band=1;
antes de determinar que no hay respuesta
      Serial.print("OK"+salto+" > "); //del vehiculo
    }
     if (Ent.equalsIgnorecase("ATWS")){ //Realiza un reseteo mas rapido que el
ATZ
      echo=1;
      header="";
      HEAD=0;
      salto="\r\n";
      Band=1;
      Serial.print("ELM327v2,0"+salto+" > ");
    }
    if (Ent.equalsIgnorecase("ATZ")){ //Realiza un reseteo al igual que ocurriría
      echo=1; //si se apaga y se prende el ELM
      header="";
      HEAD=0;
      salto="\r\n";
      Band=1;
      Serial.print("ELM327v2,0"+salto+" > ");
```

```
    }
    if (Ent.equalsIgnorecase("AT@1")){ //Muestra la descripcion del dispositivo
      Band=1;
      Serial.print("OBDIIEmulatortoRS232interpreter"+salto+" > ");
    }
     if (Ent.equalsIgnorecase("AT@2")){ //Muestra una descripcion guardada de
12
      Band=1; //caracteres
      Serial.print("FECHA12/2014"+salto+" > ");
    }
////////////////////////////////////
/////////// Comandos OBD ////////////
////////////////////////////////////
    if (Ent.equalsIgnorecase("0100")){ //Solicitud del PID 00 el cual muestra todos
los PIDs
      Band=1; //admitidos hasta el PID 20
      if (HEAD==1){
        PCI="06"+spc;
      }else{
        PCI="";
      }
      Serial.print(header+PCI+"41"+spc+"00"+spc+"00"+spc+"18"+spc+
"00"+spc+"01"+salto+" > ");
    }
    if (Ent.equalsIgnorecase("0120")){ //Solicitud del PID 20 el cual muestra todos
los PIDs
      Band=1; //admitidos hasta el PID 40
      if (HEAD==1){
        PCI="06"+spc;
      }else{
        PCI="";
      }
```

```
        Serial.print(header+PCI+"41"+spc+"20"+spc+"00"+spc+"02"+spc+
"00"+spc+"01"+salto+" > ");
    }
    if (Ent.equalsIgnorecase("0140")){ //Solicitud del PID 20 el cual muestra todos
los PIDs
        Band=1; //admitidos hasta el PID 60
        if (HEAD==1){
          PCI="06"+spc;
        }else{
          PCI="";
        }
        Serial.print(header+PCI+"41"+spc+"40"+spc+"00"+spc+"00"+spc+
"00"+spc+"40"+salto+" > ");
    }
////////////////////////////////////////
///////////////// RPM /////////////////
////////////////////////////////////////
    if (Ent.equalsIgnorecase("010C")){
        Band=1;
        if (HEAD==1){
          PCI="04"+spc;
        }else{
          PCI="";
        }
        Serial.print(header+PCI+"41"+spc+"0C"+spc);
          if (A<16){
            Serial.print('0');
          }
          Serial.print(A,HEX);
          Serial.print(spc);
          if (B<16){
          Serial.print('0');
```

```
        }
        Serial.print(B,HEX);
        Serial.print(salto+" > ");
    }
////////////////////////////////////////////
///////////////// Velocidad //////////////
////////////////////////////////////////////
    if (Ent.equalsIgnorecase("010D")){
        Band=1;
        if (HEAD==1){
            PCI="03"+spc;
        }else{
            PCI="";
        }
        Serial.print(header+PCI+"41"+spc+"0D"+spc);
        if (VelV<16){
            Serial.print('0');
        }
        Serial.print(VelV,HEX);
        Serial.print(salto+" > ");
    }
////////////////////////////////////////////
///////////////// Gasolina //////////////
////////////////////////////////////////////
    if (Ent.equalsIgnorecase("012F")){
        Band=1;
        if (HEAD==1){
            PCI="03"+spc;
        }else{
            PCI="";
        }
        Serial.print(header+PCI+"41"+spc+"2F"+spc);
```

```
        if (GasSalida<16){
          Serial.print('0');
        }
        Serial.print(GasSalida,HEX);
        Serial.print(salto+" > ");
      }
////////////////////////////////////////
/////////////// Aceleracion //////////////
////////////////////////////////////////
      if (Ent.equalsIgnorecase("015A")){ //Solicitud de la Aceleracion
        Band=1;
        if (HEAD==1){
          PCI="03"+spc;
        }else{
          PCI="";
        }
        Serial.print(header+PCI+"41"+spc+"5A"+spc);
        if (AccSalida<16){
          Serial.print('0');
        }
        Serial.print(AccSalida,HEX);
        Serial.print(salto+" > ");
      }
////////////////////////////////////////////////
////////////// Múltiples parámetros ////////////////////
////////////////////////////////////////////////
      if ((Ent.substring(0,2)=="01")&&(Band==0)){
        Ent.replace("01","");
        if (Ent.length()>2){
          Band=1;
          multiple(Ent,spc,header,salto,A,B,GasSalida,AccSalida,VelV);
        }
```

```
    }
    if (Band==0){
      Serial.print("NODATA"+salto+" > ");
    }
  }
////////////////////////////////////////////////////
////////////// Calculo de los parametros ///////////////////
////////////////////////////////////////////////////
//////////////////////////////// RPM
  RPM=FuncRPM(RPMLect);
  if (RPM>=2000){
    for (int i=0; i <3; i++){
      RPM=RPM+FuncRPM(RPMLect);
    }
  }
  RPM=RPM*4/4;
  A=RPM/256;
  B=RPM %256;
//////////////////////////////// Velocidad
  VelV=FuncVel(VelLect);
  if ((VelV>=20)&&(VelV<40)){
    for (int i=0; i <2; i++){
      VelV=VelV+FuncVel(VelLect);
    }
    VelV=VelV/3;
  }else if (VelV>=40){
    for (int i=0; i <5; i++){
      VelV=VelV+FuncVel(VelLect);
    }
    VelV=VelV/6;
  }
//////////////////////////////// Aceleración
```

```
  AccSalida = FuncAcc(AccLect);
  AccSalida = AccSalida*255/100;
//////////////////////////// Gasolina
  if (millis()-contador2>20000){
    if (GasCont = 0){
      GasValor = FuncGas(GasLect);
      GasSalida = GasValor;
      GasCont = 1;
    }
    if (GasCont <15){
      GasCont++;
      GasValor = GasValor + FuncGas(GasLect);
    }else if (GasCont = 15){
      GasSalida = GasValor/15;
      GasCont=1;
      GasValor = FuncGas(GasLect);
    }
    GasSalida = GasSalida*255/100;
    contador2=millis();
  }
}

//////////////////////////// Final del programa

//////////////////////////// Funcion de múltiples parámetros
void multiple(String Ent, String spc, String header, String salto, int A, int B, int Gas-
Salida, int AccSalida, int VelV){
  int NroBytesRecibidos;
  int NroBytesEnviados;
  int Ref=0;
  int Bandera=0;
  NroBytesRecibidos=Ent.length()/2;
```

```
NroBytesEnviados=1;
for (int cont=0; cont<Ent.length(); cont=cont+2){
  if (Ent.substring(cont,cont+2)==("0C")){
    NroBytesEnviados=NroBytesEnviados+3;
  }else{
    NroBytesEnviados=NroBytesEnviados+2;
  }
}
if (NroBytesRecibidos<=2){
  Serial.print(header+spc+"0");
  Serial.print(NroBytesEnviados,HEX);
}else{
  Serial.print(header+spc+"10"+spc+"0");
  if (NroBytesEnviados<10){
    Serial.print(NroBytesEnviados,HEX);
  }else{
    Serial.print("A");
  }
}
Serial.print(spc+"41"+spc);
NroBytesEnviados=0;
while(Ref<NroBytesRecibidos){
  if (Ent.substring(Ref*2,Ref*2+2)==("5A")){////////////Aceleracion
    if (NroBytesEnviados<5){
      Serial.print("5A"+spc);
      NroBytesEnviados++;
    }else {
      if (Bandera==0){
        Bandera=1;
        Serial.print(salto+header+spc+"21"+spc);
      }
      Serial.print("5A"+spc);
```

```
    NroBytesEnviados++;
  }
  if (NroBytesEnviados<5){
    if (AccSalida<16){
      Serial.print("0");
      Serial.print(AccSalida,HEX);
      Serial.print(spc);
    }else{
      Serial.print(AccSalida,HEX);
      Serial.print(spc);
    }
    NroBytesEnviados++;
  }else{
    if (Bandera==0){
      Bandera=1;
      Serial.print(salto+header+spc+"21"+spc);
    }
      if (AccSalida<16){
        Serial.print("0");
        Serial.print(AccSalida,HEX);
        Serial.print(spc);
      }else{
        Serial.print(AccSalida,HEX);
        Serial.print(spc);
      }
      NroBytesEnviados++;
    }
  }
  if (Ent.substring(Ref*2,Ref*2+2)==("2F")){////////////Gasolina
    if (NroBytesEnviados<5){
      Serial.print("2F"+spc);
      NroBytesEnviados++;
```

```
}else {
  if (Bandera==0){
    Bandera=1;
    Serial.print(salto+header+spc+"21"+spc);
  }
  Serial.print("2F"+spc);
  NroBytesEnviados++;
}
if (NroBytesEnviados<5){
  if (GasSalida<16){
    Serial.print("0");
    Serial.print(GasSalida,HEX);
    Serial.print(spc);
  }else{
    Serial.print(GasSalida,HEX);
    Serial.print(spc);
  }
  NroBytesEnviados++;
}else{
  if (Bandera==0){
    Bandera=1;
    Serial.print(salto+header+spc+"21"+spc);
  }
  if (GasSalida<16){
    Serial.print("0");
    Serial.print(GasSalida,HEX);
    Serial.print(spc);
  }else{
    Serial.print(GasSalida,HEX);
    Serial.print(spc);
  }
  NroBytesEnviados++;
```

```
    }
  }
  if (Ent.substring(Ref*2,Ref*2+2)==("0C")){////////////RPM
    if (NroBytesEnviados<5){
      Serial.print("0C"+spc);
      NroBytesEnviados++;
    }else {
      if (Bandera==0){
        Bandera=1;
        Serial.print(salto+header+spc+"21"+spc);
      }
      Serial.print("0C"+spc);
      NroBytesEnviados++;
    }
    if (NroBytesEnviados<5){
      if (A<16){
        Serial.print("0");
        Serial.print(A,HEX);
        Serial.print(spc);
      }else{
        Serial.print(A,HEX);
        Serial.print(spc);
      }
      NroBytesEnviados++;
    }else{
      if (Bandera==0){
        Bandera=1;
        Serial.print(salto+header+spc+"21"+spc);
      }
      if (A<16){
        Serial.print("0");
        Serial.print(A,HEX);
```

```
      Serial.print(spc);
    }else{
      Serial.print(A,HEX);
      Serial.print(spc);
    }
    NroBytesEnviados++;
  }
  if (NroBytesEnviados<5){
    if (B<16){
      Serial.print("0");
      Serial.print(B,HEX);
      Serial.print(spc);
    }else{
      Serial.print(B,HEX);
      Serial.print(spc);
    }
    NroBytesEnviados++;
  }else{
    if (Bandera==0){
      Bandera=1;
      Serial.print(salto+header+spc+"21"+spc);
    }
    if (B<16){
      Serial.print("0");
      Serial.print(B,HEX);
      Serial.print(spc);
    }else{
      Serial.print(B,HEX);
      Serial.print(spc);
    }
    NroBytesEnviados++;
  }
```

```
}
if (Ent.substring(Ref*2,Ref*2+2)==("0D")){//////////////Velocidad
  if (NroBytesEnviados<5){
    Serial.print("0D"+spc);
    NroBytesEnviados++;
  }else {
    if (Bandera==0){
      Bandera=1;
      Serial.print(salto+header+spc+"21"+spc);
    }
    Serial.print("0D"+spc);
    NroBytesEnviados++;
  }
  if (NroBytesEnviados<5){
    if (VelV<16){
      Serial.print("0");
      Serial.print(VelV,HEX);
      Serial.print(spc);
    }else{
      Serial.print(VelV,HEX);
      Serial.print(spc);
    }
    NroBytesEnviados++;
  }else{
    if (Bandera==0){
      Bandera=1;
      Serial.print(salto+header+spc+"21"+spc);
    }
    if (VelV<16){
      Serial.print("0");
      Serial.print(VelV,HEX);
      Serial.print(spc);
```

```
            }else{
              Serial.print(VelV,HEX);
              Serial.print(spc);
            }
            NroBytesEnviados++;
          }
        }
        Ref++;
    }
    Ref=NroBytesEnviados;
    Serial.print(salto+" > ");
    if (Ref<=6){
      while(Ref<5){
        Serial.print("00"+spc);
        Ref++;
      }
      Serial.print(salto+" > ");
    }else{
      while(Ref<12){
        Serial.print("00"+spc);
        Ref++;
      }
      Serial.print(salto+" > ");
    }
  }
```

Segundo código: Comunición remota de 1 solo parámetro y comunicación Bluetooth.

```
#include <SoftwareSerial.h>
#define RxD 8
#define TxD 9
SoftwareSerial Bluetooth(RxD, TxD);
////////////////////////////////////////////////////
////////////// Funcion de las RPM ////////////////////
////////////////////////////////////////////////////
int FuncRPM(int RPMLect){
/////////////// Constantes y variables de RPM/////////////////////
unsigned long RPMT1 = 0; //Tiempos para el calculo de las RPM
unsigned long RPMT2 = 0;
int RPMBand1;
int RPMBand2;
unsigned long RPMAux;
unsigned long RPMK;
int RPM;
int RPMmin = 1200;
unsigned long RPMTv;
RPMBand1=RPMBand2=RPM=0;
RPMTv=60000/RPMmin;
RPMK = RPMTv*1000;
  if (digitalRead(RPMLect)==HIGH){
    delay(12);
  }
  RPMAux=micros();
  while ((micros()-RPMAux)<RPMK){
    switch (digitalRead(RPMLect)){
      case HIGH:{
        delaymicroseconds(100);
```

```
        if (digitalRead(RPMLect)==HIGH){
          if (RPMBand1==0){ //Cuando es un primer pulso se toma el tiempo
            RPMT1=micros(); //de dicho pulso
            RPMAux=micros();
            RPMBand1=1;
            delay(12);
            break;
          }
          if (RPMBand2==1){//Cuando es un segundo pulso, se toma dicho
            RPMT2=micros(); //tiempo y ya se puede obtener el periodo de
            RPMK=0; //la señal
            break;
          }
        }
        break;
      }
      case LOW:{
        if (RPMBand1==1){
          RPMBand2=1;
        }
        break;
      }
    }
  }
  if (RPMT2!=0){
    RPM = int(60000000/(RPMT2-RPMT1));
  }
  return RPM;
}
//////////////////////////////////////////////////////
////////// Funcion de la velocidad ////////////////
//////////////////////////////////////////////////////
```

```
int FuncVel(int VelLect){
unsigned long VelTv = 100;
int VelLimInf = 507;
int VelLimSup = 508; /*Valores de referencia para comparar con la entrada
        corresponde al voltaje con resolucion de 1024=5V*/
int VelBand = 0;
unsigned long VelT1;
unsigned long VelT2;
unsigned long VelAux;
int VelV;
const float VelTheta = 0.6981317; // Periodo angular en radianes de de pulso a pulso
float VelR = 29.718; // Radio de las ruedas del prototipo (cm)
int VelValLect; //Valor que se lee de la entrada analógica
int K, cont;
  VelTv = VelTv*1000;
  VelR = VelR/100.0;
  VelBand=VelV=VelT1=VelT2=K=0;
  VelAux=micros();
  K = analogRead(VelLect);
  while ((micros()-VelAux)<VelTv){
    VelValLect = cont = 0;
    VelValLect = analogRead(VelLect);
    if ((VelValLect>=K-1)&&(VelValLect<=K+1)){
      if (VelValLect<VelLimInf){
        VelBand = 1;
        if ((VelBand == 1)&&(VelT1 != 0)){
          VelBand = 2;
        }
      } else if (VelValLect>VelLimSup){
        if ((VelBand == 1)&&(VelT1 == 0)){
          VelT1 = micros();
```

```
            VelAux = micros();
            delaymicroseconds(8300);
          }
          if (VelBand == 2){
            VelT2 = micros();
            VelAux=0;
          }
        }
      }
      K = analogRead(VelLect);
    }
    if (VelT2 != 0){
      VelV = int(VelR*VelTheta*3600000.0/(VelT2-VelT1));
    }
    return VelV;
}
////////////////////////////////////////////////////
//////// Funcion de la Aceleracion ////////////////
////////////////////////////////////////////////////
int FuncAcc(int AccLect){
float Acc; //Variable de la entrada analógica obtenida y que
int AccLeida = 0; //Aceleracion leida del ADC
float AccSens = 0.3465;//(1.704-1.353)=0.351... (2.046-1.704)=0.342
float AccV0 = 1.699;
int AccSalida; //Valor final de salida de la aceleracion
    AccLeida = analogRead(AccLect);
    Acc = (((5.0*AccLeida-1024.0*AccV0)*9.80665)/(1024.0*AccSens));
    if (Acc<=0){
      Acc=Acc*-1;
    }
    AccSalida = int(Acc*255/100);
    return AccSalida;
```

```
}
//////////////////////////////////////////////////////
////////// Funcion de la Gasolina ////////////////
//////////////////////////////////////////////////////
int FuncGas(int GasLect){
int GasNivel = 0;
int GasSalida = 0;
  delay(5);
  GasNivel = analogRead(GasLect);
  if (GasNivel <= 510){
    GasSalida = 92;
  } else if ((GasNivel >510)&&(GasNivel <= 511)){
    GasSalida = 84;
  } else if ((GasNivel >511)&&(GasNivel <= 512)){
    GasSalida = 76;
  } else if ((GasNivel >512)&&(GasNivel <= 513)){
    GasSalida = 69;
  } else if ((GasNivel >513)&&(GasNivel <= 515)){
    GasSalida = 61;
  } else if ((GasNivel >515)&&(GasNivel <= 518)){
    GasSalida = 53;
  } else if ((GasNivel >= 518)&&(GasNivel <= 528)){
    GasSalida = 46;
  } else if ((GasNivel >= 528)&&(GasNivel <= 553)){
    GasSalida = 38;
  } else if ((GasNivel >= 553)&&(GasNivel <= 742)){
    GasSalida = 30;
  } else if (GasNivel >742){
    GasSalida = 0;
  }
  return GasSalida;
}
```

```
//////////////////////////////////////////////////
////////////// Comienzo del programa ///////////////
//////////////////////////////////////////////////
String Ent,salto,saltoBT,header,headerBT,spc,spcBT,PCI,PCIBT;
int Band,Band2,echo,echoBT,TOut,TOutBT,HEAD,HEADBT;
unsigned long contadorBT = 0;
//////// Constantes y variables de RPM//////////////
const int RPMLect = A4; //Pin lector del voltaje analógico
int RPM=0; //Variable de salida de RPM
int A,B;
//////// Constantes y variables de velocidad ///////
const int VelLect = A1;
int VelV = 0;
unsigned long contador;
//////// Constantes y variables de la gasolina /////
const int GasLect = A0; // Pin lector del voltaje analógico
int GasValor;
int GasCont = 0;
int GasSalida=0;
//Variable de la salida final correspondiente a la altura
//del tanque en %
//////// Constantes y variables de la aceleracion //
const int AccLect = A3; // Pin lector del voltaje analógico
int AccSalida = 0; //Valor final de salida de la aceleracion
void setup(){
  contador=contadorBT= millis();
  TOut=TOutBT=5;
  echo=echoBT=1;
  HEAD=HEADBT=0;
  salto=saltoBT="\r\n";
  header=headerBT="";
  spc=spcBT="";
```

```
  PCI=PCIBT="";
  pinMode(RPMLect, INPUT);
  pinMode(VelLect, INPUT);
  pinMode(GasLect, INPUT);
  pinMode(AccLect, INPUT);
  analogReference(DEFAULT);
  Bluetooth.flush();
  delay(500);
  Bluetooth.begin(9600);
  Serial.begin(9600);
  Serial.setTimeout(TOut);
  Bluetooth.setTimeout(TOutBT);
  delay(2000);
  Serial.print(salto);
  delay(1000);
  Serial.print("B"+salto);
  delay(1000);
  Serial.print("ELM327v2,0"+salto+" > ");
}
void loop(){
  Band=Band2=0;
  Ent="";
  if ((Serial.available() >0) | | (Bluetooth.available() >0)){
    if ((Bluetooth.available() >0)&&(( millis()-contadorBT)>50)){
      Band2=2;
      Ent = Bluetooth. readString();
      Bluetooth.println();
      if (echoBT==1){
        Bluetooth.println(Ent);
      }
      contadorBT= millis();
    } else if (Serial.available() >0){
```

```
      Band2=1;
      Ent = Serial. readString();
      Serial.println();
      if (echo==1){
        Serial.println(Ent);
      }
    }
    Ent.trim();
    Ent.replace(""{,"");
    if (Ent.substring(0,4)=="ATAT"){ //Control adaptativo de tiempo
      Band=1;
      imprimir(saltoBT,salto,Band2);
    }
     if (Ent.substring(0,5)=="ATCAF"){ //Elimina o activa el autoformateo CAN,
es
      Band=1; //decir los mensajes se imprimen tal cual
           //como los recibe el ELM. Ningun efecto
      imprimir(saltoBT,salto,Band2);
    }
    if (Ent.substring(0,5)=="ATCFC"){ //Elimina los mensajes de flow control del
      Band=1; //protolo CAN. Ningun efecto
      imprimir(saltoBT,salto,Band2);
    }
    if (Ent.substring(0,5)=="ATCRA"){ //Ajusta la mascara CAN
      Band=1;
      imprimir(saltoBT,salto,Band2);
    }
    if (Ent.equalsIgnorecase("ATD")){ //Restaura todos los valores de fábrica
      Band=1;
      if (Band2==2){
        echoBT=1;
        HEADBT=0;
```

```
        saltoBT="\r\n";
        headerBT="";
        spcBT="";
        PCIBT="";
      }else{
        echo=1;
        HEAD=0;
        salto="\r\n";
        header="";
        spc="";
        PCI="";
      }
      imprimir(saltoBT,salto,Band2);
    }
    if (Ent.equalsIgnorecase("ATDP")){ //Muestra la descripcion el protocolo pre-
sente
      Band=1;
      imprimir2("AUTO, ISO15765 − 4(CAN11/250)",saltoBT,salto,Band2);
    }
    if (Ent.equalsIgnorecase("ATDPN")){ //Muestra el numero del protocolo pre-
sente
      Band=1;
      imprimir2("A8",saltoBT,salto,Band2);
    }
    if (Ent.equalsIgnorecase("ATE0")){ //Desactiva el eco de los comandos
      Band=1; //(retransmitir cada comando recibido)
      imprimir(saltoBT,salto,Band2);
      if (Band2==2){
        echoBT=0;
      }else{
        echo=0;
      }
```

```
}
if (Ent.equalsIgnorecase("ATE1")){ //Activa el eco de los comandos
  Band=1;
  imprimir(saltoBT,salto,Band2);
  if (Band2==2){
    echoBT=1;
  }else{
    echo=1;
  }
}
if (Ent.equalsIgnorecase("ATH0")){ //Desactiva los headers de la respuesta del
        //vehiculo
  Band=1;
  if (Band2==2){
    headerBT="";
    HEADBT=0;
  }else{
    header="";
    HEAD=0;
  }
  imprimir(saltoBT,salto,Band2);
}
if (Ent.equalsIgnorecase("ATH1")){ //Activa los headers de la respuesta del
        //vehiculo
  Band=1;
  if (Band2==2){
    headerBT="7E8"+spc;
    HEADBT=1;
  }else{
    header="7E8"+spc;
    HEAD=1;
  }
```

```
      imprimir(saltoBT,salto,Band2);
    }
    if (Ent.equalsIgnorecase("ATI")){ //Hace que el chip se identifique a si mismo
      Band=1;
      imprimir2("ELM327v2,0",saltoBT,salto,Band2);
    }
    if (Ent.equalsIgnorecase("ATL0")){ //Desactiva el salto de línea luego de en-
viar
            //una respuesta
      Band=1;
      if (Band2==2){
        saltoBT="\r";
      }else{
        salto="\r";
      }
      imprimir(saltoBT,salto,Band2);
    }
    if (Ent.equalsIgnorecase("ATL1")){ //Activa el salto de línea luego de enviar
            //una respuesta
      Band=1;
      if (Band2==2){
        saltoBT="\r\n";
      }else{
        salto="\r\n";
      }
      imprimir(saltoBT,salto,Band2);
    }
    if (Ent.substring(0,3)=="ATM"){ //Desactiva o activa la memoria no volatil
      Band=1;
//de los protocolos
      imprimir(saltoBT,salto,Band2);
    }
```

```
    if (Ent.equalsIgnorecase("ATRV")){ //Muestra el voltaje de entrada del ELM
      Band=1;
      imprimir2("5V",saltoBT,salto,Band2);
    }
    if (Ent.equalsIgnorecase("ATS0")){ //Elimina los espacios entre los caracteres
      Band=1;
//de las respuestas
      if (Band2==2){
        spcBT="";
      }else{
        spc="";
      }
      imprimir(saltoBT,salto,Band2);
    }
    if (Ent.equalsIgnorecase("ATS1")){ //Añade los espacios entre los caracteres
      Band=1;
//de las respuestas
      if (Band2==2){
        spcBT="";
      }else{
        spc="";
      }
      imprimir(saltoBT,salto,Band2);
    }
    if (Ent.substring(0,4)=="ATSH"){ //Establece el header a usar en CAN
      Band=1; //Pero para este caso como se simula una
              //sola ECU el header de respuesta
              //permanece igual
      imprimir(saltoBT,salto,Band2);
    }
    if (Ent.substring(0,4)=="ATSP"){ //Selecciona la lectura del protocolo
      Band=1; //introducido, como se simula un solo
```

```
            //protocolo, se hace caso
            //omiso al numero y se mantiene el protocolo
      imprimir2("OK\r\nSEARCHING...",saltoBT,salto,Band2);
    }
    if (Ent.substring(0,4)=="ATST"){ //Selecciona el tiempo de espera del ELM
      Band=1; //antes de determinar que no hay respuesta
            //del vehiculo
      imprimir(saltoBT,salto,Band2);
    }
     if (Ent.equalsIgnorecase("ATWS")){ //Realiza un reseteo mas rapido que el
ATZ
      Band=1;
      if (Band2==2){
        echoBT=1;
        headerBT="";
        HEADBT=0;
        saltoBT="\r\n";
      }else{
        echo=1;
        header="";
        HEAD=0;
        salto="\r\n";
      }
      imprimir2("ELM327v2,0",saltoBT,salto,Band2);
    }
    if (Ent.equalsIgnorecase("ATZ")){ //Realiza un reseteo al igual que ocurriria
      Band=1; //si se apaga y se prende el ELM
      if (Band2==2){
        headerBT="";
        HEADBT=0;
        saltoBT="\r\n";
        echoBT=1;
```

```
        }else{
          header="";
          HEAD=0;
          salto="\r\n";
          echo=1;
        }
        imprimir2("ELM327v2,0",saltoBT,salto,Band2);
    }
    if (Ent.equalsIgnorecase("AT@1")){ //Muestra la descripcion del dispositivo
        Band=1;
        imprimir2("OBDIIEmulatortoRS232interpreter",saltoBT,salto,Band2);
    }
    if (Ent.equalsIgnorecase("AT@2")){//Muestra una descripcion guardada de 12
        Band=1; //caracteres
        imprimir2("FECHA12/2014",saltoBT,salto,Band2);
    }
////////////////////////////////////////////////
////////////// Comandos OBDII //////////////////
////////////////////////////////////////////////
    if (HEAD==1){
        PCI="06"+spc;
    }else{
        PCI="";
    }
    if (HEADBT==1){
        PCIBT="06"+spcBT;
    }else{
        PCIBT="";
    }
    if (Ent.equalsIgnorecase("0100")){ //Solicitud del PID 00 el cual muestra todos
los PIDs
        Band=1; //admitidos hasta el PID 20
```

```
        if (Band2==2){
          Bluetooth.print(headerBT+PCIBT+"41"+spcBT+"00"+spcBT+"00"+spcBT+
"18"+spcBT+"00"+spcBT+"01"+saltoBT+" > ");
        }else{
          Serial.print(header+PCI+"41"+spc+"00"+spc+"00"+spc+
"18"+spc+"00"+spc+"01"+salto+" > ");
        }
      }
      if (Ent.equalsIgnorecase("0120")){ //Solicitud del PID 20 el cual muestra todos
los PIDs
        Band=1; //admitidos hasta el PID 40
        if (Band2==2){
          Bluetooth.print(headerBT+PCIBT+"41"+spcBT+"20"+spcBT+"00"+spcBT+
"02"+spcBT+"00"+spcBT+"01"+saltoBT+" > ");
        }else{
          Serial.print(header+PCI+"41"+spc+"20"+spc+"00"+spc+
"02"+spc+"00"+spc+"01"+salto+" > ");
        }
      }
      if (Ent.equalsIgnorecase("0140")){ //Solicitud del PID 20 el cual muestra todos
los PIDs
        Band=1; //admitidos hasta el PID 60
        if (Band2==2){
          Bluetooth.print(headerBT+PCIBT+"41"+spcBT+"40"+spcBT+"00"+spcBT
+"00"+spcBT+"00"+spcBT+"40"+saltoBT+" > ");
        }else{
          Serial.print(header+PCI+"41"+spc+"40"+spc+"00"+spc+
"00"+spc+"00"+spc+"40"+salto+" > ");
        }
      }
////////////////////////////////////////////////////
////////////////// RPM ////////////////////////////
```

```
////////////////////////////////////////////////////
    if (Ent.equalsIgnorecase("010C")){
      Band=1;
      if (HEAD==1){
        PCI="04"+spc;
      }else{
        PCI="";
      }
      if (HEADBT==1){
        PCIBT="04"+spcBT;
      }else{
        PCIBT="";
      }
      if (Band2==2){
        Bluetooth.print(headerBT+PCIBT+"41"+spcBT+"0C"+spcBT);
      }else{
        Serial.print(header+PCI+"41"+spc+"0C"+spc);
      }
      if (A<16){
        if (Band2==2){
          Bluetooth.print('0');
        }else{
          Serial.print('0');
        }
      }
      if (Band2==2){
        Bluetooth.print(A,HEX);
        Bluetooth.print(spcBT);
      }else{
        Serial.print(A,HEX);
        Serial.print(spc);
      }
```

```
        if (B<16){
          if (Band2==2){
            Bluetooth.print('0');
          }else{
            Serial.print('0');
          }
        }
        if (Band2==2){
          Bluetooth.print(B,HEX);
          Bluetooth.print(salto+" > ");
        }else{
          Serial.print(B,HEX);
          Serial.print(salto+" > ");
        }
      }
////////////////////////////////////////////////////
////////////////// Velocidad //////////////////////
////////////////////////////////////////////////////
      if (HEADBT==1){ //Como todos los parámetros siguientes tienen un
        PCIBT="03"+spcBT; //solo byte de respuesta se define el PCI de una vez
      }else{
        PCIBT="";
      }
      if (HEAD==1){
        PCI="03"+spc;
      {else{
        PCI="";
      }
      if (Ent.equalsIgnorecase("010D")){
        Band=1;
        if (Band2==2){
          Bluetooth.print(headerBT+PCIBT+"41"+spcBT+"0D"+spcBT);
```

```
        }else{
          Serial.print(header+PCI+"41"+spc+"0D"+spc);
        }
        if (VelV<16){
          if (Band2==2){
            Bluetooth.print('0');
          }else{
            Serial.print('0');
          }
        }
        if (Band2==2){
          Bluetooth.print(VelV,HEX);
          Bluetooth.print(salto+" > ");
        }else{
          Serial.print(VelV,HEX);
          Serial.print(salto+" > ");
        }
      }
////////////////////////////////////////////////////
////////////////// Gasolina ////////////////////////
////////////////////////////////////////////////////
      if (Ent.equalsIgnorecase("012F")){
        Band=1;
        if (GasCont = 0){
          GasValor = FuncGas(GasLect);
          GasSalida = GasValor;
          GasCont = 1;
        }
        if (GasCont <15){
          GasCont++;
          GasValor = GasValor + FuncGas(GasLect);
        } else if (GasCont = 15){
```

```
        GasSalida = GasValor/15;
        GasCont=1;
        GasValor = FuncGas(GasLect);
      }
      GasSalida = k;
      GasSalida = GasSalida*255/100;
      if (Band2==2){
        Bluetooth.print(headerBT+PCIBT+"41"+spcBT+"2F"+spcBT);
      }else{
        Serial.print(header+PCI+"41"+spc+"2F"+spc);
      }
      if (GasSalida<16){
        if (Band2==2){
          Bluetooth.print('0');
        }else{
          Serial.print('0');
        }
      }
      if (Band2==2){
        Bluetooth.print(GasSalida,HEX);
        Bluetooth.print(saltoBT+" > ");
      }else{
        Serial.print(GasSalida,HEX);
        Serial.print(salto+" > ");
      }
    }
////////////////////////////////////////////////////
////////////////// Aceleracion ////////////////////
////////////////////////////////////////////////////
    if (Ent.equalsIgnorecase("015A")){ //Solicitud de la Aceleracion
      Band=1;
      AccSalida = FuncAcc(AccLect);
```

```
        AccSalida = AccSalida*255/100;
        if (Band2==2){
          Bluetooth.print(headerBT+PCIBT+"41"+spcBT+"5A"+spcBT);
        }else{
          Serial.print(header+PCI+"41"+spc+"5A"+spc);
        }
        if (AccSalida<16){
          if (Band2==2){
            Bluetooth.print('0');
          }else{
            Serial.print('0');
          }
        }
        if (Band2==2){
          Bluetooth.print(AccSalida,HEX);
          Bluetooth.print(saltoBT+" > ");
        }else{
          Serial.print(AccSalida,HEX);
          Serial.print(salto+" > ");
        }
      }
////////////////////////////////////////////////////
////////// Final y cálculo de parámetros//////////////////
////////////////////////////////////////////////////
//////////////////////////// RPM
      if (Band==0){
        imprimir2("NODATA",saltoBT,salto,Band2);
      }
    }
    if ( millis()-contador>200){
      RPM=FuncRPM(RPMLect);
      if (RPM>=2000){
```

```
        for (int i=0; i <3; i++){
          RPM=RPM+FuncRPM(RPMLect);
        }
      }
      RPM=RPM*4/4;
      A=RPM/256;
      B=RPM %256;
//////////////////////////// Velocidad
      VelV=FuncVel(VelLect);
      if ((VelV>=20)&&(VelV<40)){
        for (int i=0; i <2; i++){
          VelV=VelV+FuncVel(VelLect);
        }
        VelV=VelV/3;
      } else if (VelV>=40){
        for (int i=0; i <5; i++){
          VelV=VelV+FuncVel(VelLect);
        }
        VelV=VelV/6;
      }
      contador= millis();
    }
}
////////////////////////////////////////////////////
////////// Funciones de impresion /////////////
////////////////////////////////////////////////////
////////// Función de impresión 1/////////////
void imprimir( String saltoBT, String salto, int Band2){
  if (Band2==2){
    Bluetooth.print("OK"+saltoBT+" > ");
  }else{
    Serial.print("OK"+salto+" > ");
```

```
  }
}

////////// Función de impresión 2//////////////
void imprimir2( String cadena, String saltoBT, String salto, int Band2){
  if (Band2==2){
    Bluetooth.print(cadena+saltoBT+" > ");
  }else{
    Serial.print(cadena+salto+" > ");
  }
}
```

Apéndice E

Descripción de los pines usados con el UTP CAT 6 para la interconexión de los Sensores

El configuración usada para el cableado UTP que hace posible la interconexión entre los sensores y el microcontrolador fue el B, y se puede apreciar en la figura E.1.

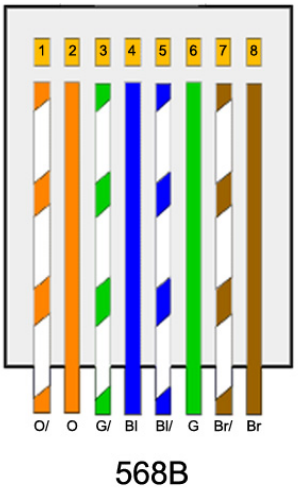


Figura E.1: Estándar EIA/TIA-568B

La siguiente tabla muestra la correspondencia de cada pin del estándar con la conexión de los sensores.

Tabla E.1: Asignación de los pines del estandar UTP-B a los sensores.

Pines	Sensor	Conexión
1	Aceleración	GND
2		Vcc
3		Salida
4	Gasolina	Vcc
5		GND
6		Salida
7	RPM del Motor	GND
8		Salida

Para cada señal se requiere de una entrada analógica en el Arduino, las entradas usadas en la placa del microcontrolador fueron las que se muestran en la siguiente tabla.

Tabla E.2: Pines de entrada al microcontrolador

Conexión al Arduino	
A0	Gasolina
A1	Velocidad
A2	RPM
A3	Aceleración

Referencias Bibliográficas

[1] DIGI. *XBEE-PRO 900HP and XSC RF Modules S3 and S3B*, .

[2] DIGI. *Wireless Mesh Networking ZigBee vs DigiMesh*, .

[3]

[4] UIT. *Recomendación UIT-R P.526-13: Propagación por difracción*. 2013.

[5] URL http://arduino.cc/en/uploads/Main/Arduino_WirelessProtoShield_Front.jpg.

[6] . URL http://www.amazon.com/SainSmart-Adapter-Arduino-Mega2560-Duemilanove/dp/B0085J99CI.

[7] Society of Automotive Engineers, editor. *Vehicle Dynamics Terminology SAE J670e*. 1976.

[8] Elm Electronics. *ELM327 datasheet*.

[9] URL http://www.users.miamioh.edu/jamiespa/teaching/ECE_387/Final_Projects-May_25_2011/99ca1491821df6a57efa22e6d77ba324.jpg.

[10] URL http://www.tinyosshop.com/image/cache/data/XBEE/xbee%20pro%20900%20HP-1-600x600.jpg.

[11] URL http://botscience.net/store/image/cache/data/products/MODCOM/TEL0002/bluetooth-modulo-serial-microcontrolador-pic-arduino-atmel_mlm-o-2636188463_042012-500x500.jpg.

[12] . URL http://arduino.cc/en/uploads/Main/ArduinoUno_R3_Front.jpgv.

[13] N. Hernández, J. Quero, N. Lartiguez, J. Ramírez, and E. Monagas. *Acta constitutiva de la Organización SAE UC Venezuela*. 2003.

[14] J. Cárdenas and N. Márquez. *Desarrollo de un sistema de instrumentación para el proyecto BAJA de la Organización SAE UC Venezuela*. Universidad de Carabobo, 2009.

[15] J. Marín. *Diseño e implementación de un sistema de embebido de telemetría para vehículos BAJA SAE*. Universidad Simón Bolívar, 2012.

[16] D. Ramos. *Diseño de un sistema de monitoreo OBDII con comunicación GSM*. 2014.

[17] SAE. *SAE STANDARD J1850, Class B Data Communication Network Interface*. SAE.

[18] ISO. *Road vehicles - Diagnostic communication over Controller Area Network*. International Standard.

[19] DIGI. *Guía de Usuario Xbee Series 1*, .

[20] DIGI. *XBee-PRO 900HP*, . URL http://www.digi.com/pdf/wp_zigbeevsdigimesh.pdf.

[21] DIGI. *The digimesh networking protocol. Technical report*, . URL http://www.digi.com/technology/digimesh/.

[22] S. Tumanski. *Principles of Electrical Measurement*. CRC Press, 2006.

[23] R. Pallas. *Sensores y acondicionadores de señal*. Marcombo S.A. 4ta edicion, 2003.

[24] SAE International. *2015 Collegiate Design Series Baja SAE Rules*, 2014.

[25] Measurement Computing. *Data Acquisition Handbook*. USA, 2004-2012.

[26] UIT. *Recomendación UIT-R P.529-3: MÉTODOS DE PREDICCIÓN REQUERIDOS PARA EL SERVICIO MÓVIL TERRESTRE TERRENAL EN LAS BANDAS DE ONDAS MÉTRICAS Y DECIMÉTRICAS*. 1999.

[27] UIT. *Recomendación UIT-R P.525-2: CÁLCULO DE LA ATENUACIÓN EN EL ESPACIO LIBRE*. 1994.

[28] UIT. *Recomendación UIT-R P.341-5: NOCIÓN DE PÉRDIDAS DE TRANSMISIÓN EN LOS ENLACES RADIOELÉCTRICOS*. 1999.

[29] P. Rice A. Longley. *Prediction of Tropospheric Radio Tansmission Loss Over Irregular Terrain, A Computer Method*. U. S. DEPARTMENT OF COMMERCE, 1968.

[30] Carlos Reyes. *Microcontroladores PIC Programación en Basic*. RISPERGRAF, tercera edición edition, 2008.

[31] RAE. *Diccionario de la Real Academia Española*. 2001.

[32] Alejandro Aragón-Zavala Simon Saunders. *Antennas and propagation for wireless comunication systems*. 2007.

Anexo A

Gráficas de cobertura obtenidas de las simulaciones.

Figura a.1: Mapa de cobertura del receptor (PC) de la primera simulación para la pista 1.

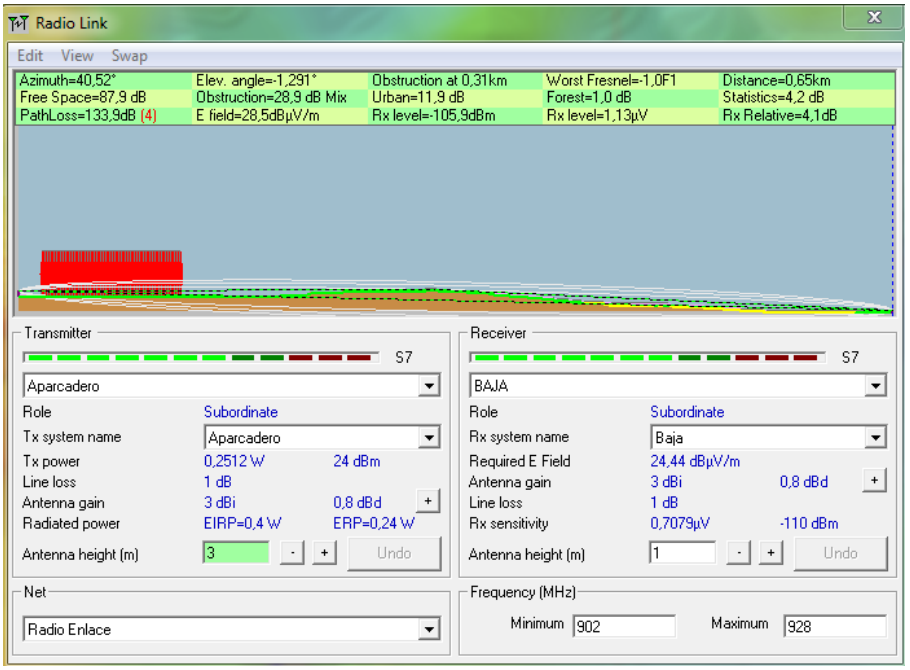


Figura a.2: Resultado del radioenlace de la primera simulación para la pista 1.

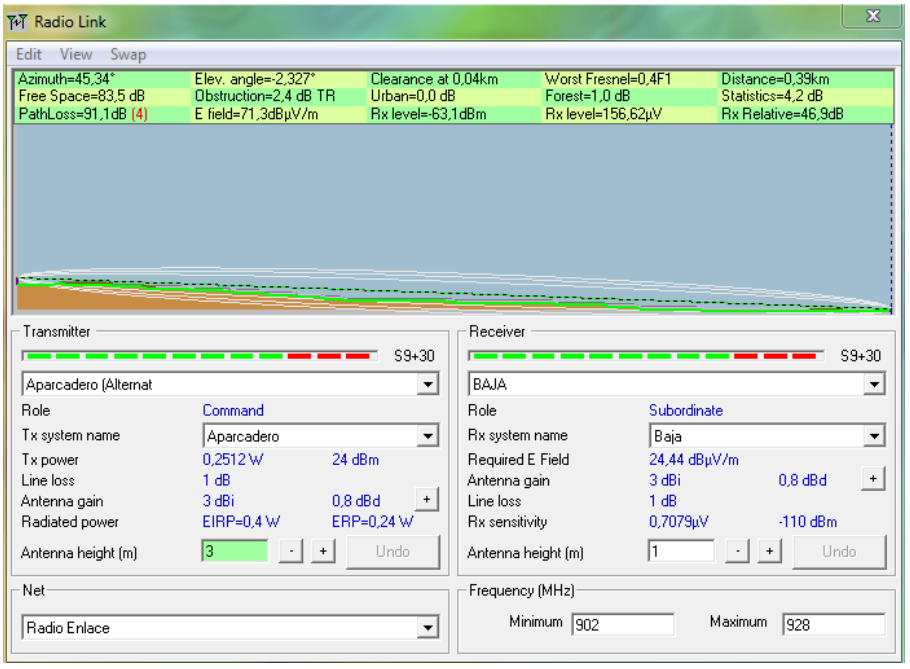


Figura a.3: Resultado del radioenlace de la segunda simulación para la pista 1.

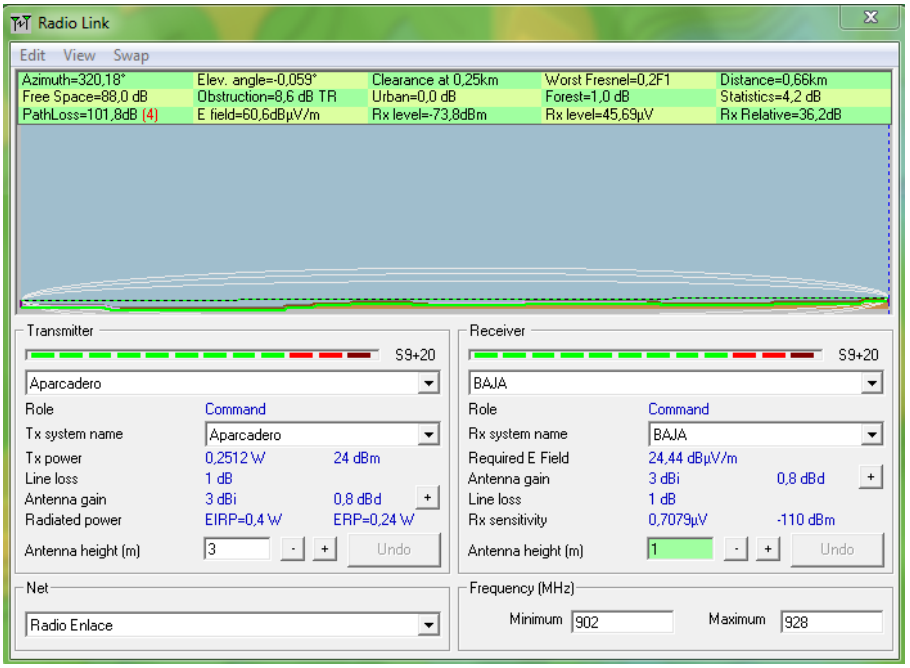


Figura a.4: Resultado del radioenlace de la primera simulación para la pista 2.

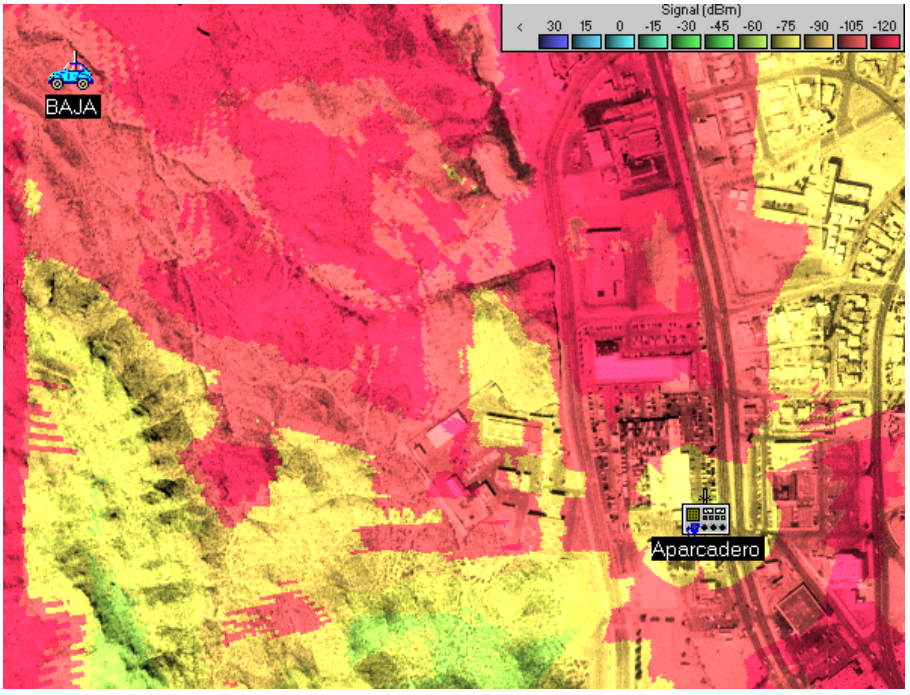


Figura a.5: Mapa de cobertura del receptor (PC) de la primera simulación para la pista 3.

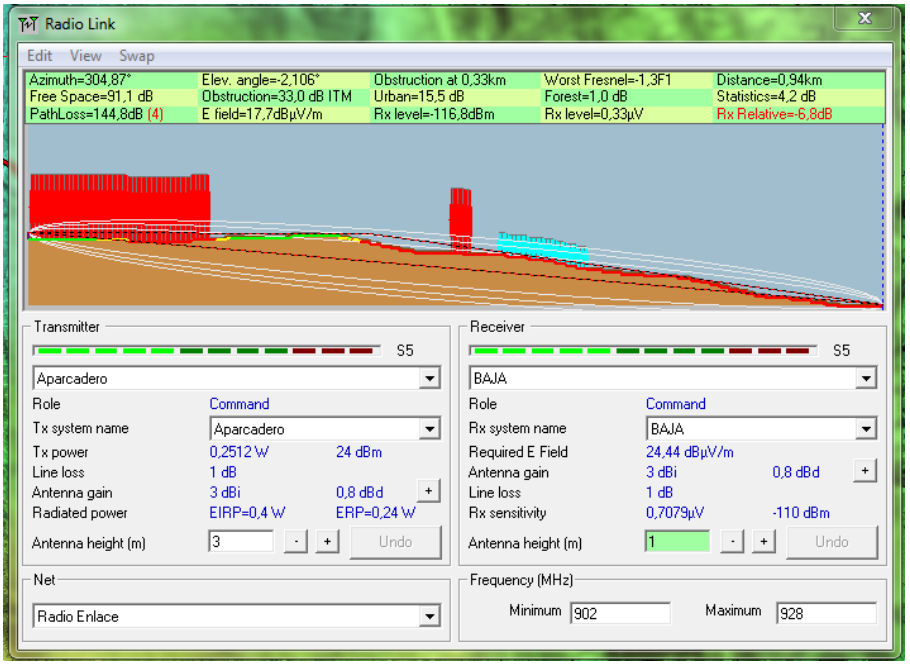


Figura a.6: Resultado del radioenlace de la primera simulación para la pista 3.

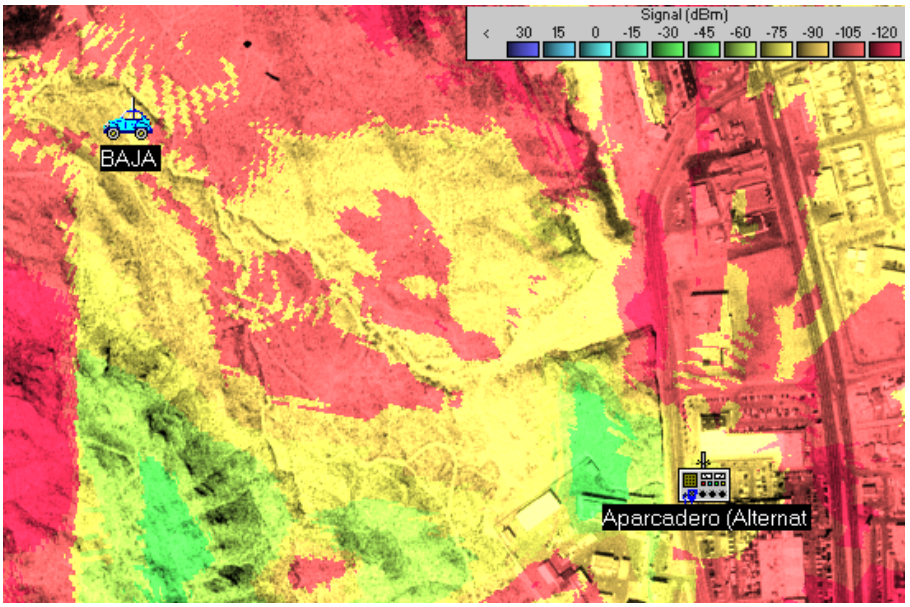


Figura a.7: Mapa de cobertura del receptor (PC) de la segunda simulación para la pista 3.

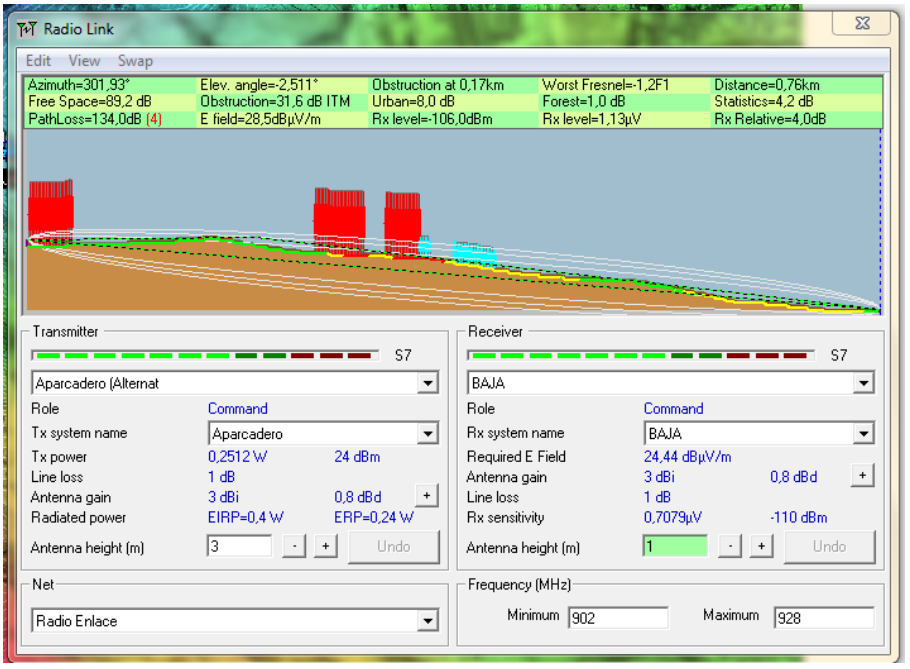


Figura a.8: Resultado del radioenlace de la segunda simulación para la pista 3.

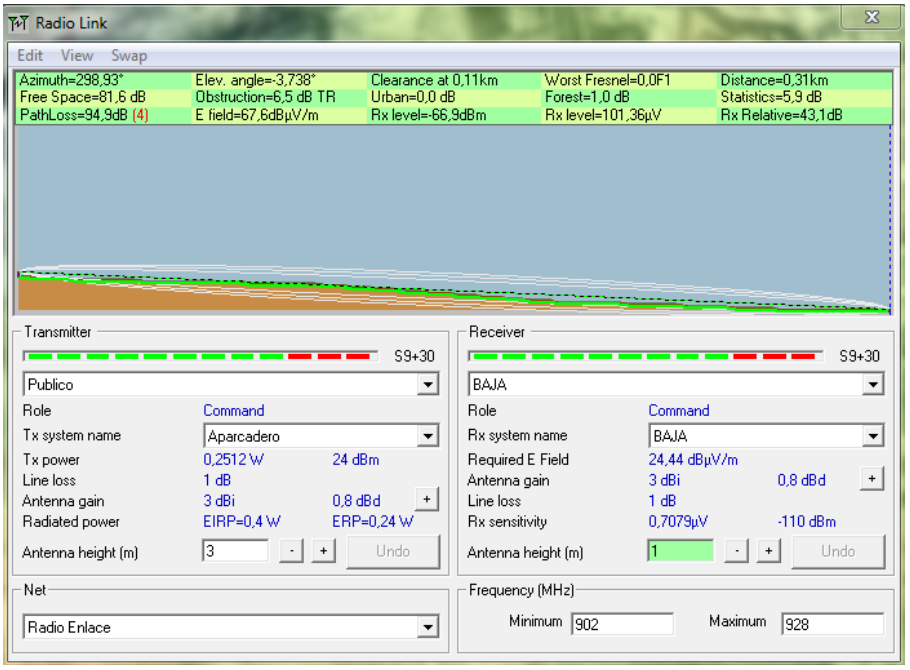
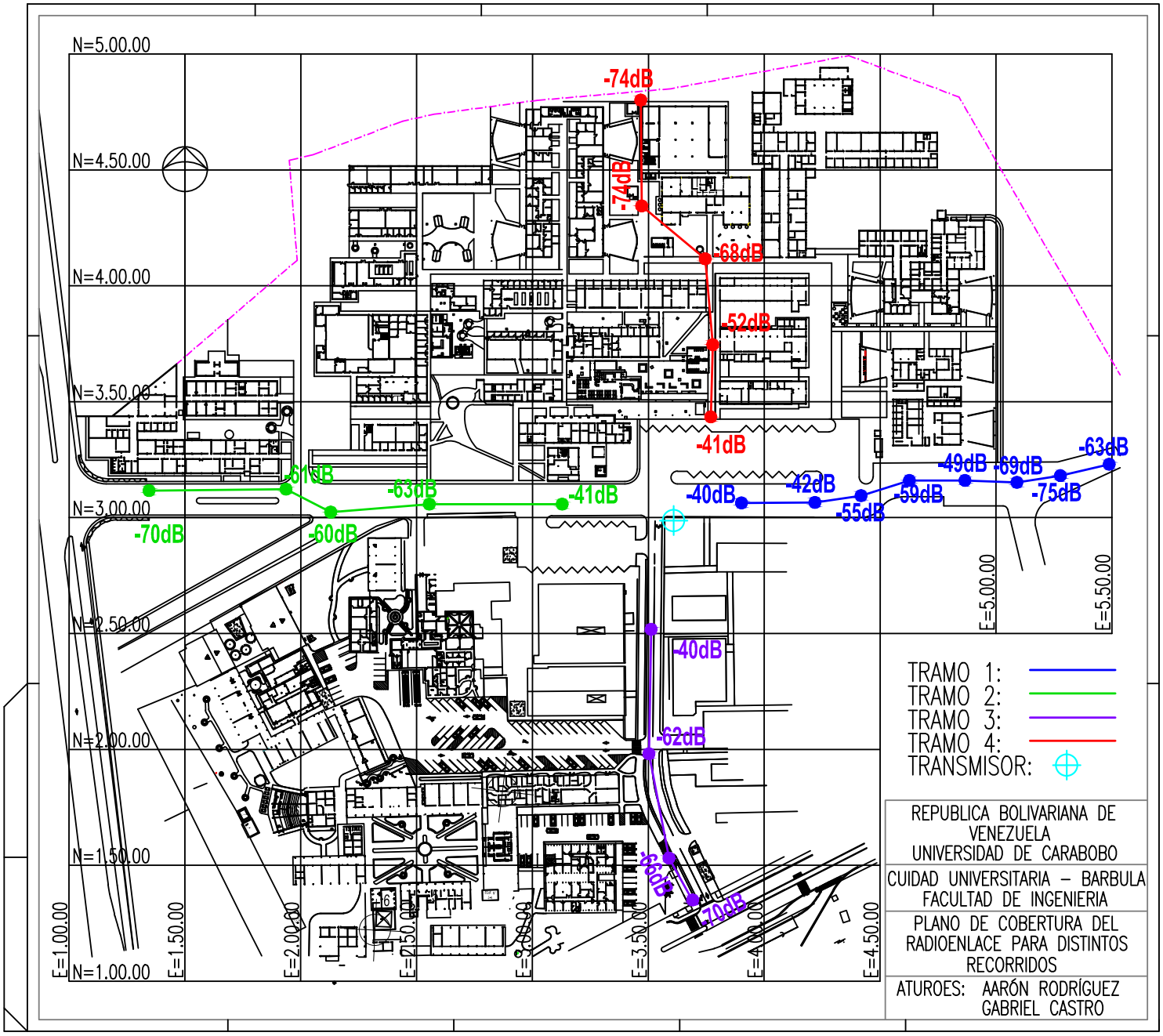


Figura a.9: Resultado del radioenlace de la tercera simulación para la pista 3.

Anexo B

Plano del recorrido realizado para el rango de cobertura del sistema.


N=5.00.00
N=4.50.00
N=4.00.00
N=3.50.00
N=3.00.00
N=2.50.00
N=2.00.00
N=1.50.00
N=1.00.00
E=1.00.00
E=1.50.00
E=2.00.00
E=2.50.00
E=3.00.00
E=3.50.00
E=4.00.00
E=4.50.00
E=5.00.00
E=5.50.00
-74dB
-74dB
-68dB
-52dB
-41dB
-61dB
-63dB
-41dB
-70dB
-60dB
-40dB
-42dB
-55dB
-59dB
-49dB
-69dB
-75dB
-63dB
-40dB
-62dB
-66dB
-70dB
TRAMO 1:
TRAMO 2:
TRAMO 3:
TRAMO 4:
TRANSMISOR:
REPUBLICA BOLIVARIANA DE VENEZUELA
UNIVERSIDAD DE CARABOBO
CUIDAD UNIVERSITARIA – BARBULA
FACULTAD DE INGENIERIA
PLANO DE COBERTURA DEL RADIOENLACE PARA DISTINTOS RECORRIDOS
ATUROES: AARÓN RODRÍGUEZ
GABRIEL CASTRO