UNIVERSIDAD POLITÉCNICA SALESIANA

SEDE CUENCA

CARRERA INGENIERÍA MECÁNICA AUTOMOTRIZ

"DISEÑO Y CONSTRUCCIÓN DE UN EQUIPO PROTOTIPO DE DIAGNÓSTICO DE LA COMPRESIÓN DE MOTORES DE COMBUSTIÓN INTERNA CICLO OTTO CON SISTEMA MPFI MULTEC DELPHI MEDIANTE EL ANÁLISIS DE LOS SENSORES CKP, MAP Y AUXILIARES"

TESIS DE GRADO PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO MECÁNICO AUTOMOTRIZ

AUTORES:

EDISSON PAÚL PARAPI PATIÑO

FREDDY MAURICIO ZUÑA RODRÍGUEZ

DIRECTOR:

INGENIERO NÉSTOR RIVERA

CUENCA, 8 DE JULIO DEL 2013

DECLARACIÓN

Nosotros, Édison Paúl Parapi Patino, Freddy Mauricio Zuña Rodríguez, declaramos bajo juramento que el trabajo aquí descrito es de nuestra autoría; que no ha sido previamente presentado para ningún grado o calificación profesional, también cedemos todos los derechos a la Universidad Politécnica Salesiana para fines académicos.

Cuenca, 8 de Julio del 2013

Edisson Paúl Parapi Patiño

Freddy Mauricio Zuña Rodríguez

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por los señores Edisson Paul Parapi Patiño y Freddy Mauricio Zuña Rodríguez, bajo mi supervisión.

.

Ing. Néstor Diego Rivera Campoverde

DIRECTOR DE PROYECTO

DEDICATORIA

Este proyecto de investigación dedico a mi hermano RICARDO JAVIER PARAPI PATIÑO, un amigo digno de profunda admiración y respeto que me acompaña en todo momento brindándome fuerza para continuar siempre hacia adelante, su recuerdo nunca se desvanecerá de mi mente, gracias hermano y amigo por tu ejemplo de lucha.

De igual forma, dedico este proyecto a mi padre por el apoyo incondicional a pesar de cualquier adversidad ha sabido mantener su confianza en mí y mantener una familia unida. A mi madre por saber guiarme en aquellos momentos difíciles de la vida, pilar fundamental en toda mi carrera de estudio. A mis hermanos por soportar mi actitud de indiferencia, y por brindar mucha alegría en el hogar.

Dedico también, a mis abuelos, tíos, primos, cuñados, sobrinos y demás familiares por estar junto a mí en los momentos buenos y malos de la vida. Al grupo de amigos de la esquina que siempre compartimos ideas y momentos más gratos de la universidad. A mis profesores por compartir sus conocimientos y sabiduría en el desarrollo de mi vida profesional.

EDISSON PAÚL PARAPI PATIÑO

AGRADECIMIENTO.

Primeramente agradezco a mis padres por haberme dado la vida y estar a mi lado para guiarme. Ellos son ejemplo de lucha y perseverancia al no declinar ante los problemas que vivimos, por todo el cariño y respeto hacia todos sus hijos y familiares, mi eterna gratitud. A mi hermano por siempre ser mi ejemplo y por toda la protección que me has brindado.

Agradezco a mi hermana Mónica por su paciencia y especialmente a su esposo por su humildad al haberme permitido usar su computador todo este tiempo. A mi hermana Elizabeth por siempre alegrarme la vida y a todos mis familiares.

Agradezco a mi cuñada Miriam Auquilla por todo su cariño y amor que le tuvo a mi hermano al nunca dejarlo solo, siempre estaré agradecido.

A la familia de mi compañero Freddy Zuña por la paciencia que han tenido en el transcurso de nuestro proyecto.

A nuestro director de tesis Ing. Néstor Rivera por ayudarnos a resolver los problemas que se presentó en la realización de nuestro proyecto.

Al grupo de amigos de la esquina, sepan ellos que siempre contaran con mi apoyo. Un especial agradecimiento a nuestro paciente bodeguero al señor Hernán, por la alegría y honestidad que tiene para todos los alumnos que han pasado por el laboratorio de Ingeniería Mecánica Automotriz.

EDISSON PAUL PARAPI PATIÑO

AGRADECIMIENTO.

Primeramente me gustaría agradecer a Dios por cada escalón y tropiezo que ha puesto en mi vida, pues gracias a ellos he podido superar muchos obstáculos, agradezco también el haber puesto en mi camino a muchas personas que me han brindado su ayuda incondicional.

A mis Padres por haber financiado esta carrera que hoy se convierte en una profesión, a mis hermanas por su apoyo moral, sentimental y económico. A mi primo Rolando Suna, quién ha colaborado económicamente y desinteresadamente en el desarrollo de este proyecto. Así como a mi tío Luis Zuña por regalarme la herramienta que facilitó mucho mis estudios.

A mi director de tesis, Ing. Néstor Rivera, por compartir ideas que hoy se están haciendo realidad al culminar este proyecto de tesis. A todos los profesores y laboratoristas que han impartido sus conocimientos durante mi carrera profesional. A mi amigo Yulian Zapata Marín, quién me facilitó mucha información en el transcurso de realización del proyecto de tesis. A mi amigo Diego Vintimilla Espinoza por su apoyo en aspectos de programación.

Y muchas personas más que han formado parte en esta etapa final de mi carrera, a quienes no necesito nombrar pues me faltaría espacio el mismo que es limitado, pero tanto ellas como yo sabemos que desde lo más profundo de mi corazón les agradezco por todo el apoyo, colaboración, ánimo pero sobre todo su cariño y su valiosa amistad.

PARA ELLOS: MUCHAS GRACIAS Y QUE DIOS LOS BENDIGA.

FREDDY MAURICIO ZUÑA RODRÍGUEZ

INDICE GENERAL

CONTENIDOS **PAG**

Declaración.....I

Certificación.....II

Dedicatoria.....III

Agradecimiento.....IV

Agradecimiento.....V

Índice general.....VI

Índice de figuras.....XI

Índice de tablas.....XV

CAPITULO I

1. Análisis del comportamiento de las señales de los sensores CKP, MAP y auxiliares…1

1.1. Antecedentes 1

1.2. Sensores.....2

1.2.1. Clasificación de los sensores.....3

1.2.1.1. Sensores según su función.....4

1.2.1.2.Sensores según la señal de salida.....4

1.3. Análisis.....5

1.3.1. Sensor de presión absoluta en el colector de admisión (MAP).....5

1.3.1.1. Funcionamiento sensor MAP.....6

1.3.1.2. Tipos de sensor MAP.....6

1.3.2. Señal de inyección.....7

1.3.2.1. Funcionamiento del sensor MAP para verificar el estado del motor.....8

1.3.2.1.1. Falla en el cilindro número uno.....10

1.3.2.1.2. Falla en el cilindro número dos.....11

1.3.2.1.3. Falla en el cilindro número tres.....12

1.3.2.1.4. Falla en el cilindro número cuatro.....13

1.4. Fase de compresión de un motor ciclo Otto.....15

CAPÍTULO II

2. Diseño del equipo de diagnóstico para motores ciclo Otto.....18
2.1. Introducción.....18
2.2. Razón del diseño del equipo.....18
2.2.1. Fallas eléctricas.....18
2.2.2. Fallas mecánicas.....19
2.3. Características del diseño.....20
2.3.1. Movilidad.....20
2.3.2. Debe responder efectivamente y no ocupar mucha memoria.....20
2.3.3. Interfaz intuitiva y fácil de usar.....20
2.3.4. Posibilidad de ampliación de funciones e información.....20
2.4. Diseño inicial.....21
2.5. Software libre.....22
2.5.1. Cuadro comparativo.....23
2.6. Hardware y software arduino….....24
2.6.1 Hardware arduino.....24
2.6.2. Software arduino.....26
2.7. Tarjeta arduino para implementar nuestro dispositivo.....28
2.7.1. Características de la tarjeta arduino mega.....29
2.7.1.1. Pin Vin.....29
2.7.1.2. Pin 5v.....29
2.7.1.3. Pin 3.3v.....29
2.7.1.4.Memoria de la placa.....29
2.7.1.5.Pines de entrada y salida.....30
2.7.1.6. Pines serial.....30
2.7.1.7. Pines de interrupciones externas.....30
2.8. Android.....30
2.9. Diseño de la interfaz.....31
2.9.1. Pantalla de bienvenida.....31
2.9.2. Pantalla inicial.....32

2.9.3. Pantalla de conexión.....33
2.9.4. Pantalla observaciones.....33
2.9.5. Pantalla de indicaciones para el análisis estático.....34
2.9.6. Pantalla análisis estático.....35
2.9.7. Pantalla de indicaciones del análisis dinámico.....36
2.9.8. Pantalla análisis dinámico.....36
2.9.9. Pantalla de resultados.....37
2.10. Comunicación entre arduino y android.....38
2.10.1. A través de comunicación wireless.....38
2.10.2. A través de comunicación con cable de datos.....39
2.10.3. A través de comunicación bluetooth.....40
2.11. Diseño final del equipo.....41

CAPÍTULO III

3. Construcción del equipo de diagnóstico para motores ciclo Otto.....43
3.1. Introducción.....43
3.2. Construyendo la aplicación android.....43
3.3. Entorno de desarrollo.....44
3.3.1. Vista package explorer.....45
3.3.2. Vista outline.....46
3.3.3. Vista console, problems.....47
3.3.4. Vista área de trabajo.....47
3.3.5. Botón para correr la aplicación.....48
3.4. Construyendo la aplicación para android.....50
3.4.1. Construyendo la pantalla de bienvenida splash.....50
3.4.2. Construyendo la pantalla inicial.....57
3.4.3. Construyendo la pantalla conexión.....59
3.4.4. Construyendo la pantalla observaciones.....62
3.4.5. Construyendo la pantalla indicaciones estático.....67

3.4.6. Construyendo la pantalla análisis estático.....71
3.4.7. Construyendo la pantalla indicaciones dinámico.....77
3.4.8. Construyendo la pantalla resultados.....88
3.5. Construyendo la aplicación en arduino.....91
3.5.1. Conectar la placa con el software.....91
3.5.2. Partes principales de la ventana.....93
3.5.2.1. Área de trabajo.....93
3.5.2.2. Botón verificar.....94
3.5.2.3. Cargar / compilar.....95
3.5.2.4. Pantalla de estado.....95
3.5.3. Compilación en proteus.....95
3.6. Construyendo el código para implementar la adquisición de datos.....97
3.6.1. Declaración de variables.....97
3.6.1.1. Variables estáticas.....97
3.6.1.2. Variables tipo byte.....97
3.6.1.3. Variables enteras y decimales.....98
3.6.2. Estado de las variables.....98
3.6.3. Función de las variables.....100
3.7. Diseño de una placa generadora de pulsos.....113
3.7.1. Placa generadora.....104
3.7.1.1 Circuito integrado 555.....106
3.7.1.1.1. Circuito monoestable.....106
3.7.1.1.2. Circuito astable.....106
3.7.1.2. Circuito integrado 555 astable.....107

CAPÍTULO IV

4. Análisis de resultados de funcionamiento del equipo de diagnóstico en vehículos dotados con el sistema MPFI MULTEC DELPHI.....111
4.1. Introducción.....111
4.2. Creando un patrón.....111
4.3. Recolectando datos Chevrolet Corsa.....111
4.4. Recolectando datos Chevrolet Luv.....116
4.5. Recolectando datos Chevrolet Aveo.....120
4.6. Recopilación de datos generados por el equipo de diagnóstico.....124

CAPÍTULO V

5. Validación del equipo de diagnóstico.....126
5.1. Introducción.....126
5.2. Pruebas de campo.....126
5.3 Validación.....127
5.3.1 Validación con el manómetro.....127
5.3.2 Validación matemática.....130
5.4 Demostración porcentual del sistema.....133
5.5. Modelado matemático.....133
5.6. Análisis en simulink.....138

ÍNDICE DE FIGURAS

CAPITULO I

Figura 1.1. Estructura de un sensor.....2
Figura 1.2. Sensores en el vehículo.....3
Figura 1.3. Tipos de Señales.....4
Figura 1.4. Sensor MAP. 5
Figura 1.5. Onda sensor MAP.....7
Figura 1.6 Señal de inyección.....7
Figura 1.7. Voltaje vs Presión. 8
Figura 1.8. Señal Sensor MAP.....9
Figura 1.9. Señal Sensor MAP.....10
Figura 1.10. Señal Sensor MAP.....11
Figura 1.11. Señal Sensor MAP.....12
Figura 1.12. Señal Sensor MAP.....13
Figura 1.13. Señal Sensor MAP a 115 PSI.....14
Figura 1.14. Señal Sensor MAP a 100 PSI.....14
Figura 1.15 Relación de calores específicos.....16
Figura1.16. Presión – Volumen de un cilindro.....16
Figura 1.17.Fases del motor.....17

CAPITULO II

Figura 2.1. Pantalla LCD 128x240.....21
Figura 2.2. Pantalla TFT 2.8 plg.....22
Figura 2.3 Miscelánea de programas de software libre.....23
Figura 2.4 Cuadro comparativo.....23
Figura 2.5. Arduino.....25
Figura 2.6 Placas de Arduino.....25

Figura 2.7 Software Arduino.....26
Figura 2.8 Elementos de soporte de un Micro controlador.....27
Figura 2.9 Monitor Serial Arduino.....27
Figura 2.10 Monitor Serial Arduino.....28
Figura 2.11. Sistema operativo ANDROID.....31
Figura 2.12. Pantalla Splash.....32
Figura 2.13. Pantalla Inicial.....32
Figura 2.14. Pantalla de conexión.....33
Figura 2.15. Pantalla Observaciones.....34
Figura 2.16. Pantalla Indicaciones Análisis Estático.....34
Figura 2.17. Pantalla Análisis Estático.....35
Figura 2.18. Pantalla Indicaciones Análisis Dinámico.....36
Figura 2.19. Pantalla Análisis Dinámico.....37
Figura 2.20. Pantalla Resultados.....38
Figura 2.21. Módulo Wireless para arduino.....39
Figura 2.22. Conexión por cable de datos.....40
Figura 2.23. Módulo Bluetooth.....41
Figura 2.24. Caja circuitos electrónicos parte 1 del equipo.....41
Figura 2.25. Celular inteligente con sistema android parte 2 del equipo.....42
Figura 2.26. Cables de conexión para los sensores.....42

CAPITULO III

Figura 3.1.Entorno de desarrollo.....43
Figura 3.2. Ventana Entorno de desarrollo.....44
Figura 3.3. Vista Package.....45
Figura 3.4. Vista Outline.....46
Figura 3.5. Vista Console, problems.....47
Figura 3.6. Vista Área de trabajo.....47
Figura 3.7. Botón Run.....48

Figura 3.8. Configuración Botón Run.....48
Figura 3.9. Configuración Botón Run.....49
Figura 3.10. Emulador del entorno de desarrollo.....50
Figura 3.11. Pantalla Splash.....51
Figura 3.12. Carpeta Layout.....51
Figura 3.13. Programación de forma visual.....52
Figura 3.14. Programación mediante códigos.....52
Figura 3.15. Carpeta Drawable.....54
Figura 3.16. Carpeta SRC.....55
Figura 3.17. Pantalla menú.....58
Figura 3.18. Pantalla de conexión.....60
Figura 3.19. Pantalla Observaciones.....64
Figura 3.20. Pantalla Observaciones función botón Análisis.....66
Figura 3.21. Pantalla indicaciones Análisis Estático.....69
Figura 3.22. Pantalla Análisis Estático.....73
Figura 3.23. Ajuste de datos de porcentaje vs presión en CFTOOL.....74
Figura 3.24. Ajuste de datos Presión vs Bits en CFTOOL.....76
Figura 3.25. Pantalla indicaciones Análisis Dinámico.....79
Figura 3.26. Análisis Dinámico.....83
Figura 3.27. Ajuste de datos CFTOOL.....84
Figura 3.28. Datos Análisis Estático.....87
Figura 3.29. Datos Análisis Dinámico.....87
Figura 3.30. Pantalla Resultados.....89
Figura 3.31. Tipo de Tarjeta Arduino.....91
Figura 3.32. Puerto y Tarjeta Utilizada.....92
Figura 3.33. Área de Trabajo.....93
Figura 3.34. Botón Verificar.....94
Figura 3.35. Líneas de Información de Error.....94
Figura 3.36. Botón Cargar.....95

Figura 3.37. Pantalla de Estado.....95
Figura 3.38 Activación del Compilador.....96
Figura 3.39. Código Compilado y Revisado en Proteus.....96
Figura 3.40. Tipo de interrupción.....99
Figura 3.41 Señal Map y Pulso de Inyección.....104
Figura 3.42. Tip 122 Armado en Proteus.....104
Figura 3.43 Simulación en proteus.....105
Figura 3.44 Chip HFE40106.....105
Figura 3.45 Circuito 555 Monoestable.....106
Figura 3.46 Circuito 555 Aestable.....107
Figura 3.47 Chip 555 configuración.....107
Figura 3.48. Diseño del circuito en Proteus.....108
Figura 3.49 Pulsos de Inyección.....109
Figura 3.50. Circuito armado en Project Board.....110
Figura 3.51. Circuito armado en una placa perforada.....110

CAPITULO IV

Figura 4.1. Chevrolet corsa.....112
Figura 4.2. Voltaje sensor MAP Chevrolet corsa.....112
Figura 4.3. Voltaje sensor MAP Chevrolet corsa.....114
Figura 4.4. Voltaje sensor MAP Chevrolet corsa.....114
Figura 4.5. Chevrolet Luv.....116
Figura 4.6. Voltaje sensor MAP Chevrolet Luv.....116
Figura 4.7. Voltaje sensor MAP Chevrolet Luv.....118
Figura 4.8. Voltaje sensor MAP Chevrolet Luv.....118
Figura 4.9. Chevrolet Aveo.....120
Figura 4.10. Voltaje sensor MAP Chevrolet Aveo.....120
Figura 4.11. Voltaje sensor MAP Chevrolet Aveo.....122
Figura 4.12. Voltaje sensor MAP Chevrolet Aveo.....122

Figura 4.13. Gráfica de dispersión de resultados Equipo vs manómetro.....124
Figura 4.14. Gráfica de dispersión de resultados Equipo.....125

CAPITULO V

Figura 5.1. Vehículos Chevrolet Corsa, Luv, Aveo.....126
Figura 5.2. Datos iniciales del equipo.....128
Figura 5.3. Señal del sensor MAP (Luv 2.2).....128
Figura 5.4. Presión del Cilindro.....129
Figura 5.5. Datos calibrados del equipo.....130
Figura 5.6. Presión en función del número de bits.....132
Figura 5.7. Estado porcentual de los cilindros.....133
Figura 5.8. Representación del sistema de admisión de aire.....134
Figura5.9. Modelado Matemático.....138
Figura 5.10 Flujo Másico.....139
Figura 5.11. Voltaje sensor MAP Chevrolet Corsa.....139
Figura 5.12. Voltaje sensor MAP Chevrolet Aveo.....140

INDICE TABLAS

CAPÍTULO IV

Tabla 4.1. Voltaje Sensor MAP Chevrolet corsa.....107
Tabla 4.2. Bits referentes al Sensor MAP Chevrolet corsa.....107
Tabla 4.3. Presión de cada cilindro Chevrolet corsa. 107
Tabla 4.4. Presión Manómetro vs Presión del equipo.....110
Tabla 4.5. Voltaje Sensor MAP Chevrolet Luv.....111
Tabla 4.6. Bits referentes al Sensor MAP Chevrolet Luv.....111
Tabla 4.7. Presión de cada cilindro Chevrolet Luv.....112
Tabla 4.8. Presión Manómetro vs Presión del equipo.....113

Tabla 4.9. Voltaje Sensor MAP Chevrolet Aveo.....115
Tabla 4.10. Bits referentes al Sensor MAP Chevrolet Aveo.....115
Tabla 4.11. Presión de cada cilindro Chevrolet Aveo.....117
Tabla 4.12. Presión Manómetro vs Presión del equipo.....117

CAPÍTULO V

Tabla 5.1: Presión de los cilindros.....119
Tabla 5.2 Presión – Voltaje.....121
Tabla 5.3 Voltaje –Presión.....123
Tabla 5.4 Presión – Número de Bits del Sistema.....123

CONCLUSIONES.....142

RECOMENDACIONES.....147

BIBLIOGRAFIA.....149

ANEXOS.....152

FICHA SENESCYT.....

CAPÍTULO I

ANÁLISIS DEL COMPORTAMIENTO DE LAS SEÑALES DE LOS SENSORES CKP, MAP Y AUXILIARES.

CAPÍTULO I

1. ANÁLISIS DEL COMPORTAMIENTO DE LAS SEÑALES DE LOS SENSORES CKP, MAP Y AUXILIARES.

1.1. ANTECEDENTES.

Hace algunos años atrás el número de vehículos existentes en el medio eran muy bajos, donde únicamente la gente adinerada podía conseguirlos, dichos vehículos carecían de muchas mejoras actualmente existentes en diferentes vehículos, pero en aquellos tiempos más que una necesidad era un símbolo de superioridad, su consumo de combustible era alto así como su contaminación, pero debido a que no existían muchos automotores el problema sobre la contaminación estaba descuidado por parte de los ingenieros, a medida que pasaron los años y la cantidad de vehículos fue en aumento, de esta manera la contaminación también se fue acrecentando, a raíz de esto surgieron entidades que empezaron a preocuparse por el medio ambiente, de tal manera que impusieron normas a nivel mundial sobre la contaminación que puede producir un vehículo, esta decisión provocó un gran cambio en la industria automotriz hacia un nuevo desarrollo con el fin de reducir la contaminación.

De tal manera que los ingenieros automotrices buscando una solución a la problemática implementaron una serie de sensores en los vehículos, y desarrollaron nuevas tecnologías para reducir la contaminación, es por tal motivo que la mayoría de vehículos actuales poseen varios sensores que ayudan mucho en la gestión electrónica del vehículo, reduciendo el consumo de combustible, reduciendo la contaminación y alargando los periodos para dar mantenimiento a los vehículos.

Los diversos tipos de sensores pueden ser de efecto inductivos o de efecto hall, cada uno de ellos realiza funciones muy importantes para la gestión electrónica del vehículo, puesto que cada sensor envía una distinta señal hacia la unidad de control electrónica conocida como “ECU”, con la finalidad de brindar seguridad, confort en

la conducción e incluso la reducción de contaminantes hacia el exterior del vehículo (gases de escape).

1.2. SENSORES.

Un sensor es aquel elemento que convierte una magnitud física (tales como: temperatura, revoluciones, velocidad, impacto, etc.), o química (para los gases de escape, calidad de aire, etc.), en una magnitud eléctrica para que la unidad de control "ECU" pueda reconocerla fácilmente y de esta manera pueda realizar alguna operación.

En la señal eléctrica obtenida del sensor se debe considerar la amplitud de corriente y de tensión, la frecuencia, el periodo, así como la duración de la oscilación eléctrica. Con el fin de realizar un análisis, cálculo, o algún tipo de procesamiento para mejorar el funcionamiento, en este caso del motor o del mismo vehículo.

A continuación un esquema básico de un sensor:

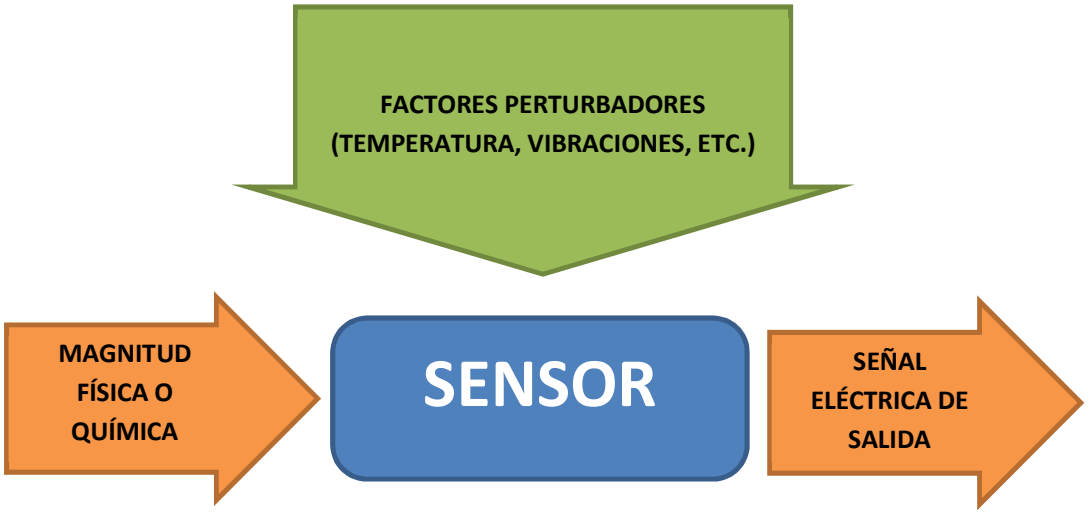


Figura 1.1. Estructura de un sensor.

Fuente: Los Autores.

1.2.1. CLASIFICACIÓN DE LOS SENSORES.

Conocido ya los aspectos importantes de funcionamiento de los sensores, estos se podrían clasificar según su función y también según la señal de salida que entrega cada sensor.

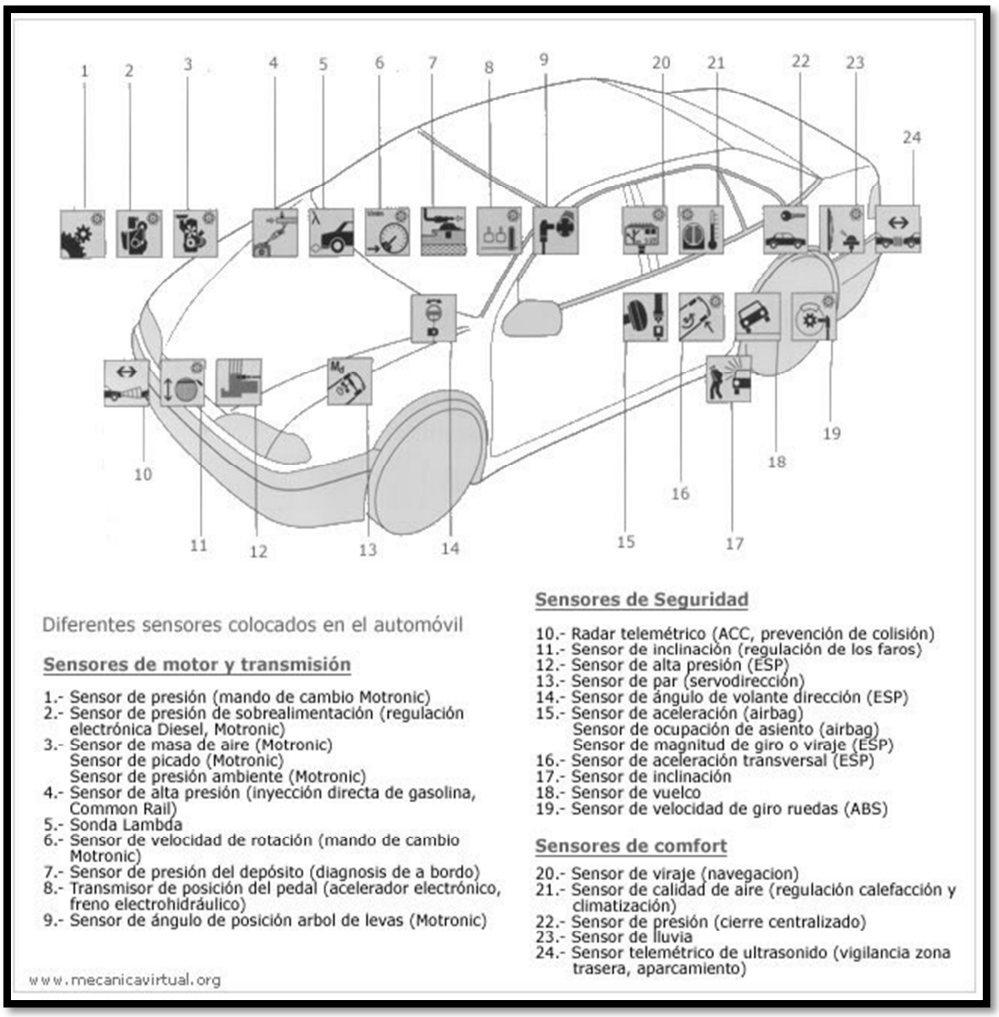


Figura 1.2. Sensores en el vehículo.

Fuente: www.mecanicavirtual.org.

1.2.1.1. SENSORES SEGÚN SU FUNCIÓN.

Hace referencia a su utilización por ejemplo en sistemas antirrobo, sistemas de diagnóstico, sistemas de mando y regulación.

1.2.1.2. SENSORES SEGÚN LA SEÑAL DE SALIDA.

Cada sensor según su aplicación se puede tener una señal analógica (sensor de temperatura, sensor CKP, sensor de aire aspirado, etc.), o también se puede tener una señal digital (sensores de efecto hall, sensor CKP, etc.)

De tal manera que cada sensor tiene una función muy importante dentro del vehículo, de acuerdo a la implementación de nuestro proyecto vamos a proceder a explicar y analizar a los sensores necesarios e importantes para la elaboración del equipo de diagnóstico.

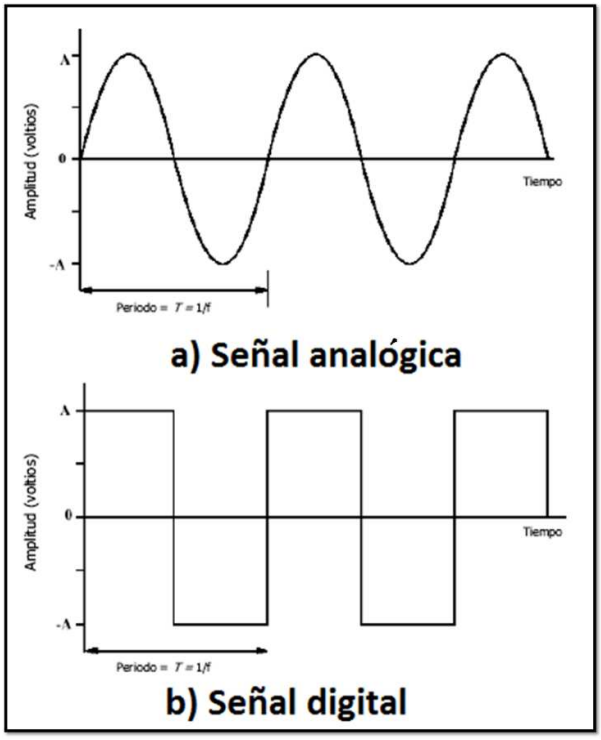


Figura 1.3. Tipos de Señales.

Fuente: Los autores.

1.3. ANÁLISIS.

Para el caso de nuestro proyecto de tesis vamos a utilizar los siguientes sensores:

1.3.1. SENSOR DE PRESIÓN ABSOLUTA EN EL COLECTOR DE ADMISIÓN (MAP).

Este sensor es el encargado de enviar un determinado valor de voltaje de acuerdo al vacío generado dentro del motor, con el fin de modificar el tiempo final de la inyección y por ende la cantidad de combustible a inyectar.

Internamente es como un chip de silicón el cuál flexiona según varíe la presión, cuyo movimiento permite un cambio en la señal de voltaje.

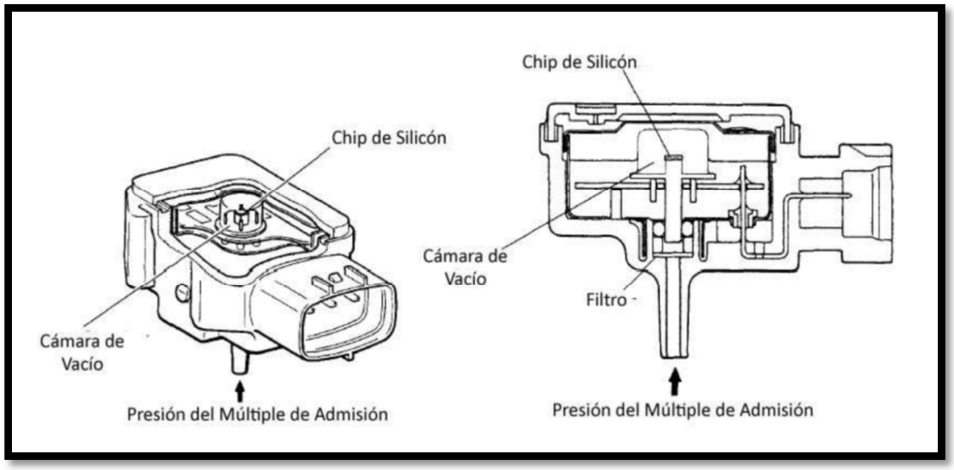


Figura 1.4. Sensor MAP.
Fuente: http://e-auto.com.mx.

Como se puede apreciar en la figura, el sensor tiene una cámara de vacío, esta puede ser de un vacío perfecto o una presión calibrada.

El sensor MAP posee tres cables y una conexión para conectar al colector de admisión:

- Un cable para tensión de alimentación de 5 voltios.
- Un cable para la conexión a masa.
- Un cable para la salida de la señal del sensor.
- Una manguera para la conexión del sensor con el colector de admisión, en algunos casos no se requiere de dicha manguera puesto que el sensor se coloca directamente en el colector de admisión.

1.3.1.1. FUNCIONAMIENTO SENSOR MAP.

Este sensor utiliza la cámara de vacío como un punto de referencia y a medida que varía la presión del colector de admisión según la carga del motor provocará una diferencia de presión entre las dos cámaras, este cambio hace que su resistencia eléctrica varíe, por ende se modificará el valor de voltaje de la señal entregada por el sensor, este cambio es captado por la unidad de control “ECU” que lo interpreta como un cambio de presión.

1.3.1.2. TIPOS DE SENSOR MAP.

Como ya se mencionó anteriormente existen sensores análogos y digitales, y este sensor MAP puede ser de los dos tipos, su aplicación difiere bastante según el tipo de vehículo. A continuación una gráfica del sensor MAP analógico:

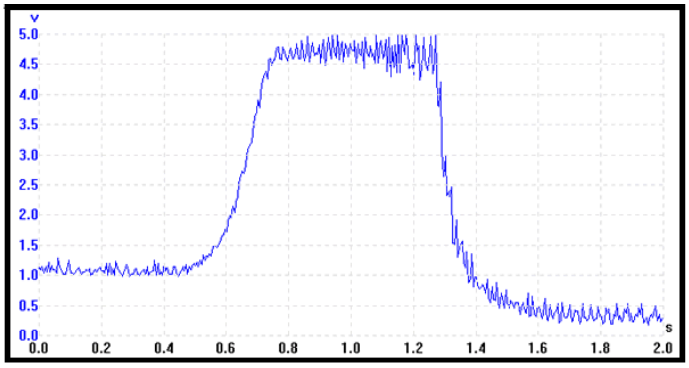


Figura 1.5. Onda sensor MAP.

Fuente: http://www.miac.es.

1.3.2. SEÑAL DE INYECCIÓN.

La señal del inyector es usado en nuestro proyecto con el fin de determinar el orden de encendido que posee el motor, esto nos ayuda al momento de programar puesto que el sensor MAP nos genera la señal para cada cilindro casi de forma idéntica, haciendo imposible determinar cuál es el primer cilindro.

Esta señal de inyección no es analizada, únicamente se reduce su voltaje para usarlo como disparo para determinar al primer cilindro del motor, como el voltaje de inyección esta por los 12 voltios, al ingresar esta señal sin ser tratada puede quemar la placa arduino.

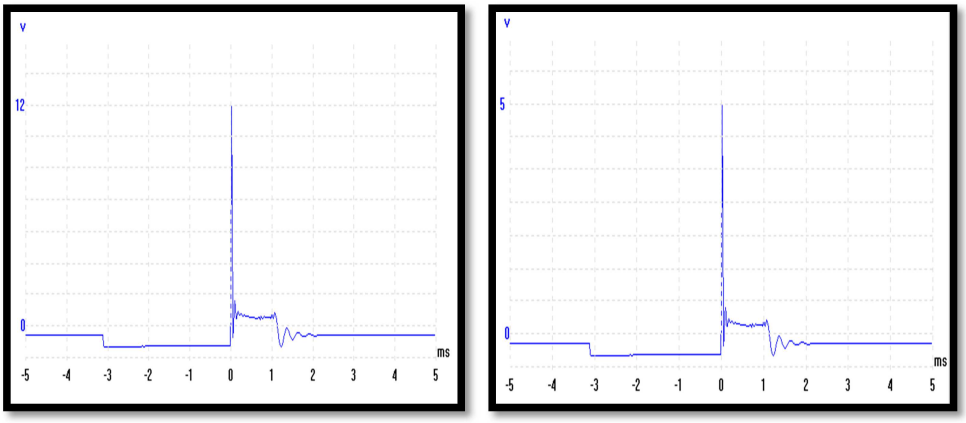


Figura 1.6 Señal de inyección.

Fuente: Los Autores.

1.3.2.1. FUNCIONAMIENTO DEL SENSOR MAP PARA VERIFICAR EL ESTADO DEL MOTOR.

Para nuestro caso el sensor MAP utilizado es uno tipo analógico, cuyo voltaje que entrega a la salida del pin de señal es proporcional a la presión interna del motor, puesto que al incrementar la presión del colector de escape el voltaje del sensor MAP incrementa, razón por la cual el voltaje obtenido del sensor MAP es alto cuando el motor está apagado, mientras que al encenderlo el voltaje que entrega el sensor disminuye debido a que la presión del colector de admisión también disminuye.

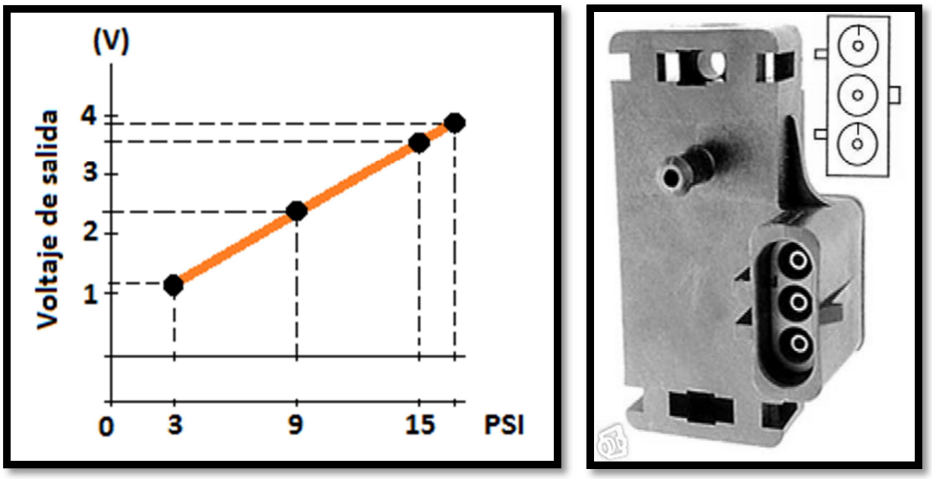


Figura 1.7. Voltaje vs Presión.

Fuente: Los Autores.

En nuestro proyecto hemos usado un sensor MAP externo, con la finalidad de usarlo como patrón para todos los vehículos a analizar, reduciendo así el número de conexiones a realizar en el motor dotado con sistema de inyección MPFI MULTEC DELPHI.

Este sensor es de gran ayuda al momento de realizar un análisis del estado del motor, puesto que al conectar el osciloscopio al sensor, nos entrega una señal analógica indicando el estado de cada cilindro que posee el motor.

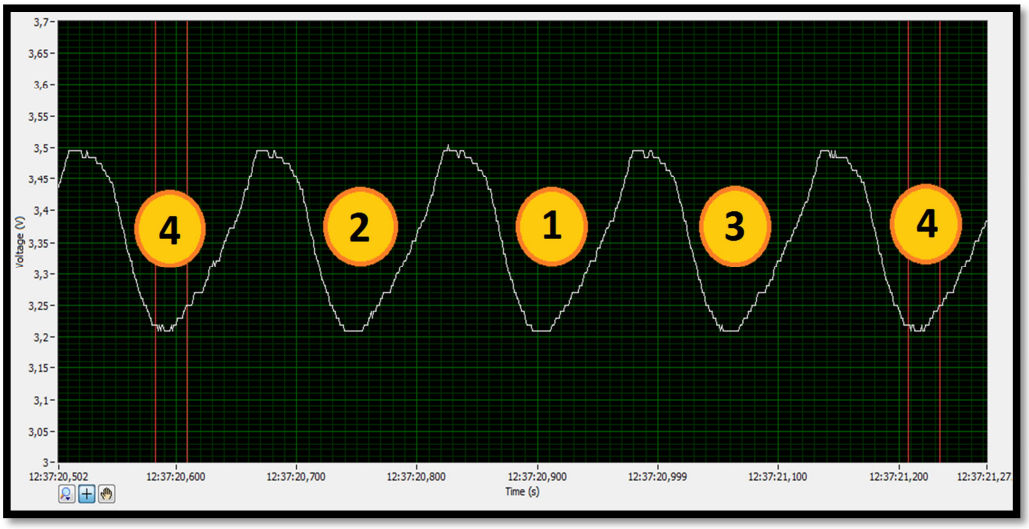


Figura 1.8. Señal Sensor MAP.

Fuente: Los Autores (Adquisición de datos).

Como se puede apreciar en la figura 1.8, se tiene la gráfica que nos genera el sensor MAP, los números indican el orden de encendido y la aspiración de cada cilindro, es decir, el número cuatro conjuntamente con las líneas rojas indica aspiración del cuarto cilindro y la inyección en el primer cilindro respectivamente. De tal manera que cuando el motor se encuentra en perfectas condiciones todos los valores mínimos de voltaje de la onda sinusoide debe ser igual u oscilar por el mismo valor de voltaje que en este caso aproximado es de 3.21 volts, esto para todos los cilindros.

Pero cuando uno de los cilindros se encuentra en mal estado o presentan fugas en la compresión indicando el ovalamiento del cilindro u otras causas para tener una deficiencia en su funcionamiento, la onda sinusoide que visualizamos en el osciloscopio va a variar indicando la falla presente en uno o varios cilindros.

Para poder visualizar las fallas de cada cilindro en el osciloscopio nosotros provocamos una falla para cada cilindro y obtuvimos las siguientes graficas:

1.3.2.1.1. FALLA EN EL CILINDRO NÚMERO UNO.

Provocando una falla en el cilindro uno se obtuvo la siguiente gráfica:

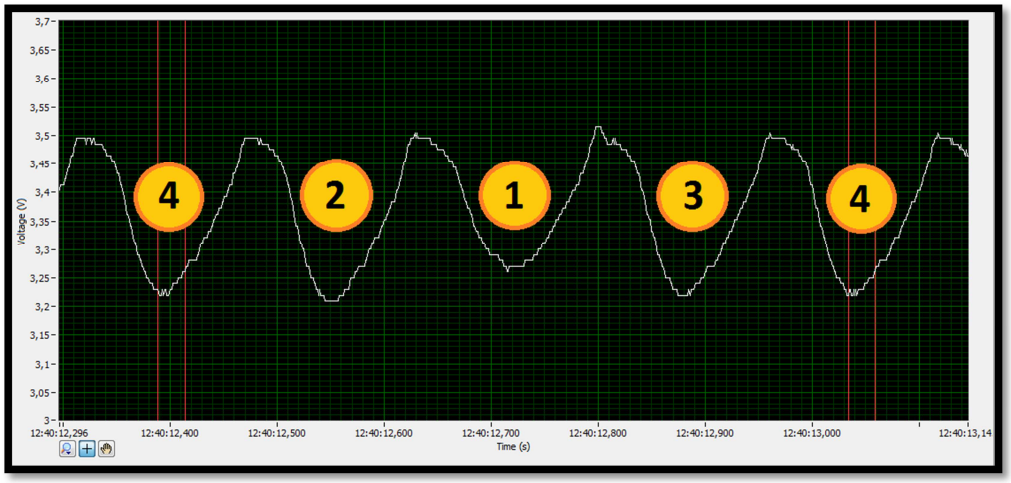


Figura 1.9. Señal Sensor MAP.

Fuente: Los Autores (Adquisición de datos).

La falla provocada en el cilindro uno nos genera una gráfica diferente, tal como se aprecia en la figura 1.9, el número uno indica la aspiración del primer cilindro, debido a la falla generada se tiene una variación en su voltaje mínimo el cual está por los ***3,27 voltios.***

1.3.2.1.2. FALLA EN EL CILINDRO NÚMERO DOS.

Provocando una falla en el cilindro dos se obtuvo la siguiente gráfica:

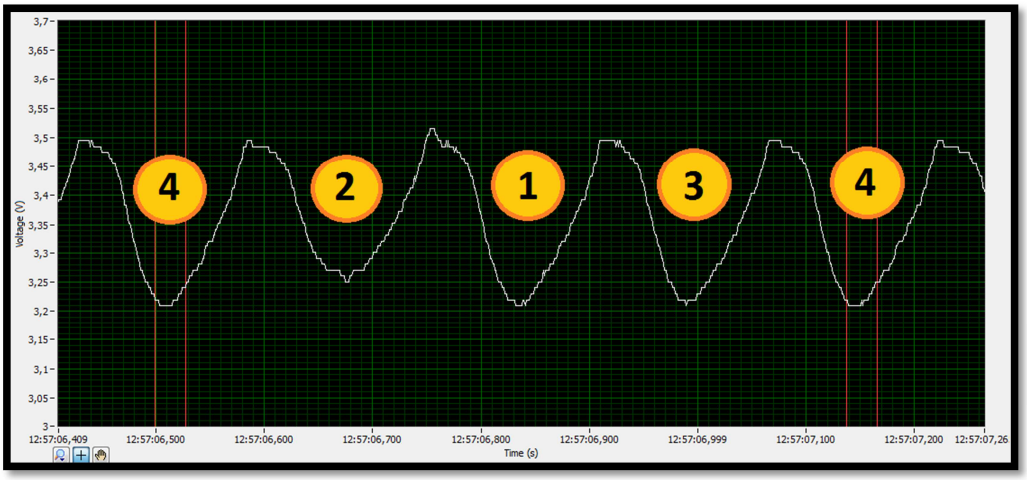


Figura 1.10. Señal Sensor MAP.

Fuente: Los Autores (Adquisición de datos).

La falla provocada en el cilindro dos nos genera una gráfica diferente, tal como se aprecia en la figura 1.10, el número dos indica la aspiración del segundo cilindro, debido a la falla generada se tiene una variación en su voltaje mínimo el cual está por los ***3,25 voltios.***

1.3.2.1.3. FALLA EN EL CILINDRO NÚMERO TRES.

Provocando una falla en el cilindro tres se obtuvo la siguiente gráfica:

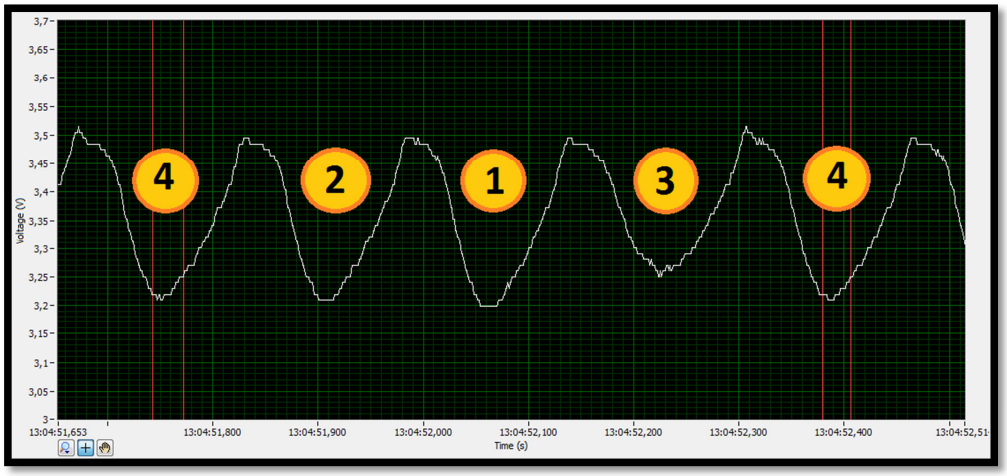


Figura 1.11. Señal Sensor MAP.

Fuente: Los Autores (Adquisición de datos)

La falla provocada en el cilindro tres nos genera una gráfica diferente, tal como se aprecia en la figura 1.11, el número tres indica la aspiración del tercer cilindro, debido a la falla generada se tiene una variación en su voltaje mínimo el cual está por los ***3,25 voltios.***

1.3.2.1.4. FALLA EN EL CILINDRO NÚMERO CUATRO.

Provocando una falla en el cilindro cuatro se obtuvo la siguiente gráfica:

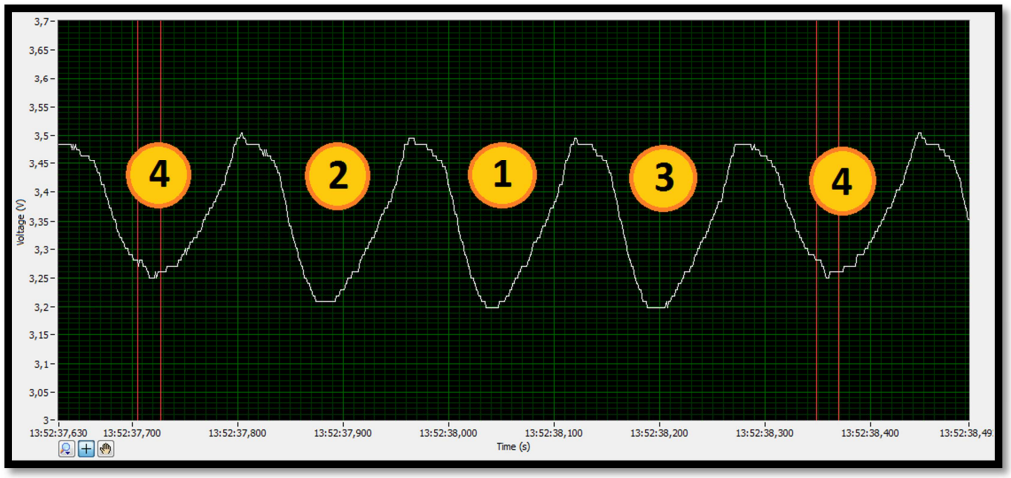


Figura 1.12. Señal Sensor MAP.

Fuente: Los Autores (Adquisición de datos).

La falla provocada en el cilindro cuatro nos genera una gráfica diferente, tal como se aprecia en la figura 1.12, el número cuatro indica la aspiración del cuarto cilindro, debido a la falla generada se tiene una variación en su voltaje mínimo el cual está por los ***3,25 voltios.***

Todas estas fallas son individuales, en el caso que los cilindros del motor se desgasten independientemente, pero como son un conjunto todos trabajan todos por igual, pudiendo existir una variación no muy exagerada de desgaste entre los cilindros, de tal manera que al comenzar a existir desgaste lo que varía es el voltaje que nos entrega el sensor MAP.

Cuando el motor del vehículo no presenta desgaste el voltaje que nos entrega el sensor MAP es alto, mientras que al presentar desgaste el valor de voltaje empieza a descender, es gracias a ello que realizamos la programación pertinente para determinar el desgaste que presenta el motor.

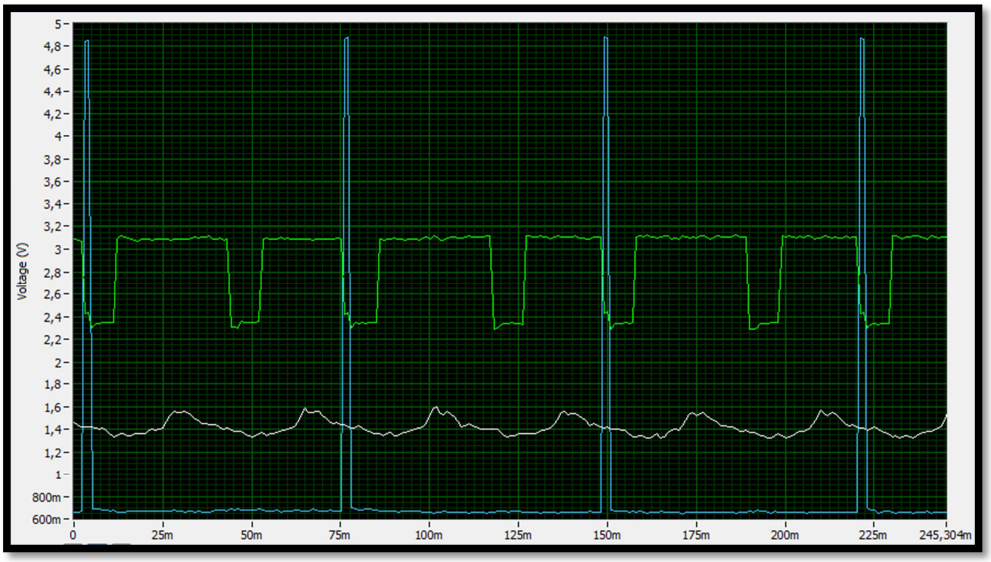


Figura 1.13. Señal Sensor MAP a 115 PSI.

Fuente: Los Autores (Adquisición de datos).

Figura 1.14. Señal Sensor MAP a 100 PSI.

Fuente: Los Autores (Adquisición de datos).

Como podemos apreciar los valores de voltaje que nos da el sensor MAP existe una variación mínima, y es la que nos ayuda para generar los datos a mostrar en el equipo de diagnóstico.

1.4. FASE DE COMPRESIÓN DE UN MOTOR CICLO OTTO.

Para analizar el ciclo de compresión de un motor a gasolina y verificar la función del sensor MAP, teniendo en cuenta que este tipo de motor es adiabático, lo que quiere decir que un fluido realiza trabajo, ser adiabático consiste también a que elementos impidan la trasferencia de calor hacia el exterior como es el caso de los cilindros.

Para un sistema adiabático tenemos la siguiente fórmula:

$$P_2 = P_1 * (\frac{V_1}{V_2})^k$$

$$T_2 = T1 * (\frac{V_1}{V_2})^{k-1}$$

Dónde:

P: Presión.
V: Volumen.
T: Temperatura.

Con un valor de relación de compresión (*r*):

$$r = \frac{Vmax}{Vmin} = \frac{V1}{V2}$$

La relación de calores específicos viene dado por $k = \frac{C_P}{C_V}$. En termodinámica se puede decir que la eficiencia térmica del ciclo Otto aumenta con la relación de calores específicos “k” del fluido de trabajo.

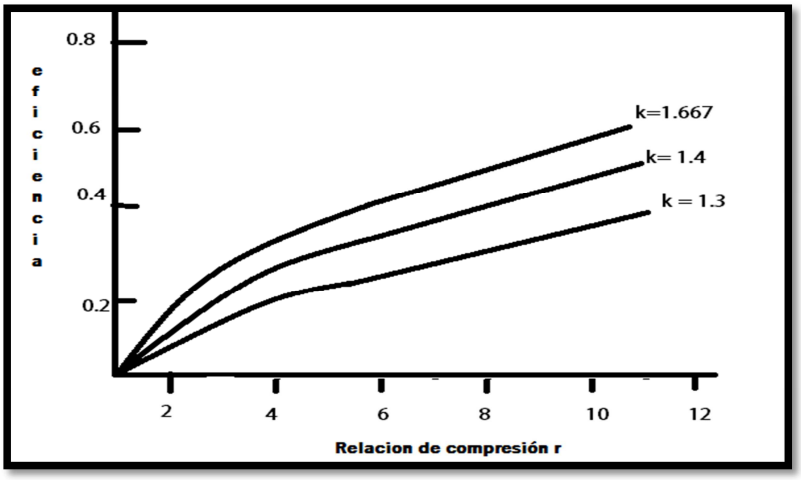


Figura 1.15 Relación de calores específicos.

Fuente: Los Autores.

Determinando la compresión en función de la relación de compresión.

$$p_2 = p_1 * r_v^{\,k} (Pa)$$

El valor para k es de 1.4, es un valor de relación de calores específicos del aire a temperatura ambiente.

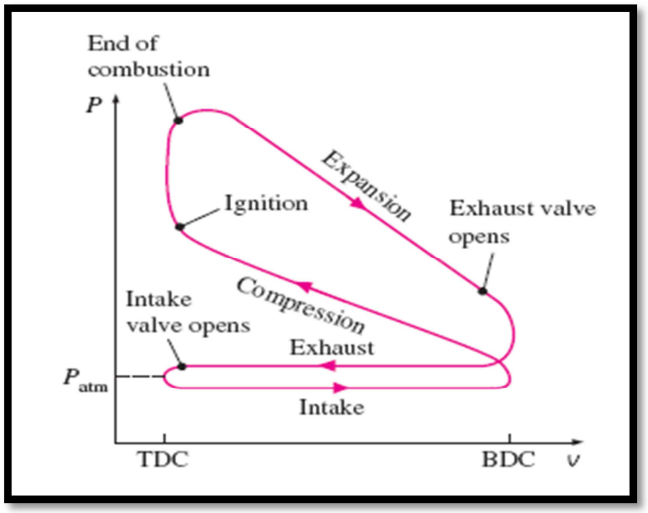


Figura1.16. Presión – Volumen de un cilindro.

Fuente: http://m.forocoches.com

Se puede observar que la presión de admisión es inferior a la presión atmosférica, este fenómeno es porque el pistón absorbe el aire atmosférico reduciendo la presión con la que ingresa en nuestro colector, una vez que la válvula de admisión se ha cerrado la presión en el cilindro permanece constante en toda su fase de admisión, para luego continuar con su fase de compresión en la cual efectivamente existe un aumento de la presión dentro de nuestro cilindro, la fase de expansión es donde se produce el trabajo o movimiento del cigüeñal, para finalizar con el ciclo de escape y permitir la salida de los gases a través de la válvula de escape.

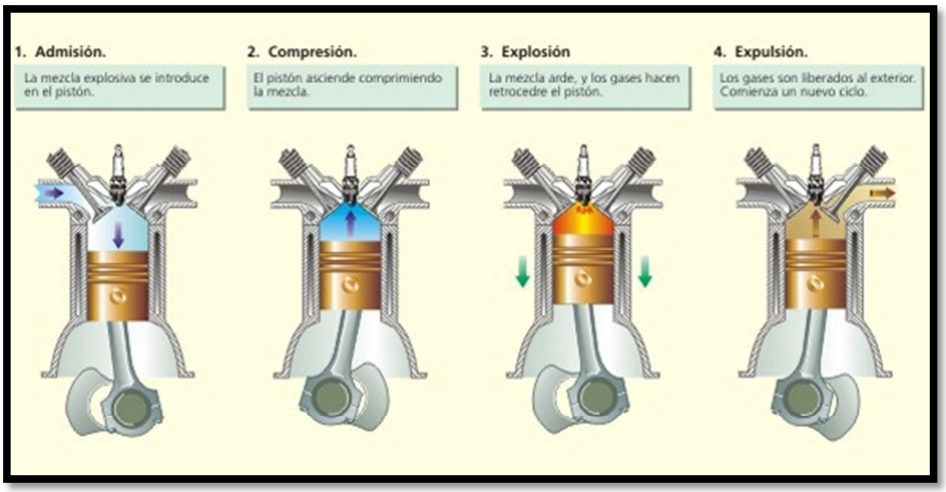


Figura 1.17.Fases del motor.

Fuente: http://examenandresgavilanes.blogspot.com/

CAPÍTULO II

DISEÑO DEL EQUIPO DE DIAGNÓSTICO PARA MOTORES CICLO OTTO.

CAPÍTULO II

2. DISEÑO DEL EQUIPO DE DIAGNÓSTICO PARA MOTORES CICLO OTTO.

2.1. INTRODUCCIÓN.

A través del tiempo la tecnología se ha incrementado a tal punto que la mayor parte de sistemas de control del vehículo son de tipo electrónicos, facilitando el manejo a los usuarios, y brindándoles mayor comodidad.

Al igual que la tecnología avanza en el control vehicular también se han desarrollado múltiples equipos para diagnosticar el estado de un vehículo gracias a la ayuda de múltiples sensores que los vehículos poseen, ayudando al Ingeniero Automotriz a detectar mucho más rápido y de manera precisa las fallas de un vehículo, así como, a reducir el tiempo de intervención al momento de detectar fallas.

2.2. RAZÓN DEL DISEÑO DEL EQUIPO.

Este proyecto pretende facilitar el diagnóstico del estado físico del tren alternativo del motor de combustión interna a gasolina mediante el análisis de la curva que proporcionan el sensor MAP y auxiliares.

Estas señales serán analizadas con el fin de obtener datos importantes, que indican el estado de cada cilindro (según el número de cilindros que tenga el vehículo) con valores porcentuales referentes al estado del tren alternativo del motor, así como también un valor de presión.

Entonces la finalidad de este proyecto es proporcionar al mecánico automotriz de un equipo capaz de indicar el estado físico del tren alternativo del motor, esto de una manera rápida y sin desmontajes previos para constatar el fallo de uno o varios

cilindros, pues lo único que tendrá que realizar el mecánico será conectar el equipo al sensor MAP y auxiliares del vehículo, y esperar el resultado que nos proporcionará el equipo.

Ya obtenidos los datos el mecánico sabrá que cilindro tiene una deficiencia de potencia, ya sea provocada por fallas eléctricas o mecánicas.

2.2.1. FALLAS ELÉCTRICAS.

Pudiera darse el caso de que la falla o deficiencia de potencia este siendo provocada por la ausencia de chispa en la bujía, o también por ausencia de inyección de combustible ya sea inyector dañado o sin pulso de inyección.

2.2.2. FALLAS MECÁNICAS

La deficiencia de potencia en el motor también radica en problemas mecánicos, ya que cuando un motor a gasolina está próximo a repararse su potencia tiende a disminuir debido a la mala estanqueidad del conjunto cilindro-pistón, bloque-cabezote, e incluso entre las válvulas y su respectivo asiento.

Actualmente los Mecánicos Automotrices para la comprobación del estado del tren alternativo del motor utilizan algunos métodos para su comprobación, utilizan un manómetro para verificar la presión que contiene cada cilindro del motor y al realizar esta operación se requieren desmontajes previos que en muchos vehículos el acceso al desmontaje es muy limitado e incómodo, generando mayor tiempo en la comprobación, que en muchos de los casos puede provocar daños en el motor o periferias.

El diseño de este proyecto dedicado a motores de combustión interna a gasolina, surge como una ayuda para el Ingeniero Mecánico Automotriz para el correcto diagnóstico del estado físico del tren alternativo del motor de manera rápida, sencilla y reduciendo las posibles daños en el motor, tan solo con establecer una conexión del

equipo de diagnóstico con unos sensores del vehículo, obteniendo de esta manera datos exactos y reduciendo el tiempo empleado en comprobación del tren alternativo.

2.3. CARACTERÍSTICAS DEL DISEÑO.

Debido a que la manipulación de este equipo será por Ingenieros Mecánicos Automotrices o personas con conocimientos sobre mecánica, los datos obtenidos serán en función a la presión de cada cilindro del motor Otto, es por esto que el equipo debe tener las siguientes características:

2.3.1. MOVILIDAD.

Para que se pueda disponer el equipo en cualquier momento, es decir, de fácil transportación en caso de que se requiera.

2.3.2. DEBE RESPONDER EFECTIVAMENTE Y NO OCUPAR MUCHA MEMORIA.

Ya que su respuesta debe ser rápida y el consumo de memoria en exceso provocan que el diagnóstico sea lento.

2.3.3. INTERFAZ INTUITIVA Y FÁCIL DE USAR.

Debe ser sencilla debido a que lo único que se requiere es conocer los porcentajes de eficiencia y presión de cada cilindro.

2.3.4. POSIBILIDAD DE AMPLIACIÓN DE FUNCIONES E INFORMACIÓN.

Con la finalidad de mejorar su funcionalidad corrigiendo errores e implementando más funciones.

2.4. DISEÑO INICIAL.

Inicialmente se optó por un diseño sencillo usando únicamente un display LCD con su respectivo controlador, para poder visualizar en esta pantalla el resultado de la programación, con gráficos en forma de barras en donde se indica el porcentaje de presión de cada cilindro.

Figura 2.1. Pantalla LCD 128x240.

Fuente: http://es.farnell.com.

Al observar que su interfaz gráfica es muy pobre se optó por una pantalla TFT con su respectivo micro controlador, puesto que esta pantalla cuenta con mucha más resolución gráfica y una gama extensa de colores haciendo mucho más vistoso el equipo de diagnóstico a más de tener una pantalla táctil lo que reduciría el número de elementos a colocar en nuestro equipo.

Figura 2.2. Pantalla TFT 2.8 plg.

Fuente: http://www.bricogeek.com.

Al hacer un cambio de pantalla se optó por realizar una interfaz gráfica adecuada para mostrar los resultados del análisis del estado de cada cilindro.

Para controlar la interfaz gráfica nos encontramos con el inconveniente que al programar en un determinado software de computadora, debíamos comprar los permisos o derechos necesarios para hacer o diseñar nuestro equipo de diagnóstico, este valor era muy alto, motivo por el cuál buscamos otras opciones y optamos por arduino y android que son OPEN SOURCE (software libre).

2.5. SOFTWARE LIBRE.

Son muchas las bondades que ofrece trabajar software libre, entre ellas podemos decir que los usuarios pueden ejecutar, editar o mejorar, el comportamiento del software a diferencia de un software propietario es necesario una licencia y no permite ver el código fuente para editarlo ni copiarlo sin restricciones, es por eso, que los creadores de software libre viendo todo el monopolio de empresas que solo piensan en un bien común se crearon los programas de código abierto, teniendo en cuenta que al no pagar derechos de licencia este dinero puede ser invertido en otras necesidades, como mejorar nuestras maquinas e incluso mejorar nuestra calidad de vida.

Existe una infinidad de programas útiles para la vida profesional, como para la enseñanza, que pueden ser usados por un ingeniero, economista, doctor, Etc.

Figura 2.3 Miscelánea de programas de software libre.

Fuente: http://freealts.com/

En el caso de la educación y particularmente en las universidades, el software libre no puede competir con un software propietario debido a los años de experiencia que tienen estos programas en el mercado, pero los docentes junto con los estudiantes pueden desarrollar programas parecidos e ir editando cada vez que este lo necesita para alguna otra actividad.

2.5.1. CUADRO COMPARATIVO.

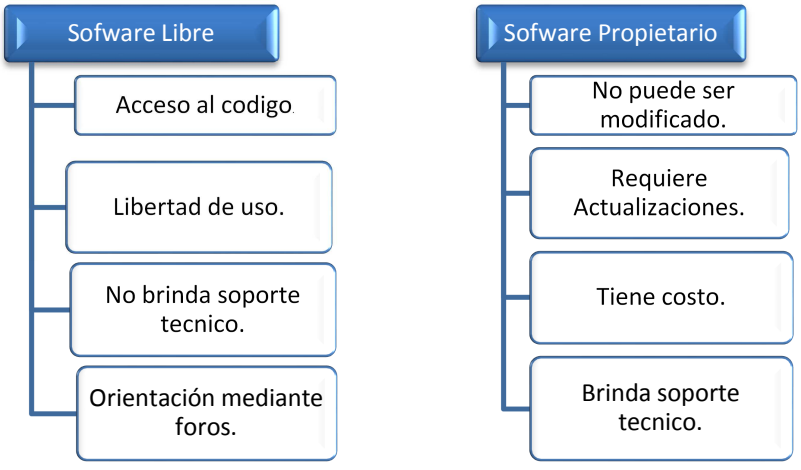


Figura 2.4 Cuadro comparativo.

Fuente: http://freealts.com/

Otro punto muy importante de manejar software libre es que los usuarios pueden compartir información con la comunidad internacional que aprovechan estos medios para el desarrollo y crecimiento mundial.

2.6. HARDWARE Y SOFTWARE ARDUINO.

Arduino es una plataforma de código abierto, basada en una sencilla placa con entradas y salidas, analógicas/ digitales, en un entorno de desarrollo que implementa el lenguaje de programación Processing muy similar al lenguaje C/C++. Está basado en un procesador Atmega, un chip sencillo y de bajo costo que permite el desarrollo de múltiples diseños.

2.6.1 HARDWARE ARDUINO.

Al ser Hardware libre tanto su diseño como su distribución, es decir, puede utilizarse para el desarrollo de cualquier tipo de proyecto sin haber adquirido ninguna licencia.

Este dispositivo nos permite conectarnos al mundo físico con el mundo virtual o al mundo analógico con el mundo digital, pudiendo ser utilizado para poner en marcha procesos simples como procesos complejos. Por ejemplo, encender un led, las luces de un teatro, construir instrumentos musicales, poner en marcha un robot entre otras funcionalidades que se le pueda dar al dispositivo.

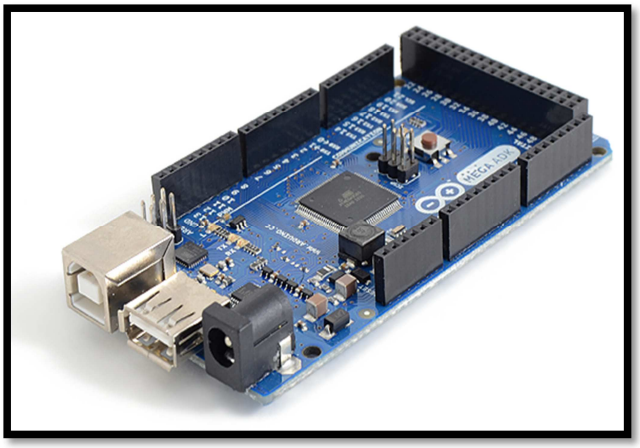


Figura 2.5. Arduino.

Fuente: http://www.arduino.cc/

La placa Arduino puede ser construida por nosotros o adquirirla en cualquier almacén electrónico, no tiene un manual específico, tenemos que acudir a foros o recopilaciones de datos que tienen algunos investigadores, el hardware de la tarjeta poseen un micro controlador AVR.

Dependiendo de las necesidades podremos elegir un tipo de tarjeta, debido a que cada placa contiene un micro controlador diferente, entre los cuales podemos citar a los más comunes, ATmega168PA, ATmega328P, ATmega164A, ATmega324P, ATmega644PA, ATmega1284P, ATmega2560, etc. Dependiendo de esto las placas Arduino toman diversos nombres, Arduino Uno, Arduino Leonardo, Arduino Mega, etc.

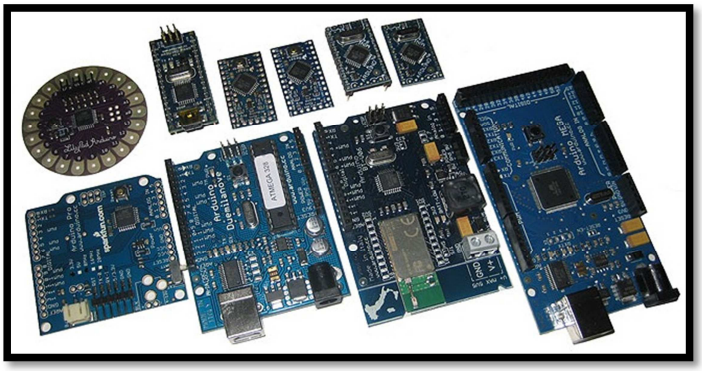


Figura 2.6 Placas de Arduino.

Fuente: http://www.robotshop.com

2.6.2. SOFTWARE ARDUINO.

Figura 2.7 Software Arduino.

Fuente: http://www.arduino.com

El Arduino además de ser una placa de hardware muy utilizada también nos presenta un software propio para la compilación de nuestros proyectos, conocido con el mismo nombre de Arduino, el cual está basado en el lenguaje de programación muy parecido a C / C++, al ser software libre podemos implementar librerías propias para recopilar líneas de código más cortas y tener espacio suficiente para implementar mejoras en nuestro diseño cuya respuesta ha de ser más rápida que cuando nos manejamos en líneas de código simultaneo o de corrido.

Muchos de los programas para revisar los resultados de los diseños de programación es necesario copilarlos y pasar a otro programa para analizar su funcionalidad, que dependerá mucho de la capacidad de nuestras computadoras para ejecutarlo, otra manera sería cargar el programa directamente al micro controlador, para lo cual no es muy eficiente a la hora de diseñar nuestros proyectos por el motivo del tiempo que toma en construir la placa con los elementos de soporte para el micro controlador además tendremos que editarlos si se cambia el código.

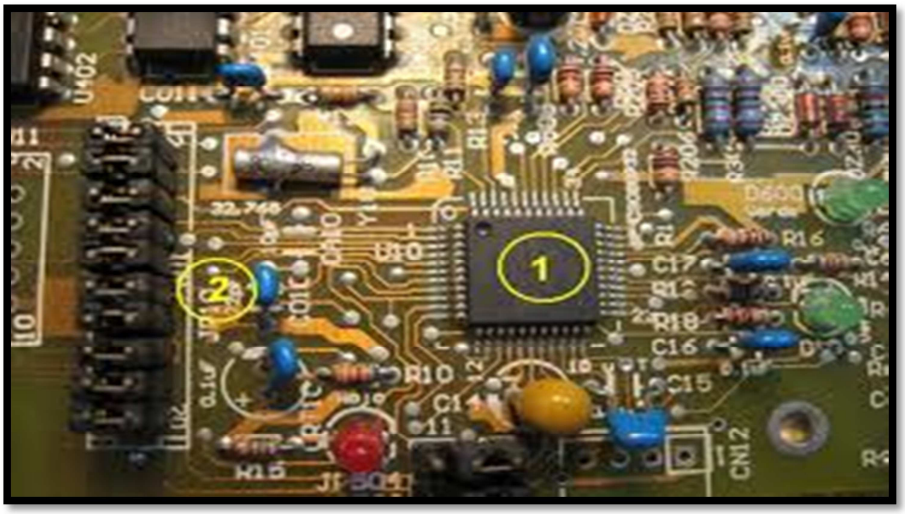


Figura 2.8 Elementos de soporte de un Micro controlador.

Fuente: http://es.m.wikipedia.org

Arduino tiene una gran ventaja ante estas problemáticas pues puede ser compilado e interpretado en tiempo real, es decir, cuando se construye el código fuente este se transforma en una especie de código de máquina y podremos analizar cualquier proyecto creado en este software mediante el monitor serial.

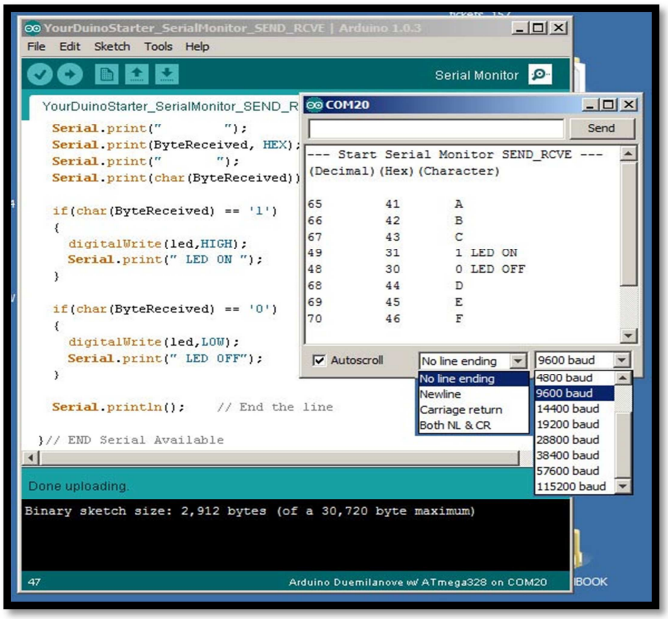


Figura 2.9 Monitor Serial Arduino.

Fuente: http://arduino-info.wikispaces.com

El programador de Arduino es una herramienta muy versátil que proporciona una gran ventaja, este puede comunicarse con otros programas o programar desde estos como es el caso de Processing, Eclipse, Matlab, Simulink, etc. Tomando en cuenta todo esto podemos decir que arduino ha sido diseñado para personas que no tengan conocimiento alguno en lenguajes de programación o aquellos que manejan otros tipos de software.

2.7. TARJETA ARDUINO PARA IMPLEMENTAR NUESTRO DISPOSITIVO.

Para implementar un dispositivo de adquisición de datos la tarjeta debe tener mínimo dos interrupciones externas para realizar un contador de eventos, para lo cual la tarjeta arduino uno cuenta con dos interrupciones y la tarjeta arduino mega con cuatro interrupciones, es por esto que optamos por la tarjeta arduino mega por tener mayor cantidad de pines para interrupciones.

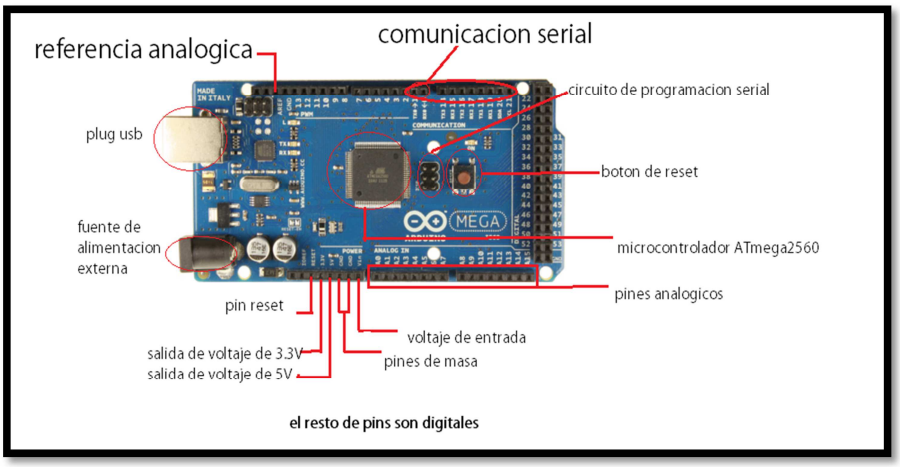


Figura 2.10 Monitor Serial Arduino.

Fuente: Los Autores

2.7.1. CARACTERÍSTICAS DE LA TARJETA ARDUINO MEGA.[1]

2.7.1.1. PIN VIN.

Es la tensión de entrada de la tarjeta arduino cuando se utiliza una fuente de alimentación externa (en lugar de una conexión USB de 5V u otra fuente de alimentación regulada).

2.7.1.2. PIN 5V.

Donde se tiene un voltaje regulado de 5V en toda la tarjeta. Esta puede ser alimentada a través de la toma de alimentación de corriente continua CC, conexión USB (5V), a través del pin VIN de la tarjeta. Un suministro directo a los pines de 5V o 3.3V podría dañar a la placa puesto que por estos pines no se regula la tensión, por lo tanto, no se recomienda realizar esta acción.

2.7.1.3.PIN 3.3V.

Aquí se tienen una tensión de 3,3 voltios, que está integrado en la placa proporcionada por el regulador de tensión. La corriente máxima de salida es de 50 mA.

2.7.1.4. MEMORIA DE LA PLACA.

El Atmega2560 tiene 256 KB de memoria flash (de los cuales 8 KB ocupados por el bootloader de Arduino). También dispone de 8 KB de SRAM y 4 KB de EEPROM, que con la librería EEPROM puede ser leído y escrito).

Fuente: *www.arduino.cc*

2.7.1.5. PINES DE ENTRADA Y SALIDA.

Cada uno de los 54 pines digitales del Arduino se puede utilizar como entrada o salida. Para esto, las funciones: pinMode (), digitalWrite () y digitalRead () están disponibles, operan con un voltaje de 5 voltios. Cada pin puede proporcionar o recibir una corriente máxima de 40 mA y tiene una resistencia pull-up de 20 a 50 ohms, que 'por defecto' no está conectado. Además, hay pasadores para funciones específicas:

La placa Mega 2560 tiene 16 entradas analógicas, con el nombre de A0 a A15 y cada uno tiene una resolución de 10 bits (es decir, 1.024 pasos). Por defecto se miden desde 0 a 5 voltios.

2.7.1.6. PINES SERIAL.

Pin **0 (RX)** y pin **1 (TX)** estos pines con datos de serie TTL recibir (RX) o transmitida (TX). Estos pasadores están conectados a las clavijas correspondientes de la ATMEGA16U2 conectado chip de serie de USB a TTL.

2.7.1.7. PINES DE INTERRUPCIONES EXTERNAS.

Siendo los pines: **2 (interrupción 0), 3 (alarma 1), 18 (alarma 5), 19 (alarma 4), 20 (interrupción 3) y 21 (alarma 2)** Estos pines se pueden configurar para que cuando un valor bajo, un ascendente o descendente, dispare una interrupción.

2.8. ANDROID.

Puesto que la pantalla a utilizar es una de tipo TFT, que son las mismas que utilizan en equipos móviles, optamos por generar una aplicación en un sistema operativo para que pueda funcionar en un teléfono celular con sistema androide, para la transferencia de datos entre la plataforma arduino y androide usamos un módulo bluetooth.

De esta forma tenemos una buena resolución gráfica y con una pantalla mucho más grande a la inicial pudiendo apreciar mejor las gráficas.

Androide es un sistema operativo basado en LINUX, que esta diseñado para dispositivos móviles con pantalla táctil o conocidos en el mercado como SMARTPHONE, además de ser un sistema operativo de código abierto no se tendría inconvenientes al momento de realizar una aplicación para este tipo de plataforma.

Figura 2.11. Sistema operativo ANDROID

Fuente: http://www.android.com/.

2.9. DISEÑO DE LA INTERFAZ.

La interfaz gráfica, parte fundamental para el usuario, debe tener colores y contrastes adecuados, facilitando su lectura y visualización. Se planteó realizar la interfaz gráfica con varias pantallas o conocidas en el mundo de la programación java como actividades las mismas que presentan a continuación.

2.9.1. PANTALLA DE BIENVENIDA.

Este es el diseño para la pantalla inicial o de bienvenida, que se presenta al inicio de la aplicación mientras se carga toda la documentación del programa.

Figura 2.12. Pantalla Splash.

Fuente: Los Autores.

2.9.2. PANTALLA INICIAL.

Este es el diseño para la pantalla inicial, en donde se puede observar el nombre de la aplicación y un botón con el nombre de *INICIAR*, el cual nos ayuda para cambiar a otra actividad y proceder con el análisis del motor.

Figura 2.13. Pantalla Inicial.

Fuente: Los Autores.

2.9.3. PANTALLA DE CONEXIÓN.

Una vez pulsado sobre el botón *INICIAR* de la actividad anterior se presenta el segundo *activity*, con una lista de dispositivos bluetooth vinculados con el teléfono, donde el usuario podrá escoger a que dispositivo establecer conexión, cuenta también con un botón llamado *BUSCAR DISPOSITIVOS*, con el fin de ayudar al usuario a vincular al celular un nuevo dispositivo bluetooth, para luego establecer la respectiva conexión y continuar con el análisis.

Figura 2.14. Pantalla de conexión.

Fuente: Los Autores.

2.9.4. PANTALLA OBSERVACIONES.

Una vez que el usuario designe un dispositivo para la conexión y esta se realice con éxito el programa mostrará un tercer *activity* con algunas observaciones que deberá tener en cuenta el usuario antes de proceder con el análisis, aparecerá también un botón con el nombre *ANÁLISIS* que desplegará un subconjunto de tres botones para que el usuario escoja entre uno de ellos.

Figura 2.15. Pantalla Observaciones.

Fuente: Los Autores.

2.9.5. PANTALLA DE INDICACIONES PARA EL ANÁLISIS ESTÁTICO.

Después de que el usuario lea las observaciones procederá a seleccionar uno de los tres botones del análisis, en caso de que su selección sea el botón *ESTÁTICO* se desplegará una cuarta actividad donde se podrá observar una serie de indicaciones previas al análisis, así como un par de botones, uno con la imagen de una lupa que realizará el mismo trabajo del botón *ANÁLISIS* explicado anteriormente, y un botón con el nombre *OK*, este último se encargará de aceptar las indicaciones y mostrar una nueva actividad.

Figura 2.16. Pantalla Indicaciones Análisis Estático.

Fuente: Los Autores

2.9.6. PANTALLA ANÁLISIS ESTÁTICO.

Seguido de las indicaciones tenemos la pantalla del análisis que hace referencia a una quinta actividad donde se encuentra los cuatro gráficos en forma de cilindros de motor, junto con un valor porcentual para cada cilindro del motor, así como también un valor entero aproximado de su compresión en unidades de PSI, y también un botón con imagen de lupa para poder continuar con el siguiente análisis.

En este análisis estático lo que se pretende demostrar es el estado del tren alternativo del motor, mediante los porcentajes que indican la eficiencia que posee cada cilindro, siendo un valor de 90 - 100% un estado muy bueno, un valor de 75 – 89% un estado medio permisible, y por último un valor de 0 – 74 % un estado totalmente malo que indica la reparación inmediata del motor.

Así como un valor entero en PSI que indica la compresión que tiene cada cilindro, de esta manera poder también diagnosticar según su compresión interna y tener una mejor visualización de los datos.

Figura 2.17. Pantalla Análisis Estático.

Fuente: Los Autores

2.9.7. PANTALLA DE INDICACIONES DEL ANÁLISIS DINÁMICO.

En caso que el usuario seleccione el botón *DINÁMICO*, se desplegará una sexta actividad donde se podrá observar una serie de indicaciones previas al análisis, así como un par de botones, uno con la imagen de una lupa que realizará el mismo trabajo del botón *ANÁLISIS* explicado anteriormente, y un botón con el nombre *OK*, este último se encargará de aceptar las indicaciones y mostrar una nueva actividad.

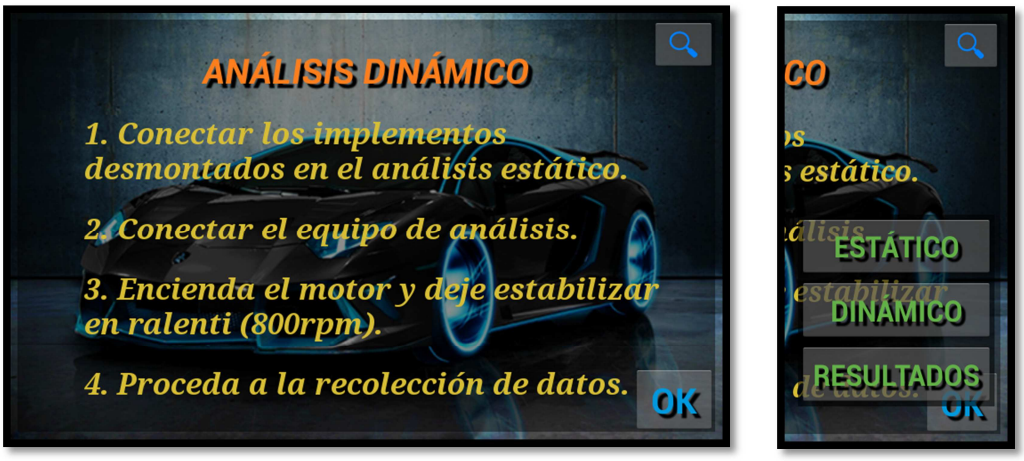


Figura 2.18. Pantalla Indicaciones Análisis Dinámico.

Fuente: Los Autores.

2.9.8. PANTALLA ANÁLISIS DINÁMICO.

Seguido de las indicaciones tenemos la pantalla del análisis correspondiente a una séptima actividad en donde se encuentra unos gráficos en forma de cilindros de motor, en total cuatro gráficos que indican un valor porcentual para cada cilindro del motor, así como también un valor entero aproximado de su compresión en unidades de PSI, además de un botón con imagen de lupa para poder continuar con el siguiente análisis.

En este análisis dinámico lo que se pretende demostrar es un valor porcentual de trabajo de cada cilindro del motor, puesto que al ser cuatro cilindros el trabajo total en función de porcentaje es de 100% siendo para cada cilindro un valor de 25%, este

valor indica cuanto aporta cada cilindro, de tal manera se puede diagnosticar que cilindro no aporta el trabajo necesario. Siendo un valor aproximado de 23 - 25 % un aporte adecuado, un valor de 20 - 22% un aporte medio que ya requiere una inspección y un valor de 0 – 19% un aporte demasiado bajo por lo que se deberá tomar las medidas necesarias para solucionar este problema.

Así como un valor entero en PSI que indica el valor de compresión que tiene cada cilindro, siendo de gran ayuda para tomar medidas y solucionar el problema.

Figura 2.19. Pantalla Análisis Dinámico.

Fuente: Los Autores.

2.9.9. PANTALLA DE RESULTADOS.

En caso que el usuario seleccione el botón *RESULTADOS*, se desplegará una octava actividad donde se podrá observar los resultados del análisis de cada cilindro, así como se podrá generar un archivo de texto con los valores del análisis para su posterior envió por correo o únicamente para imprimir y tener un respaldo del análisis.

Figura 2.20. Pantalla Resultados.

Fuente: Los Autores.

2.10. COMUNICACIÓN ENTRE ARDUINO Y ANDROID.

Una vez ya de acuerdo con el hardware y software a usar se requiere de una comunicación necesaria para la transferencia de datos, desde arduino hacia android, esta comunicación se la puede realizar por varios métodos:

2.10.1. A TRAVÉS DE COMUNICACIÓN WIRELESS.

Este módulo wireless que se adapta a la plataforma Arduino permite que dicha plataforma pueda conectarse a internet a través de la red inalámbrica Wi-Fi 802.11, de tal manera que con la ayuda de este módulo, poder transmitir los datos hacia el teléfono android conectado previamente a la misma red inalámbrica Wi-Fi a la que está conectado la plataforma arduino, debido a que la transferencia de datos que se van a realizar son varios, este módulo sería el adecuado para su uso en este proyecto debido a que este permite enviar y recibir varios datos a la vez.

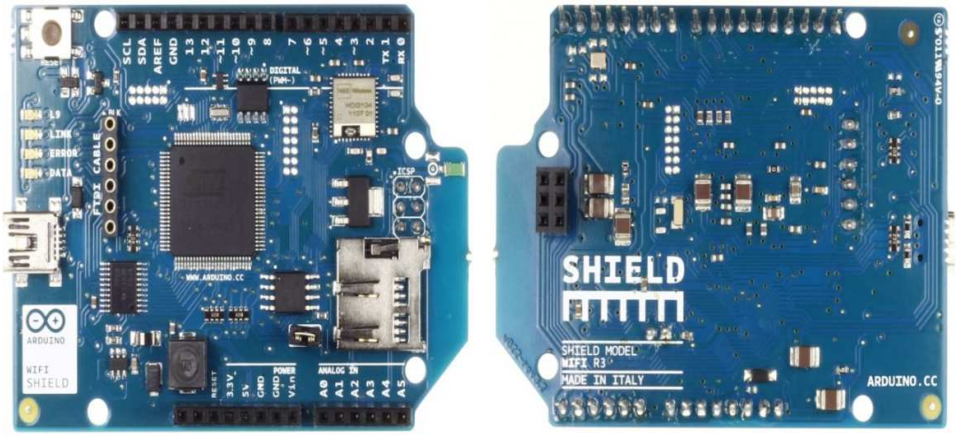


Figura 2.21. Módulo Wireless para arduino.

Fuente: http://www.arduino.cc.

2.10.2. A TRAVÉS DE COMUNICACIÓN CON CABLE DE DATOS.

Este método hace que la conexión entre la plataforma arduino y el teléfono con sistema android se realice a través de un cable, el cual hace de canal para la transmisión de datos.

Esta es una buena opción al momento de trabajar con una plataforma arduino capaz de reconocer la programación gracias a que este arduino posee de una interfaz de host USB para conectarse con los teléfonos android.

Este método quedó descartado ya que la conexión a través de un cable de datos hace que el equipo sea menos versátil pues lo que se requiere es tener la cantidad mínima de cables.

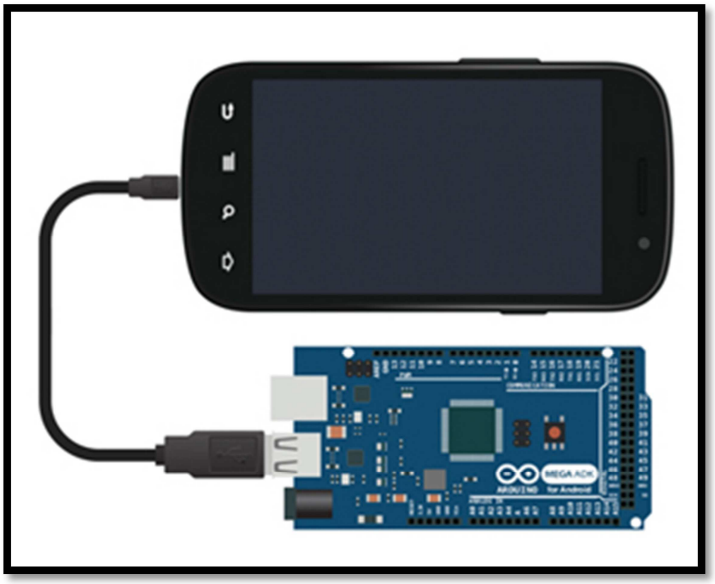


Figura 2.22. Conexión por cable de datos.

Fuente: http://www.arduino.cc

2.10.3. A TRAVÉS DE COMUNICACIÓN BLUETOOTH.

A más de las anteriores también se pueden transmitir los datos desde la plataforma arduino hacia el teléfono con sistema android mediante un módulo bluetooth, con la diferencia al módulo wireless, este módulo bluetooth puede transmitir solamente un dato a la vez, por tal motivo se debe empaquetar los datos para enviarlos por bluetooth y posterior a ello desempaquetar en el teléfono android y así poder visualizar los datos.

Se escogió la opción de usar un módulo bluetooth en lugar de los anteriores puesto que es más económico y de fácil manejo, solo que se deben tener precauciones al momento de alimentar el módulo, ya que su voltaje de trabajo esta por los 3.3 voltios, una de las ventajas de usar la plataforma arduino que incorpora una salida con el voltaje mencionado, además se puede configurar la velocidad de transferencia de los datos que por lo general esta a un valor estandar de 9600 baud.

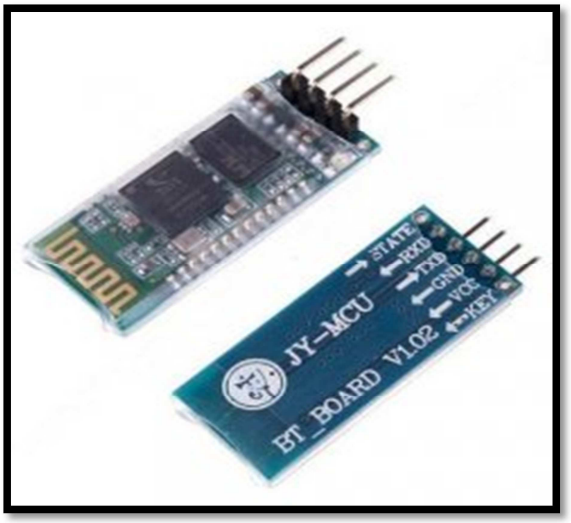


Figura 2.23. Módulo Bluetooth.

Fuente: http://blog.make-a-tronik.com/.

2.11. DISEÑO FINAL DEL EQUIPO.

Finalmente se tendría el diseño completo del equipo el cual constaría de dos partes, la primera lo que sería una caja por así llamarlo con unas conexiones para cables tipo osciloscopio que serán los que se instalarán en los sensores del vehículo, así como una conexión para una manguera, la segunda parte sería el teléfono inteligente con sistema operativo android.

Figura 2.24. Caja circuitos electrónicos parte 1 del equipo.

Fuente: Los Autores.

Figura 2.25. Celular inteligente con sistema android parte 2 del equipo.

Fuente: Los Autores.

Figura 2.26. Cables de conexión para los sensores.

Fuente: Los Autores.

CAPÍTULO III

CONSTRUCCIÓN DEL EQUIPO DE DIAGNÓSTICO PARA MOTORES CICLO OTTO.

CAPÍTULO III

3. CONSTRUCCIÓN DEL EQUIPO DE DIAGNÓSTICO PARA MOTORES CICLO OTTO.

3.1. INTRODUCCIÓN.

La construcción del equipo de diagnóstico referente al diseño ya presentado en el capítulo anterior, hace que sea necesario utilizar dos programas de computadora con el fin de realizar la programación pertinente para la obtención de los resultados requeridos, uno para android y otro para arduino.

3.2. CONSTRUYENDO LA APLICACIÓN ANDROID.

Para realizar la aplicación se hace necesario utilizar un programa para computador denominado eclipse que es un entorno de desarrollo para aplicaciones para el sistema operativo android.

Figura 3.1.Entorno de desarrollo.

Fuente: http://encuentroscontux.blogspot.com

Este entorno de desarrollo se ejecuta en la computadora, dicho entorno consta de una pantalla principal mostrada a continuación:

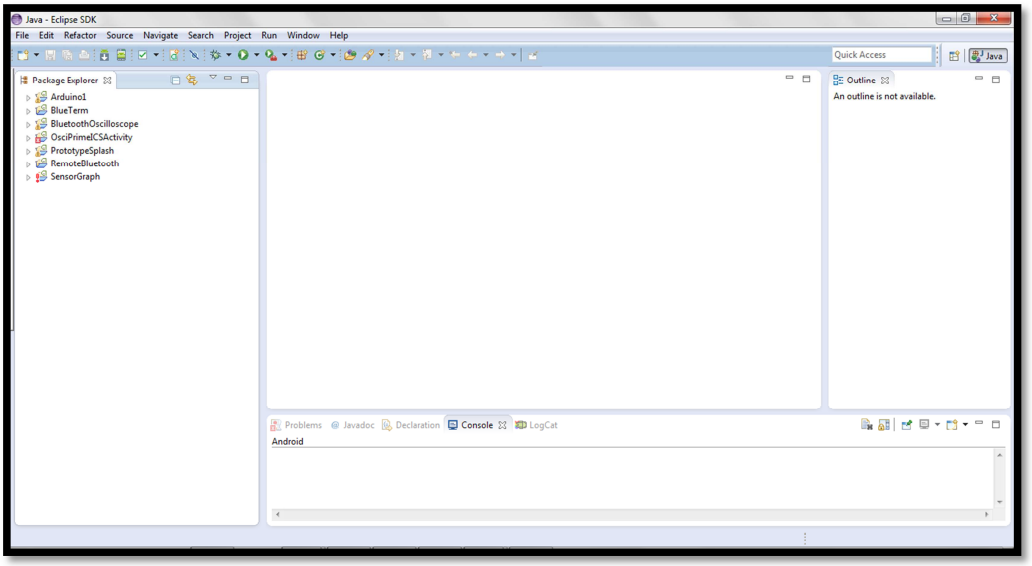


Figura 3.2. Ventana Entorno de desarrollo.

Fuente: Los Autores.

3.3. ENTORNO DE DESARROLLO.

Como ya se indicó en la figura 3.2, se pueden apreciar la ventana completa de la interfaz de usuario de Eclipse, tal como se puede observar esta ventana consta de dos tipos de elementos los que son las vistas y los editores.

Donde los editores permiten realizar una tarea en general mientras que las vistas proporcionan funciones de apoyo para el desarrollador, la función de cada una de las vistas disponibles en esta interfaz lo detallaremos a continuación, así como el botón necesario para lanzar a correr la aplicación desarrollada.

3.3.1. VISTA PACKAGE EXPLORER.

La vista del Package Explorer (Explorador de paquetes) muestra la estructura de paquetes y clases java almacenados en cada uno de los proyectos que el usuario este construyendo, donde cada paquete o llamado también carpetas fuente se las puede distinguir entre paquetes debido a su icono en forma de fichero contenido, donde sí se expande cada paquete se puede observar su contenido, ya sea para modificar o añadir líneas de programación de la aplicación que se está realizando.

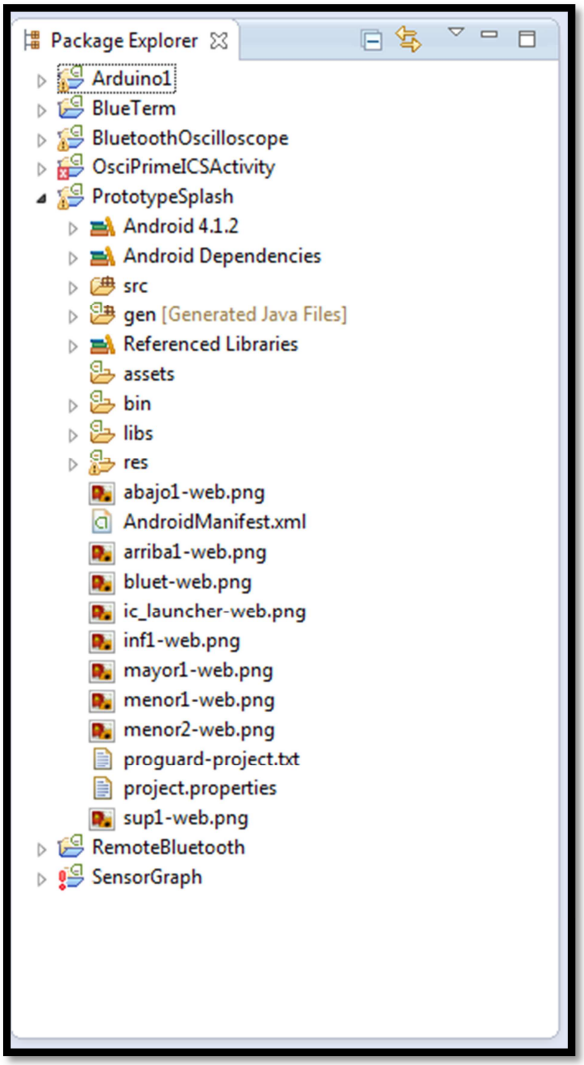


Figura 3.3. Vista Package.

Fuente: Capturas de pantalla.

3.3.2. VISTA OUTLINE.

Esta vista es de gran ayuda para poder ver los métodos y atributos que se van generando al desarrollar una aplicación en JAVA, cada icono que contiene esta vista facilitan información extra respecto a la visibilidad del atributo en general, y tal solo con dar encima un clic en uno de los iconos presentes en esta vista nos llevará directamente a la línea de código generada para la misma, en fin es muy útil en caso de que se desee navegar a través de archivos JAVA con extensas líneas de programación.

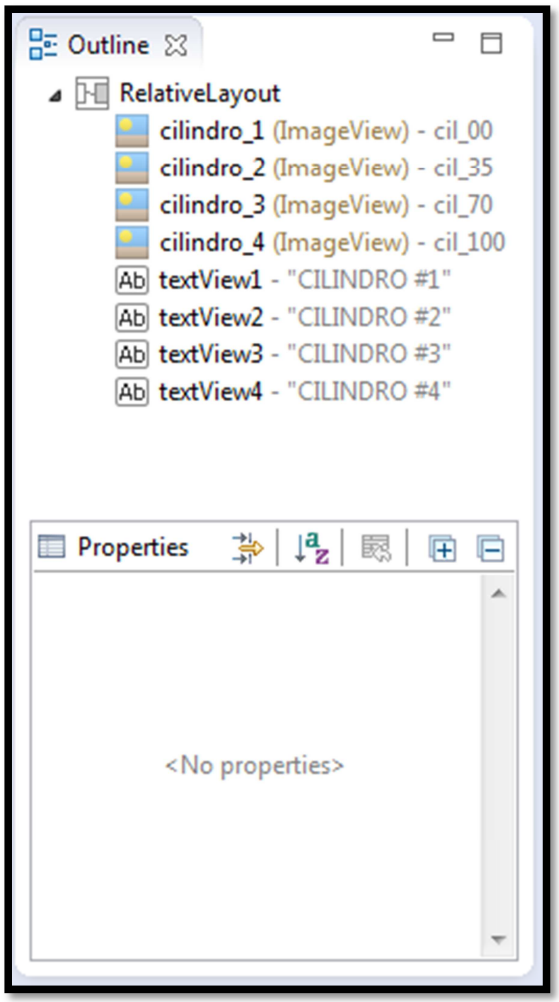


Figura 3.4. Vista Outline.

Fuente: Los Autores.

3.3.3. VISTA CONSOLE, PROBLEMS.

Esta vista es de gran ayuda al momento de lanzar o enviar a correr la aplicación puesto que en caso de algún error encontrado en la aplicación, en una de estas pestañas se genera las líneas referentes al error, lo cual es de mucha ayuda para poder corregir los errores y de esta manera tener una aplicación muy funcional.

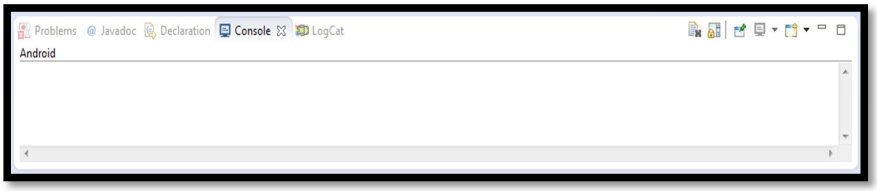


Figura 3.5. Vista Console, problems.

Fuente: Los Autores.

3.3.4. VISTA ÁREA DE TRABAJO.

Esta vista es en donde más va a estar involucrado el desarrollador, puesto que, es en ella donde se ingresa todas las líneas de programación, pudiendo almacenar algunas pestañas a la vez, de esta forma ayudando al desarrollador a tener una organización del trabajo que está realizando y evitar así alguna confusión.

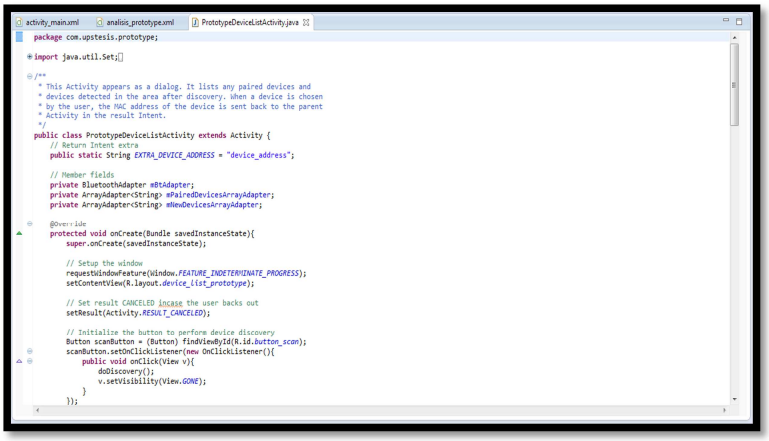


Figura 3.6. Vista Área de trabajo.

Fuente: Los Autores.

3.3.5. BOTÓN PARA CORRER LA APLICACIÓN.

Y por último tenemos a este botón muy importante para el desarrollador, el cual nos permite verificar como está quedando la aplicación, y en caso de ser necesario dar algunos ajustes en la programación para poder visualizarla de la manera que se desea.

Figura 3.7. Botón Run.

Fuente: Los Autores.

Al presionar este botón por primera vez nos despliega una segunda vista, la cual es muy necesaria para poder configurar que aplicación o proyecto correr y también para escoger en que dispositivo deseamos correr la aplicación.

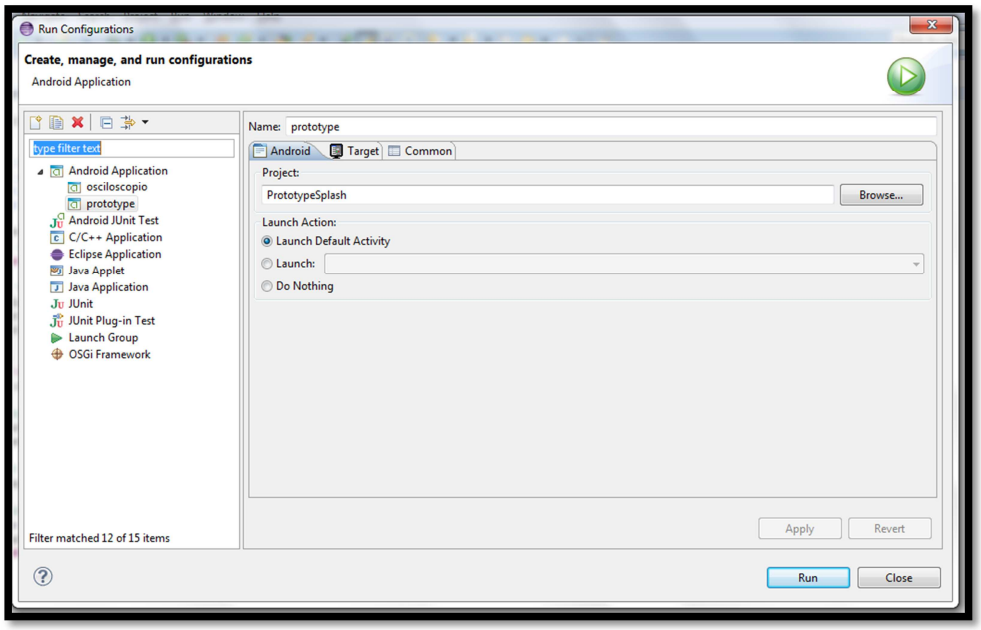


Figura 3.8. Configuración Botón Run.

Fuente: Los Autores.

En la pantalla anterior escogemos el proyecto a simular, mientras que en la siguiente figura seleccionamos en que dispositivo correr la aplicación pudiendo ser en un emulador o directamente en el mismo celular.

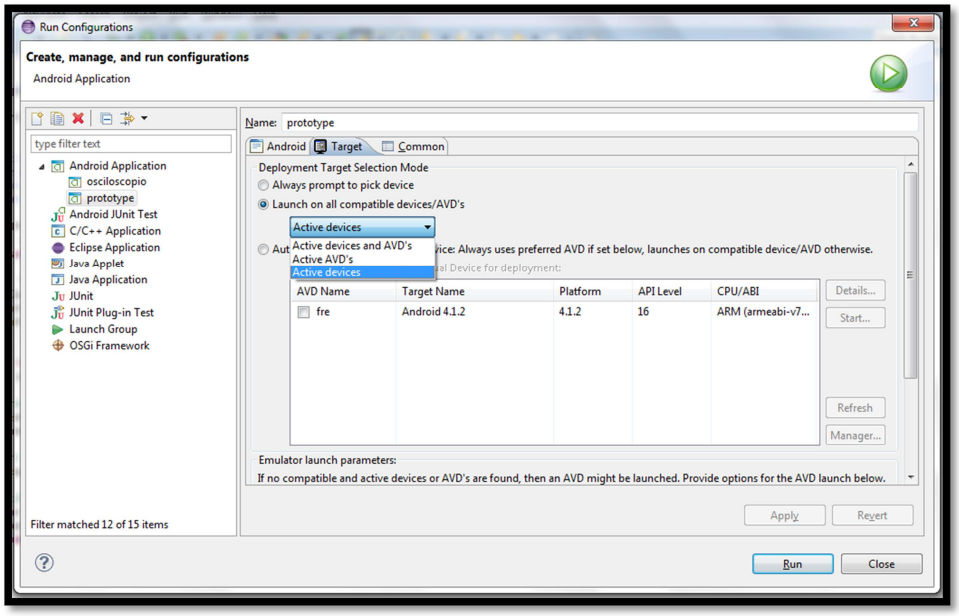


Figura 3.9. Configuración Botón Run.

Fuente: Los Autores.

Por motivos de funcionamiento de nuestra aplicación el emulador disponible no es de gran ayuda para verificar si está funcionando correctamente debido a que este emulador no posee de un módulo bluetooth para verificar nuestra aplicación, de tal manera que tuvimos que seleccionar la opción para correr directamente en nuestro celular con sistema operativo android.

Figura 3.10. Emulador del entorno de desarrollo.

Fuente: Capturas de pantalla.

3.4. CONSTRUYENDO LA APLICACIÓN PARA ANDROID.

Puesto que el diseño ya se explicó en el capítulo anterior, ahora vamos a detallar como se va a realizar cada una de las pantallas o actividades que posee la aplicación por lo cual empezaremos por las siguientes:

3.4.1. CONSTRUYENDO LA PANTALLA DE BIENVENIDA SPLASH.

Esta pantalla va a presentarse al momento que se lanza a correr la aplicación, por lo cual nos hemos preocupado que dé una buena impresión al usuario, no muy vistoso pero tampoco muy sencilla en gráficos y colores.

Para empezar explicamos que la pantalla de bienvenida se mostrará durante un determinado intervalo de tiempo, con el fin de que la aplicación se cargue por completo, cuyo tiempo de inicialización dependerá del dispositivo android en el cual se encuentre instalado.

Ahora procedemos a indicar como se construyó esta pantalla splash:

Figura 3.11. Pantalla Splash.

Fuente: Los Autores.

La pantalla se divide en dos partes, la primera es el círculo azul que gira durante la carga previa de la aplicación.

Al crear un nuevo proyecto en el entorno de desarrollo se generan un paquete con varias subcarpetas que contienen archivos necesarios para la programación, entre uno de ellos está la carpeta Layout:

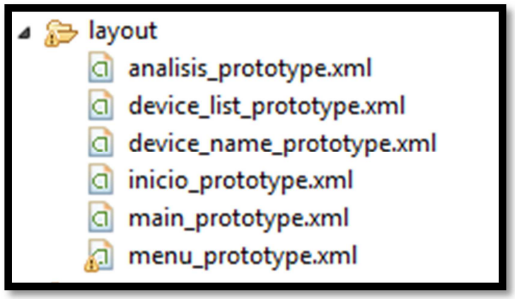


Figura 3.12. Carpeta Layout.

Fuente: Los Autores.

Esta carpeta contiene los ficheros o archivos con terminación .xml, de las diferentes pantallas de la interfaz gráfica, estos archivos se pueden programar de dos maneras: Una de forma visual donde se puede escoger los botones, cuadros de imágenes según se desee tal como se aprecia a continuación:

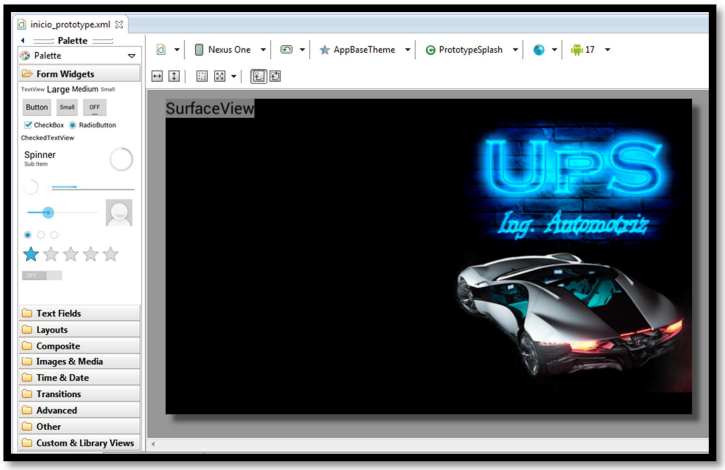


Figura 3.13. Programación de forma visual.

Fuente: Los Autores.

Y otra es mediante líneas de código donde únicamente se escribe los códigos referentes al tipo de objeto que desee el desarrollador colocar tal como botones, cuadro de imágenes, etc.

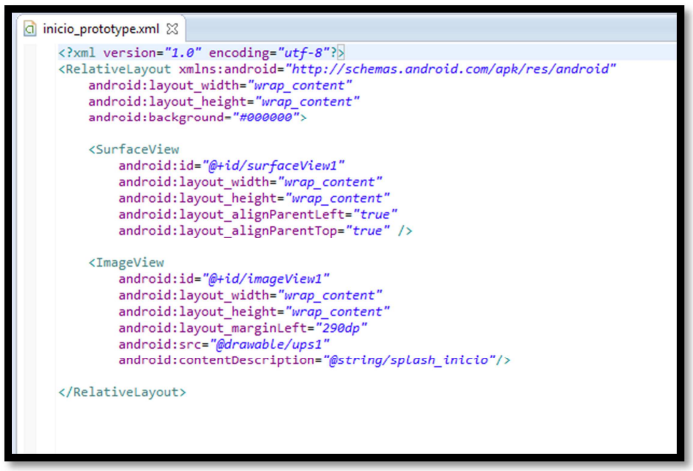


Figura 3.14. Programación mediante códigos.

Fuente: Los Autores.

Cualquiera de las dos formas son válidas y elegidas según lo requiera el desarrollador. Con esto vamos a proceder a indicar como se realizó la primera pantalla.

Como ya se mencionó esta pantalla está dividido en dos partes, la primera consta de un objeto conocido como ***Surface view*** que nos sirve para proporcionar un modo más directo de mostrar imágenes en forma de animación, para lo cual en un archivo .xml colocamos la siguiente línea de programación para usar esta clase llamada surfaceview

```
<SurfaceView
        android:id="@+id/surfaceView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true" />
```

Cada línea de programación tiene su función que detallaremos a continuación:

```
        android:id="@+id/surfaceView1"
```

Esta línea hace que la clase surfaceView pueda ser cambiada de nombre por lo que respecta el nombre seleccionado es el mismo únicamente añadido un número 1.

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

Estas dos líneas van de la mano, puesto que definen el tamaño de la clase surfaceView donde *width* es la anchura y *heigth* es la altura, al momento que colocamos wrap_content estamos condicionando a la clase a tomar el tamaño de la imagen a renderizar.

```
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
```

Estas dos líneas son de ayuda para alinear a esta clase en la pantalla según la posición deseada. Mientras que la segunda parte de la pantalla Splash consta de un cuadro de imagen llamado ImageView cuyas líneas de programación son:

```
<ImageView
        android:id="@+id/imageView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="290dp"
        android:src="@drawable/ups1"
        android:contentDescription="@string/splash_inicio"/>
```

Lo nuevo de estas líneas de programación es la línea `android:src="@drawable/ups1"` la misma que nos ayuda a encontrar una imagen determinada en la carpeta drawable para mostrarle en la clases Image view, es en esta carpeta donde se guardan todo tipo de imágenes que se van a utilizar en la aplicación.

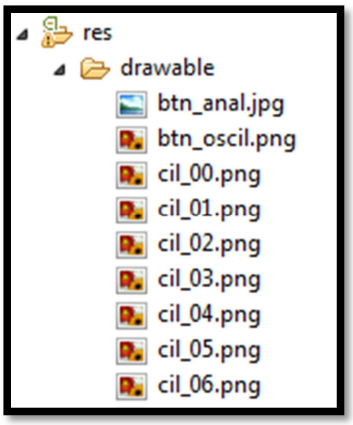


Figura 3.15. Carpeta Drawable.

Fuente: Los Autores.

Todas las líneas de programación anteriores son únicamente para crear una vista de la pantalla que se quiere mostrar en el celular, para proceder a generar el movimiento para la pantalla de inicio de la aplicación, necesitamos crear un archivo *.java*, el cual se almacena en otra carpeta con el nombre *SRC*.

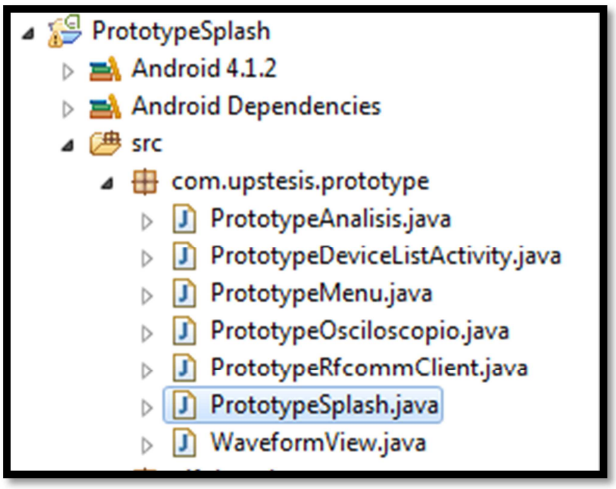


Figura 3.16. Carpeta SRC.

Fuente: Los Autores.

En esta carpeta es donde se guardan los archivos JAVA, este archivo contiene la programación que se ejecutará en el dispositivo android.

Como ya tenemos realizado el diseño en un archivo *.xml*, lo hacemos funcionar a través de un archivo JAVA, mediante el uso de las siguientes líneas que se encargarán de mostrar la pantalla de inicio en el celular:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView (R.layout.inicio_prototype);
```

Una vez mostrada la pantalla de inicio se debe cerrar la misma para posterior a ella mostrar otra pantalla, para ello utilizamos la siguiente línea:

```
Bundle extras = getIntent().getExtras();
if (extras !=null){
      if (extras.getBoolean("CERRAR", false)){finish();
      return;
      }
```

Debido a que queremos mostrar la pantalla inicial con movimiento hasta que se cargue la aplicación, usamos la clase *surfaceview* junto con las siguientes líneas de programación:

```
SurfaceView v = (SurfaceView)
findViewById(R.id.surfaceView1);
GifRun w = new GifRun();
w.LoadGiff(v, this, R.drawable.intro);
```

Para mostrar la pantalla de inicio durante la carga de la aplicación hemos estimado un tiempo con las siguientes líneas estipulando unos 10 segundos.

```
Thread Timer = new Thread(){
      public void run(){
      try {
            sleep(10000);
            startActivity (new Intent("prototype.tesis.fin.SPLASH")); }
            catch (InterruptedException e) {
            e.printStackTrace();      }
            finally {finish ();}      }
      };
      Timer.start();      }
      }
```

Y finalmente tenemos ya corriendo en el celular la primera pantalla llamada SPLASH, pero al estar manipulando en el celular notamos que al dejar de usarlo, este se colocaba en modo sleep, apagando la pantalla del celular y con el problema que al

volver a activarlo se debía reiniciar la aplicación, por este motivo se ingresaron nuevas líneas de programación las mismas que se muestran a continuación.

En primer lugar se debe tener permiso del celular para poder usar esta opción para lo cual necesitamos de la siguiente línea dentro de la clase JAVA:

```
protected PowerManager.WakeLock mWakeLock;
@SuppressLint("Wakelock")
```

Y para que se ejecute el permiso en el celular necesitamos de las siguientes líneas de programación dentro del método *OnCreate:*

```
PowerManager pM (PowerManager)
      getSystemService(Context.POWER_SERVICE);
this.mWakeLock = pM.newWakeLock
      (PowerManager.FULL_WAKE_LOCK, "My Tag");
this.mWakeLock.acquire();
```

De esta forma el celular no ingresará a modo *SLEEP* hasta que el usuario cierre o finalice la aplicación.

Una vez cargada la aplicación en el celular necesitamos de otra pantalla para continuar con la construcción de la aplicación, por lo que pasamos a la segunda pantalla nombrada Pantalla Inicial.

3.4.2. CONSTRUYENDO LA PANTALLA INICIAL.

Esta pantalla como su nombre lo indica es la inicial de toda la aplicación donde el usuario selecciona iniciar todo el proceso de análisis, indicamos el diseño del el archivo.xml, tal como se aprecia en la gráfica:

Figura 3.17. Pantalla menú.

Fuente: Los Autores.

Y para ello hemos utilizado las siguientes líneas de programación, donde cabe destacar que usamos un *RelativeLayout* puesto que este nos permite especificar la posición de cada elemento de manera relativa.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/inicio" >

      <Button
          android:id="@+id/btnConectar"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:layout_centerHorizontal="true"
          android:layout_marginLeft="25dip"
          android:layout_marginTop="260dip"
          android:onClick="Conectarse"
          android:shadowColor="@color/sombra"
```

```
        android:shadowDx="5"
        android:shadowDy="5"
        android:shadowRadius="1.5"
        android:text="@string/btnConectar"
        android:textColor="@color/azul"
        android:textSize="12pt"
        android:textStyle="bold" />

</RelativeLayout>
```

Como se puede apreciar en las líneas de programación, contamos con un nuevo elemento que es un botón el cual lo podemos observar en la pantalla inicial con el nombre de *INICIAR*, lo importante de este botón es que gracias al método *onClick* podemos cambiar entre actividades o pantallas.

```
android:onClick="Conectarse"
public void Conectarse (View view) {
      Intent analisis = new Intent(this, ActivityPrototype.class );
      startActivity(analisis);
}
```

Estas líneas de programación nos ayuda a cargar o abrir la pantalla de conexión que hace referencia a la segunda actividad.

3.4.3. CONSTRUYENDO LA PANTALLA CONEXIÓN.

La segunda actividad o pantalla de conexión usamos para conectarnos a un dispositivo bluetooth requiere de una clase extra que le nombramos DeviceListActivityPrototype, esta clase posee los parámetros necesarios para poder establecer conexión con un dispositivo bluetooth.

Figura 3.18. Pantalla de conexión.

Fuente: Los Autores.

Para poder visualizar esta pantalla se utilizaron las siguientes líneas de programación, comenzando por la clase DeviceListActivityPrototype, en cuyo layout se utilizaron 3 elementos:

TextView: Para colocar dentro de este los títulos del bloque que muestra los dispositivos vinculados y los disponibles para vincular al celular, para ello disponemos de las siguientes líneas:

```
<TextView
        android:id="@+id/title_paired_devices"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/title_paired_devices"
        android:visibility="gone"
        android:background="#666"
        android:textColor="#fff"
        android:paddingLeft="5dp"/>
```

```
<TextView
        android:id="@+id/title_new_devices"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/title_other_devices"
        android:visibility="gone"
        android:background="#666"
        android:textColor="#fff"
        android:paddingLeft="5dp"/>
```

ListView: este objeto lo usamos para que se genere una lista con todos los dispositivos vinculados al celular, y los disponibles para vincular.

```
<ListView
        android:id="@+id/paired_devices"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"/>
<ListView
        android:id="@+id/new_devices"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_weight="2"/>
```

Button: este objeto lo usamos para generar una búsqueda de todos los dispositivos bluetooth cercanos al celular, con el fin de vincular un nuevo dispositivo bluetooth en caso de ser necesario para conectarse a un nuevo equipo de diagnóstico.

```
<Button
        android:id="@+id/button_scan"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/button_scan"/>
```

3.4.4. CONSTRUYENDO LA PANTALLA OBSERVACIONES.

Una vez listo la actividad para la conexión se usó las siguientes líneas para visualizar la tercera actividad llamada pantalla observaciones, la misma que consta de dos elementos:

Un *LinearLayout* integrado con varios *TextView* para poder visualizar las observaciones referentes al análisis:

```
<LinearLayout
        android:id="@+id/Observaciones"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:layout_marginBottom="5dip"
        android:layout_marginLeft="5dip"
        android:layout_marginRight="5dip"
        android:layout_marginTop="5dip"
        android:background="#90000000"
        android:orientation="vertical">

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center_horizontal"
            android:layout_marginLeft="5dip"
            android:layout_marginRight="5dip"
            android:layout_marginTop="5dip"
            android:shadowColor="@color/sombra"
            android:shadowDx="5"
            android:shadowDy="5"
            android:shadowRadius="1.5"
            android:text="@string/tituloObservaciones"
            android:textColor="@color/titulo_menu"
            android:textSize="13pt"
```

```
        android:textStyle="bold|italic" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="30dip"
        android:layout_marginRight="20dip"
        android:layout_marginTop="10dip"
        android:text="@string/observacion1"
        android:textColor="@color/indicaciones"
        android:textSize="9pt"
        android:textStyle="bold|italic"
        android:typeface="serif" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="30dip"
        android:layout_marginRight="20dip"
        android:layout_marginTop="10dip"
        android:text="@string/observacion2"
        android:textColor="@color/indicaciones"
        android:textSize="9pt"
        android:textStyle="bold|italic"
        android:typeface="serif" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="30dip"
        android:layout_marginRight="20dip"
        android:layout_marginTop="10dip"
        android:text="@string/observacion3"
        android:textColor="@color/indicaciones"
        android:textSize="9pt"
        android:textStyle="bold|italic"
        android:typeface="serif" />
```

```
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="30dip"
        android:layout_marginRight="20dip"
        android:layout_marginTop="10dip"
        android:text="@string/observacion4"
        android:textColor="@color/indicaciones"
        android:textSize="9pt"
        android:textStyle="bold|italic"
        android:typeface="serif" />
</LinearLayout>
```

Estas líneas generan la pantalla que se visualiza a continuación:

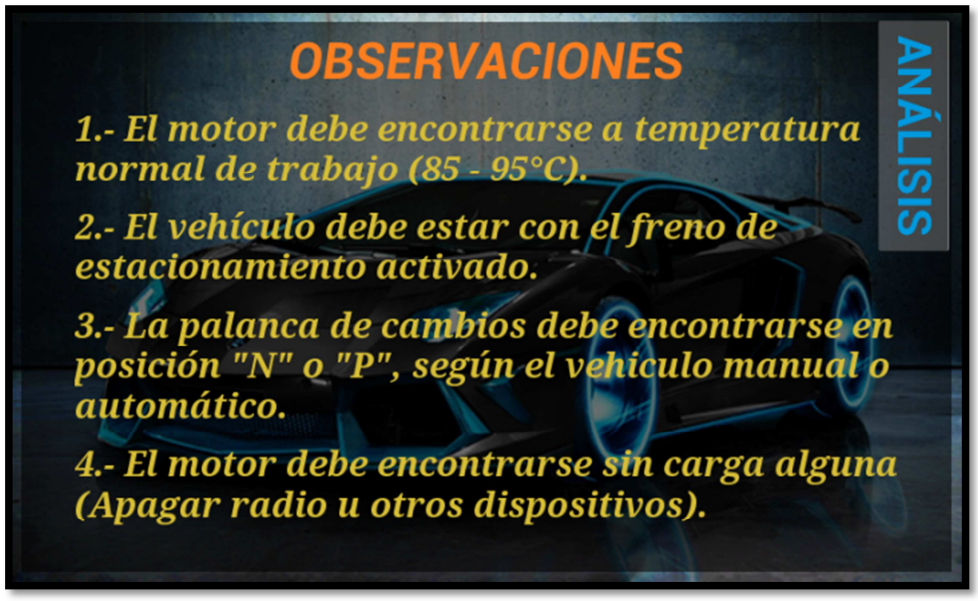


Figura 3.19. Pantalla Observaciones.

Fuente: Los Autores.

Tal como se aprecia en la figura anterior también posee de un botón con el nombre *ANÁLISIS* el cual posee las siguientes líneas de programación:

```
<Button
    android:id="@+id/btnMenuAp"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="50dip"
    android:onClick="Aparece"
    android:rotation="90"
    android:shadowColor="@color/sombra"
    android:shadowDx="5"
    android:shadowDy="5"
    android:shadowRadius="1.5"
    android:text="@string/btnMenu"
    android:textColor="@color/azul"
    android:textSize="12pt"
    android:textStyle="bold"/>

<Button
    android:id="@+id/btnMenuDe"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="50dip"
    android:onClick="Desaparece"
    android:rotation="90"
    android:shadowColor="@color/sombra"
    android:shadowDx="5"
    android:shadowDy="5"
    android:shadowRadius="1.5"
    android:text="@string/btnMenu"
    android:textColor="@color/azul"
    android:textSize="12pt"
    android:textStyle="bold"/>
```

Al colocar el código se nota que existen dos botones con el mismo nombre, pero lo que hace la diferencia es el evento *onClick* de cada uno de ellos puesto que hicimos que al presionar sobre este botón apareciera un subconjunto de opciones y a la vez si se presiona nuevamente el botón *ANÁLISIS* este subconjunto deberá desaparecer para visualizar el contenido que está detrás del mismo, para ello usamos de los siguientes eventos que su nombre mismo dice la función que realiza:

```
public void Aparece (View v){
        conjuntoMenu.setVisibility(View.VISIBLE);
        btnMenuAp.setVisibility(View.INVISIBLE);
        btnMenuDe.setVisibility(View.VISIBLE);
}

public void Desaparece (View v){
        conjuntoMenu.setVisibility(View.INVISIBLE);
        btnMenuAp.setVisibility(View.VISIBLE);
        btnMenuDe.setVisibility(View.INVISIBLE);
}
```

Aparece y desaparece el subconjunto de opciones.

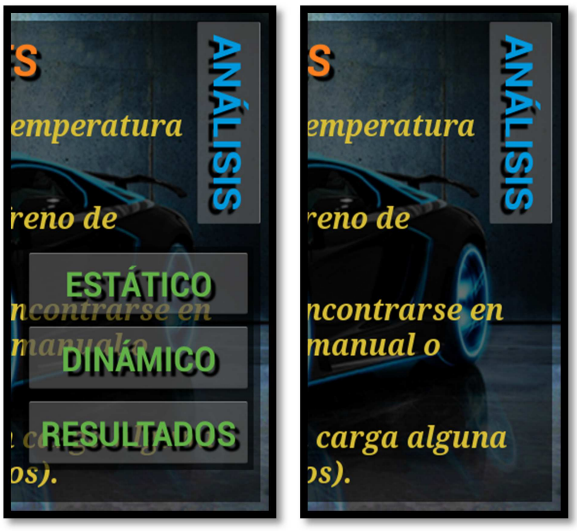


Figura 3.20. Pantalla Observaciones función botón Análisis.

Fuente: Los Autores.

3.4.5. CONSTRUYENDO LA PANTALLA INDICACIONES ESTÁTICO.

Seguido a la pantalla observaciones procedemos a indicar las líneas de programación para la actividad o pantalla indicaciones Estático, de igual manera esta actividad cuenta con un *LinearLayout* integrado con varios TextView, que muestra las indicaciones previas a realizar el análisis:

```
<LinearLayout
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_marginBottom="5dip"
    android:layout_marginLeft="5dip"
    android:layout_marginRight="5dip"
    android:layout_marginTop="5dip"
    android:background="#70000000"
    android:orientation="vertical" >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:layout_marginTop="10dip"
        android:shadowColor="@color/sombra"
        android:shadowDx="5"
        android:shadowDy="5"
        android:shadowRadius="1.5"
        android:text="@string/IndicacionTituloEstatico"
        android:textColor="@color/titulo_menu"
        android:textSize="13pt"
        android:textStyle="bold|italic"
        android:typeface="serif" />

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
```

```
        android:layout_marginLeft="55dip"
        android:layout_marginRight="40dip"
        android:layout_marginTop="10dip"
        android:text="@string/Indicacion1Estatico"
        android:textColor="@color/indicaciones"
        android:textDirection="anyRtl"
        android:textSize="10pt"
        android:textStyle="bold|italic"
        android:typeface="serif" />

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_marginLeft="55dip"
        android:layout_marginRight="40dip"
        android:layout_marginTop="10dip"
        android:text="@string/Indicacion2Estatico"
        android:textColor="@color/indicaciones"
        android:textSize="10pt"
        android:textStyle="bold|italic"
        android:typeface="serif" />

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_marginLeft="55dip"
        android:layout_marginRight="40dip"
        android:layout_marginTop="10dip"
        android:text="@string/Indicacion3Estatico"
        android:textColor="@color/indicaciones"
        android:textSize="10pt"
        android:textStyle="bold|italic"
        android:typeface="serif" />

    <TextView
```

```
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:layout_marginLeft="55dip"
            android:layout_marginRight="40dip"
            android:layout_marginTop="10dip"
            android:text="@string/Indicacion4Estatico"
            android:textColor="@color/indicaciones"
            android:textSize="10pt"
            android:textStyle="bold|italic"
            android:typeface="serif" />
    </LinearLayout>
```

Cuyas líneas generan la pantalla de indicaciones para el análisis estático:

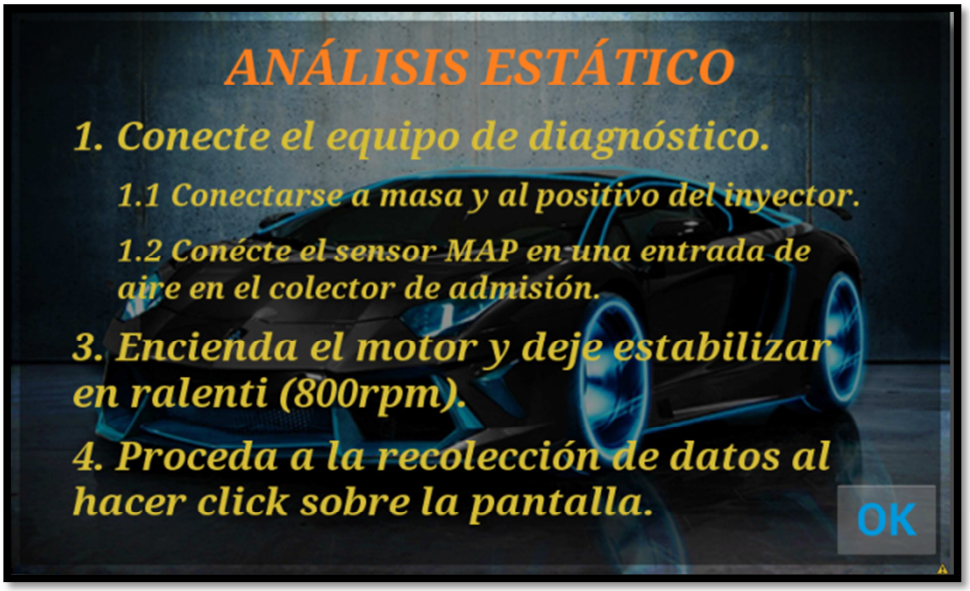


Figura 3.21. Pantalla indicaciones Análisis Estático.

Fuente: Los Autores.

Además en esta actividad se aprecia dos botones, el primero con una imagen de una lupa hace la misma función que el botón *ANÁLISIS* de la pantalla observaciones, solo que se ha reducido el nombre a una imagen únicamente por motivos de espacio en las siguientes actividades. Mientras que para el botón con el nombre *OK* se utilizó las siguientes líneas:

```
<Button
      android:id="@+id/btnOkEstatico"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_marginLeft="465dip"
      android:layout_marginTop="265dip"
      android:onClick="AnalisisEstatico"
      android:shadowColor="@color/sombra"
      android:shadowDx="5"
      android:shadowDy="5"
      android:shadowRadius="1.5"
      android:text="@string/btnOk"
      android:textColor="@color/azul"
      android:textSize="12pt"
      android:textStyle="bold" />
```

Donde lo más importante radica en el evento *onClick* que es el encargado de mostrar la siguiente actividad llamada Pantalla Análisis Estático.

```
public void AnalisisEstatico (View v){
             AnalisisDinamico.setVisibility(View.INVISIBLE);
             AnalisisEstatico.setVisibility(View.VISIBLE);
             indicacionesEstatico.setVisibility(View.INVISIBLE);
             conjuntoCilindrosEstatico.setVisibility(View.VISIBLE);
             conjuntoCilindrosDinamico.setVisibility(View.INVISIBLE);
             conjuntoBarrasBadEstatico.setVisibility(View.VISIBLE);
             conjuntoBarrasHalfEstatico.setVisibility(View.VISIBLE);
             conjuntoBarrasOkEstatico.setVisibility(View.VISIBLE);
             conjuntoBarrasBadDinamico.setVisibility(View.INVISIBLE);
             conjuntoBarrasHalfDinamico.setVisibility(View.INVISIBLE);
             conjuntoBarrasOkDinamico.setVisibility(View.INVISIBLE);
             btnMenuAp1.setVisibility(View.VISIBLE);
             btnMenuDe1.setVisibility(View.INVISIBLE);
             }
```

3.4.6. CONSTRUYENDO LA PANTALLA ANÁLISIS ESTÁTICO.

Para ello usamos el evento *onClick* del botón *OK* de la actividad anterior, que mostrará la quinta actividad, así como ocultar y mostrar los objetos necesarios para dicho análisis, donde se tienen las siguientes líneas de programación, por motivo de extensas línea al tener cuatro cilindros explicaremos solo el de un cilindro los tres restantes tienen gran similitud.

Este *FrameLayout* contiene la imagen que usará el programa para visualizar el porcentaje según el análisis.

```
<FrameLayout
    android:layout_width="120dip"
    android:layout_height="250dip"
    android:layout_gravity="center_horizontal" >
    <ImageView
        android:id="@+id/Cilindro_1Estatico"
        android:layout_width="120dip"
        android:layout_height="250dip"
        android:layout_gravity="center_horizontal"
        android:contentDescription="@string/cilindro1"
        android:src="@drawable/cil_0" />
</FrameLayout>
```

Este *TextView* se usa únicamente para colocar el nombre del cilindro.

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal"
    android:text="@string/cilindro1"
    android:textSize="7pt"
    android:textStyle="bold"
    android:typeface="serif" />
```

Este *LinearLayout* contiene las barras de progreso usados para mostrar el porcentaje de cada cilindro, realizamos tres grupos para tener una barra para cada estado de análisis, un estado bueno, uno malo y otro regular, con el fin de variar los colores de la misma.

```
<LinearLayout
android:id="@+id/barrasCilindro1BADEstatico"
android:layout_width="wrap_content"
android:layout_height="match_parent"
android:layout_marginLeft="120dip"
android:layout_marginTop="40dip"
android:orientation="horizontal" >

<ProgressBar
    android:id="@+id/progressBar0Estatico"
    style="@style/Widget.ProgressBar.VerticalBAD"
    android:layout_width="7dip"
    android:layout_height="250dip" />

 <ProgressBar
    android:id="@+id/progressBar1Estatico"
    style="@style/Widget.ProgressBar.VerticalHALF"
    android:layout_width="7dip"
    android:layout_height="250dip"
    android:visibility="invisible" />

 <ProgressBar
    android:id="@+id/progressBar2Estatico"
    style="@style/Widget.ProgressBar.VerticalOK"
    android:layout_width="7dip"
    android:layout_height="250dip"
    android:visibility="invisible" />
```

Las líneas mencionadas hacen referencia a esta actividad tal como se aprecia las barras de progreso tienen su respectivo color según el estado de cada cilindro:

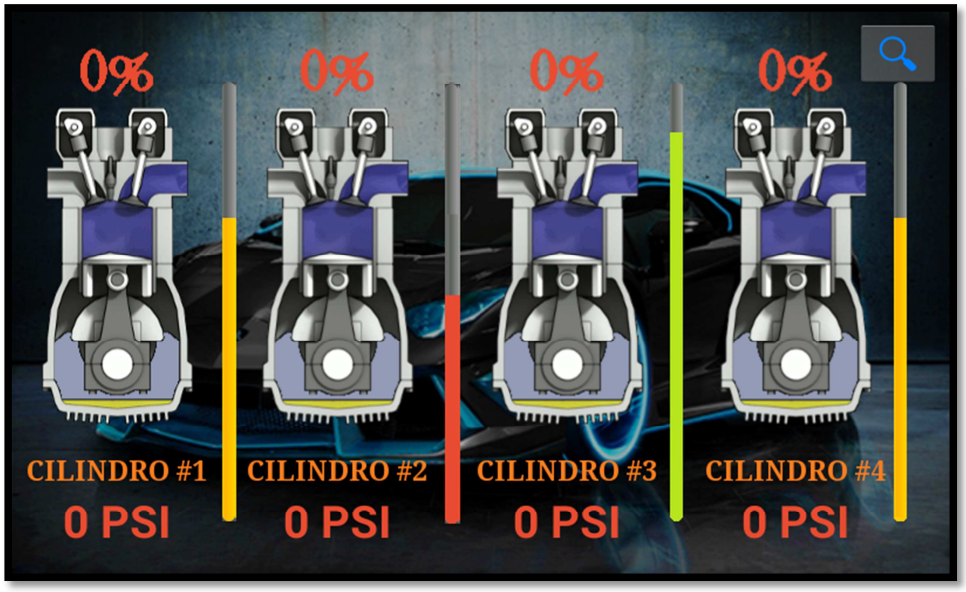


Figura 3.22. Pantalla Análisis Estático.

Fuente: Los Autores.

El botón con la imagen de lupa como se indicó anteriormente hace la misma función que el botón *ANÁLISIS* de la pantalla observaciones.

Las líneas de programación para mostrar los respectivos valores del análisis estático son las siguientes:

```
private void AnalisisTiempoRealEstatico (int[] datos) {
try {
////////// Porcentaje cilindro 1 //////////
int valor1 = datos[0];
double fx1 = ((p1*((valor1)^2))+(p2*valor1)+p3);
int porcentaje1 = (int)fx1;
InputStream is1 = getAssets().open("cil_"+porcentaje1+".png");
Cilindro1Estatico.setImageDrawable(Drawable.createFromStream(is1,
null));
```

```
InputStream is1 = getAssets().open("cil_" + porcentaje1 + ".png");
        } catch (IOException e) {
                e.printStackTrace();
                System.out.println("Error en la LECTURA");
}
```

Estas líneas son usadas para generar el valor de porcentaje para cada cilindro, puesto que el voltaje que entrega el sensor MAP está en un intervalo de 1,26 – 1,56 volts, representado en bits tenemos un intervalo de muestras 220 – 300, equivalentes a 55 – 75 bits, correspondiente al valor porcentual de 100 – 0 %, respectivamente, de tal manera que se realizó un ajuste de curvas usando la herramienta CFTOOL del programa MATLAB nos dió la siguiente ecuación con su respectiva gráfica:

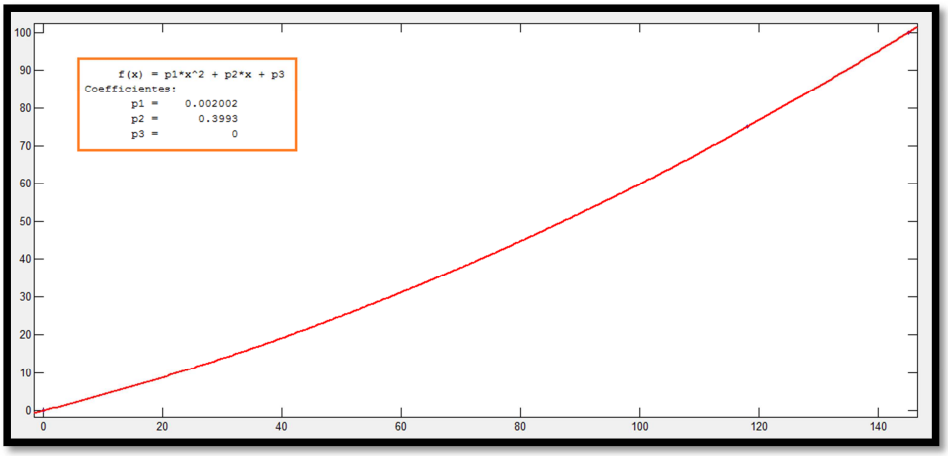


Figura 3.23. Ajuste de datos de porcentaje vs presión en CFTOOL .

Fuente: Los Autores.

Como se aprecia en la *figura 3.23*, tenemos la fórmula para la aproximación del valor porcentual correspondiente de 55 bits igual a 145 PSI y al 100%, para ello se tiene la fórmula:

$$f(x) = p1*x^2 + p2*x + p3$$

Dónde:

f(x): nos entrega el valor del porcentaje de cada cilindro.
p1, p2, p3: coeficientes de aproximación obtenidos realizando en el ajuste.
x: valor correspondiente de bits o presión que genera el vehículo.

Y así gracias al ajuste que logramos obtener el valor porcentual para cada cilindro.

Ahora para mostrar el valor en las barras de progreso se utilizó los mismos datos obtenidos con la herramienta CFTOOL y para ello tenemos las siguientes líneas de programación:

```
int a = datos[0];
double fxA = ((p1*((a)^2))+(p2*a)+p3);
int i1 = (int)fxA;
if (i1 < 75) {
      ////////// Para las barras de progreso. //////////
      malo1.setProgress(i1);
      malo1.setVisibility(View.VISIBLE);
      medio1.setVisibility(View.GONE);
      bueno1.setVisibility(View.GONE);
} else if (i1 < 90) {
      medio1.setProgress(i1);
      medio1.setVisibility(View.VISIBLE);
      malo1.setVisibility(View.GONE);
      bueno1.setVisibility(View.GONE);

} else if (i1 <= 100) {
      bueno1.setProgress(i1);
      bueno1.setVisibility(View.VISIBLE);
```

```
            malo1.setVisibility(View.GONE);
            medio1.setVisibility(View.GONE);
        }
```

En el caso de la presión referente al primer cilindro se realizó las siguientes líneas de programación:

```
double Pp1 = ((pUno*((i1)^2))+(pDos * (i1)));
int Presion1 = (int)Pp1;
String DatoPresion1 = String.valueOf(Presion1);
txtDatoPresion1.setText(DatoPresion1 + " PSI");
```

Puesto que el voltaje que entrega el sensor MAP es proporcional a la presión interna del motor se realizó otro ajuste de datos con la herramienta CFTOOL.

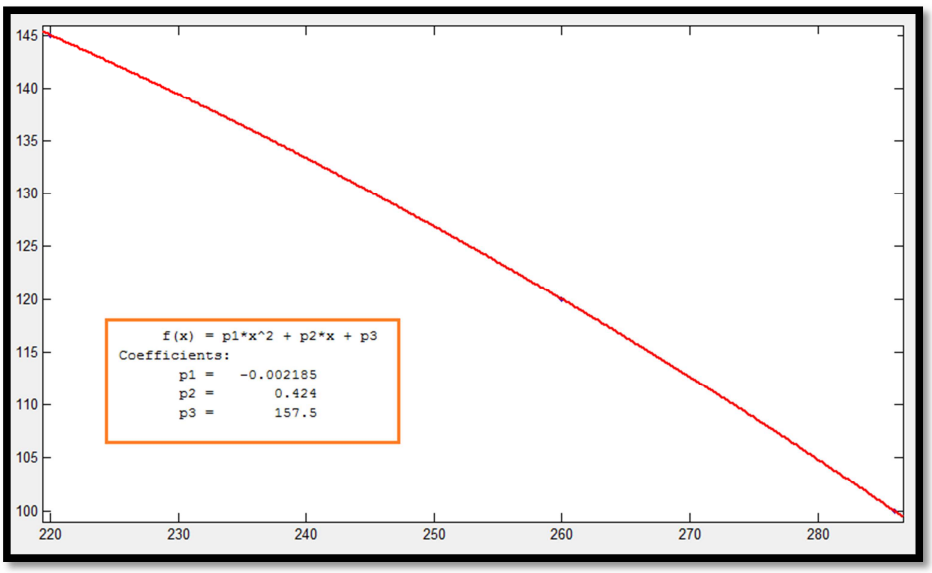


Figura 3.24. Ajuste de datos Presión vs Bits en CFTOOL.

Fuente: Los Autores.

Este ajuste tal como se aprecia nos dio la fórmula con los mismo datos realizados anteriormente con la única diferencia que el valor de **x** corresponde a los bits que genera el sensor MAP al tener el vehículo encendido.

3.4.7. CONSTRUYENDO LA PANTALLA INDICACIONES DINÁMICO.

Al continuar con los botones del subconjunto del botón *ANÁLISIS*, contamos también con un análisis dinámico, al seleccionar esta opción el usuario debe tener en cuenta las indicaciones necesarias dispuestas en una nueva actividad denominada pantalla Indicaciones Dinámico, cuya actividad cuenta con un *LinearLayout* con varios *TextView* que muestran las indicaciones previas al análisis:

```
<LinearLayout
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_marginBottom="5dip"
    android:layout_marginLeft="5dip"
    android:layout_marginRight="5dip"
    android:layout_marginTop="5dip"
    android:background="#70000000"
    android:orientation="vertical"
    tools:ignore="TooManyViews" >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:layout_marginTop="20dip"
        android:shadowColor="@color/sombra"
        android:shadowDx="5"
        android:shadowDy="5"
        android:shadowRadius="1.5"
        android:text="@string/indicacionTituloDin"
        android:textColor="@color/titulo_menu"
        android:textSize="12pt"
        android:textStyle="bold|italic" />

    <TextView
        android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:layout_marginLeft="50dip"
        android:layout_marginRight="40dip"
        android:layout_marginTop="15dip"
        android:text="@string/indicacion5"
        android:textColor="@color/indicaciones"
        android:textSize="10pt"
        android:textStyle="bold|italic"
        android:typeface="serif" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="50dip"
        android:layout_marginRight="40dip"
        android:layout_marginTop="15dip"
        android:text="@string/indicacion6"
        android:textColor="@color/indicaciones"
        android:textSize="10pt"
        android:textStyle="bold|italic"
        android:typeface="serif" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="50dip"
        android:layout_marginRight="40dip"
        android:layout_marginTop="15dip"
        android:text="@string/indicacion7"
        android:textColor="@color/indicaciones"
        android:textSize="10pt"
        android:textStyle="bold|italic"
        android:typeface="serif" />

    <TextView
```

```
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginLeft="50dip"
            android:layout_marginRight="40dip"
            android:layout_marginTop="15dip"
            android:text="@string/indicacion8"
            android:textColor="@color/indicaciones"
            android:textSize="10pt"
            android:textStyle="bold|italic"
            android:typeface="serif" />
    </LinearLayout>
```

Estas líneas generan la pantalla indicaciones dinámico:

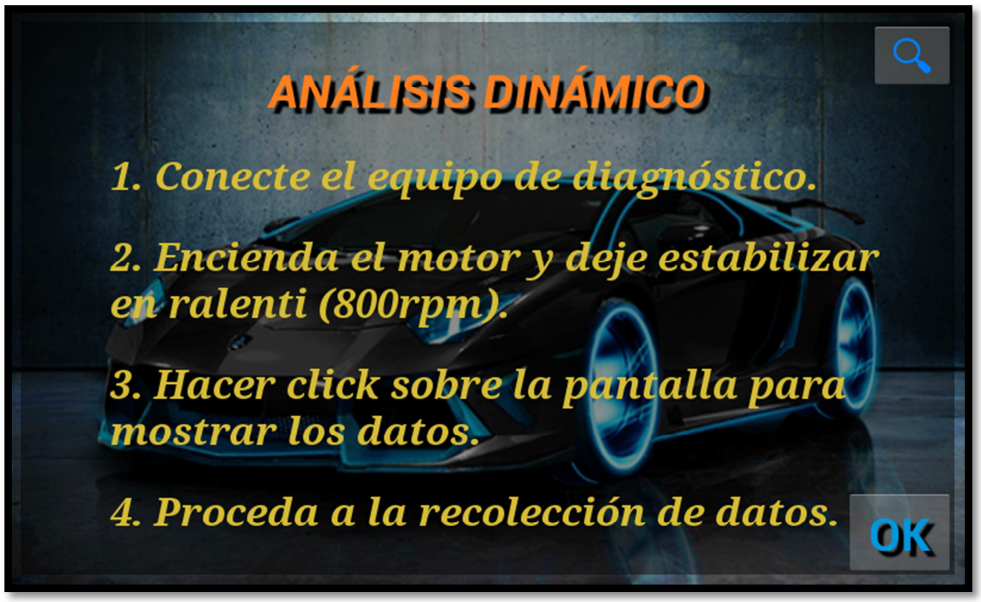


Figura 3.25. Pantalla indicaciones Análisis Dinámico.

Fuente: Los Autores.

Tal como se puede apreciar en la figura anterior, esta actividad cuenta con dos botones, el primero con una imagen de una lupa hace la misma función que el botón *ANÁLISIS* de la pantalla observaciones, mientras que para el botón con el nombre *OK* se utilizó para cerrar las indicaciones y proceder a la siguiente actividad denominada Análisis Dinámico.

```
<Button
    android:id="@+id/btnOkDinamico"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginLeft="465dip"
    android:layout_marginTop="265dip"
    android:onClick="AnalisisDinamico"
    android:shadowColor="@color/sombra"
    android:shadowDx="5"
    android:shadowDy="5"
    android:shadowRadius="1.5"
    android:text="@string/btnOk"
    android:textColor="@color/azul"
    android:textSize="12pt"
    android:textStyle="bold" />
```

Indicando nuevamente la parte fundamental de este objeto es el evento *onClick* que se encarga de mostrar la siguiente actividad de análisis, así como también cerrar otros objetos innecesarios para este análisis.

```
public void AnalisisDinamico (View v){

    AnalisisEstatico.setVisibility(View.INVISIBLE);
    AnalisisDinamico.setVisibility(View.VISIBLE);
    indicacionesDinamico.setVisibility(View.INVISIBLE);
    conjuntoCilindrosDinamico.setVisibility(View.VISIBLE);
    conjuntoCilindrosEstatico.setVisibility(View.INVISIBLE);
    conjuntoBarrasBadDinamico.setVisibility(View.VISIBLE);
    conjuntoBarrasHalfDinamico.setVisibility(View.VISIBLE);
    conjuntoBarrasOkDinamico.setVisibility(View.VISIBLE);
    conjuntoBarrasBadEstatico.setVisibility(View.INVISIBLE);
    conjuntoBarrasHalfEstatico.setVisibility(View.INVISIBLE);
    conjuntoBarrasOkEstatico.setVisibility(View.INVISIBLE);
    btnMenuAp1.setVisibility(View.VISIBLE);
    btnMenuDe1.setVisibility(View.INVISIBLE);    }
```

Donde los más importantes que se muestran son:

Un *FrameLayout* que contiene una imagen en movimiento de un cilindro del motor, así como una imagen con el valor porcentual para cada cilindro.

```
<FrameLayout
      android:layout_width="120dip"
      android:layout_height="250dip" >
   <ImageView
      android:id="@+id/Cilindro_1Dinamico"
      android:layout_width="100dip"
      android:layout_height="220dip"
      android:layout_gravity="center_horizontal|center_vertical"
      android:contentDescription="@string/cilindro4"
      android:src="@drawable/cildin_0" />
   <com.gif.animation.VisorGif
      android:id="@+id/cilindro1Din"
      android:layout_width="88dip"
      android:layout_height="181dip"
      android:layout_marginLeft="16.8dip"
      android:layout_marginTop="53.8dip"  /
</FrameLayout>
```

Un *TextView* únicamente para colocar el nombre del cilindro.

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/cilindro1"
    android:textColor="@color/cilindros"
    android:textSize="7pt"
    android:textStyle="bold"
    android:typeface="serif" />
```

Este *LinearLayout* contiene las barras de progreso usados para mostrar el porcentaje de cada cilindro, realizamos tres grupos para tener una barra para cada estado de análisis, un estado bueno, uno malo y otro regular, con el fin de variar los colores de la misma.

```
<LinearLayout
    android:id="@+id/conjuntoBarrasBadDinamico"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal"
    android:visibility="invisible" >

        <ProgressBar
            android:id="@+id/progressBar0Dinamico"
            style="@style/Widget.ProgressBar.VerticalBAD"
            android:layout_width="7dip"
            android:layout_height="250dip" />

        <ProgressBar
            android:id="@+id/progressBar1Dinamico"
            style="@style/Widget.ProgressBar.VerticalHALF"
            android:layout_width="7dip"
            android:layout_height="250dip"
            android:visibility="invisible" />

        <ProgressBar
            android:id="@+id/progressBar2Dinamico"
            style="@style/Widget.ProgressBar.VerticalOK"
            android:layout_width="7dip"
            android:layout_height="250dip"
            android:visibility="invisible" />
    </LinearLayout>
```

De esta manera tenemos la actividad para el análisis dinámico.

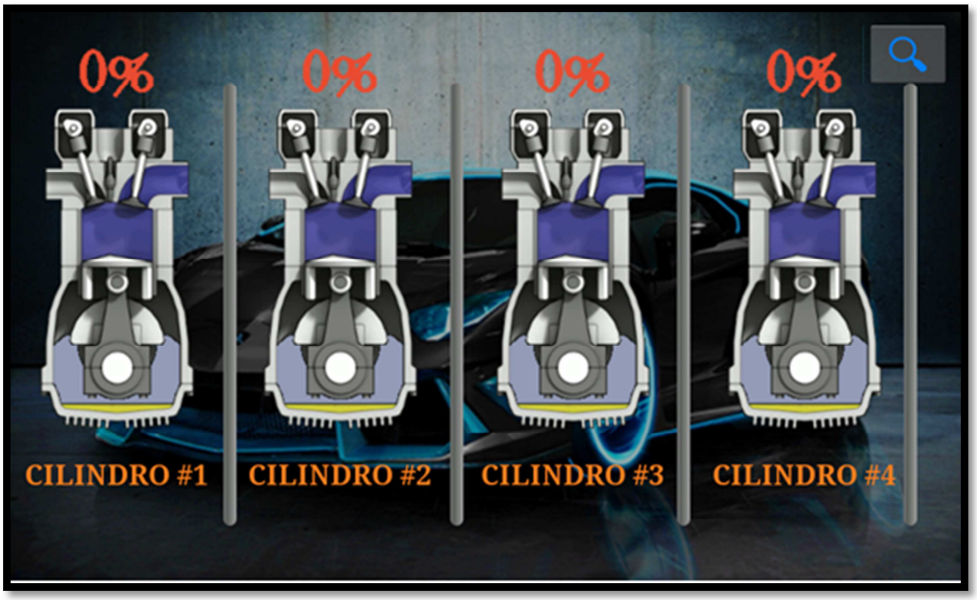


Figura 3.26. Análisis Dinámico.

Fuente: Los Autores.

Esto en cuanto a los objetos necesarios a mostrar para realizar el análisis, mientras que la programación para determinar cada valor hace referencia en las siguientes líneas:

```
private void AnalisisTiempoRealDinamico (int[] datos) {
try {
////////// Porcentaje cilindro 1 //////////
double fx1Din = ((p1Din*((valor1Din)^2))+(p2Din*valor1Din)+p3Din);
int porcentaje1Din = (int)fx1Din;
system.out.println("porcentaje1:" + porcentaje1Din);
InputStream is1Din = getAssets().open("cildin_" + porcentaje1Din +
".png");
cilindro1Dinamico.setImageDrawable(Drawable.createFromStream(is1Din,
null));
```

Estas líneas son usadas para generar un valor porcentual de trabajo que genera cada cilindro, puesto que una fase de trabajo del motor correspondería al 100%, pero debido a la presencia de cuatro cilindros este valor se divide en cuatro partes, siendo un valor de 25% que entrega cada cilindro.

Para ello realizamos un ajuste de datos con la ayuda de la herramienta CFTOOL, puesto que tenemos los siguientes datos del sensor MAP 1,26 – 1,56 volts, los debemos ajustar al valor porcentual de 25 -0%, respectivamente.

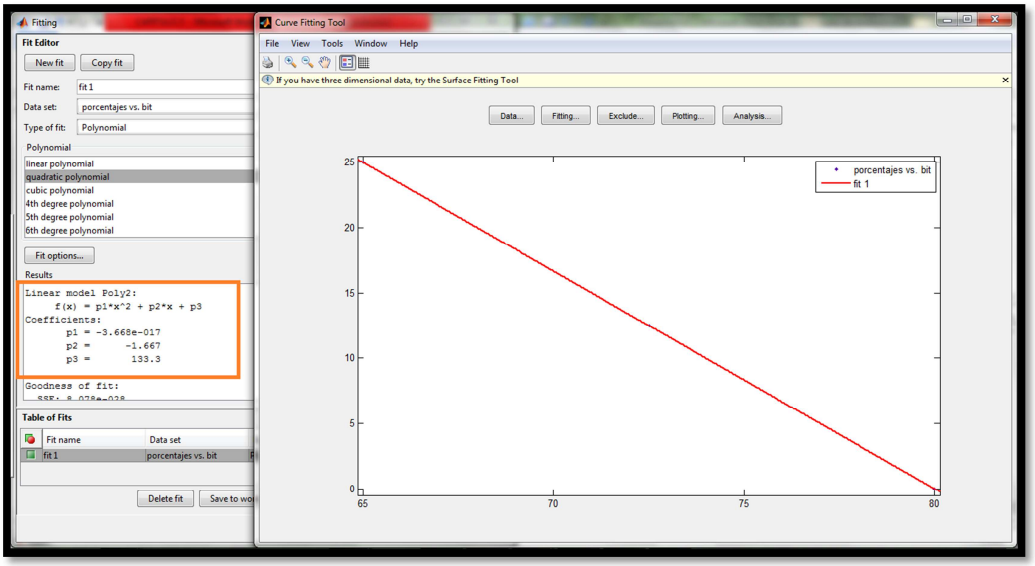


Figura 3.27. Ajuste de datos CFTOOL.

Fuente: Los Autores.

En cuanto para las barras de progreso se usan los mismos datos obtenidos en el CFTOOL, con las siguientes líneas de programación:

```
int aDin = datos[0];
double fxADin = ((p1Din*((aDin)^2))+(p2Din*aDin)+p3Din);
int i1Din = (int)fxADin;

if (i1Din < 20) {
      ////////// Para las barras de progreso. //////////
```

```
            malo1Din.setProgress(i1Din);
            malo1Din.setVisibility(View.VISIBLE);
            medio1Din.setVisibility(View.GONE);
            bueno1Din.setVisibility(View.GONE);

        } else if (i1Din < 23) {
            medio1Din.setProgress(i1Din);
            medio1Din.setVisibility(View.VISIBLE);
            malo1Din.setVisibility(View.GONE);
            bueno1Din.setVisibility(View.GONE);

        } else if (i1Din <= 25) {
            bueno1Din.setProgress(i1Din);
            bueno1Din.setVisibility(View.VISIBLE);
            malo1Din.setVisibility(View.GONE);
            medio1Din.setVisibility(View.GONE);
        }
```

Una vez terminado los cálculos necesarios para el ajuste y transformación de datos se procede a realizar la programación para recolectar los datos enviados desde arduino, los datos se transfieren a través de la comunicación bluetooth desde el arduino hacia el celular, cuyos datos son procesados para posterior a ellos permitir realizar al programa los cálculos ya establecidos anteriormente, cuyas líneas son:

```
    private void enviaTexto(String TextoEscrito) {
        if (conectado) {if (mySocket != null) {
            try {
            strBufferIn = "";
            MyOutStream = mySocket.getOutputStream();
            } catch (IOException e)
{Toast.makeText(ActivityPrototype.this,
            ("\nERROR:  " + e.getMessage()), Toast.LENGTH_SHORT)
            .show();
            }
            try {
```

```
                strBufferIn = "";
                MyOutStream.write((TextoEscrito).getBytes());
                }           catch           (IOException           e)
        {Toast.makeText(ActivityPrototype.this,
                ("\nERROR:          "          +          e.getMessage()),
Toast.LENGTH_SHORT).show();
                }     }     }     }
```

Estas líneas se encargan de enviar datos hacia el arduino a través del bluetooth, para que este responda con los datos necesarios para el proceso de cálculo, pero los datos obtenidos los tenemos que realizar una transformación para poder usarlos y para ello usamos las siguientes líneas:

```
try {
        while ((line = br.readLine()) != null) {
                datos[acum] = Integer.parseInt(line);
                acum++;
                if (acum == 4)
                AnalisisTiempoRealEstatico(datos);
                AnalisisTiempoRealDinamico(datos);
}
```

Donde únicamente recoge cuatro datos que son de los cuatro cilindros, y los transforma en variables numéricas enteras, de esta manera tenemos los valores en cada una de las actividades, tanto para el análisis estático, como para el análisis dinámico.

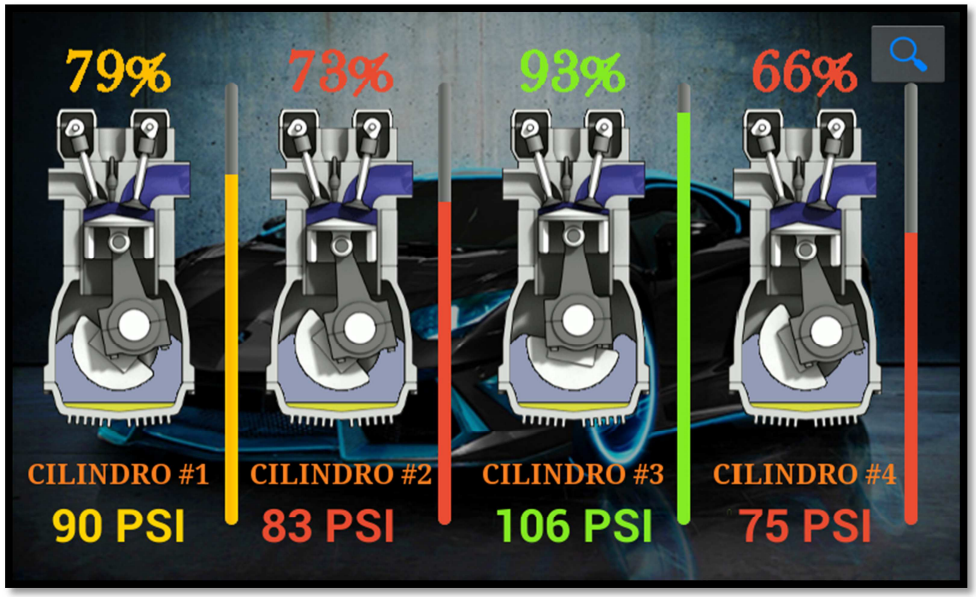


Figura 3.28. Datos Análisis Estático.

Fuente: Los Autores.

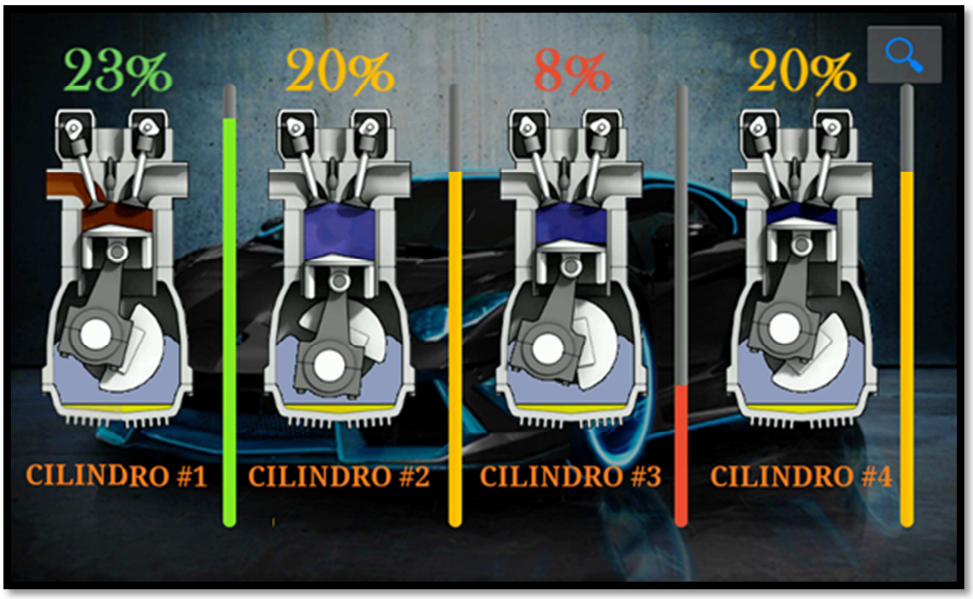


Figura 3.29. Datos Análisis Dinámico.

Fuente: Los Autores.

3.4.8. CONSTRUYENDO LA PANTALLA RESULTADOS.

Por último tenemos la recolección de los datos que nos dio el análisis, estos datos pueden ser visualizados en la actividad de Resultados o Pantalla de Resultados, donde se aprecia el valor porcentual y la presión de cada cilindro, esta visualización es con el fin de obtener un archivo de texto con el resultado del análisis, cuyo archivo poder imprimir o enviar por correo según sea lo necesario.

Esta actividad consta de los siguientes elementos:

Un *TableRow*, como su nombre lo dice genera una tabla para la visualización de los datos en su respectivo *TextView*.

```
<TableRow
    android:id="@+id/tableRow1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:padding="10dip" >
    <TextView
        android:layout_width="60dip"
        android:layout_height="wrap_content"
        android:layout_marginLeft="12dip"
        android:text="@string/cilin1"
        android:textColor="@color/indicaciones"
        android:textSize="9pt"
        android:textStyle="bold"
        android:typeface="serif" />
    <TextView
        android:id="@+id/txtDato1Estatico"
        android:layout_width="60dip"
        android:layout_height="wrap_content"
        android:textColor="@color/azul"
        android:textSize="9pt"
        android:textStyle="bold"
```

```
            android:typeface="monospace" />
            <TextView
                android:id="@+id/txtDato1EstaticoPresion"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:textColor="@color/azul"
                android:textSize="9pt"
                android:textStyle="bold"
                android:typeface="monospace" />
        </TableRow>
```

Y es así como se genera la actividad para la visualización de los resultados.

Figura 3.30. Pantalla Resultados.

Fuente: Los Autores.

Esto en cuanto a la visualización, pero para guardar o generar el archivo de texto usamos las siguientes líneas:

```
try {
	File ruta_sd = Environment.getExternalStorageDirectory();
	File f = new File(ruta_sd.getAbsolutePath(), "ProToTyPe.txt");
	OutputStreamWriter fout = new OutputStreamWriter(new
	FileOutputStream(f));

	fout.write("ANALISIS DEL MOTOR" + "\n " + "\n"
	                                + "# CILINDRO" + "        " +
	"EFICIENCIA (%)" + "     " + " PRESÓN (PSI) " + "\n" + "\n"
	                                +   "Cilindro   1"   +   "
	" + porcentaje1 +" %" + "          " + PresionCil1 + " PSI" + "
	" + "\n"
	                                +   "Cilindro   2"   +   "
	" + porcentaje2 +" %" + "          " + PresionCil2 + " PSI" + "
	" + "\n"
	                                +   "Cilindro   3"   +   "
	" + porcentaje3 +" %" + "          " + PresionCil3 + " PSI" + "
	" + "\n"
	                                +   "Cilindro   4"   +   "
	" + porcentaje4 +" %" + "          " + PresionCil4 + " PSI" + "
	" + "\n");
	fout.close();

} catch (IOException e) {
	e.printStackTrace();
	System.out.println("Error al guardar Datos"); }
```

Y es así como se obtienen todos los datos de cada análisis realizado, puesto que es un prototipo y está en proceso de mejoras, estimamos un error de 10% en cada análisis y se pretende reducir a un mínimo aceptable, donde los datos sean unos 99% correctos.

3.5. CONSTRUYENDO LA APLICACIÓN EN ARDUINO.

Para construir el código de la tarjeta de adquisición de datos, usamos el programador Arduino, que maneja un código muy similar a C/C++, lenguaje de programación orientado para la implementación de sistemas operativo. A partir de esto podremos describir el software Arduino.

3.5.1. CONECTAR LA PLACA CON EL SOFTWARE.

Para trabajar en esta plataforma lo principal es instalar el Driver USB en el computador y así poder cargar el código directamente a nuestra placa, además para que el software reconozca la placa es necesario elegir qué tipo de tarjeta tenemos, como hemos dicho anteriormente cada tarjeta tiene un micro controlador distinto.

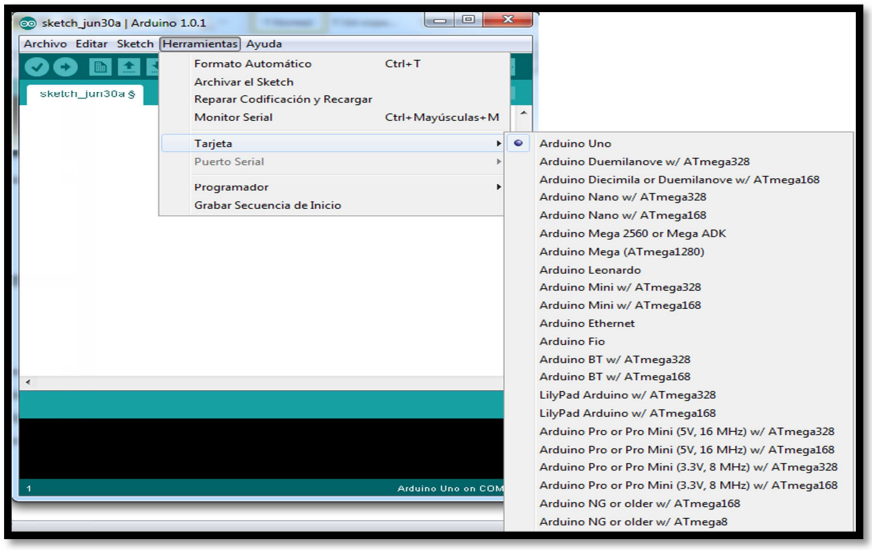


Figura 3.31. Tipo de Tarjeta Arduino.

Fuente: Los Autores.

Realizado todo esto el software nos informa que tipo de puerto USB tenemos que usar para realizar cualquier proyecto además de la placa que estamos utilizando.

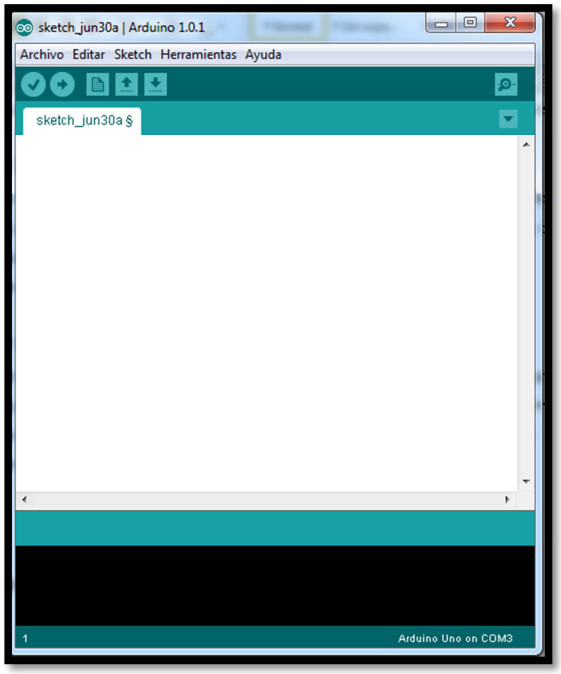


Figura 3.32. Puerto y Tarjeta Utilizada.

Fuente: Los Autores**.**

3.5.2. PARTES PRINCIPALES DE LA VENTANA.

El software como todo programa cuenta con un menú principal, a continuación citamos lo más importante de este programa.

3.5.2.1. ÁREA DE TRABAJO:

Es la parte principal del software en el cual el programador trabaja con el código directamente y se lo puede editar.

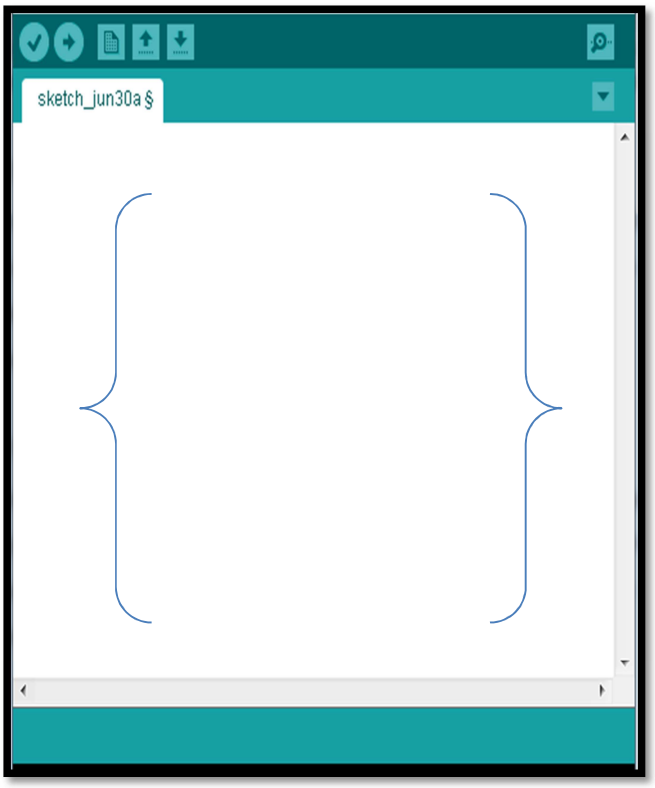


Figura 3.33. Área de Trabajo.

Fuente: Los Autores

3.5.2.2. BOTÓN VERIFICAR.

Este comando como su nombre indica nos sirve para verificar el código creado antes de enviarlo a compilar en la tarjeta y poder editarlo en caso de ser necesario puesto que revisa todas las líneas de código una por una y resalta con color amarillo si la línea de código está mal.

Figura 3.34. Botón Verificar.

Fuente: Los Autores.

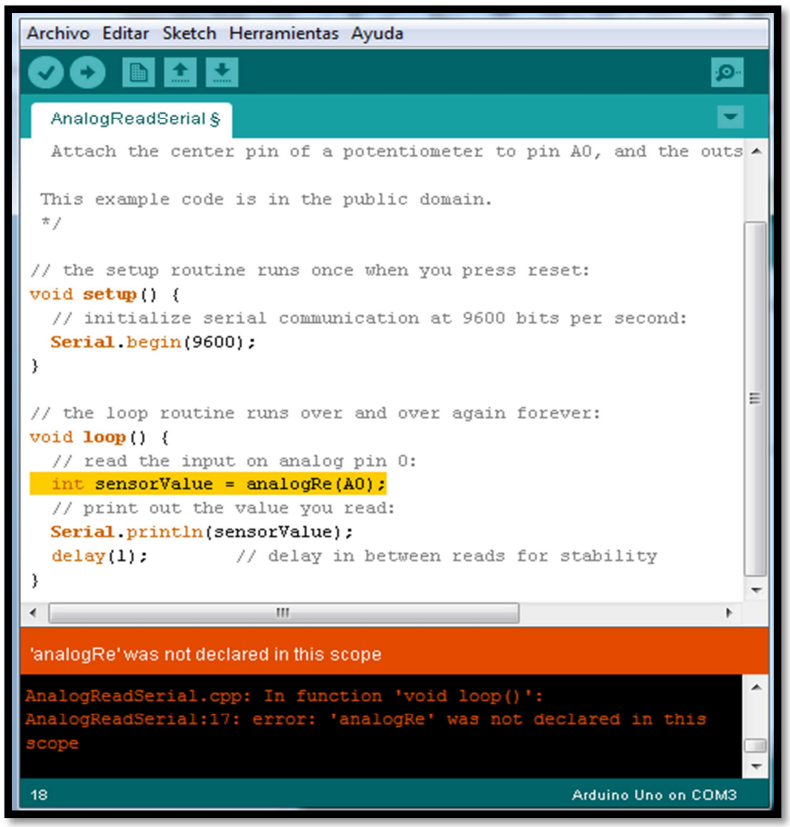


Figura 3.35. Líneas de Información de Error.

Fuente: Los Autores.

3.5.2.3. CARGAR / COMPILAR:

Compila el código directamente a nuestra tarjeta, además de realizar su función principal también revisa el código línea a línea resaltando el error como en el caso anterior.

Figura 3.36. Botón Cargar.

Fuente: Los Autores.

3.5.2.4. PANTALLA DE ESTADO:

Es la pantalla que informa el estado de la carga así como el tamaño de nuestro código.

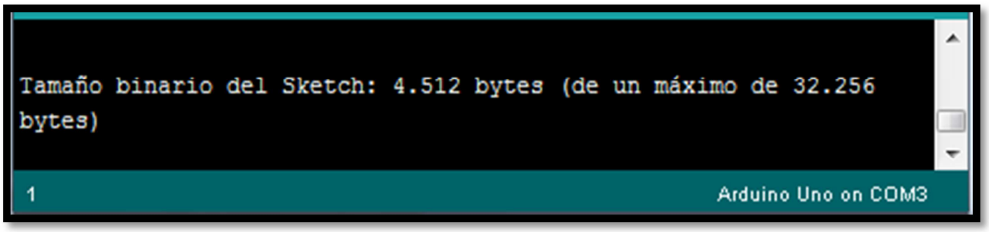


Figura 3.37. Pantalla de Estado.

Fuente: Los Autores.

3.5.3. COMPILACIÓN EN PROTEUS.

Parte importante de todo código de programación es poder compilarlo y así revisar con otro programa su estado, como es el caso de proteus, este nos permite conectar todo los dispositivos electrónicos necesarios para analizar el estado de

funcionamiento de nuestro código, para esto es necesario activar el compilador propio del software.

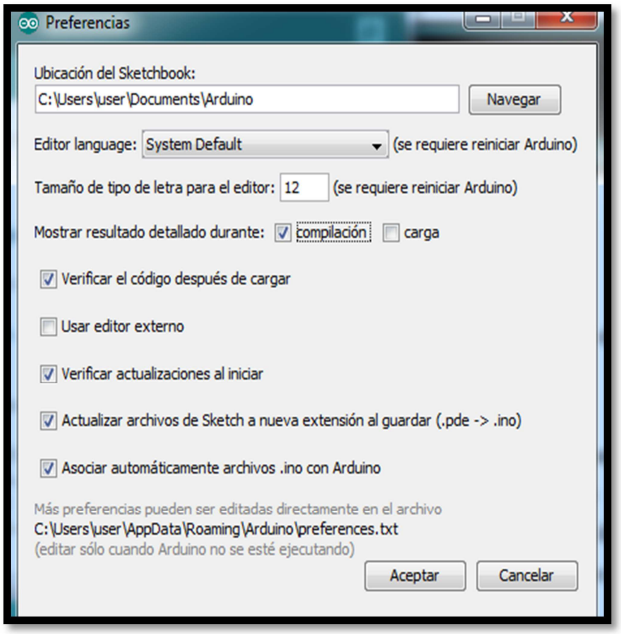


Figura 3.38 Activación del Compilador.

Fuente: Los Autores.

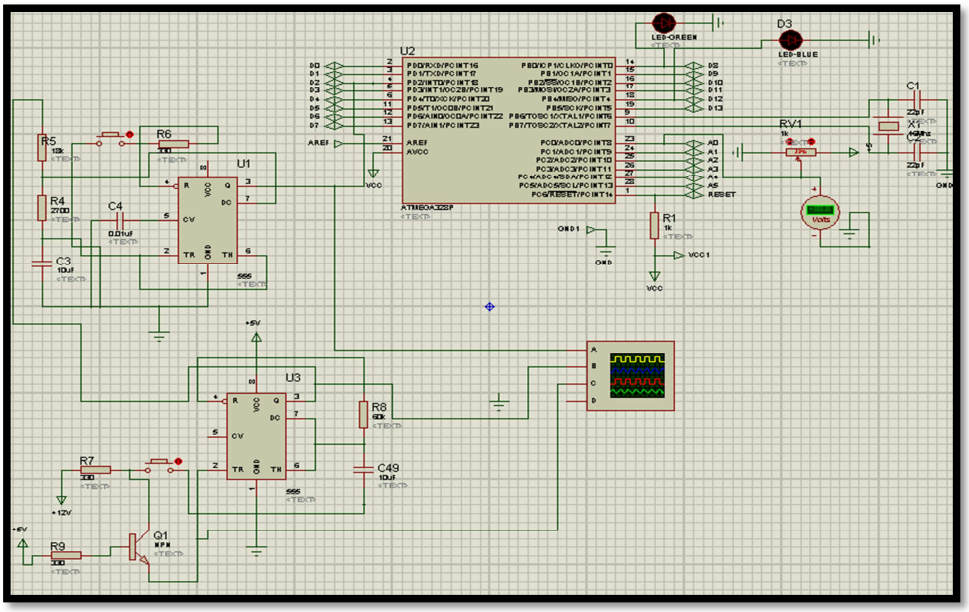


Figura 3.39. Código Compilado y Revisado en Proteus.

Fuente: Los Autores.

3.6. CONSTRUYENDO EL CÓDIGO PARA IMPLEMENTAR LA ADQUISICIÓN DE DATOS.

3.6.1. DECLARACIÓN DE VARIABLES.

Para declarar variables debemos tener en cuenta que tipo de dato queremos implementar o recibir ya que éstas pueden ser de carácter, binario, entero, párrafos, decimal, etc.

3.6.1.1. VARIABLES ESTÁTICAS.

En un código es necesario colocar líneas de comentarios entre este símbolo (//) para recordar que función cumple cada línea del código, estas variables son de tipo entero y almacena valores numéricos de 16 bits sin decimales en un rango de -32.767 a 32.767 que en arduino se declara “int”, sirven para informar si la señal esta activada o no.

```
// Variables estáticas//
      int pinBoton = 2;
      int verde = 12 ;
      int rojo = 8;
```

3.6.1.2. VARIABLES TIPO BYTE.

Variables de tipo “byte” almacenan valores numéricos de 8 bits en un rango de 0 a 255 sin decimales, sirven para enviar los datos por bluetooth a nuestro celular.

```
byte cilindroA=0;
byte cilindroB=0;
byte cilindroC=0;
byte cilindroD=0;
```

3.6.1.3. VARIABLES ENTERAS Y DECIMALES.

Para este último caso de la declaración de variables tenemos valores enteros y decimales (coma flotante) que en arduino se declara como “float”, es decir nos acepta valores decimales comprendido entre `3.4028235e +38` a `-3.4028235e +38`, la función que cumplen estas variables en nuestro caso es verificar en qué estado está el inyector “estadoBoton”, la “entrada” recibe los valores del sensor MAP y “conta” contará el número de pulsos enviado por el sistema.

```
int estadoBoton = 0;
float entrada =0;
int conta=0;
```

3.6.2. ESTADO DE LAS VARIABLES.

Una vez declarada las variables procedemos a especificar si son entradas / salidas o están en estado bajo / alto, para ello arduino nos pide iniciar el código de programación con el comando *void setup{},* este es la cabeza de nuestro código en donde especifica en qué estado se encuentran las variables, además de iniciar con la comunicación entre la placa arduino y el celular.

Además es aquí donde se coloca la programación pertinente para la comunicación serial de la placa arduino.

```
Serial.begin(9600);
Serial.println("Iniciando lectura de datos...");
```

Estas líneas nos envía toda la información desde nuestra tarjeta hacia nuestro celular vía comunicación serial bluetooth, además se puede verificar si la información enviada es la correcta usando una herramienta propia de arduino que es el monitor serial explicado anteriormente, así corroboramos la veracidad de los datos.

Seguido de la comunicación serial declaramos las variables “verde” y “rojo” como salidas como se dijo son variables de información de estado, también declaramos un pin de entrada con el nombre de “pinBoton”.

```
pinMode(verde,OUTPUT);
pinMode(rojo,OUTPUT);
pinMode(pinBoton,INPUT);
```

Declaración de interrupciones externas, para ello primero habilitamos las interrupciones globales “sei();” además activamos el “pin digital 2” que corresponde a la interrupción “0” que funcionará cada vez que la señal este en estado bajo.

```
sei();
      EIMSK |= (1<<INT0);
      EICRA |= (1<<ISC01)
```

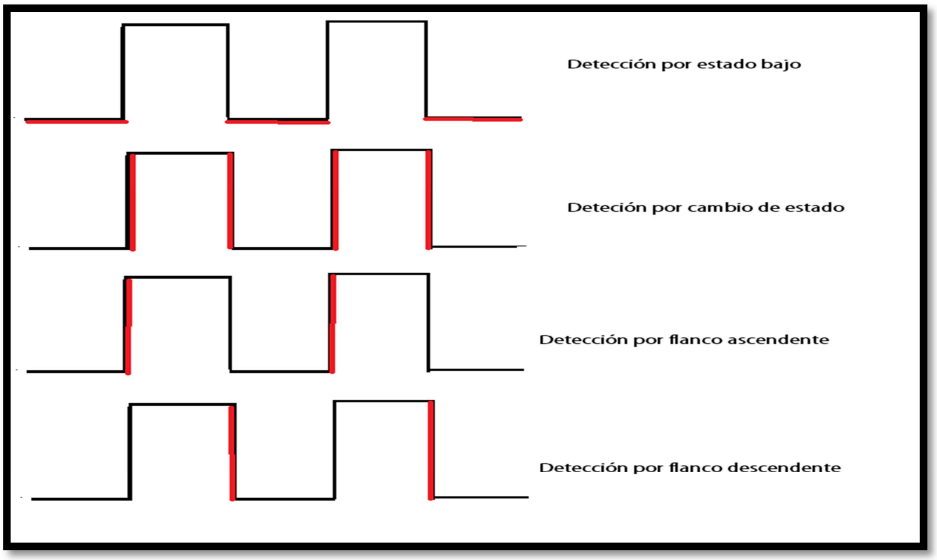


Figura 3.40. Tipo de interrupción.

Fuente: Los Autores

Además colocamos la función *void sleepNow(){}*, que permite apagar nuestra placa así como toda sus funciones como es el caso de los timer, el convertidor analógico digital, etc. Con la finalidad de reducir el consumo de energía como los recursos, evitando asi que la placa se sature de información.

```
void sleepNow()
    {
      set_sleep_mode(SLEEP_MODE_IDLE);

      sleep_enable(); //ACTIVA EL MOD
     power_adc_disable(); //DESACTIVA POWER_ADC
      power_spi_disable();//DESACTIVA
      power_timer0_disable();//DESACTIVA
      power_timer1_disable();//DESACTIVA
      power_timer2_disable();//DESACTIVA
      power_twi_disable();//DESACTIVA
      sleep_mode();
      sleep_disable();
      power_all_enable();
```

3.6.3. FUNCIÓN DE LAS VARIABLES.

Para definir qué función cumple cada variable, arduino crea el comando *void loop(){}*, que es la parte más importante del código, en esta decidimos los cambios a realizarse con cualquier evento para que la variable cambie de estado o realice otra actividad.

```
      if (estadoBoton == HIGH) {
          digitalWrite(verde, HIGH);
      }
      else {
          digitalWrite(verde, LOW);
      }
```

Con las líneas de programación anterior decidimos que: si el “estadoBoton” es alto es decir la alimentación del pin llega a 5V coloque a “verde” también en estado alto, caso contrario verde se mantendrá en estado bajo.

Ahora si el “estadoBoton” es bajo empieza el conteo del pulso mediante la variable “conta” que empieza desde el cero y termina en cuatro con saltos de uno en uno.

```
if (estadoBoton == LOW) {
   conta++;
   conta = 0  (conta<=4)? conta:1;
}
```

La sentencia “switch / case” sirve para ejecutar diferentes códigos en función de varias condiciones, es decir, compara el valor de una variable con el valor especificado de la sentencia “case”, siempre que se use estas sentencias es necesario cerrarlos con un “break”, caso contrario permanecerá ejecutando la misma parte del código.

```
switch (conta) {
     case 1:
   if(conta=1){
     cilindroA =map(entrada,0,1024,0,255);
    Serial.println("cilindro 1 = ");
     Serial.println(cilindroA,DEC ) ;
  }
  break;
```

Empezamos con la sentencia “switch” en función de la variable “conta” y tomando como primer caso la misma variable, provocando así que la variable “cilindroA” transforme o sea mapeado su valor de entrada desde valores de 0 a 1024 muestras y transformados a 8 bits para poder enviar el valor por nuestro módulo bluetooth al celular.

Para el caso 2, 3, 4 las líneas de programación son idénticas, con la única diferencia que las variables tienen su respectivo nombre: “cilindroB”, “cilindroC” y “cilindroD”, estas variables son de gran ayuda puesto que recogen los valores que envía el sensor MAP.

```
if (Serial.available()) {
    int val = Serial.read();
    if (val == 'S') {
        Serial.println("Serial: Entering Sleep mode");
        digitalWrite(6,LOW);
        delay(100);
        contador = 0;
        sleepNow();
    }
    if (val == 'A') {
        digitalWrite(6,HIGH);
        sleep_disable();
    }
}
```

Nuevamente usamos la función *Serial* pero no para enviar datos al celular si no para recibir un carácter denominado “S” que sirve para apagar la tarjeta y las funciones descritas anteriormente.

```
if (conta >= 4) {
    Serial.println("Timer: Entering Sleep mode");
    delay(100);
    conta = 0;
    sleepNow();
   digitalWrite(6,LOW);
}
```

En este caso cada vez que “conta” llega al número cuatro espera un tiempo de 100ms y se reinicia o vuelve a cero, para empezar nuevamente el conteo.

```
ISR(INT0_vect){
      if(entrada >= 677.06){
         digitalWrite(rojo,HIGH);
      }
      if(entrada < 677.06){
         digitalWrite(rojo,LOW);
      }
}
```

El código de la interrupción puede ir al principio como al final de nuestro programa, esta parte reconoce que es lo que debe realizar el código, en nuestro caso cada vez que el pulso de inyección este en estado bajo tomará los valores del sensor MAP los imprime al monitor serial y los envía por bluetooth.

3.7. DISEÑO DE UNA PLACA GENERADORA DE PULSOS.

La parte más importante de nuestro código es el contador de eventos, para poder activarlos es necesario de cuatro pulsos pero nuestro sistema consta de un pulso de inyección de 12V que marca el primer cilindro, partiendo de esto se procedió por construir una placa que genere los pulsos necesarios de 5V para evitar que nuestra placa se dañe.

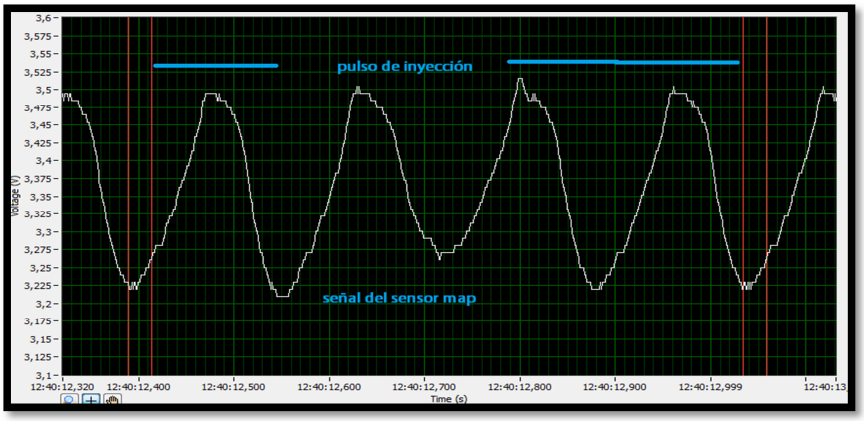


Figura 3.41 Señal Map y Pulso de Inyección

Fuente: Los Autores

3.7.1. PLACA GENERADORA.

Lo primero en realizar fue bajar el pulso de 12V a 5V para que nuestra tarjeta pueda recibirla, para ello usamos a un transistor Darlington Tip 122.

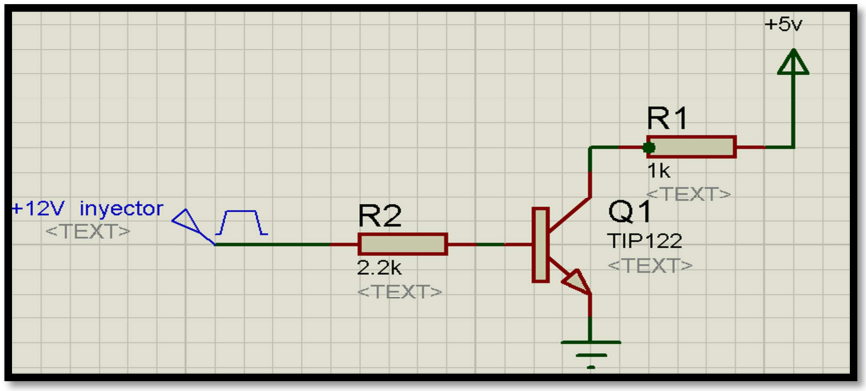


Figura 3.42. Tip 122 Armado en Proteus.

Fuente: Los Autores

El tip 122 soporta voltajes de 100 V y una corriente de 600mA que en nuestro diseño es más que suficiente, el voltaje que queremos manejar es de 12V, para verificar su funcionamiento en proteus dando un valor máximo de 2.5V.

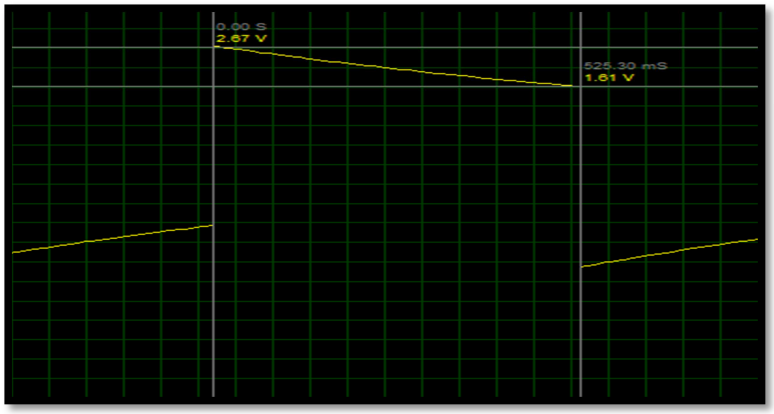


Figura 3.43 Simulación en proteus.

Fuente: Los Autores

La señal generada del inyector es alto por lo cual con el transistor Darlington lo reducimos a un valor comprendido entre 0 – 5 voltios, nuestras interrupciones funcionan cada vez que el pulso sea bajo para esto se usó un chip HFE40106, que es una compuerta NOT que invierte la señal de entrada.

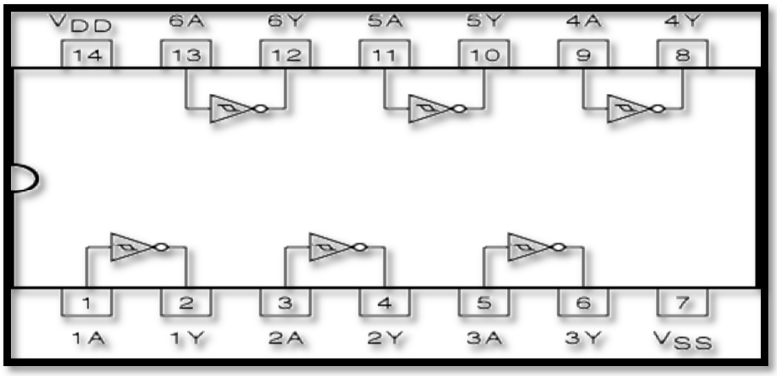


Figura 3.44 Chip HFE40106

Fuente: http://www.datasheetarchive.com

Esta señal invertida se ingresa al pin de alimentación de un circuito integrado 555 usado para generar los pulsos correspondientes simulados para cada cilindro.

3.7.1.1 CIRCUITO INTEGRADO 555.

Este elemento electrónico puede ser configurado de manera monoestable o astable dependiendo las necesidades.

3.7.1.1.1. CIRCUITO MONOESTABLE.

En este caso el circuito genera un solo pulso con un ancho predefinido por el diseñador.

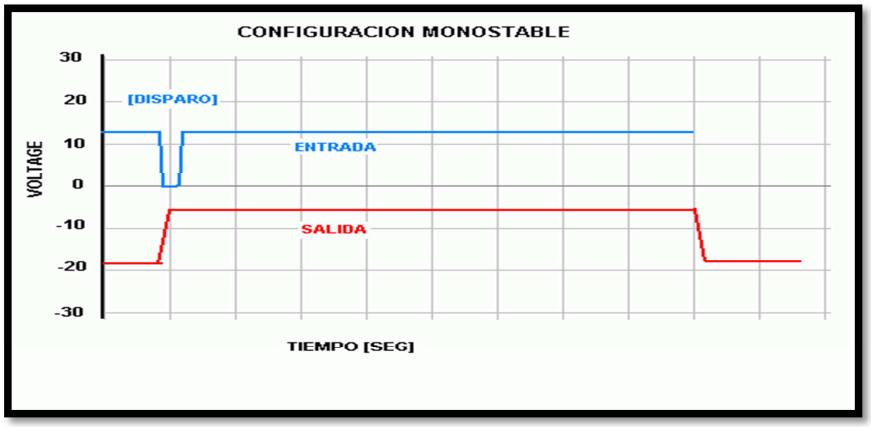


Figura 3.45 Circuito 555 Monoestable

Fuente: http://www.ieespain.com

3.7.1.1.2. CIRCUITO ASTABLE.

Este a diferencia del anterior nos permite generar mayor cantidad de pulsos con un determinado tiempo en estado alto y otro en estado bajo.

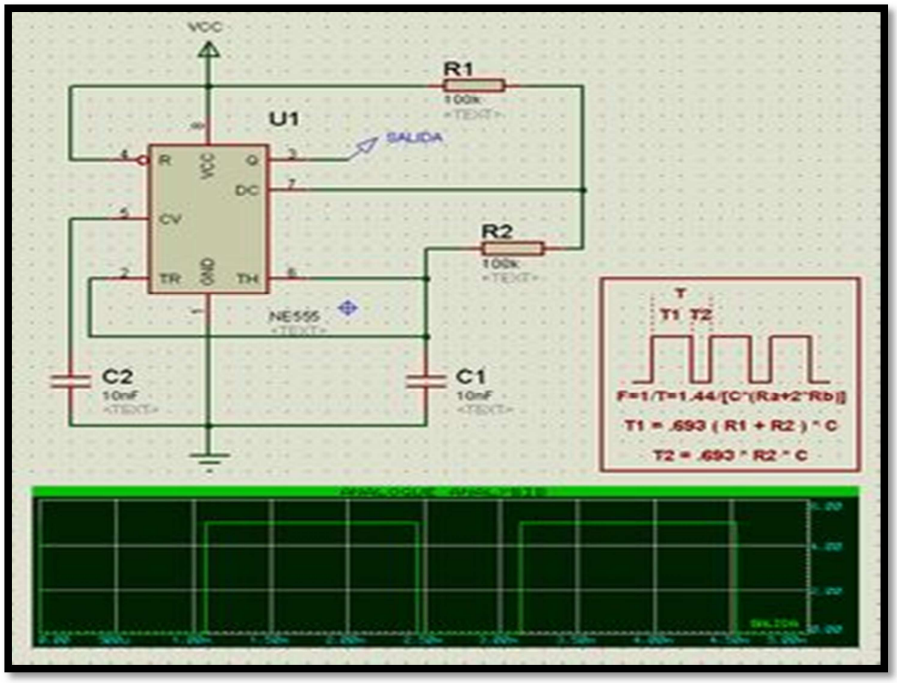


Figura 3.46 Circuito 555 Aestable

Fuente: http://www.ieespain.com

3.7.1.2. CIRCUITO INTEGRADO 555 ASTABLE.

Cuando se realizaron las pruebas en un motor obtuvimos el tiempo de duración de la inyección, siendo en estado alto de *10 ms* y un estado bajo de *30 ms*, usando estos datos se procedió a realizar el cálculo de diseño del circuito.

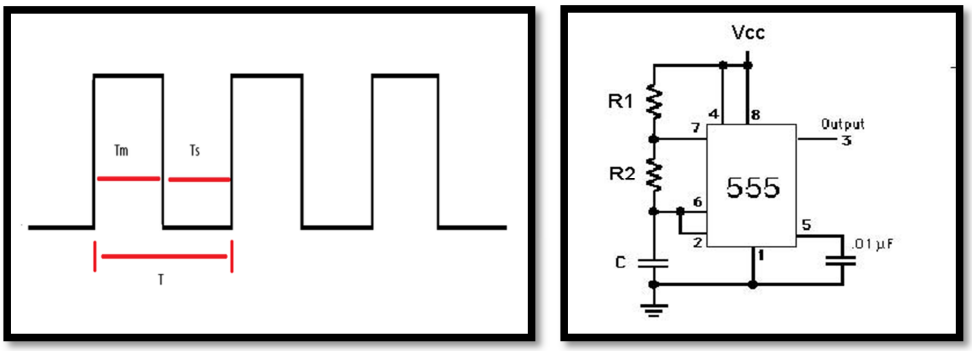


Figura 3.47 Chip 555 configuración.

Fuente: Los Autores

Fórmulas para manejar el 555 de modo astable.

$$T = 0.7 * (R1 + 2R2)C$$

$$F = 1/T$$

$$T = Tm + Ts$$

$$Tm = 0.7 * (R1 + R2) * C$$

$$Ts = 0.7 * R2 * C$$

$$R2 = 0.7/(F * C)$$

Con estas ecuaciones se procedió a realizar los siguientes cálculos y obtuvimos los siguientes datos:

Ts = 10ms.

Tm=30ms

R2=1.2 KΩ

R1=800Ω

C=22uf

Con dichos valores procedimos a realizar nuestro diseño.

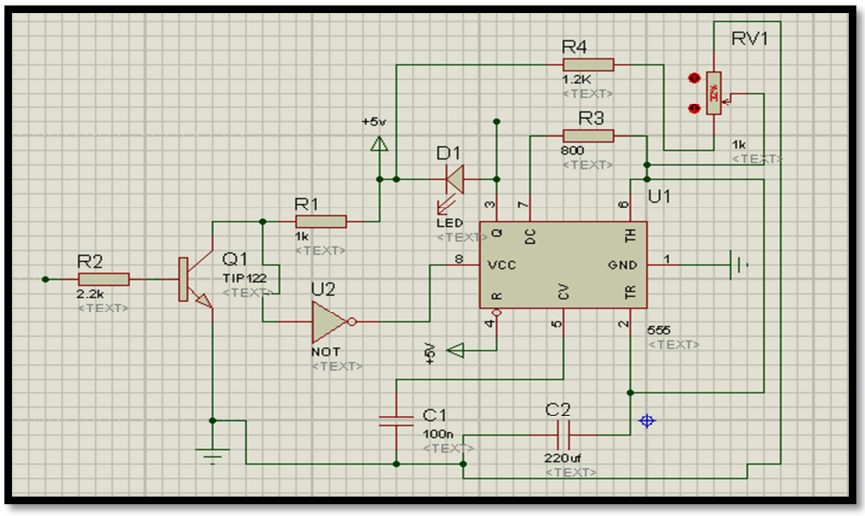


Figura 3.48. Diseño del circuito en Proteus.

Fuente: Los Autores

El trabajo del circuito astable una vez que entra el pulso de inyección es alimentar al condensador, es decir, lo carga dependiendo de las resistencias, vamos a tener más o menos tiempo en un estado alto o bajo del pulso, luego pasa el pulso positivo por el chip NOT y obtenemos un valor negativo y lleva a masa al circuito 555 permitiendo generar los pulsos necesarios.

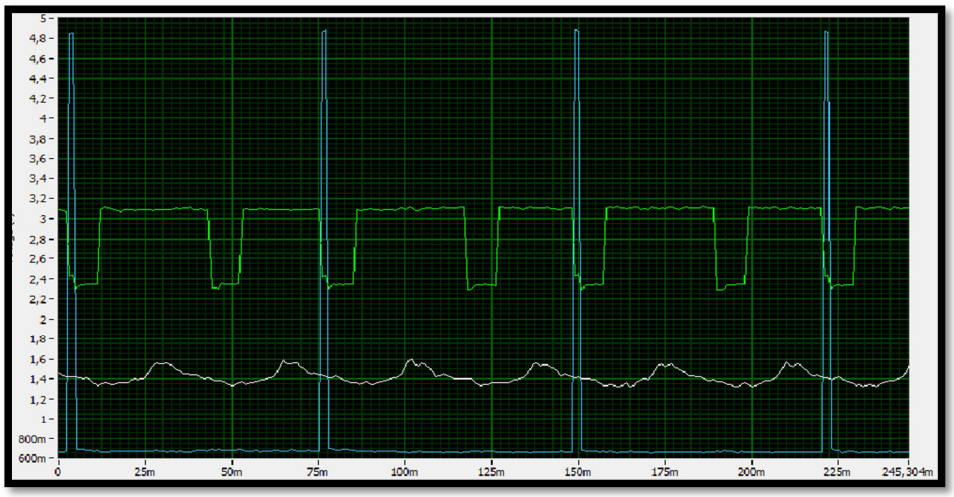


Figura 3.49 Pulsos de Inyección.

Fuente: Los Autores.

Ya funcionando el diseño del circuito en el programa Proteus, realizamos una conexión física de todos los elementos para comprobar su funcionamiento, para ello ensamblamos todos los elementos usados en un project board.

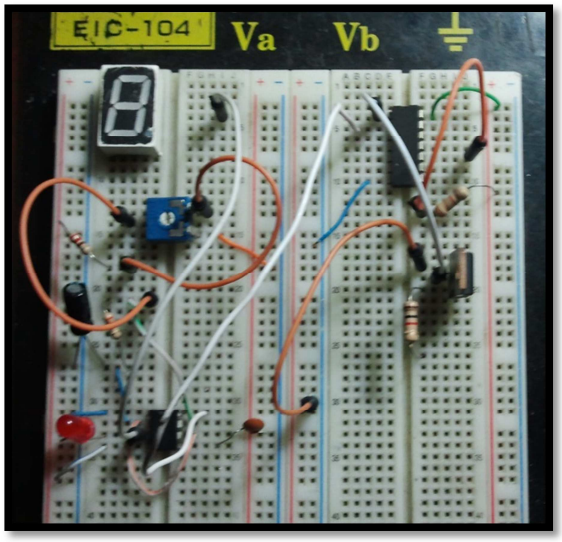


Figura 3.50. Circuito armado en Project Board.

Fuente: Los Autores.

Ya realizada la comprobación del circuito armado en el Project board y seguros de su funcionalidad procedimos a construir el circuito en una placa perforada.

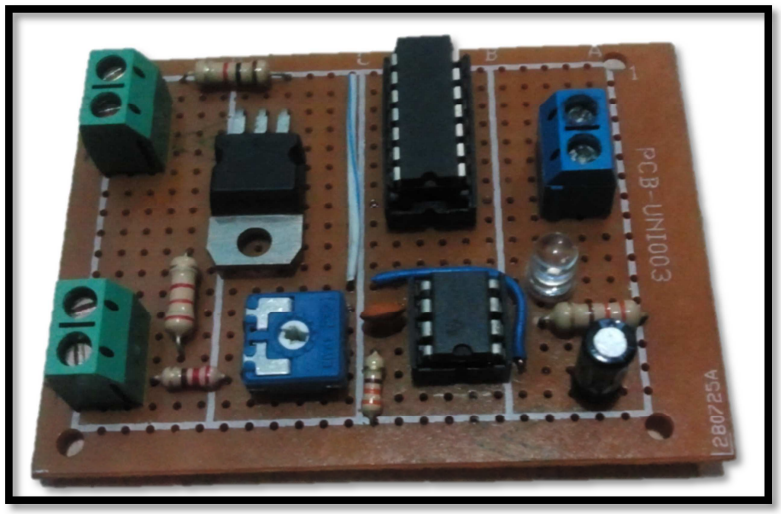


Figura 3.51. Circuito armado en una placa perforada.

Fuente: Los Autores.

CAPÍTULO IV

ANÁLISIS DE RESULTADOS DE FUNCIONAMIENTO DEL EQUIPO DE DIAGNÓSTICO EN VEHÍCULOS DOTADOS CON EL SISTEMA MPFI MULTEC DELPHI.

CAPÍTULO IV

4. ANÁLISIS DE RESULTADOS DE FUNCIONAMIENTO DEL EQUIPO DE DIAGNÓSTICO EN VEHÍCULOS DOTADOS CON EL SISTEMA MPFI MULTEC DELPHI.

4.1. INTRODUCCIÓN.

Finalizada la construcción del equipo de diagnóstico, se procedió a realizar algunas pruebas sobre su funcionamiento, esto se realizó en tres vehículos, con el fin de determinar errores y así corregirlos.

4.2. CREANDO UN PATRÓN.

Al momento de generar un patrón para el diagnóstico en otros vehículos dotados con el mismo sistema de inyección, hicimos una recolección de datos de tres vehículos, al poseer todos el sistema MPFI MULTEC DELPHI los datos deben ser los mismos.

4.3. RECOLECTANDO DATOS CHEVROLET CORSA.

Este vehículo de la casa comercial Chevrolet es un motor con un cilindraje 1364 cm^3, con una relación de compresión de **10:1** que genera una presión de compresión de **145 PSI,** con sistema de alimentación MPFI MULTEC DELPHI**.**

Figura 4.1. Chevrolet corsa.

Fuente: Los Autores.

Se procedió a recolectar los datos correspondientes de voltaje que nos genera el sensor MAP en este vehículo, los mismos que son:

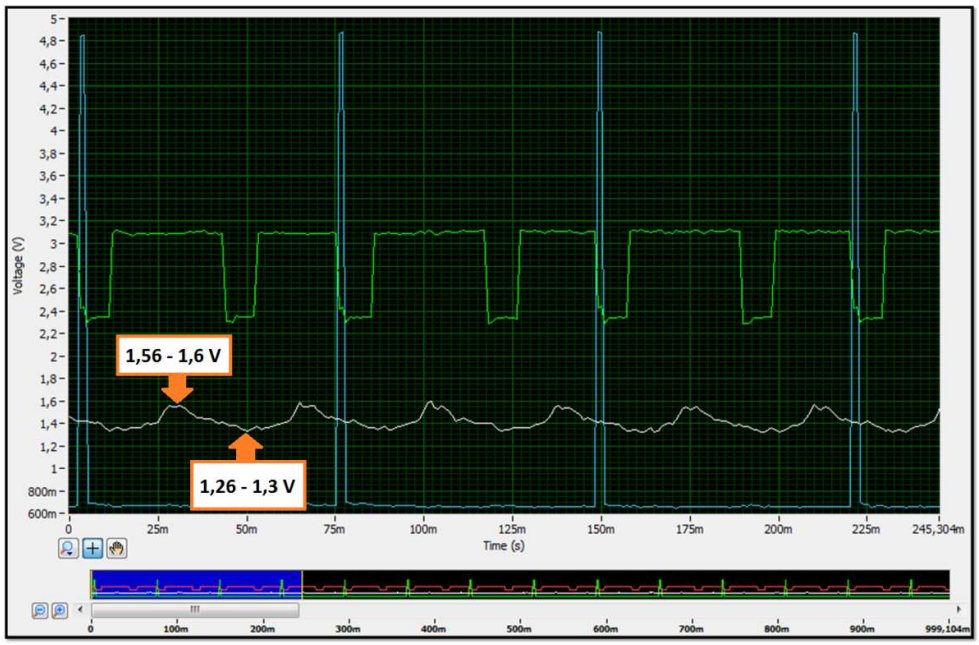


Figura 4.2. Voltaje sensor MAP Chevrolet corsa.

Fuente: Los autores.

Los valores adquiridos de voltaje tal como se indica en la figura anterior son:

Voltaje Máximo	1,56 V.	1,6 V.
Voltaje Mínimo	1,26 V.	1.3 V.

Tabla 4.1. Voltaje Sensor MAP Chevrolet corsa.

Fuente: Los autores.

Estos valores de voltaje los transformamos en bytes, para que los reconozca el programa teniendo los siguientes:

Bits Máximo	80	320
Bits Mínimo	65	260

Tabla 4.2. Bits referentes al Sensor MAP Chevrolet corsa.

Fuente: Los autores.

Mientras que los valores de compresión de cada cilindro es de:

# De Cilindro	Presión PSI.
Cilindro 1	120
Cilindro 2	123
Cilindro 3	118
Cilindro 4	120

Tabla 4.3. Presión de cada cilindro Chevrolet corsa.

Fuente: Los autores.

Al comparar con los datos del fabricante se tiene una diferencia de 30 PSI, lo que indica que este motor ya tiene un desgaste considerado, para comprobar con los datos que nos entrega el equipo de diagnóstico se procedió a conectar el equipo de

diagnóstico donde realizamos un análisis estático y se obtuvieron los siguientes resultados.

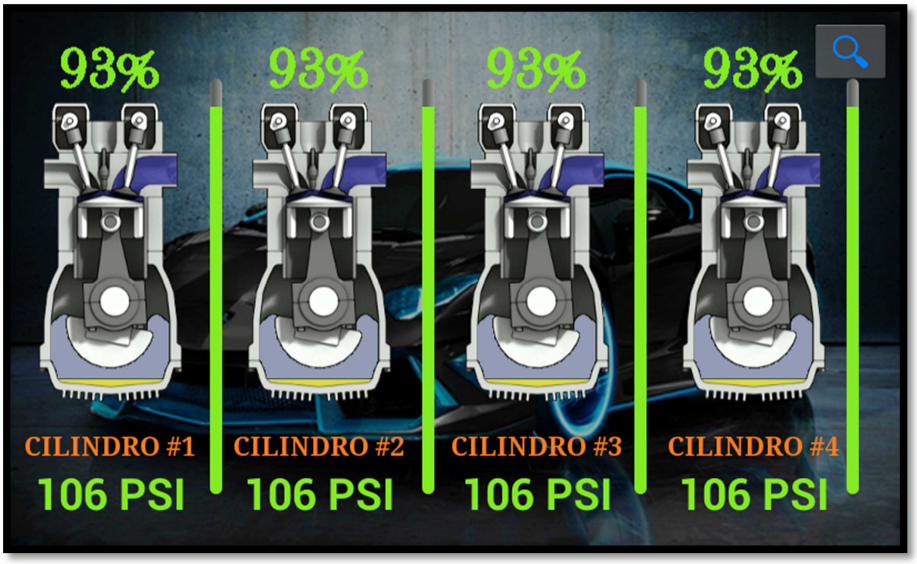


Figura 4.3. Voltaje sensor MAP Chevrolet corsa.

Fuente: Los autores.

Así mismo se realizó un análisis dinámico y obtuvimos los siguientes datos:

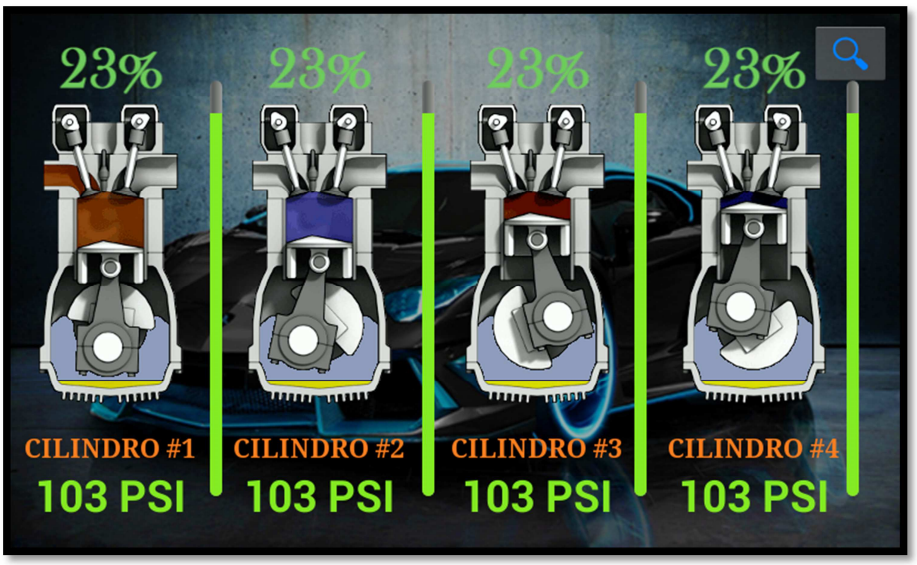


Figura 4.4. Voltaje sensor MAP Chevrolet corsa.

Fuente: Los autores.

Estos datos obtenidos en ambos análisis nos da un valor de porcentaje que indica que el motor se encuentra en buen estado, pero al analizar las presiones obtenidas con las del fabricante notamos que están muy bajo por el límite, indicando que el motor está muy próximo a una reparación, se realizó una comparación de los datos obtenidos con el equipo conjuntamente con los valores obtenidos al medir con el manómetro.

PRESIÓN MANÓMETRO (PSI)	PRESIÓN EQUIPO DE DIAGNOSTICO (PSI)
120	106
123	106
118	106
120	106

Tabla 4.4. Presión Manómetro vs Presión del equipo.

Fuente: Los autores.

4.4. RECOLECTANDO DATOS CHEVROLET LUV.

Este vehículo de la casa comercial Chevrolet es un motor con un cilindraje 2200 cm^3, con una relación de compresión de **10:1** que genera una presión de compresión de **145 PSI,** con sistema de alimentación MPFI MULTEC DELPHI**.**

Figura 4.5. Chevrolet Luv.

Fuente: Los autores.

Recolectamos los siguientes datos correspondientes al voltaje que nos genera el sensor MAP de este vehículo:

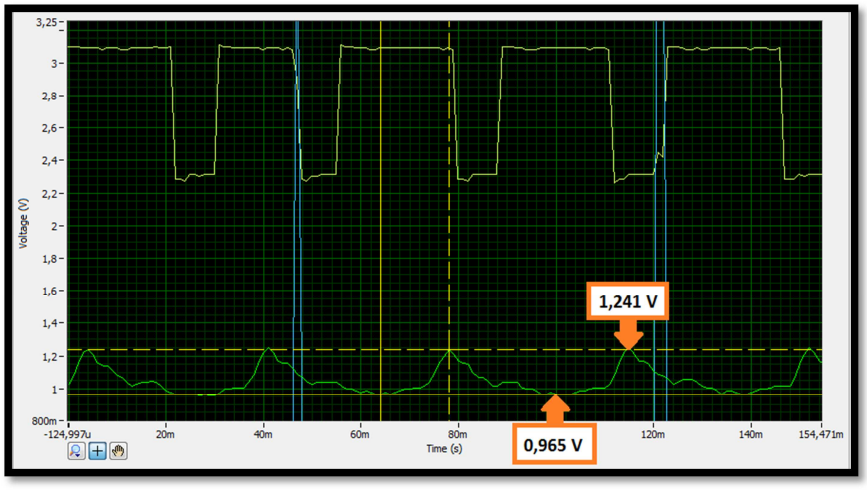


Figura 4.6. Voltaje sensor MAP Chevrolet Luv.

Fuente: Los autores.

Los valores adquiridos de voltaje tal como se indica en la figura anterior son:

Voltaje Máximo	1,241 V.
Voltaje Mínimo	0,965 V.

Tabla 4.5. Voltaje Sensor MAP Chevrolet Luv.

Fuente: Los autores.

Estos valores de voltaje los transformamos en bytes, para que los reconozca el programa teniendo los siguientes:

Bits Máximo	65	260
Bits Mínimo	50	200

Tabla 4.6. Bits referentes al Sensor MAP Chevrolet Luv.

Fuente: Los autores.

Mientras que los valores de compresión de cada cilindro es de:

# De Cilindro	Presión PSI.
Cilindro 1	100
Cilindro 2	118
Cilindro 3	115
Cilindro 4	115

Tabla 4.7. Presión de cada cilindro Chevrolet Luv.

Fuente: Los autores.

Al comparar con los datos del fabricante se tiene una diferencia de 30 PSI aproximadamente, lo que indica que este motor ya tiene un desgaste considerado, para comprobar con los datos que nos entrega el equipo de diagnóstico se procedió a

conectar el equipo de diagnóstico donde realizamos un análisis estático y se obtuvieron los siguientes resultados.

Figura 4.7. Voltaje sensor MAP Chevrolet Luv.

Fuente: Los autores.

Así mismo se realizó un análisis dinámico y obtuvimos los siguientes datos:

Figura 4.8. Voltaje sensor MAP Chevrolet Luv.

Fuente: Los autores.

Estos datos obtenidos en ambos análisis el dato obtenido de porcentaje indican que el motor se encuentra en un estado regular por así decirlo, pero al analizar las presiones obtenidas están muy bajo por el límite, indicando que el motor está muy próximo a una reparación, se comprobó los datos obtenidos con el equipo juntamente con los valores obtenidos al medir con el manómetro, la diferencia entre presiones es de unos 20 PSI aproximados.

PRESIÓN MANÓMETRO (PSI)	PRESIÓN EQUIPO DE DIAGNOSTICO (PSI)
100	83
118	90
115	90
115	90

Tabla 4.8. Presión Manómetro vs Presión del equipo.

Fuente: Los autores.

4.5. RECOLECTANDO DATOS CHEVROLET AVEO.

Este vehículo de la casa comercial Chevrolet es un motor con un cilindraje 1598 cm^3, con una relación de compresión de **10:1** que genera una presión de compresión de **145 PSI,** con sistema de alimentación MPFI MULTEC DELPHI**.**

Figura 4.9. Chevrolet Aveo.

Fuente: Los Autores.

Se procedió a recolectar los datos correspondientes de voltaje que nos genera el sensor MAP en este vehículo, los mismos que son:

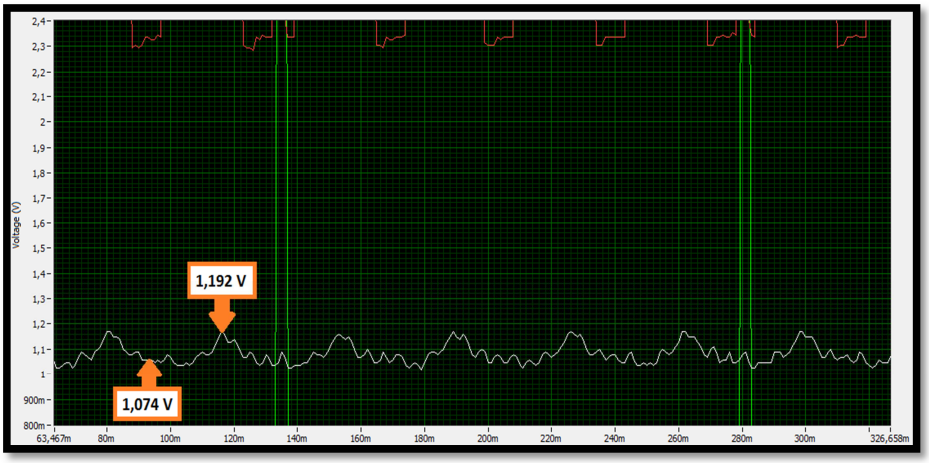


Figura 4.10. Voltaje sensor MAP Chevrolet Aveo.

Fuente: Los Autores.

Los valores adquiridos de voltaje tal como se indica en la figura anterior son:

Voltaje Máximo	1,192 V.
Voltaje Mínimo	1,074 V.

Tabla 4.9. Voltaje Sensor MAP Chevrolet Aveo.

Fuente: Los Autores.

Estos valores de voltaje los transformamos en bytes, para que los reconozca el programa teniendo los siguientes:

Bits Máximo	61	245
Bits Mínimo	55	220

Tabla 4.10. Bits referentes al Sensor MAP Chevrolet Aveo.

Fuente: Los Autores.

Mientras que los valores de compresión de cada cilindro es de:

# De Cilindro	Presión PSI.
Cilindro 1	145
Cilindro 2	145
Cilindro 3	145
Cilindro 4	145

Tabla 4.11. Presión de cada cilindro Chevrolet Aveo.

Fuente: Los Autores.

Al comparar con los datos del fabricante no presenta diferencia alguna, lo que indica que este motor está en buenas condiciones, para comprobar con los datos que nos

entrega el equipo de diagnóstico se procedió a conectar el equipo de diagnóstico donde realizamos un análisis estático y se obtuvieron los siguientes resultados.

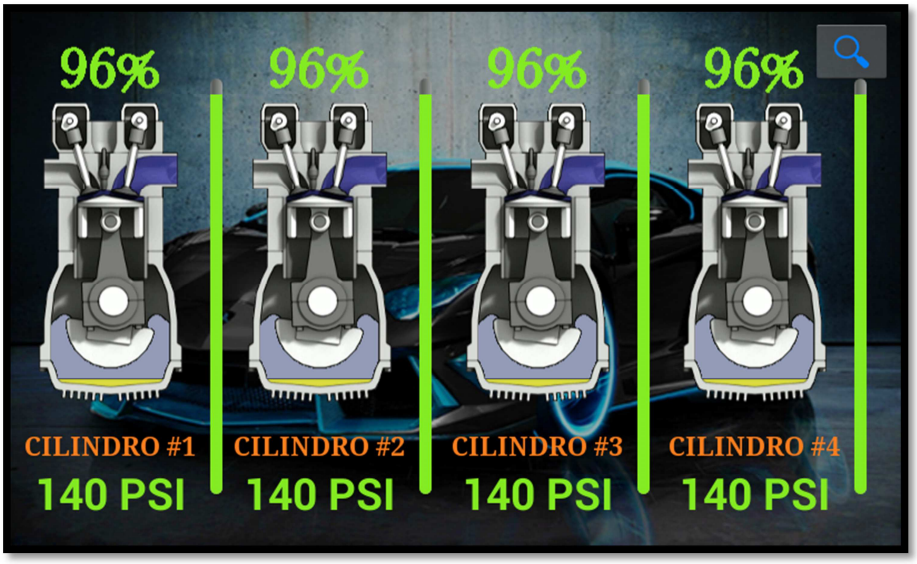


Figura 4.11. Voltaje sensor MAP Chevrolet Aveo.

Fuente: Los Autores.

Así mismo se realizó un análisis dinámico y obtuvimos los siguientes datos:

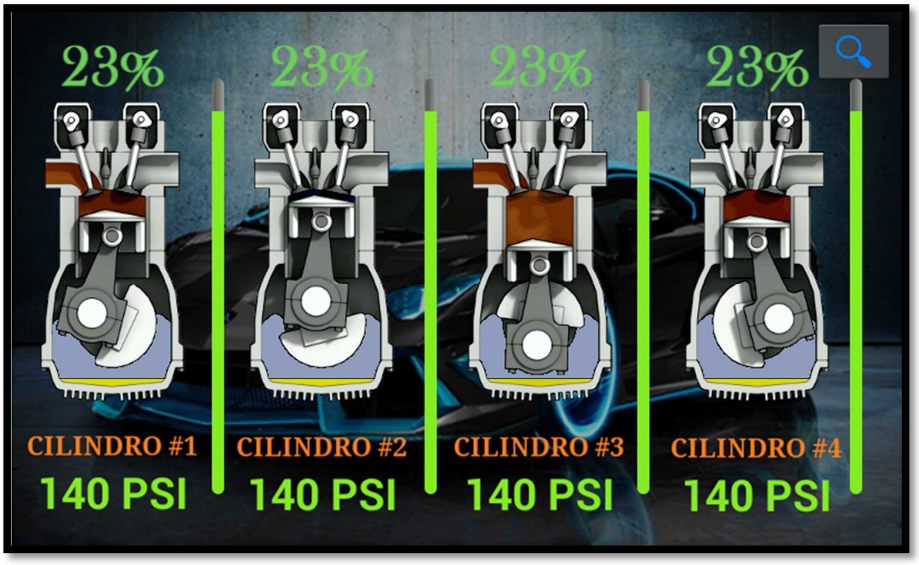


Figura 4.12. Voltaje sensor MAP Chevrolet Aveo.

Fuente: Los Autores.

Estos datos obtenidos en ambos análisis, el valor de porcentaje nos indica que el motor se encuentra en buen estado, y no requiere de una reparación, se realizó una comparación de los datos obtenidos con el equipo conjuntamente con los valores obtenidos al medir con el manómetro, y se tuvo los siguientes datos:

PRESIÓN MANÓMETRO (PSI)	PRESIÓN EQUIPO DE DIAGNOSTICO (PSI)
145	140
145	140
145	140
145	140

Tabla 4.12. Presión Manómetro vs Presión del equipo.

Fuente: Los Autores.

4.6. RECOPILACIÓN DE DATOS GENERADOS POR EL EQUIPO DE DIAGNÓSTICO.

Una vez finalizada las pruebas en los vehículos mencionados anteriormente se procedió a realizar unas gráficas de dispersión para visualizar los resultados obtenidos de medir la compresión de cada motor usando el manómetro.

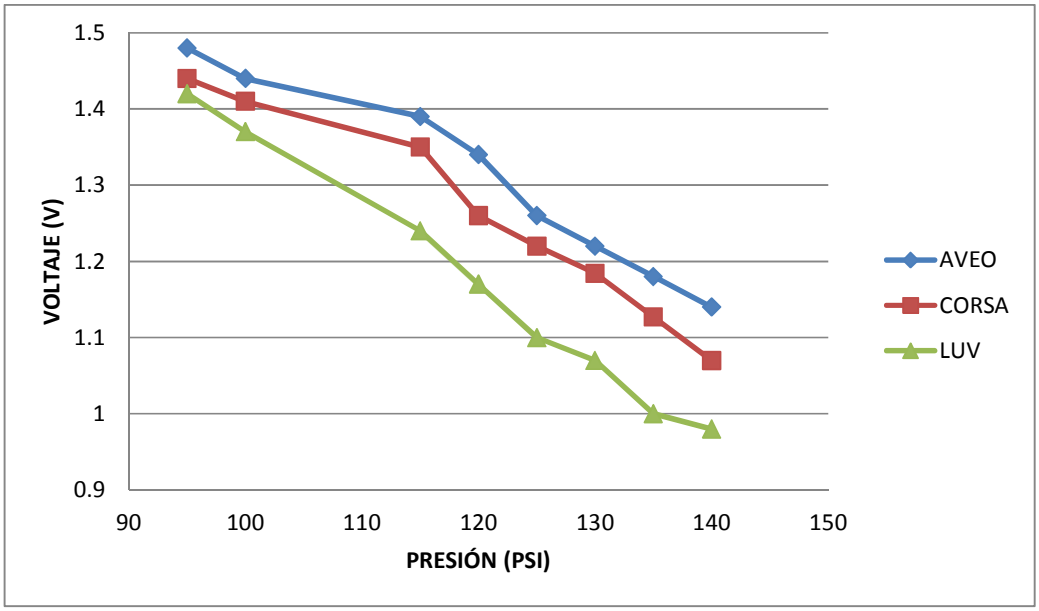


Figura 4.13. Gráfica de dispersión de resultados con el manómetro.

Fuente: Los Autores.

Como apreciamos en esta grafica de dispersión el motor del vehículo Aveo tiene mayor voltaje según la presión que posee su cilindro, le sigue el motor de vehículo corsa y por último el motor del vehículo luv, es por tal motivo que el equipo de diagnóstico posee un determinado porcentaje de error, pues lo que se trata de hacer es ajustar las curvas al mismo voltaje con el fin de determinar que cilindro posee menor presión.

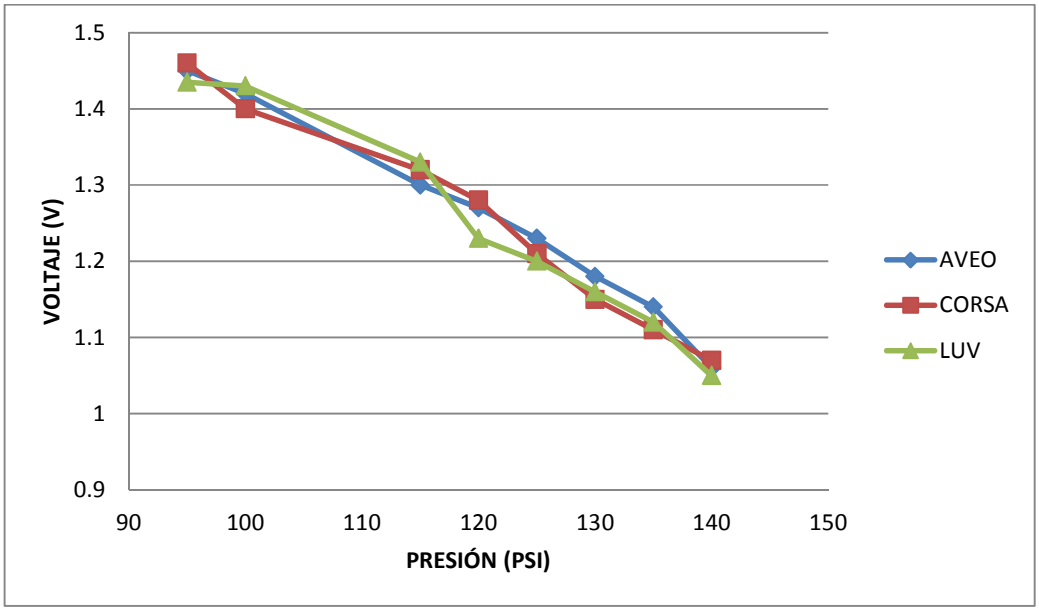


Figura 4.14. Gráfica de dispersión de resultados Equipo.

Fuente: Los Autores.

Como se mencionó anteriormente al equipo de diagnóstico se realizó un ajuste de curvas con la finalidad de tener un patrón para varios vehículos, siempre que posean sistema de alimentación MPFI MULTEC DELPHI, obviamente teniendo un margen de error tolerable, el cual puede ser reducido al seguir profundizando en la investigación y generar una base de datos de cada vehículo que se desea analizar.

Es así como finalizan las pruebas para verificar la veracidad del equipo de diagnóstico.

CAPITULO V

VALIDACIÓN DEL EQUIPO DE DIAGNÓSTICO MEDIANTE CÁLCULO MATEMÁTICO, ESTADÍSTICO Y COMPARACIÓN CON OTROS EQUIPOS ANALÓGICOS.

CAPITULO V

5. VALIDACIÓN DEL EQUIPO DE DIAGNÓSTICO.

5.1. INTRODUCCIÓN.

Todo equipo antes de introducirse al mercado debe ser comprobado realizando las pruebas de campo o validación con otros equipos de alta o media gama, además de esto debe ser justificado su diseño antes de hacerlo público.

5.2. PRUEBAS DE CAMPO.

La comprobación de nuestro equipo se realizó en los vehículos con los sistemas MPFI MULTEC DELPHI para saber el estado en el que se encuentran, específicamente los vehículos en los que realizamos las pruebas fueron en una camioneta Chevrolet Luv con motor de 2200 cm^3, un Corsa con motor de 1400 cm^3 y un Aveo Emotion con motor de 1600 cm^3.

Figura 5.1. Vehículos Chevrolet Corsa, Luv, Aveo.

Fuente: Los Autores.

Para la comprobación de nuestro equipo usamos un manómetro para saber la presión de los cilindros del motor, el valor estaba en un promedio de 100 a 115 psi con una diferencia de 15 PSI aproximado entre los distintos cilindros del motor.

# De Cilindros	Presión(PSI)
1	100
2	118
3	115
4	115

Tabla 5.1: Presión de los cilindros.

Fuente: Los Autores

5.3 VALIDACIÓN.

5.3.1 VALIDACIÓN CON EL MANÓMETRO.

Con el valor de presión de cada cilindro de la camioneta Chevrolet Luv procedimos a comprobar con los valores obtenidos desde nuestro equipo, estos datos obtenidos eran demasiado altos, de tal manera que partiendo desde este punto procedimos a recalibrar el dispositivo, nuevamente estudiamos el estado de la curva enviada por el sensor MAP de cada uno de los motores y así determinar en qué punto se deben activar los eventos de la adquisición de datos para enviar el valor correspondiente al celular.

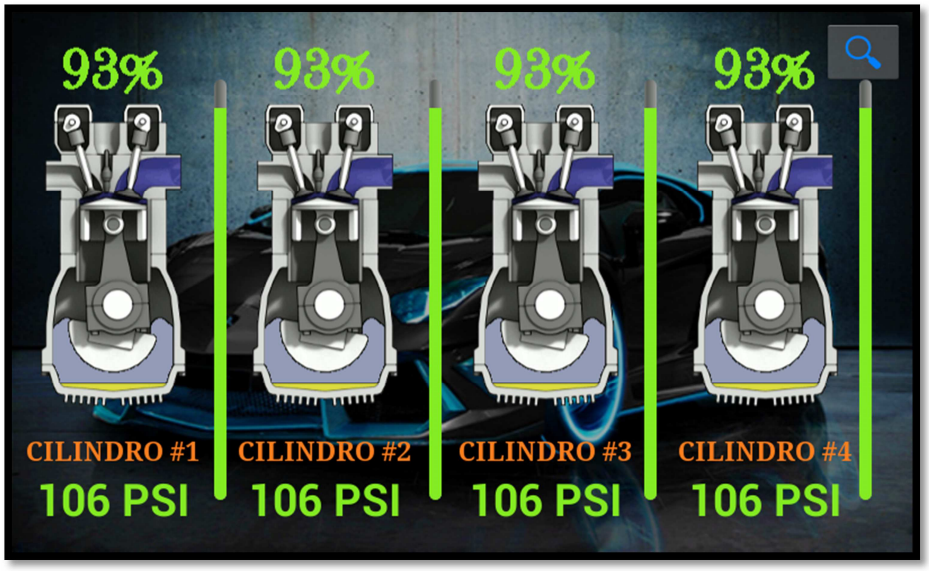


Figura 5.2. Datos iniciales del equipo.

Fuente: Los Autores.

Datos del sensor MAP recolectados para calibrar el equipo de diagnóstico.

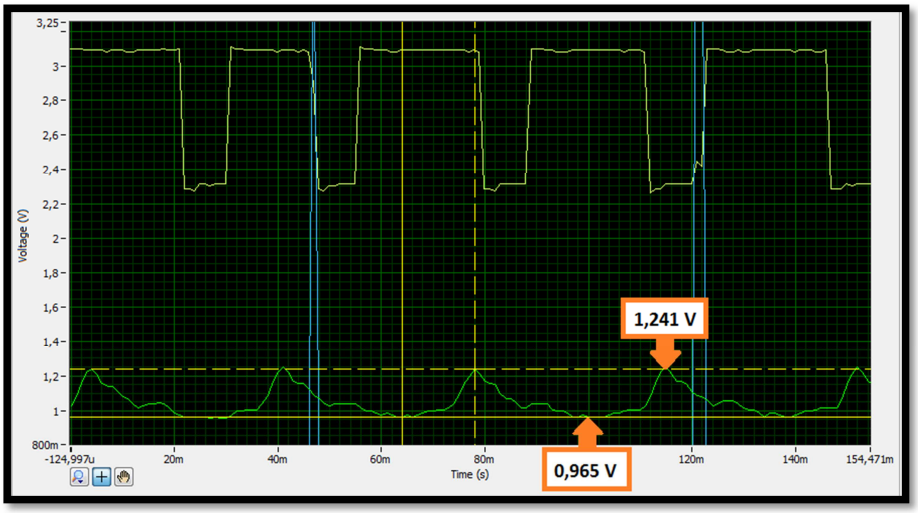


Figura 5.3. Señal del sensor MAP (Luv 2.2).

Fuente: Los Autores.

Figura 5.4. Presión del Cilindro.

Fuente: Los Autores.

Observamos en la curva del sensor MAP sus valores comprendidos entre 0,965 – 1,241 V, siendo el voltaje más bajo la presión que poseen los cilindros, obteniendo los siguientes valores aproximados:

Cilindros	Presión(psi)	Voltaje(V)
1	100	1
2	118	1
3	115	1
4	115	1

Tabla 5.2 Presión – Voltaje.

Fuente: Los Autores

Con estos valores de presión reajustamos nuestro dispositivo para que nos muestre la presión real aproximada de cada uno de los cilindros.

Figura 5.5. Datos calibrados del equipo.

Fuente: Los Autores.

5.3.2 VALIDACIÓN MATEMÁTICA.

La manera de diferenciar la exactitud de cualquier proyecto que pasa de ser analógico a dispositivo digital es mediante una demostración matemática, para obtener una ecuación matemática fue necesario los valores de presión y número de bits promedio en cada uno de los cilindros de cada motor.

Para obtener el número de bits de un sistema debemos tomar en cuenta la resolución del convertidor analógico digital, para el caso de nuestra tarjeta arduino cada pin tiene una resolución de 10 bits, es decir toma los valores de entrada de 0 a 5 voltios para el cual tendremos un valor de 0000000000 igual a cero decimal para una entrada de 0 voltios y de 1111111111 igual a 1023 decimal para una entrada de 5 voltios, estos valores en función a los vehículos analizados son:

	Motor 2.2 cm^3	Motor 1.4 cm^3	Motor 1.6 cm^3
Voltaje(V)	1,4	1.26	1.074
Presión(psi)	100	120	145

Tabla 5.3 Voltaje –Presión

Fuente: Los Autores.

La ecuación formulada para obtener el número de bits dependiendo del voltaje que tenemos en la entrada es de:

$$b = \frac{Vent.* 1024}{5}$$

Donde b son los bits del sistema, *Vent.*es el voltaje de entrada en nuestro sensor MAP, *1024* es el tamaño de la muestra del convertidor analógico digital y *5* el voltaje de alimentación del sistema.

Teniendo esto en cuenta podemos decir que la cantidad de bits son:

	Motor 2.2 cm^3	Motor 1.4 cm^3	Motor 1.6 cm^3
Presión(psi)	100	120	145
Bits	287	258	220

Tabla 5.4 Presión – Número de Bits del Sistema.

Fuente: Los Autores.

Valores que fueron introducidos en el CFTOOL de MATLAB, básicamente es una herramienta de aproximación por series numéricas, que nos facilita una ecuación de segundo grado que podrá ser usada para los sensores MAP de cualquier vehículo siempre y cuando se determine el número de bits que ingresa al sistema.

$$P = -0.002185 * b^2 + 0.424 * b + 157.5$$

Donde P es el valor de la presión de los cilindros y b el número de bits que envía el sistema.

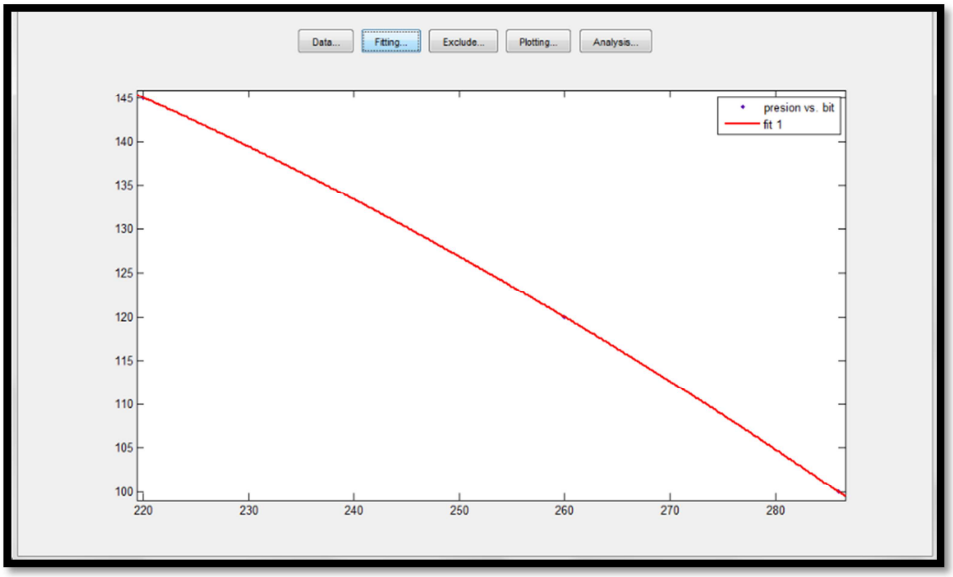


Figura 5.6. Presión en función del número de bits.

Fuente: Los Autores.

5.4 DEMOSTRACIÓN PORCENTUAL DEL SISTEMA.

El sistema a más de mostrar los valores de presión de cada cilindro, muestra su estado porcentual para poder diagnosticar y evitar sacar relaciones entre ellos, facilitando el diagnóstico y la lectura de la información ante cualquier persona no entendida en nuestra profesión.

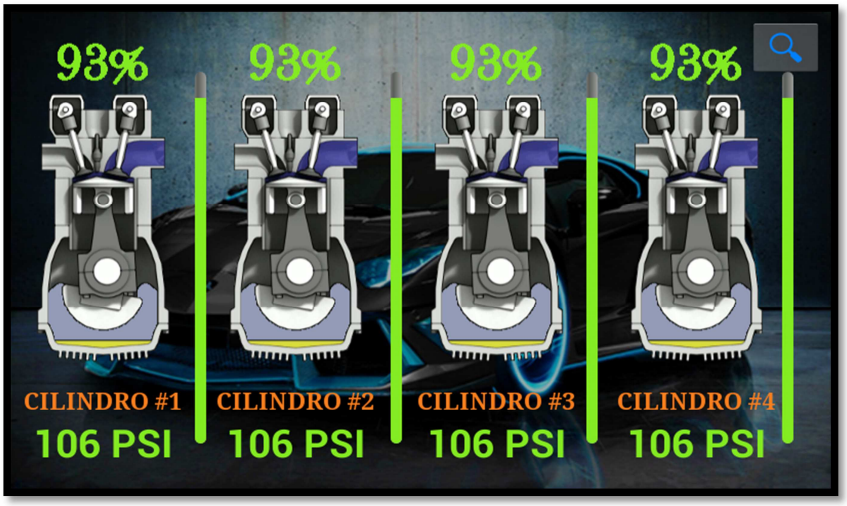


Figura 5.7. Estado porcentual de los cilindros.

Fuente: Los Autores.

Estas relaciones son sacadas en función de la presión de cada cilindro para lo cual cada uno debe tener una presión aproximada de 145 psi equivalente a un 100%, el cual difiere según el estado en el que se encuentre los cilindros.

5.5. MODELADO MATEMÁTICO.

La demostración principal de nuestro proyecto se realiza mediante un modelado matemático realizado en la herramienta SIMULINK de MATLAB, este modelado puede ser representado en cualquier otro motor, el principal objetivo del modelado matemático del sistema de admisión es para verificar la variación del flujo de aire a través del ducto de admisión.

Para que se produzca una buena combustión del aire – combustible, la relación estequiométrica debe ser de 14.7 : 1, por tanto la masa y la velocidad de aire que ingresa en los cilindros es de gran importancia en el funcionamiento de los motores de combustión interna, para analizar el flujo másico como la presión en nuestro colector de admisión se debe tomar en cuenta el material del cual es diseñado el colector, pudiendo ser de aluminio, plástico u otro material, además de la capacidad de succión del pistón.

Los conductos de admisión anchos y cortos repercuten favorablemente en la admisión en altas rpm, mientras que los largos y delgados mejoran la admisión de bajas rpm.

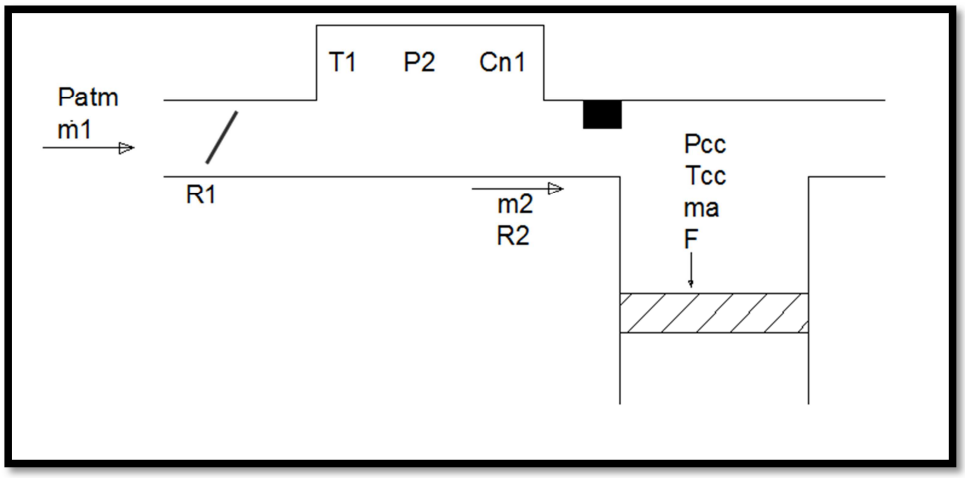


Figura 5.8. Representación del sistema de admisión de aire.

Fuente: Los Autores

La grafica hace relación a un colector de admisión con su respectivo cilindro.

$$R = \frac{d(\Delta P)}{dQ}$$

$$R_2 = \frac{\dot{m}_1 - \dot{m}_2}{Q}$$

$$\sum \dot{m} = C_n \frac{dP}{dt}$$

$$C d\dot{m}_2 = Q dt$$

$$Q = C \frac{d\dot{m}_2}{dt}$$

$$R_2 = \frac{\dot{m}_1 - \dot{m}_2}{C \frac{d\dot{m}_2}{dt}}$$

$$R_2 * C * D * \dot{m}_2 - \dot{m}_1 + \dot{m}_2 = 0$$

$$R_2 * Cs * \dot{m}_2(s) - \dot{m}_1(s) + \dot{m}_2(s) = 0$$

$$\dot{m}_2(s) = \frac{\dot{m}_1}{R_2 Cs + 1}(s)$$

$$\dot{m}_2(s) = \frac{\dot{m}_1}{R_2 \frac{V}{RT} s + 1}(s)$$

Dónde:

$\dot{\boldsymbol{m}}$: $Flujo\ másico$.

P: Presión.

R: Resistencia neumática.

C: Capacitancia neumática.

T: Temperatura.

V: Volumen.

Donde el flujo másico $\dot{m}_2$ se dirige hacia la cámara de combustión y según sus condiciones aportará de una manera que garantice su desempeño de la combustión.

Teniendo estos datos en cuenta nuestras ecuaciones de flujo másico de admisión en el colector y el cilindro son:

$$\dot{m}_2\,(s) = \frac{P_1 - P_2}{R_2}$$

$$P = \dot{m}_2 R_2 + P_2$$

$$DP_1 = \ddot{m}_2 R_2 + DP_2$$

$$\dot{m}_1 - \dot{m}_2 = CDP_1$$

$$\frac{\dot{m}_1 - \dot{m}_2}{C} = DP_1$$

$$PV = mRT$$

$$P_2 = \frac{mRT}{AL}$$

$$DP_2 = \frac{\dot{m}_2 RT}{AL}$$

$$\frac{\dot{m}_1 - \dot{m}_2}{C} = \ddot{m}_2 R_2 + \frac{\dot{m}_2 RT}{AL}$$

$$\dot{m}_1 \dot{-} \dot{m}_2 = C\ddot{m}_2 R_2 + C\frac{\dot{m}_2 RT}{AL}$$

$$L[\dot{m}_1] = L[\dot{C}\ddot{m}_2 R_2 + m_2\{C\frac{RT}{AL} + 1\}]$$

$$m_1 s_1 = m_2[CR_2 s_2 + (C\frac{RT}{AL} + 1)s]$$

$$m_2(s_2) = [\frac{1}{CR_2 s^2 + \left(C\frac{RT}{AL} + 1\right)s}]m_1(s_1)$$

$$\dot{m}_2(s_2) = [\frac{1}{CR_2 s^2 + \left(C\frac{RT}{AL} + 1\right)s}]\dot{m}_1(s_1)$$

$$\frac{\dot{m}_2(s_1)}{\dot{m}_1(s_1)} = [\frac{1}{\frac{V_1}{RT}R_2 s^2 + (\frac{V_1}{AL} + 1)s}]$$

Cabe mencionar también el aire es considerado como un gas ideal en donde la masa molar del aire $M = 28.97\ g/mol$, su constante de gas ideal $R = 8.134 \frac{m^3 * Pa}{°K * mol}$ y presión atmosférica en la ciudad de Cuenca es de 85000 Pa.

Tomando todos estos datos a consideración sacamos el flujo másico a partir de:

$$m = \int \dot{m}\, \mathrm{dt}$$

$$m = \frac{P * V * M}{R * T}$$

$$m = \frac{V * \frac{28.97 g}{mol} * P}{8.314 \frac{m^3 * Pa}{°K * mol} * 290°K}$$

$$\dot{m} = \frac{d}{dt}(V * P * 0.12)$$

$$\dot{m} = (\frac{dV}{dt} P + \frac{dP}{dt} V)0.12$$

Obteniendo la ecuación de transferencia.

$$m(s) = \frac{1}{0.004448s + 1}$$

5.6. ANÁLISIS EN SIMULINK.

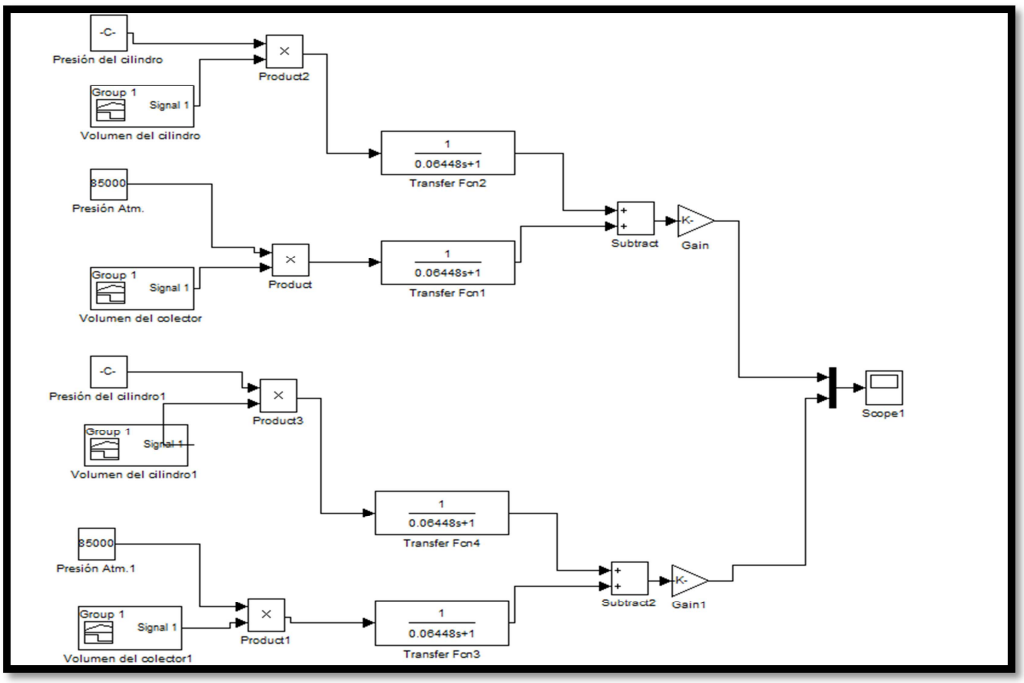


Figura5.9. Modelado Matemático.

Fuente: Los Autores

El análisis dependerá mucho de la cilindrada del motor así como del volumen del colector de admisión, ya que si el colector es más grande, absorberá mayor cantidad de aire incrementando su flujo o si la mariposa está más o menos abierta de igual forma pasa con la cilindrada.

En nuestro caso realizamos la prueba con la presión atmosférica que es la que entra en el colector de admisión siendo una presión en la ciudad de Cuenca 85000 Pa, mientras los valores de volumen son variables por motivos antes presentados

.

Para cual los resultados obtenidos del flujo másico son de:

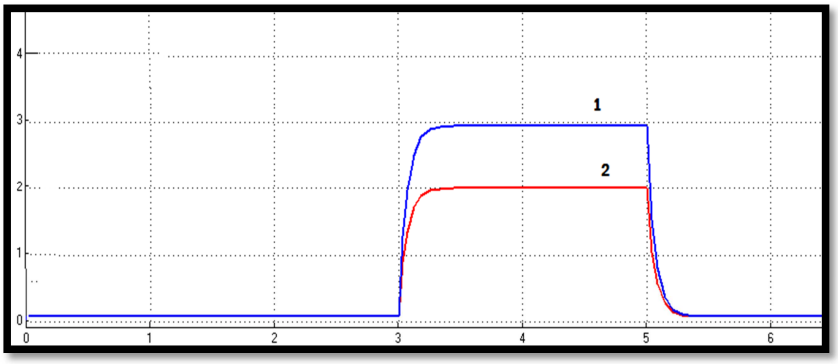


Figura 5.10 Flujo Másico.

Fuente: Los Autores.

En la primera curva se puede apreciar un mayor flujo másico provocado por una mayor cantidad de aire - combustible es de (3g), esto se debe a que la presión absoluta en el interior del cilindro es menor.

Observamos también que en la curva dos el flujo de masa que ingresa al cilindro disminuye a 2g por ciclo, esto es provocado por la depresión generada por el pistón dentro del cilindro.

Este valor de flujo fue analizado con el voltaje que genera el sensor MAP, de un motor Cora 1400cc y un Aveo con motor 1600cc.

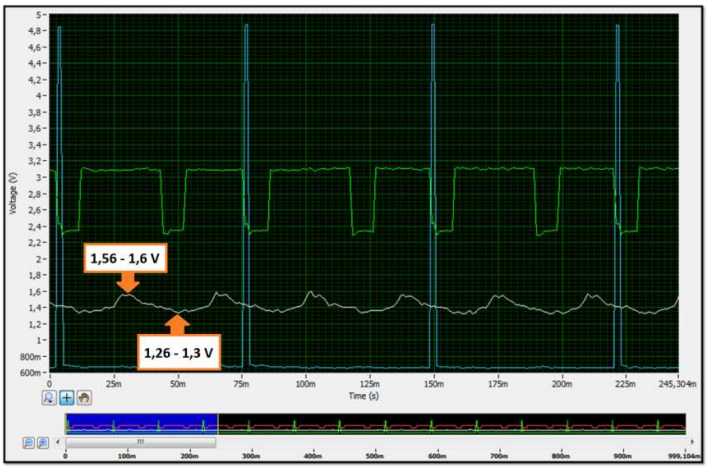


Figura 5.11. Voltaje sensor MAP Chevrolet Corsa.

Fuente: Los Autores.

En este primer motor del análisis podemos apreciar la figura de nuestro sensor Map, pudiendo decir que la válvula de admisión abre en un voltaje de 1.26V, por lo que el flujo másico de nuestro sistema para este vehículo es de 2g por ciclo de admisión.

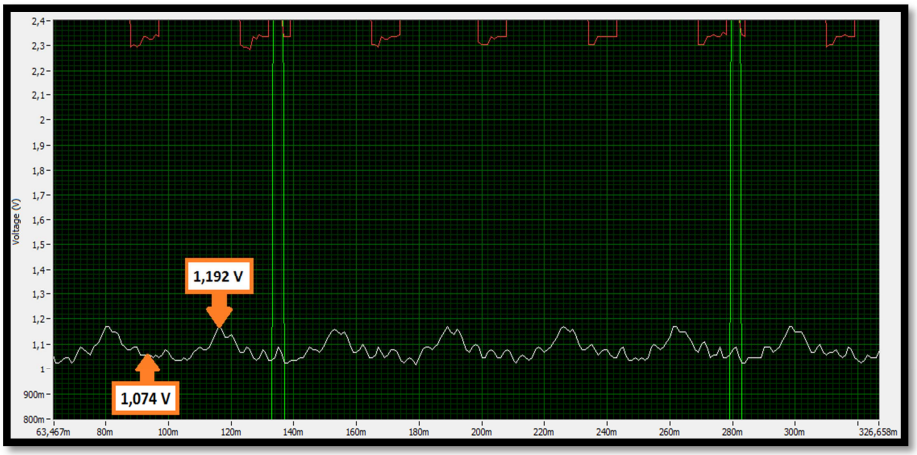


Figura 5.12. Voltaje sensor MAP Chevrolet Aveo.

Fuente: Los autores.

En la señal del sensor Map se puede observar que la válvula de admisión abre en un voltaje de 1.074V, por lo que su flujo másico obtenido en el análisis matemático es de un ingreso de 3g por ciclo de admisión.

Entonces la presión final de la admisión es:

$$\boldsymbol{P = \frac{n * R * T}{Vc}}$$

Dónde:

P: Presión absoluta.

Vc: Volumen del cilindro.

R: Constante universal de los gases ideales.

T: Temperatura absoluta.

n: Número de moles de gas = m/M

m: masa.

M: masa molar.

Para nuestro vehículo Corsa 1400 cc tenemos:

$$\boldsymbol{n = \frac{2g}{28.97g/mol}}$$

$$\boldsymbol{n = 0.069mol}$$

$\boldsymbol{Vc = 3.5^{-4}m^3}$

$\boldsymbol{T \approx 290°k}$

$R = 8.134 \frac{m^3 * Pa}{°K * mol}$

$$P \approx \frac{0.069mol * 8.134 \frac{m^3 * Pa}{°K * mol} * \boldsymbol{290°k}}{\boldsymbol{3.5^{-4}cm^3}}$$

$$P \approx 67.42psi$$

Esta presión difiere mucho por el motivo de una aproximación de temperatura.

Para nuestro vehículo Aveo 1600cc tenemos:

$$\boldsymbol{n = \frac{3g}{28.97g/mol}}$$

$$\boldsymbol{n = 0.1035mol}$$

$\boldsymbol{Vc = 4^{-5}m^3}$

$\boldsymbol{T \approx 290°k}$

$R = 8.134 \frac{m^3 * Pa}{°K * mol}$

$$P \approx \frac{0.1035mol * 8.134 \frac{m^3 * Pa}{°K * mol} * \boldsymbol{290°k}}{\boldsymbol{4^{-4}cm^3}}$$

$$P \approx 88.5psi$$

Esta presión difiere mucho por el motivo de una aproximación de temperatura.

CONCLUSIONES.

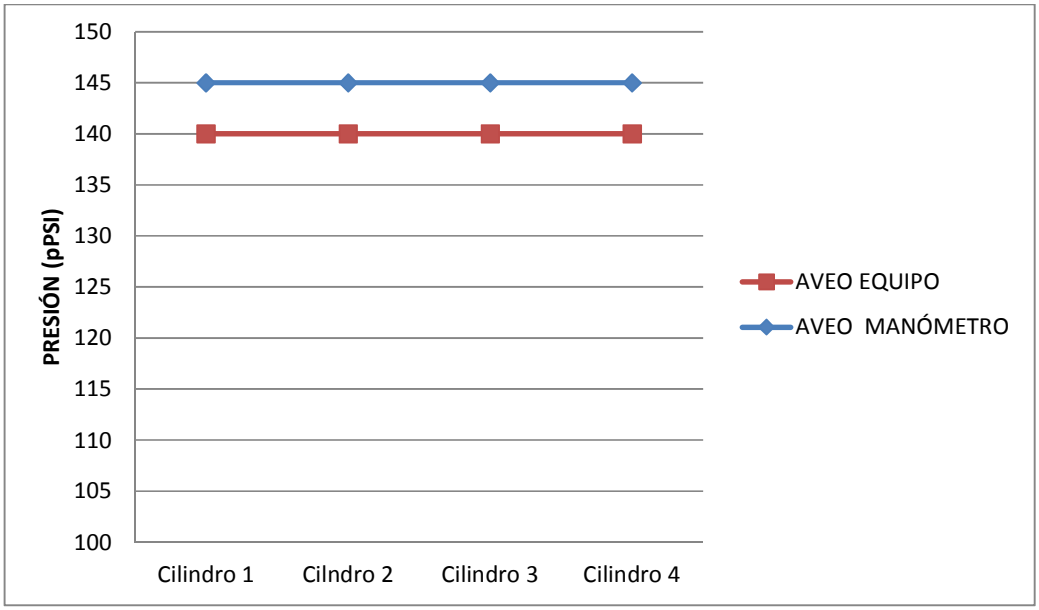
Al finalizar este proyecto podemos concluir lo siguiente:

- Nuestro proyecto de investigación ha tomado en cuenta las necesidades de todas las personas que trabajan en el campo automotriz, ya que es un dispositivo cómodo y de fácil manejo, diseñado para la mayor demanda de vehículos de nuestro mercado como es el caso de la marca Chevrolet dotados con sistema de alimentación MPFI MULTEC DELPHI.

- Analizar el estado de la compresión de cuatro cilindros a la vez dependiendo la eficiencia del motor genera una versatilidad mayor, debido a que nos evitamos sacar las bujías y colocar el manómetro u otro dispositivo de medición tradicional, reduciendo el tiempo de diagnóstico de un motor antes de su reparación.

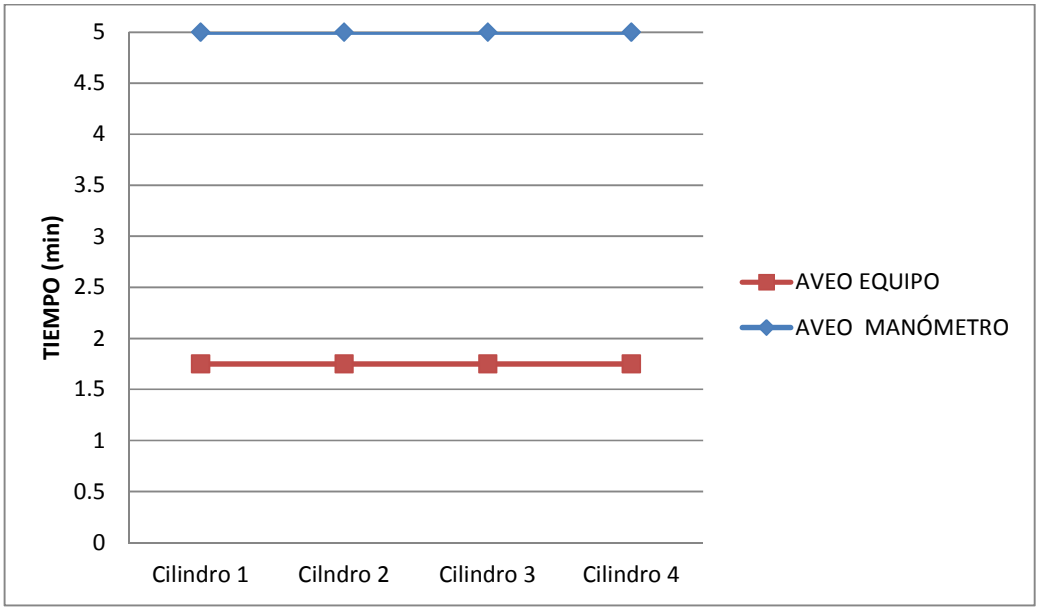
- Al culminar las pruebas en los distintos vehículos podemos comprobar que los resultados entre nuestro diseño con los resultados del manómetro las diferencias son mínimas con un error del 10% entre todos los cilindros, errores que pueden ser provocados por la inestabilidad de nuestros motores o el retraso de los eventos.

- Realizando una adquisición de datos del equipo y el manometro en el moto 1600 cc AVEO:



Como podemos apreciar los valores obtenidos con el manómetro y el equipo de diagnóstico se encuentran a una diferencia de 5 PSI, correspondiente a un error porcentual aproximado de 4% para cada uno de los cilindro del motor. Este valor de error lo podemos seguir reduciendo hasta tener una lectura lo más exacta posible a un 99%, profundizando en la investigación.

- Comparando los tiempos de diagnostico entre los equipos obtuvimos los siguientes resultados.



Los tiempos necesarios para la recolección de datos entre los dos equipos tienen una gran diferencia, en el caso de la comprobación con el manómetro el tiempo es elevado por el motivo que se requiera extraer cada bujía de los cilindros del motor así como elementos que impiden su acceso como lo es en el vehículo Aveo, en caso del equipo de diagnóstico el tiempo aproximado es de 7 minutos puesto que también es necesario desmontar la tapa del motor para poder adquirir la señal de inyección y conectar el equipo a una entrada de vacío. Demostrando así la eficiencia en cuanto al tiempo de diagnóstico.

- El sensor MAP muy importante en el funcionamiento de un motor genera un voltaje que es proporcional a la presión interna del motor, por tal motivo al aumentar la presión este voltaje también aumenta. Pero en el caso de nuestro proyecto la presión es inversamente proporcional al voltaje, es decir, cuando el motor del vehículo se encuentra apagado pero con el interruptor accionado se tiene el máximo voltaje en el sensor, mientras que al estar el

motor funcionando el voltaje que se tiene en el sensor MAP es la más baja, puesto que la presión del colector de admisión también es baja.

- Al momento de realizar las diversas pruebas en vehículos dotados con sistema MPFI Multec Delphi, pudimos constatar que los tiempos de inyección para cada motor son diferentes por tal motivo se debe calibrar el equipo según el vehículo a ser diagnosticado, con estos datos se podría ampliar el campo de investigación para modificar la programación y así expandir el diagnóstico a varias marcas de vehículos existentes en el mercado.

- Al momento de generar fallas en cada uno de los cilindros para verificar la veracidad del equipo de diagnóstico los datos conseguidos no eran los correctos por el motivo que al general la falla el motor empieza a desestabilizarse, impidiendo que el sensor MAP se mantenga estable para que el equipo pueda recolectar un dato correcto, es por tal motivo que al momento de diagnosticar el motor debe encontrarse a temperatura normal de trabajo y además debe estar funcionando de manera estable.

- Manejar software libre hoy en día nos acerca cada vez a un mundo democratizado y de mejor acceso al conocimiento e investigación evitando así ser controlados por empresas monopolistas que solo piensan en su propio binenestar.

- Arduino ha sido diseñado para personas que no tengan muchos conocimientos en programación o tengan pocos conocimientos en simulink, matlab, eclipse, processing, pues gracias a estos software de ayuda para un desarrollador se puede crear líneas de programación, con la ventaja que arduino permite verificar resultados en tiempo real sin necesidad de quemar pics o armar un circuito externo.

- La comunicación bluetooth es de gran ayuda porque nos permite enviar un dato a la vez o mayor cantidad de información mediante la comprensión de datos que pueden ser recibidos a una distancia de 50 metros entre el teléfono móvil y nuestro módulo bluetooth, además nos evita la utilización de cables que podrían causar datos erróneos si estos llegaran a aplastarse o hacer un falso contacto ya sea por estar rotos o cortocircuitados.

RECOMENDACIONES.

- Arduino en su software y hardware podría ser útil en la carrera de ingeniería automotriz, la tarjeta puede ser diseñada por nosotros mismo, pudiendo cambiar su microcontrolador original por uno de mayor capacidad según las necesidades y así realizar un control muy parecido a la de una computadora automotriz o a partir de su propio microcontrolador manejar cada uno de los sistemas del automóvil independientemente

- La ventaja principal de la tarjeta arduino es que son accesibles por su bajo costo, la tarjeta arduino de mayor valor es la arduino mega ADK con un costo de 80$ y además nos sirve como tarjeta de adquisición de datos ante programas sofisticados para ingeniería como es el caso de Matlab y Labview cuyo valor de sus tarjetas de adquisición de datos es por encima de los 300$.

- La plataforma Android también generada en código abierto, es hoy en día la más utilizada para programación de dispositivos móviles, entre sus funciones principales está la de enviar y recibir información por mensajes de texto, bluetooth o internet dependiendo las necesidades del desarrollador.

- Los dispositivos móviles mejoran el estatus de diseño en la presentación de proyectos, debido a que estos se manejan por pantallas táctiles evitando el uso de botones para desplazarnos en nuestro entorno de desarrollo, además de ampliar la visualización en cualquier ambiente que nos encontremos.

- ANDROID puede ser diseñado por bloques de programación como es el caso de App Inventor, código básico para aplicaciones de poco alcance al no tener los permisos suficientes para manejar un código de mayor complejidad como es el caso de Eclipse cuyo entorno de desarrollo se maneja por lenguaje JAVA, código de programación más completo para el desarrollo de proyectos.

- El lenguaje de programación JAVA está orientado a objetos el cual nos permite crear código en clases secundarias para luego poder unirlas a una clase principal y mostrar los datos necesarios en nuestro diseño final.

- Eclipse maneja todos los permisos necesarios que utiliza nuestro dispositivo móvil, esto nos ayuda a mejorar la programación de nuestra aplicación, tan solo se necesita tener permisos de usuario para poder activarlos.

BIBLIOGRAFÍA.

Libros.

- GARBENO, Jorge Alberto, Tratado de Electrónica Automotriz, 1ª Edición, México Pearson Educación, 2010.
- REYES, Carlos A., Micro controladores PIC, programación en Basic, 2da Edición, Quito-Ecuador: Rispergraf, 2006.
- CENGEL, YUNES A., Termodinámica, 10ma Edición, México: McGraw Hill / Interamericana Editores, 2012.
- BOYLESTAD, Robert, Electrónica Teoría de Circuitos, 10ma Edición, México: Pearson Educación, 2004.
- PALACIOS, Enrique, Microcontrolador, 3ª Edición, México: Alfaomega, 2009.
- DORF, Richard C., Sistemas de control moderno, 10ma Edición, Madrid: Pearson Prentice Hall, 2005.

Páginas Web.

- http://sistemasdealimenta.blogspot.com/2011/04/mpfi-inyeccion-de-combustible-mpfi.html
- http://dc316.4shared.com/doc/nx6bwig_/preview.html
- http://www.manualdemecanica.info/El_sensor_MAP.html
- http://grupos.emagister.com/debate/map_maf_enhancer_o_efies__burladores_de_senal_y_pwm__necesarios_/5621-725149
- http://e-auto.com.mx/manual_detalle.php?manual_id=221
- http://www.manualesdemecanica.com/foros/4-problemas-mecanicos-y-consultas/21138-falla-chevrolet-corsa-opel.html
- http://developer.android.com/intl/es/guide/topics/connectivity/bluetooth.html#ConnectingAsAServer
- http://blog.bricogeek.com/noticias/arduino/el-adk-de-google-en-un-arduino-uno/

- http://www.elecfreaks.com/677.html
- http://shareinfobymukta.blogspot.com/2011/10/plotting-sensor-data-on-android.html
- http://www.maestrodelacomputacion.net/curso-de-java-gratuito-y-en-espanol-videotutoriales/
- http://www.diegotecnology.es/arduino-android-bluetoot/
- http://www.diegotecnology.es/tutoriales-arduino/
- http://www.diegotecnology.es/tutorial-comunicacion-serie/
- http://www.geekytheory.com/android-arduino-andruino/
- http://playground.arduino.cc/Learning/arduinoSleepCode
- http://rubenlaguna.com/wp/2008/10/15/arduino-sleep-mode-waking-up-when-receiving-data-on-the-usart/
- http://www.webtutoriales.com/articulos/java-stringtokenizer-y-split
- http://arduino.cc
- http://pruebasarduino.blogspot.com/2013/02/bluetooth.html
- http://stackoverflow.com/questions/tagged/android
- http://www.imaginaformacion.com/tutoriales/configura-una-carga-de-datos-actualizando-la-interfaz-en-android/
- http://www.nosinmiubuntu.com/2012/07/como-hacer-una-pantalla-de-presentacion.html
- http://jonsegador.com/2012/11/mostrar-pantalla-splash-android-durante-unos-segundos-iniciar-aplicacion/
- http://www.ajpdsoft.com/modules.php?name=News&file=print&sid=604
- http://www.jagsaund.com/blog/2011/11/25/vertical-progress-bar-for-android.html
- http://www.javaya.com.ar/androidya/detalleconcepto.php?codigo=144&inicio=
- http://www.javaya.com.ar
- http://sekthdroid.wordpress.com/2013/02/08/guardar-datos-de-texto-en-memoria-interna-con-android/
- http://androideity.com/2011/08/18/personalizar-las-fuentes-en-android/
- http://e-auto.com.mx/manual_detalle.php?manual_id=209
- http://android.alorse.net/2012/11/27/crear-actividades-i/

- http://www.todoautos.com.pe/portal/auto/mecanica/2372-motor-sistema-inyeccion-mpfi
- http://sistemampfi-davidarturo.blogspot.com/
- http://stackoverflow.com/questions/5523064/android-finishing-activity-from-another-activity
- http://android-videos.blogspot.in/2011/10/android-bluetooth-sample-app.html
- http://transporte.comohacerpara.com/n1145/como-medir-la-compresion-del-motor.html
- http://codigofacilito.com/videos/programacion_android_hola_mundo
- http://perso.wanadoo.es/emiliotoboso/android/suma.htm

- http://books.google.com.co/books?id=c8QwhM9DyhcC&printsec=frontcover&dq=arduino+adk&hl=es&sa=X&ei=AGuiULLEItDO0QGYoYDgCA&redir_esc=y#v=onepage&q&f=false
- http://www.colourlovers.com/palettes/search
- http://www.javaya.com.ar/androidya/detalleconcepto.php?codigo=141&inicio=
- http://www.amarino-toolkit.net/index.php/home.html

ANEXOS

ANEXOS

Código java de programación de los cálculos, recolección, transformación, comunicación y visualización de datos en el entorno de desarrollo eclipse:

```
public class ActivityPrototype extends Activity {

	////////// Necesarios para el calculo de porcentaje Estatico. //////////
	public double p1 = -1.467e-016;
	public double p2 = -6.667;
	public double p3 = 533.3;

	public double p1Din = -3.668e-017;
	public double p2Din = -1.667;
	public double p3Din = 133.3;


	////////////////////////////////////////////////////////////

	//////////Necesarios para el calculo de presion. //////////

	public double pUno = 4.136e-18;
	public double pDos =  1.15;


	public double ppOk1= 5.425e-16;
	public double ppOk2= 1.5;
	public double ppOk3= -35;

	public double ppHalf1= -7.698e-17;
	public double ppHalf2= 0.6429;
	public double ppHalf3= 41.79;

	public double ppBad1= 5.035e-18;
	public double ppBad2= 1.203;
	public double ppBad3= 7.411e-15;
	////////////////////////////////////////////////////////////

	////////////Arreglo y almacenado de datos para mostrar el valor
porcentual en las graficas. //////////
	int datos[] = new int[4];
	////////////////////////////////////////////////////////////////////////////
////////////////////////////

	boolean conectado = false;
	String strBufferIn = "";
	private InputStream MyInStream;
	private OutputStream MyOutStream;
	protected BluetoothSocket mySocket = null;
	int bytesIn = 0;
```

```
        String strTempIn;
        byte[] bufferIn = new byte[1024];

        //////////Crea las variables para manejar las imagenes de cada
cilindro.//////////
        private ImageView Cilindro1Estatico;
        private ImageView Cilindro2Estatico;
        private ImageView Cilindro3Estatico;
        private ImageView Cilindro4Estatico;

        private ImageView cilindro1Dinamico;
        private ImageView cilindro2Dinamico;
        private ImageView cilindro3Dinamico;
        private ImageView cilindro4Dinamico;
        ////////////////////////////////////////////////////////////////////////
/////////


        ///// Varialbes para la comunicación bluetooth. /////
        private static final int REQUEST_CONNECT_DEVICE = 1;
        private static final int REQUEST_ENABLE_BT = 2;
        public static final int MESSAGE_STATE_CHANGE = 1;
        public static final int MESSAGE_DEVICE_NAME = 4;
        public static final int MESSAGE_TOAST = 5;
        public static final int MESSAGE_READ = 2;
        public static final int MESSAGE_WRITE = 3;

        public static final String DEVICE_NAME = "device_name";
        public static final String TOAST = "toast";
        private BluetoothAdapter mBluetoothAdapter;
        private BluetoothDevice mBluetoothDevice = null;
        Thread hilo1 = null;
        ////////////////////////////////////////////////

        ///////// ProgressBAr CILINDRO 1 //////////

        private ProgressBar Barra_show1; // Para usar una barra vacia al iniciar
el programa.
        private ProgressBar Barra_show2; // Para usar una barra vacia al iniciar
el programa.
        private ProgressBar Barra_show3; // Para usar una barra vacia al iniciar
el programa.
        private ProgressBar Barra_show4; // Para usar una barra vacia al iniciar
el programa.

        private ProgressBar malo1; // Para usar el ProgressBar cuando el estado
del cilindro es malo.
        private ProgressBar medio1; // Para usar el ProgressBar cuando el estado
del cilindro es medio.
        private ProgressBar bueno1; // Para usar el ProgressBar cuando el estado
del cilindro esta bien.

        private ProgressBar malo1Din; // Para usar el ProgressBar cuando el
estado del cilindro es malo.
```

```
        private ProgressBar medio1Din; // Para usar el ProgressBar cuando el
estado del cilindro es medio.
        private ProgressBar bueno1Din; // Para usar el ProgressBar cuando el
estado del cilindro esta bien.


        ///////// ProgressBAr CILINDRO 2 //////////
        private ProgressBar malo2; // Para usar el ProgressBar cuando el estado
del cilindro es malo.
        private ProgressBar medio2; // Para usar el ProgressBar cuando el estado
del cilindro es medio.
        private ProgressBar bueno2; // Para usar el ProgressBar cuando el estado
del cilindro esta bien.

        private ProgressBar malo2Din; // Para usar el ProgressBar cuando el
estado del cilindro es malo.
        private ProgressBar medio2Din; // Para usar el ProgressBar cuando el
estado del cilindro es medio.
        private ProgressBar bueno2Din; // Para usar el ProgressBar cuando el
estado del cilindro esta bien.


        ///////// ProgressBAr CILINDRO 3 //////////
        private ProgressBar malo3; // Para usar el ProgressBar cuando el estado
del cilindro es malo.
        private ProgressBar medio3; // Para usar el ProgressBar cuando el estado
del cilindro es medio.
        private ProgressBar bueno3; // Para usar el ProgressBar cuando el estado
del cilindro esta bien.

        private ProgressBar malo3Din; // Para usar el ProgressBar cuando el
estado del cilindro es malo.
        private ProgressBar medio3Din; // Para usar el ProgressBar cuando el
estado del cilindro es medio.
        private ProgressBar bueno3Din; // Para usar el ProgressBar cuando el
estado del cilindro esta bien.

        ///////// ProgressBAr CILINDRO 4 //////////
        private ProgressBar malo4; // Para usar el ProgressBar cuando el estado
del cilindro es malo.
        private ProgressBar medio4; // Para usar el ProgressBar cuando el estado
del cilindro es medio.
        private ProgressBar bueno4; // Para usar el ProgressBar cuando el estado
del cilindro esta bien.

        private ProgressBar malo4Din; // Para usar el ProgressBar cuando el
estado del cilindro es malo.
        private ProgressBar medio4Din; // Para usar el ProgressBar cuando el
estado del cilindro es medio.
        private ProgressBar bueno4Din; // Para usar el ProgressBar cuando el
estado del cilindro esta bien.

        ////////// Para mostrar los conjuntos de cilindros y barras. //////////
        private FrameLayout conjuntoCilindrosEstatico;
        private FrameLayout conjuntoCilindrosDinamico;
```

```
        //////////Para mostrar las indicaciones y observaciones. //////////
        private LinearLayout observaciones;
        private RelativeLayout indicacionesEstatico;
        private RelativeLayout indicacionesDinamico;

        //////////Para mostrar el menu de analisis y sus botones. //////////
        private RelativeLayout btnMenu;
        private LinearLayout conjuntoMenu;
        private Button btnMenuAp;
        private Button btnMenuDe;
        private ImageButton btnMenuAp1;
        private ImageButton btnMenuDe1;

        ////////// Para recolectar los datos porcentaje de cada cilindro.
////////////
        private TextView txtDato1EstaticoOK;
        private TextView txtDato1EstaticoHALF;
        private TextView txtDato1EstaticoBAD;

        private TextView txtDato2EstaticoOK;
        private TextView txtDato2EstaticoHALF;
        private TextView txtDato2EstaticoBAD;

        private TextView txtDato3Estatico;
        private TextView txtDato3EstaticoHALF;
        private TextView txtDato3EstaticoBAD;

        private TextView txtDato4Estatico;
        private TextView txtDato4EstaticoHALF;
        private TextView txtDato4EstaticoBAD;

        ////////Para recolectar los datos presion de cada cilindro. ////////////

        private TextView txtPresionDato1;
        private TextView txtPresionDato2;
        private TextView txtPresionDato3;
        private TextView txtPresionDato4;


        private TextView txtDatoPresion1Ok;
        private TextView txtDatoPresion2Ok;
        private TextView txtDatoPresion3Ok;
        private TextView txtDatoPresion4Ok;

        private TextView txtDatoPresion1OkDin;
        private TextView txtDatoPresion2OkDin;
        private TextView txtDatoPresion3OkDin;
        private TextView txtDatoPresion4OkDin;

        private TextView txtDatoPresion1Half;
        private TextView txtDatoPresion2Half;
        private TextView txtDatoPresion3Half;
        private TextView txtDatoPresion4Half;
```

```
        private TextView txtDatoPresion1HalfDin;
        private TextView txtDatoPresion2HalfDin;
        private TextView txtDatoPresion3HalfDin;
        private TextView txtDatoPresion4HalfDin;

        private TextView txtDatoPresion1Bad;
        private TextView txtDatoPresion2Bad;
        private TextView txtDatoPresion3Bad;
        private TextView txtDatoPresion4Bad;

        private TextView txtDatoPresion1BadDin;
        private TextView txtDatoPresion2BadDin;
        private TextView txtDatoPresion3BadDin;
        private TextView txtDatoPresion4BadDin;

        private TextView txt1Presion150;
        private TextView txt1Presion0;
        private TextView txt2Presion150;
        private TextView txt2Presion0;
        private TextView txt3Presion150;
        private TextView txt3Presion0;
        private TextView txt4Presion150;
        private TextView txt4Presion0;

        private TextView txt1Presion115Din;
        private TextView txt1Presion0Din;
        private TextView txt2Presion115Din;
        private TextView txt2Presion0Din;
        private TextView txt3Presion115Din;
        private TextView txt3Presion0Din;
        private TextView txt4Presion115Din;
        private TextView txt4Presion0Din;

        private FrameLayout AnalisisEstatico;
        private FrameLayout AnalisisDinamico;

        @Override
        protected void onCreate(Bundle savedInstanceState) {
              ////////// Muestra la actividad para esta clase //////////
              super.onCreate(savedInstanceState);
              setContentView(R.layout.activity_prototype);
              ////////////////////////////////////////////////////////

              /////////Necesarios para conectarse a un dispositivo. //////////
              mBluetoothDevice = null;
              mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
              Intent serverIntent = new
Intent(this,DeviceListActivityPrototype.class);
              startActivityForResult(serverIntent, REQUEST_CONNECT_DEVICE);
              ////////////////////////////////////////////////////////////////

              ////////////////////// ANIMACION PRIMER CILINDRO.
//////////////////////////
              final VisorGif gif1 = (VisorGif) findViewById(R.id.cilindro1Din);
              gif1.setMovieResource(R.drawable.animcilindrouno);
```

```
            // ////////////////// ANIMACION SEGUNDO CILINDRO.
//////////////////////////
            final VisorGif gif2 = (VisorGif) findViewById(R.id.cilindro2Din);
            gif2.setMovieResource(R.drawable.animcilindrodos);
            ////////////////////// ANIMACION TERCER CILINDRO.
//////////////////////////
            final VisorGif gif3 = (VisorGif) findViewById(R.id.cilindro3Din);
            gif3.setMovieResource(R.drawable.animcilindrotres);
            ////////////////////// ANIMACION CUARTO CILINDRO.
//////////////////////////
            final VisorGif gif4 = (VisorGif) findViewById(R.id.cilindro4Din);
            gif4.setMovieResource(R.drawable.animcilindrocuatro);

      ////////////////////////////////////////////////////////////////////////
////

            //////////Llama a cada imagen para su posterior uso. //////////

            cilindro1Dinamico = (ImageView)
findViewById(R.id.Cilindro_1Dinamico);
            cilindro2Dinamico = (ImageView)
findViewById(R.id.Cilindro_2Dinamico);
            cilindro3Dinamico = (ImageView)
findViewById(R.id.Cilindro_3Dinamico);
            cilindro4Dinamico = (ImageView)
findViewById(R.id.Cilindro_4Dinamico);
            ////////////////////////////////////////////////////////////

            //////////Llama al conjunto de cilindros y barras. //////////
            conjuntoCilindrosEstatico = (FrameLayout)
findViewById(R.id.conjuntoCilindrosEstatico);
            conjuntoCilindrosDinamico = (FrameLayout)
findViewById(R.id.conjuntoCilindrosDinamico);

            Barra_show1 = (ProgressBar) findViewById(R.id.ProgressBarVacia1);
            Barra_show2 = (ProgressBar) findViewById(R.id.ProgressBarVacia2);
            Barra_show3 = (ProgressBar) findViewById(R.id.ProgressBarVacia3);
            Barra_show4 = (ProgressBar) findViewById(R.id.ProgressBarVacia4);
            ////////////////////////////////////////////////////////

            /////////// Llama a los conjuntos de indicaciones  observaciones.
//////////
            observaciones = (LinearLayout)findViewById(R.id.Observaciones);
            indicacionesEstatico = (RelativeLayout)
findViewById(R.id.IndicacionesEstatico);

            indicacionesDinamico = (RelativeLayout)
findViewById(R.id.IndicacionesDinamico);

      ////////////////////////////////////////////////////////////////////////
///

            /////////// Llama al conjunto menu y sus botones //////////
            conjuntoMenu = (LinearLayout)findViewById(R.id.ConjuntoMenu);
            btnMenu = (RelativeLayout)findViewById(R.id.btnMenu);
```

```
            btnMenuAp = (Button)findViewById(R.id.btnMenuAp);
            btnMenuDe = (Button)findViewById(R.id.btnMenuDe);
            btnMenuAp1 = (ImageButton)findViewById(R.id.btnMenuAp1);
            btnMenuDe1 = (ImageButton)findViewById(R.id.btnMenuDe1);

      ////////////////////////////////////////////////////////////////////

            /////////// PAra recolectar los datos de presion de cada cilindro.
            txtPresionDato1 =
(TextView)findViewById(R.id.DatoCilindro1Dinamico);
            txtPresionDato2 =
(TextView)findViewById(R.id.DatoCilindro2Dinamico);
            txtPresionDato3 =
(TextView)findViewById(R.id.DatoCilindro3Dinamico);
            txtPresionDato4 =
(TextView)findViewById(R.id.DatoCilindro4Dinamico);

            txtDatoPresion1OkDin =
(TextView)findViewById(R.id.txtPresion1OkDin);
            txtDatoPresion2OkDin =
(TextView)findViewById(R.id.txtPresion2OkDin);
            txtDatoPresion3OkDin =
(TextView)findViewById(R.id.txtPresion3OkDin);
            txtDatoPresion4OkDin =
(TextView)findViewById(R.id.txtPresion4OkDin);


            txtDatoPresion1HalfDin = (TextView)
findViewById(R.id.txtPresion1HalfDin);
            txtDatoPresion2HalfDin = (TextView)
findViewById(R.id.txtPresion2HalfDin);
            txtDatoPresion3HalfDin = (TextView)
findViewById(R.id.txtPresion3HalfDin);
            txtDatoPresion4HalfDin = (TextView)
findViewById(R.id.txtPresion4HalfDin);


            txtDatoPresion1BadDin = (TextView)
findViewById(R.id.txtPresion1BadDin);
            txtDatoPresion2BadDin = (TextView)
findViewById(R.id.txtPresion2BadDin);
            txtDatoPresion3BadDin = (TextView)
findViewById(R.id.txtPresion3BadDin);
            txtDatoPresion4BadDin = (TextView)
findViewById(R.id.txtPresion4BadDin);


            txt1Presion115Din = (TextView) findViewById(R.id.t1xt150Din);
            txt1Presion0Din = (TextView) findViewById(R.id.t1xt0Din);
            txt2Presion115Din = (TextView) findViewById(R.id.t2xt150Din);
            txt2Presion0Din = (TextView) findViewById(R.id.t2xt0Din);
            txt3Presion115Din = (TextView) findViewById(R.id.t3xt150Din);
            txt3Presion0Din = (TextView) findViewById(R.id.t3xt0Din);
            txt4Presion115Din = (TextView) findViewById(R.id.t4xt150Din);
            txt4Presion0Din = (TextView) findViewById(R.id.t4xt0Din);
```

```
            AnalisisEstatico =
(FrameLayout)findViewById(R.id.AnalisisEstatico);
            AnalisisDinamico= (FrameLayout)findViewById(R.id.AnalisisDinamico);

            ////////////////////////////////////////////////////////////////
      }

//////////Activa el menu de opciones. //////////
@Override
public boolean onCreateOptionsMenu(Menu menu) {
      MenuInflater inflater = getMenuInflater();
      inflater.inflate(R.menu.connect_menu_prototype, menu);
      return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
      switch (item.getItemId()) {
      case R.id.Conectar:
            Intent serverIntent = new
Intent(this,DeviceListActivityPrototype.class);
            startActivityForResult(serverIntent, REQUEST_CONNECT_DEVICE);
            mBluetoothAdapter.enable();

            return true;
      }
      return false;
}
////////////////////////////////////////////////////////////////

      ////////// Aparece y desaparece el menu analisis. //////////
      public void Aparece (View v){
            conjuntoMenu.setVisibility(View.VISIBLE);
            btnMenuAp.setVisibility(View.INVISIBLE);
            btnMenuDe.setVisibility(View.VISIBLE);
      }

      public void Desaparece (View v){
            conjuntoMenu.setVisibility(View.INVISIBLE);
            btnMenuAp.setVisibility(View.VISIBLE);
            btnMenuDe.setVisibility(View.INVISIBLE);
      }

      public void Aparece1 (View v){
            conjuntoMenu.setVisibility(View.VISIBLE);
            btnMenuAp1.setVisibility(View.INVISIBLE);
            btnMenuDe1.setVisibility(View.VISIBLE);
      }

      public void Desaparece1 (View v){
            conjuntoMenu.setVisibility(View.INVISIBLE);
            btnMenuAp1.setVisibility(View.VISIBLE);
            btnMenuDe1.setVisibility(View.INVISIBLE);
      }
```

```
	////////////////////////////////////////////////////////

	////////// Muestra el analisis estatico. ////////
	public void Estatico (View v){

		AnalisisDinamico.setVisibility(View.INVISIBLE);
		AnalisisEstatico.setVisibility(View.VISIBLE);

		indicacionesEstatico.setVisibility(View.VISIBLE);
		indicacionesDinamico.setVisibility(View.INVISIBLE);
		conjuntoMenu.setVisibility(View.INVISIBLE);
		btnMenu.setVisibility(View.INVISIBLE);
		observaciones.setVisibility(View.INVISIBLE);
		conjuntoCilindrosEstatico.setVisibility(View.INVISIBLE);
		conjuntoCilindrosDinamico.setVisibility(View.INVISIBLE);
		btnMenuAp1.setVisibility(View.VISIBLE);
		btnMenuDe1.setVisibility(View.INVISIBLE);

	}

	public void AnalisisEstatico (View v){

		AnalisisDinamico.setVisibility(View.INVISIBLE);
		AnalisisEstatico.setVisibility(View.VISIBLE);
		indicacionesEstatico.setVisibility(View.INVISIBLE);
		conjuntoCilindrosEstatico.setVisibility(View.VISIBLE);
		conjuntoCilindrosDinamico.setVisibility(View.INVISIBLE);
		Barra_show1.setVisibility(View.VISIBLE);
		Barra_show2.setVisibility(View.VISIBLE);
		Barra_show3.setVisibility(View.VISIBLE);
		Barra_show4.setVisibility(View.VISIBLE);

		btnMenuAp1.setVisibility(View.VISIBLE);
		btnMenuDe1.setVisibility(View.INVISIBLE);
	}
	////////////////////////////////////////////////

	////////// Envia un caracter correspondiente al analisis estatico.
//////////
	public void EnviaEstatico (View v) {
		enviaTexto("E");
	}
	////////////////////////////////////////////////////////////////////////
///


	////////// Muestra el analisis Dinamico. ////////
	public void Dinamico (View v){

		AnalisisEstatico.setVisibility(View.INVISIBLE);
		AnalisisDinamico.setVisibility(View.VISIBLE);

		indicacionesDinamico.setVisibility(View.VISIBLE);
		indicacionesEstatico.setVisibility(View.INVISIBLE);
```

```
            conjuntoMenu.setVisibility(View.INVISIBLE);
            btnMenu.setVisibility(View.INVISIBLE);
            observaciones.setVisibility(View.INVISIBLE);
            conjuntoCilindrosEstatico.setVisibility(View.INVISIBLE);
            conjuntoCilindrosDinamico.setVisibility(View.INVISIBLE);
            btnMenuAp1.setVisibility(View.VISIBLE);
            btnMenuDe1.setVisibility(View.INVISIBLE);
      }

      public void AnalisisDinamico (View v){

            AnalisisEstatico.setVisibility(View.INVISIBLE);
            AnalisisDinamico.setVisibility(View.VISIBLE);

            indicacionesDinamico.setVisibility(View.INVISIBLE);
            conjuntoCilindrosDinamico.setVisibility(View.VISIBLE);
            conjuntoCilindrosEstatico.setVisibility(View.INVISIBLE);
            btnMenuAp1.setVisibility(View.VISIBLE);
            btnMenuDe1.setVisibility(View.INVISIBLE);
      }
      ////////////////////////////////////////////////

      ////////// Envia un caracter correspondiente al analisis estatico.
//////////
      public void EnviaDinamico (View v) {
            enviaTexto("E");

      }
      ////////////////////////////////////////////////////////////////////////
///

      //////////Envia por bluetooth una letra para iniciar la recoleccion de
datos. //////////
      private void enviaTexto(String TextoEscrito) {
            if (conectado) {
                  if (mySocket != null) {
                        try {
                              strBufferIn = "";
                              MyOutStream = mySocket.getOutputStream();
                        } catch (IOException e) {
                              Toast.makeText(ActivityPrototype.this,
                                          ("\nERROR: " + e.getMessage()),
Toast.LENGTH_SHORT)
                                          .show();
                        }
                        try {
                              strBufferIn = "";
                              MyOutStream.write((TextoEscrito).getBytes());
                        } catch (IOException e) {
                              Toast.makeText(ActivityPrototype.this,
                                          ("\nERROR: " + e.getMessage()),
Toast.LENGTH_SHORT)
                                          .show();
                        }
                  }
```

```
			}
		}
		//////////Instancia el Handler Asociado con el hilo principal. //////////
		@SuppressLint("HandlerLeak")
		private Handler messageHandler = new Handler() {
			@Override
			public void handleMessage(Message msg) {
				String cadena = "";
				cadena = String.valueOf(strBufferIn);/////Convierte la
variable numerica en una varialble string./////
				int acum = 0;
				try {
					InputStream is = new
ByteArrayInputStream(cadena.getBytes());
					BufferedReader br = new BufferedReader(
							new InputStreamReader(is));
					String line;
					try {
						while ((line = br.readLine()) != null) {
							datos[acum] = Integer.parseInt(line);
							acum++;
							if (acum == 4)
							AnalisisTiempoRealEstatico(datos);
							AnalisisTiempoRealDinamico(datos);
						}
					} catch (IOException e) {
						e.printStackTrace();
					}
					try {
						br.close();
					} catch (IOException e) {
						e.printStackTrace();
					}
				} catch (NumberFormatException nfe) {
					System.out.println("Error en la conversion del tipo de
dato");
				}
			}
		};
		////////////////////////////////////////////////////////////////////////

		//////////Recibe los datos luego de hacer el pedido. //////////
		public void recibe() {
			try {
				bytesIn = MyInStream.read(bufferIn);
				strTempIn = new String(bufferIn, 0, bytesIn);
				strBufferIn += strTempIn;
				} catch (IOException e) {
					e.printStackTrace();
				}
			messageHandler.sendMessage(Message.obtain(messageHandler, 1));

		}
		////////////////////////////////////////////////////////////////
```

```
	//////////Conecta al dispositivo Bluetooth seleccionado. //////////
	private void conectar() {
		try {
			if (mBluetoothAdapter.isEnabled()) {
				mBluetoothAdapter.startDiscovery();
				BluetoothSocket tmp = null;
				Method m = mBluetoothDevice.getClass().getMethod(
						"createRfcommSocket", new Class[] {
int.class });
				tmp = (BluetoothSocket) m.invoke(mBluetoothDevice,
						Integer.valueOf(1));
				mySocket = tmp;
				mySocket.connect();
				MyInStream = mySocket.getInputStream();
				/////Espera a que se realize una conexión. /////
				if (hilo1 == null) {
					hilo1 = new Thread() {
						@Override
						public void run() {
							///// Bucle infinito que se
establece en la clase en caso de un error. /////
							while (true) {
								recibe();
							}
						};
					};
				}
				hilo1.start();
				Toast.makeText(
						ActivityPrototype.this,
						("Conectado a:" + "\n" +
mBluetoothDevice.toString()
								+ "  " +
mBluetoothDevice.getName()),
						Toast.LENGTH_LONG).show();

				////////// Para poner visible las observaciones y el
boton menu analisis. //////////
				observaciones.setVisibility(View.VISIBLE);
				btnMenu.setVisibility(View.VISIBLE);
				btnMenuAp.setVisibility(View.VISIBLE);
			}
		} catch (Exception e) {

			Toast.makeText(ActivityPrototype.this,
					"ERROR AL CONECTAR CON EL DISPOSITIVO",
Toast.LENGTH_SHORT)
					.show();
		}
	}
	////////////////////////////////////////////////////////////////

	//////////Para conectarse con el dispositivo Bluetooth /////////
	@Override
```

```
	public void onActivityResult(int requestCode, int resultCode, Intent
data) {
		switch (requestCode) {

		case REQUEST_CONNECT_DEVICE:
			///// Cuando el DeviceListActivityPrototype muestra los
dispositivos disponibles para conexión. /////
			if (resultCode == Activity.RESULT_OK) {
				///// Línea para conseguir la direccion MAC del
dispositivo BT. /////
				String address = data.getExtras().getString(

	DeviceListActivityPrototype.EXTRA_DEVICE_ADDRESS);
				///// Linea para conectarse con el dispositivo
Bluetooth. /////
				mBluetoothDevice  =
mBluetoothAdapter.getRemoteDevice(address);
				///// Linea de intento de conexion con el BT./////
				conectar();
				conectado = true;
			}
			break;

		case REQUEST_ENABLE_BT:
			if (resultCode == Activity.RESULT_OK) {
			} else {
				/////Línea para indicar que no esta habilitado el BT o
que se produjo un error./////
				Toast.makeText(this, R.string.bt_not_enabled_leaving,
						Toast.LENGTH_SHORT).show();
				finish();
			}
		}
	}
	//////////////////////////////////////////////////////////////


	txtDato2Estatico = (TextView)findViewById(R.id.DatoCilindro2Estatico);
	txtDato3Estatico = (TextView)findViewById(R.id.DatoCilindro3Estatico);
	txtDato4Estatico = (TextView)findViewById(R.id.DatoCilindro4Estatico);

	public void AnalisisTiempoRealEstatico (int[] datos) {
		System.out.println("entro analisis Tiempo Real");

		try {


			txtDatoPresion1Ok =
(TextView)findViewById(R.id.txtPresion1Ok);
			txtDatoPresion1Half = (TextView)
findViewById(R.id.txtPresion1Half);
			txtDatoPresion1Bad = (TextView)
findViewById(R.id.txtPresion1Bad);
			txt1Presion150 = (TextView) findViewById(R.id.t1xt150);
			txt1Presion0 = (TextView) findViewById(R.id.t1xt0);
```

```
                    txtDatoPresion2Ok =
(TextView)findViewById(R.id.txtPresion2Ok);
                    txtDatoPresion2Half = (TextView)
findViewById(R.id.txtPresion2Half);
                    txtDatoPresion2Bad = (TextView)
findViewById(R.id.txtPresion2Bad);
                    txt2Presion150 = (TextView) findViewById(R.id.t2xt150);
                    txt2Presion0 = (TextView) findViewById(R.id.t2xt0);

                    txtDatoPresion3Ok =
(TextView)findViewById(R.id.txtPresion3Ok);
                    txtDatoPresion3Half = (TextView)
findViewById(R.id.txtPresion3Half);
                    txtDatoPresion3Bad = (TextView)
findViewById(R.id.txtPresion3Bad);
                    txt3Presion150 = (TextView) findViewById(R.id.t3xt150);
                    txt3Presion0 = (TextView) findViewById(R.id.t3xt0);


                    txtDatoPresion4Ok =
(TextView)findViewById(R.id.txtPresion4Ok);
                    txtDatoPresion4Half = (TextView)
findViewById(R.id.txtPresion4Half);
                    txtDatoPresion4Bad = (TextView)
findViewById(R.id.txtPresion4Bad);
                    txt4Presion150 = (TextView) findViewById(R.id.t4xt150);
                    txt4Presion0 = (TextView) findViewById(R.id.t4xt0);

                    ////////// CONSTANTES PARA EL CALCULO DE LAS PRESIONES.
////////

                    double AjPresion1 = -0.002185;
                    double AjPresion2 = 0.424;
                    double AjPresion3 = 157.5;


                    //////// PARA LAS PRESIONES DEL PRIMER CILINDRO. //////////
                    int bit1 = datos[0];
                    double PCil1 =
((AjPresion1*(bit1*bit1))+(AjPresion2*bit1)+AjPresion3);
                    int Presion1 = (int)PCil1;

                    if (Presion1 > 150){

                          txt1Presion150.setVisibility(View.VISIBLE);
                          txt1Presion0.setVisibility(View.INVISIBLE);
                          txtDatoPresion1Ok.setVisibility(View.INVISIBLE);
                          txtDatoPresion1Half.setVisibility(View.INVISIBLE);
                          txtDatoPresion1Bad.setVisibility(View.INVISIBLE);

                    }

                    if (Presion1 <= 150){

                          String DatoPresionCil1 = String.valueOf(Presion1);
```

```
                        txtDatoPresion1Ok.setText(DatoPresionCil1 + " PSI");
                        txtDatoPresion1Ok.setVisibility(View.VISIBLE);

                        txtDatoPresion1Half.setVisibility(View.INVISIBLE);
                        txtDatoPresion1Bad.setVisibility(View.INVISIBLE);
                        txt1Presion150.setVisibility(View.INVISIBLE);
                        txt1Presion0.setVisibility(View.INVISIBLE);

                    }

                    if (Presion1 <130){

                        String DatoPresionCil1 = String.valueOf(Presion1);
                        txtDatoPresion1Half.setText(DatoPresionCil1 + " PSI");
                        txtDatoPresion1Half.setVisibility(View.VISIBLE);

                        txtDatoPresion1Ok.setVisibility(View.INVISIBLE);
                        txtDatoPresion1Bad.setVisibility(View.INVISIBLE);
                        txt1Presion150.setVisibility(View.INVISIBLE);
                        txt1Presion0.setVisibility(View.INVISIBLE);

                    }

                    if (Presion1 <118){

                        String DatoPresionCil1 = String.valueOf(Presion1);
                        txtDatoPresion1Bad.setText(DatoPresionCil1 + " PSI");
                        txtDatoPresion1Bad.setVisibility(View.VISIBLE);


                        txtDatoPresion1Ok.setVisibility(View.INVISIBLE);
                        txtDatoPresion1Half.setVisibility(View.INVISIBLE);
                        txt1Presion150.setVisibility(View.INVISIBLE);
                        txt1Presion0.setVisibility(View.INVISIBLE);

                    }

                    if (Presion1 <0){

                        txt1Presion0.setVisibility(View.VISIBLE);
                        txt1Presion150.setVisibility(View.INVISIBLE);
                        txtDatoPresion1Ok.setVisibility(View.INVISIBLE);
                        txtDatoPresion1Half.setVisibility(View.INVISIBLE);
                        txtDatoPresion1Bad.setVisibility(View.INVISIBLE);

                        InputStream imBad = getAssets().open("cil_0.png");

      Cilindro1Estatico.setImageDrawable(Drawable.createFromStream(imBad,null))
;

                    }

                        //////// PARA LAS PRESIONES DEL SEGUNDO CILINDRO.
                        int bit2 = datos[0];
```

```
                        double PCil2 =
((AjPresion1*(bit2*bit2))+(AjPresion2*bit2)+AjPresion3);
                        int Presion2 = (int)PCil2;

                        if (Presion2 > 150){

                              txt2Presion150.setVisibility(View.VISIBLE);
                              txt2Presion0.setVisibility(View.INVISIBLE);
                              txtDatoPresion2Ok.setVisibility(View.INVISIBLE);

      txtDatoPresion2Half.setVisibility(View.INVISIBLE);

      txtDatoPresion2Bad.setVisibility(View.INVISIBLE);

                        }

                        if (Presion2 <= 150){

                              String DatoPresionCil2 =
String.valueOf(Presion2);
                              txtDatoPresion2Ok.setText(DatoPresionCil2 + "
PSI");
                              txtDatoPresion2Ok.setVisibility(View.VISIBLE);


      txtDatoPresion2Half.setVisibility(View.INVISIBLE);

      txtDatoPresion2Bad.setVisibility(View.INVISIBLE);
                              txt2Presion150.setVisibility(View.INVISIBLE);
                              txt2Presion0.setVisibility(View.INVISIBLE);

                        }

                        if (Presion2 <130){

                              String DatoPresionCil2 =
String.valueOf(Presion2);
                              txtDatoPresion2Half.setText(DatoPresionCil2 + "
PSI");
                              txtDatoPresion2Half.setVisibility(View.VISIBLE);

                              txtDatoPresion2Ok.setVisibility(View.INVISIBLE);

      txtDatoPresion2Bad.setVisibility(View.INVISIBLE);
                              txt2Presion150.setVisibility(View.INVISIBLE);
                              txt2Presion0.setVisibility(View.INVISIBLE);

                        }

                        if (Presion2 <118){

                              String DatoPresionCil2 =
String.valueOf(Presion2);
                              txtDatoPresion2Bad.setText(DatoPresionCil2 + "
PSI");
```

```
                                    txtDatoPresion2Bad.setVisibility(View.VISIBLE);


                                    txtDatoPresion2Ok.setVisibility(View.INVISIBLE);

        txtDatoPresion2Half.setVisibility(View.INVISIBLE);
                                    txt2Presion150.setVisibility(View.INVISIBLE);
                                    txt2Presion0.setVisibility(View.INVISIBLE);

                                }

                                if (Presion2 <0){

                                    txt2Presion0.setVisibility(View.VISIBLE);
                                    txt2Presion150.setVisibility(View.INVISIBLE);
                                    txtDatoPresion2Ok.setVisibility(View.INVISIBLE);

        txtDatoPresion2Half.setVisibility(View.INVISIBLE);

        txtDatoPresion2Bad.setVisibility(View.INVISIBLE);

                                    InputStream imBad =
getAssets().open("cil_0.png");

        Cilindro2Estatico.setImageDrawable(Drawable.createFromStream(imBad,null))
;

                                }


                                //////// PARA LAS PRESIONES DEL TERCER CILINDRO.
                                int bit3 = datos[0];
                                double PCil3 =
((AjPresion1*(bit3*bit3))+(AjPresion2*bit3)+AjPresion3);
                                int Presion3 = (int)PCil3;

                                if (Presion3 > 150){

                                    txt3Presion150.setVisibility(View.VISIBLE);
                                    txt3Presion0.setVisibility(View.INVISIBLE);
                                    txtDatoPresion3Ok.setVisibility(View.INVISIBLE);

        txtDatoPresion3Half.setVisibility(View.INVISIBLE);

        txtDatoPresion3Bad.setVisibility(View.INVISIBLE);

                                }

                                if (Presion3 <= 150){

                                    String DatoPresionCil3 =
String.valueOf(Presion3);
                                    txtDatoPresion3Ok.setText(DatoPresionCil3 + "
PSI");
                                    txtDatoPresion3Ok.setVisibility(View.VISIBLE);
```

```
        txtDatoPresion3Half.setVisibility(View.INVISIBLE);

        txtDatoPresion3Bad.setVisibility(View.INVISIBLE);
                                txt3Presion150.setVisibility(View.INVISIBLE);
                                txt3Presion0.setVisibility(View.INVISIBLE);

                            }

                            if (Presion3 <130){

                                String DatoPresionCil3 =
String.valueOf(Presion3);
                                txtDatoPresion3Half.setText(DatoPresionCil3 + "
PSI");
                                txtDatoPresion3Half.setVisibility(View.VISIBLE);

                                txtDatoPresion3Ok.setVisibility(View.INVISIBLE);

        txtDatoPresion3Bad.setVisibility(View.INVISIBLE);
                                txt3Presion150.setVisibility(View.INVISIBLE);
                                txt3Presion0.setVisibility(View.INVISIBLE);

                            }

                            if (Presion3 <118){

                                String DatoPresionCil3 =
String.valueOf(Presion3);
                                txtDatoPresion3Bad.setText(DatoPresionCil3 + "
PSI");
                                txtDatoPresion3Bad.setVisibility(View.VISIBLE);


                                txtDatoPresion3Ok.setVisibility(View.INVISIBLE);

        txtDatoPresion3Half.setVisibility(View.INVISIBLE);
                                txt3Presion150.setVisibility(View.INVISIBLE);
                                txt3Presion0.setVisibility(View.INVISIBLE);

                            }

                            if (Presion3 <0){

                                txt3Presion0.setVisibility(View.VISIBLE);
                                txt3Presion150.setVisibility(View.INVISIBLE);
                                txtDatoPresion3Ok.setVisibility(View.INVISIBLE);

        txtDatoPresion3Half.setVisibility(View.INVISIBLE);

        txtDatoPresion3Bad.setVisibility(View.INVISIBLE);

                                InputStream imBad =
getAssets().open("cil_0.png");
```

```
      Cilindro3Estatico.setImageDrawable(Drawable.createFromStream(imBad,null))
;

                        }

                        //////// PARA LAS PRESIONES DEL CUARTO CILINDRO.
//////////
                        int bit4 = datos[0];
                        double PCil4 =
((AjPresion1*(bit4*bit4))+(AjPresion2*bit4)+AjPresion3);
                        int Presion4 = (int)PCil4;

                        if (Presion4 > 150){

                              txt4Presion150.setVisibility(View.VISIBLE);
                              txt4Presion0.setVisibility(View.INVISIBLE);
                              txtDatoPresion4Ok.setVisibility(View.INVISIBLE);

      txtDatoPresion4Half.setVisibility(View.INVISIBLE);

      txtDatoPresion4Bad.setVisibility(View.VISIBLE);

                        }

                        if (Presion4 <= 150){

                              String DatoPresionCil4 =
String.valueOf(Presion4);
                              txtDatoPresion4Ok.setText(DatoPresionCil4 + "
PSI");
                              txtDatoPresion4Ok.setVisibility(View.VISIBLE);


      txtDatoPresion4Half.setVisibility(View.INVISIBLE);

      txtDatoPresion4Bad.setVisibility(View.INVISIBLE);
                              txt4Presion150.setVisibility(View.INVISIBLE);
                              txt4Presion0.setVisibility(View.INVISIBLE);

                        }

                        if (Presion4 <130){

                              String DatoPresionCil4 =
String.valueOf(Presion4);
                              txtDatoPresion4Half.setText(DatoPresionCil4 + "
PSI");
                              txtDatoPresion4Half.setVisibility(View.VISIBLE);

                              txtDatoPresion4Ok.setVisibility(View.INVISIBLE);

      txtDatoPresion4Bad.setVisibility(View.INVISIBLE);
                              txt4Presion150.setVisibility(View.INVISIBLE);
                              txt4Presion0.setVisibility(View.INVISIBLE);
```

```
                        }

                        if (Presion4 <118){

                                String DatoPresionCil4 =
String.valueOf(Presion4);
                                txtDatoPresion4Bad.setText(DatoPresionCil4 + "
PSI");
                                txtDatoPresion4Bad.setVisibility(View.VISIBLE);


                                txtDatoPresion4Ok.setVisibility(View.INVISIBLE);

      txtDatoPresion4Half.setVisibility(View.INVISIBLE);
                                txt4Presion150.setVisibility(View.INVISIBLE);
                                txt4Presion0.setVisibility(View.INVISIBLE);

                        }

                        if (Presion4 <0){

                                txt4Presion0.setVisibility(View.VISIBLE);
                                txt4Presion150.setVisibility(View.INVISIBLE);
                                txtDatoPresion4Ok.setVisibility(View.INVISIBLE);

      txtDatoPresion4Half.setVisibility(View.INVISIBLE);

      txtDatoPresion4Bad.setVisibility(View.INVISIBLE);

                                InputStream imBad =
getAssets().open("cil_0.png");

      Cilindro4Estatico.setImageDrawable(Drawable.createFromStream(imBad,null))
;

                        }

                  ////////////////////// ELEMENTOS PARA MOSTRAR EL VALOR DE
PORCENTAJE EN LA BARRA. /////////////

                  malo1 = (ProgressBar)
findViewById(R.id.progressBar0Estatico); // Para tomar el progressBar adecuado.
                  medio1 = (ProgressBar)
findViewById(R.id.progressBar1Estatico); // Para tomar el progressBar adecuado.
                  bueno1 = (ProgressBar)
findViewById(R.id.progressBar2Estatico); // Para tomar el progressBar adecuado.

                  malo1.setMax(100); // Colocar un maximo de 100 al
ProgressBar.
                  medio1.setMax(100); // Colocar un maximo de 100 al
ProgressBar.
                  bueno1.setMax(100); // Colocar un maximo de 100 al
ProgressBar.
```

```
                    malo1.setProgress(0); // Coloca como minimo de 0 al
ProgressBar.
                    medio1.setProgress(0);// Coloca como minimo de 0 al
ProgressBar.
                    bueno1.setProgress(0);// Coloca como minimo de 0 al
ProgressBar.

                    malo2 = (ProgressBar)
findViewById(R.id.progressBar3Estatico); // Para tomar el progressBar adecuado.
                    medio2 = (ProgressBar)
findViewById(R.id.progressBar4Estatico); // Para tomar el progressBar adecuado.
                    bueno2 = (ProgressBar)
findViewById(R.id.progressBar5Estatico); // Para tomar el progressBar adecuado.

                    malo2.setMax(100); // Colocar un maximo de 100 al
ProgressBar.
                    medio2.setMax(100); // Colocar un maximo de 100 al
ProgressBar.
                    bueno2.setMax(100); // Colocar un maximo de 100 al
ProgressBar.

                    malo2.setProgress(0); // Coloca como minimo de 0 al
ProgressBar.
                    medio2.setProgress(0);// Coloca como minimo de 0 al
ProgressBar.
                    bueno2.setProgress(0);// Coloca como minimo de 0 al
ProgressBar.

                    malo3 = (ProgressBar)
findViewById(R.id.progressBar6Estatico); // Para tomar el progressBar adecuado.
                    medio3 = (ProgressBar)
findViewById(R.id.progressBar7Estatico); // Para tomar el progressBar adecuado.
                    bueno3 = (ProgressBar)
findViewById(R.id.progressBar8Estatico); // Para tomar el progressBar adecuado.

                    malo3.setMax(100); // Colocar un maximo de 100 al
ProgressBar.
                    medio3.setMax(100); // Colocar un maximo de 100 al
ProgressBar.
                    bueno3.setMax(100); // Colocar un maximo de 100 al
ProgressBar.

                    malo3.setProgress(0); // Coloca como minimo de 0 al
ProgressBar.
                    medio3.setProgress(0);// Coloca como minimo de 0 al
ProgressBar.
                    bueno3.setProgress(0);// Coloca como minimo de 0 al
ProgressBar.

                    malo4 = (ProgressBar)
findViewById(R.id.progressBar9Estatico); // Para tomar el progressBar adecuado.
                    medio4 = (ProgressBar)
findViewById(R.id.progressBar10Estatico); // Para tomar el progressBar adecuado.
                    bueno4 = (ProgressBar)
findViewById(R.id.progressBar11Estatico); // Para tomar el progressBar adecuado.
```

```
                    malo4.setMax(100); // Colocar un maximo de 100 al
ProgressBar.
                    medio4.setMax(100); // Colocar un maximo de 100 al
ProgressBar.
                    bueno4.setMax(100); // Colocar un maximo de 100 al
ProgressBar.

                    malo4.setProgress(0); // Coloca como minimo de 0 al
ProgressBar.
                    medio4.setProgress(0);// Coloca como minimo de 0 al
ProgressBar.
                    bueno4.setProgress(0);// Coloca como minimo de 0 al
ProgressBar.

                    ////////////////// ELEMENTOS USADOS PARA MOSTRAR EL PORCENTAJE
EN LAS IMAGENES. ///////////
                    Cilindro1Estatico = (ImageView)
findViewById(R.id.Cilindro_1Estatico);
                    Cilindro2Estatico = (ImageView)
findViewById(R.id.Cilindro_2Estatico);
                    Cilindro3Estatico = (ImageView)
findViewById(R.id.Cilindro_3Estatico);
                    Cilindro4Estatico = (ImageView)
findViewById(R.id.Cilindro_4Estatico);

                    ////////// CONSTANTES PARA EL CALCULO DE PORCENTAJE.
//////////
                    double AjPorcentaje1 = -0.001507;
                    double AjPorcentaje2 =  0.2791;
                    double AjPorcentaje3 =   111.5;

                    //////// PARA LOS PORCENTAJES DEL PRIMER CILINDRO. //////////
                    int bar1 = datos[0];
                    double BCil1 =
((AjPorcentaje1*(bar1*bar1))+(AjPorcentaje2*bar1)+AjPorcentaje3);
                    int Barra1 = (int)BCil1;


                    if (Barra1 > 100){

                          int Max = 100;
                          bueno1.setProgress(Max);
                          bueno1.setVisibility(View.VISIBLE);
                          malo1.setVisibility(View.INVISIBLE);
                          medio1.setVisibility(View.INVISIBLE);
                          Barra_show1.setVisibility(View.INVISIBLE);

                          InputStream imOk = getAssets().open("cil_"+ Max
+".png");

      Cilindro1Estatico.setImageDrawable(Drawable.createFromStream(imOk,null));

                    }
```

```
                    if (Barra1 <= 100){
                          bueno1.setProgress(Barra1);
                          bueno1.setVisibility(View.VISIBLE);
                          malo1.setVisibility(View.INVISIBLE);
                          medio1.setVisibility(View.INVISIBLE);
                          Barra_show1.setVisibility(View.INVISIBLE);

                          InputStream imOk = getAssets().open("cil_"+ Barra1 +
".png");

      Cilindro1Estatico.setImageDrawable(Drawable.createFromStream(imOk,null));
                    }

                    if (Barra1 < 90){
                          medio1.setProgress(Barra1);
                          medio1.setVisibility(View.VISIBLE);
                          malo1.setVisibility(View.INVISIBLE);
                          bueno1.setVisibility(View.INVISIBLE);
                          Barra_show1.setVisibility(View.INVISIBLE);

                          InputStream imHalf = getAssets().open("cil_"+ Barra1 +
".png");

      Cilindro1Estatico.setImageDrawable(Drawable.createFromStream(imHalf,null)
);
                    }

                    if (Barra1 < 75){
                          malo1.setProgress(Barra1);
                          malo1.setVisibility(View.VISIBLE);
                          medio1.setVisibility(View.INVISIBLE);
                          bueno1.setVisibility(View.INVISIBLE);
                          Barra_show1.setVisibility(View.INVISIBLE);

                          InputStream imBad = getAssets().open("cil_"+ Barra1 +
".png");

      Cilindro1Estatico.setImageDrawable(Drawable.createFromStream(imBad,null))
;

                    }


                    //////// PARA LOS PORCENTAJES DEL SEGUNDO CILINDRO.
//////////

                          int bar2 = datos[0];
                          double BCil2 =
((AjPorcentaje1*(bar2*bar2))+(AjPorcentaje2*bar2)+AjPorcentaje3);
                          int Barra2 = (int)BCil2;


                          if (Barra2 > 100){

                                int Max = 100;
```

```
                                bueno2.setProgress(Max);
                                bueno2.setVisibility(View.VISIBLE);
                                malo2.setVisibility(View.INVISIBLE);
                                medio2.setVisibility(View.INVISIBLE);
                                Barra_show2.setVisibility(View.INVISIBLE);

                                InputStream imOk = getAssets().open("cil_"+ Max
+".png");

        Cilindro2Estatico.setImageDrawable(Drawable.createFromStream(imOk,null));

                        }

                        if (Barra2 <= 100){
                                bueno2.setProgress(Barra2);
                                bueno2.setVisibility(View.VISIBLE);
                                malo2.setVisibility(View.INVISIBLE);
                                medio2.setVisibility(View.INVISIBLE);
                                Barra_show2.setVisibility(View.INVISIBLE);

                                InputStream imOk = getAssets().open("cil_"+
Barra2 + ".png");

        Cilindro2Estatico.setImageDrawable(Drawable.createFromStream(imOk,null));
                        }

                        if (Barra2 < 90){
                                medio2.setProgress(Barra2);
                                medio2.setVisibility(View.VISIBLE);
                                malo2.setVisibility(View.INVISIBLE);
                                bueno2.setVisibility(View.INVISIBLE);
                                Barra_show2.setVisibility(View.INVISIBLE);

                                InputStream imHalf = getAssets().open("cil_"+
Barra2 + ".png");

        Cilindro2Estatico.setImageDrawable(Drawable.createFromStream(imHalf,null)
);
                        }

                        if (Barra2 < 75){
                                malo2.setProgress(Barra2);
                                malo2.setVisibility(View.VISIBLE);
                                medio2.setVisibility(View.INVISIBLE);
                                bueno2.setVisibility(View.INVISIBLE);
                                Barra_show2.setVisibility(View.INVISIBLE);

                                InputStream imBad = getAssets().open("cil_"+
Barra2 + ".png");

        Cilindro2Estatico.setImageDrawable(Drawable.createFromStream(imBad,null))
;

                        }
```

```
                        //////// PARA LOS PORCENTAJES DEL TERCER CILINDRO.

                        int bar3 = datos[0];
                        double BCil3 =
((AjPorcentaje1*(bar3*bar3))+(AjPorcentaje2*bar3)+AjPorcentaje3);
                        int Barra3 = (int)BCil3;


                        if (Barra3 > 100){

                              int Max = 100;
                              bueno3.setProgress(Max);
                              bueno3.setVisibility(View.VISIBLE);
                              malo3.setVisibility(View.INVISIBLE);
                              medio3.setVisibility(View.INVISIBLE);
                              Barra_show3.setVisibility(View.INVISIBLE);

                              InputStream imOk = getAssets().open("cil_"+ Max
+".png");

      Cilindro3Estatico.setImageDrawable(Drawable.createFromStream(imOk,null));

                        }

                        if (Barra3 <= 100){
                              bueno3.setProgress(Barra3);
                              bueno3.setVisibility(View.VISIBLE);
                              malo3.setVisibility(View.INVISIBLE);
                              medio3.setVisibility(View.INVISIBLE);
                              Barra_show3.setVisibility(View.INVISIBLE);

                              InputStream imOk = getAssets().open("cil_"+
Barra3 + ".png");

      Cilindro3Estatico.setImageDrawable(Drawable.createFromStream(imOk,null));
                        }

                        if (Barra3 < 90){
                              medio3.setProgress(Barra3);
                              medio3.setVisibility(View.VISIBLE);
                              malo3.setVisibility(View.INVISIBLE);
                              bueno3.setVisibility(View.INVISIBLE);
                              Barra_show3.setVisibility(View.INVISIBLE);

                              InputStream imHalf = getAssets().open("cil_"+
Barra3 + ".png");

      Cilindro3Estatico.setImageDrawable(Drawable.createFromStream(imHalf,null)
);
                        }

                        if (Barra3 < 75){
                              malo3.setProgress(Barra3);
                              malo3.setVisibility(View.VISIBLE);
```

```
                                medio3.setVisibility(View.INVISIBLE);
                                bueno3.setVisibility(View.INVISIBLE);
                                Barra_show3.setVisibility(View.INVISIBLE);

                                InputStream imBad = getAssets().open("cil_"+
Barra3 + ".png");

      Cilindro3Estatico.setImageDrawable(Drawable.createFromStream(imBad,null))
;

                          }

                          //////// PARA LOS PORCENTAJES DEL CUARTO CILINDRO.

                          int bar4 = datos[0];
                          double BCil4 =
((AjPorcentaje1*(bar4*bar4))+(AjPorcentaje2*bar4)+AjPorcentaje3);
                          int Barra4 = (int)BCil4;


                          if (Barra4 > 100){

                                int Max = 100;
                                bueno4.setProgress(Max);
                                bueno4.setVisibility(View.VISIBLE);
                                malo4.setVisibility(View.INVISIBLE);
                                medio4.setVisibility(View.INVISIBLE);
                                Barra_show4.setVisibility(View.INVISIBLE);

                                InputStream imOk = getAssets().open("cil_"+ Max
+".png");

      Cilindro4Estatico.setImageDrawable(Drawable.createFromStream(imOk,null));

                          }

                          if (Barra4 <= 100){
                                bueno4.setProgress(Barra4);
                                bueno4.setVisibility(View.VISIBLE);
                                malo4.setVisibility(View.INVISIBLE);
                                medio4.setVisibility(View.INVISIBLE);
                                Barra_show4.setVisibility(View.INVISIBLE);

                                InputStream imOk = getAssets().open("cil_"+
Barra4 + ".png");

      Cilindro4Estatico.setImageDrawable(Drawable.createFromStream(imOk,null));
                          }

                          if (Barra4 < 90){
                                medio4.setProgress(Barra4);
                                medio4.setVisibility(View.VISIBLE);
                                malo4.setVisibility(View.INVISIBLE);
                                bueno4.setVisibility(View.INVISIBLE);
                                Barra_show4.setVisibility(View.INVISIBLE);
```

```
                                InputStream imHalf = getAssets().open("cil_"+
Barra4 + ".png");

      Cilindro4Estatico.setImageDrawable(Drawable.createFromStream(imHalf,null)
);
                           }

                           if (Barra4 < 75){
                                 malo4.setProgress(Barra4);
                                 malo4.setVisibility(View.VISIBLE);
                                 medio4.setVisibility(View.INVISIBLE);
                                 bueno4.setVisibility(View.INVISIBLE);
                                 Barra_show4.setVisibility(View.INVISIBLE);

                                 InputStream imBad = getAssets().open("cil_"+
Barra4 + ".png");

      Cilindro4Estatico.setImageDrawable(Drawable.createFromStream(imBad,null))
;

                           }


            /////////// Para guardar los datos en un archivo .txt. ///////////
            File ruta_sd = Environment.getExternalStorageDirectory();
            File f = new File(ruta_sd.getAbsolutePath(), "ProToTyPe.txt");

//          double Gprc1 =
((AjPorcentual1*((valor1)^2))+(AjPorcentual2*valor1)+AjPorcentual3);
//          double Gprc2 =
((AjPorcentual1*((valor2)^2))+(AjPorcentual2*valor2)+AjPorcentual3);

            OutputStreamWriter fout = new OutputStreamWriter(new
FileOutputStream(f));

            fout.write("ANALISIS DEL MOTOR" + "\n " + "\n"
                        + "# CILINDRO" + "       " + "EFICIENCIA (%)" + "
" + " PRESÓN (PSI) " + "\n" + "\n"
                        + "Cilindro 1" + "            " + Gprc1 +" %"
                  + "            " + PresionCil1 + " PSI" + "
" + "\n"
                        + "Cilindro 2" + "            " + Gprc2 +" %"
+ "            " + PresionCil2 + " PSI" + "                        " + "\n"
                        + "Cilindro 3" + "            " + porcentaje3 +" %" +
"            " + PresionCil3 + " PSI" + "                            " + "\n"
                        + "Cilindro 4" + "            " + porcentaje4 +" %" +
"            " + PresionCil4 + " PSI" + "                            " + "\n"
                        );
            fout.close();


            } catch (IOException e) {
```

```
                    e.printStackTrace();
                    System.out.println("Error en la LECTURA ");
              }

       }

       ////////// PORCENTAJE PARA EL PRIMER CILINDRO  //////////

                    int valor1 = datos[0];
                    if (valor1<252){

                          double AjPorcentualBAD1 = 2.188e-19;
                          double AjPorcentualBAD2 = 0.2948;
                          double AjPorcentualBAD3 = 0;

                          double xBad =
((AjPorcentualBAD1*((valor1)^2))+(AjPorcentualBAD2*valor1)+AjPorcentualBAD3);
                          int porcentaje1 = (int)xBad;
                          InputStream imBAD = getAssets().open("cil_" +
porcentaje1 + ".png");

       Cilindro1Estatico.setImageDrawable(Drawable.createFromStream(imBAD,null))
;

                          ///// Recoge el valor en un text view para usarlo en
pantalla resultados.//////
                          String DatoCilPorBAD1 = String.valueOf(porcentaje1);
                          txtDato1EstaticoBAD.setText(DatoCilPorBAD1);
                          System.out.println("Porcentaje 1 : " + porcentaje1);

       ////////////////////////////////////////////////////////////////////////
//////

                    } else if (valor1 < 279){
                          double AjPorcentualHALF1 = -2.139e-17;
                          double AjPorcentualHALF2 = 0.5385;
                          double AjPorcentualHALF3 = -60.69;

                          double xHalf
=((AjPorcentualHALF1*((valor1)^2))+(AjPorcentualHALF2*valor1)+AjPorcentualHALF3)
;
                          int porcentaje1 = (int)xHalf;
                          InputStream imHALF = getAssets().open("cil_" +
porcentaje1 + ".png");

       Cilindro1Estatico.setImageDrawable(Drawable.createFromStream(imHALF,null)
);

                          ///// Recoge el valor en un text view para usarlo en
pantalla resultados.//////
                          String DatoCilPorHALF1 = String.valueOf(porcentaje1);
                          txtDato1EstaticoHALF.setText(DatoCilPorHALF1);
                          System.out.println("Porcentaje 1 : " + porcentaje1);
```

```
                    }else if(valor1 <= 300){

                          double AjPorcentualOK1 = 0.006614;
                          double AjPorcentualOK2 = -3.353;
                          double AjPorcentualOK3 = 510.7;

                          double xOK
=((AjPorcentualOK1*((valor1)^2))+(AjPorcentualOK2*valor1)+AjPorcentualOK3);
                          int porcentaje1 = (int)xOK;
                          InputStream imOK = getAssets().open("cil_" +
porcentaje1 + ".png");

      Cilindro1Estatico.setImageDrawable(Drawable.createFromStream(imOK,null));

                          ///// Recoge el valor en un text view para usarlo en
pantalla resultados//////
                          String DatoCilPorOK1 = String.valueOf(porcentaje1);
                          txtDato1EstaticoOK.setText(DatoCilPorOK1);
                          System.out.println("Porcentaje 1 : " + porcentaje1);

      ////////////////////////////////////////////////////////////////////////
//////
                    }


              //////////Porcentaje cilindro 2 //////////
              int valor2 = datos[1];

              double fx2 =
((AjPorcentual1*((valor2)^2))+(AjPorcentual2*valor2)+AjPorcentual3);

              if (fx2>100){
                    int porcentajeOK = 100;
                    InputStream isOK = getAssets().open("cil"+ porcentajeOK +
".png");

      Cilindro2Estatico.setImageDrawable(Drawable.createFromStream(isOK,
null));
                    txtDatoPresion2Ok.setVisibility(View.INVISIBLE);
                    txtDatoPresion2Half.setVisibility(View.INVISIBLE);
                    txtDatoPresion2Bad.setVisibility(View.INVISIBLE);
              }else {

              int porcentaje2 = (int)fx2;
              InputStream is2 = getAssets().open("cil_" + porcentaje2 + ".png");
              Cilindro2Estatico.setImageDrawable(Drawable.createFromStream(is2,
null));
              String DatoEstatico2 = String.valueOf(porcentaje2);
              txtDato2Estatico.setText(DatoEstatico2);

              System.out.println("porcentaje2:" + porcentaje2);
              }

              //////////Porcentaje cilindro 3 //////////
              int valor3 = datos[3];
```

```
            double fx3 =
((AjPorcentual1*((valor3)^2))+(AjPorcentual2*valor3)+AjPorcentual3);

            if (fx3>100){
                  int porcentajeOK = 100;
                  InputStream isOK = getAssets().open("cil"+ porcentajeOK +
".png");

      Cilindro3Estatico.setImageDrawable(Drawable.createFromStream(isOK,
null));
                  txtDatoPresion3Ok.setVisibility(View.INVISIBLE);
                  txtDatoPresion3Half.setVisibility(View.INVISIBLE);
                  txtDatoPresion3Bad.setVisibility(View.INVISIBLE);
            }

            int porcentaje3 = (int)fx3;
            InputStream is3 = getAssets().open("cil_" + porcentaje3 + ".png");
            Cilindro3Estatico.setImageDrawable(Drawable.createFromStream(is3,
null));
            String DatoEstatico3 = String.valueOf(porcentaje3);
            txtDato3Estatico.setText(DatoEstatico3);

            System.out.println("porcentaje3:" + porcentaje3);

      //////////Realiza la conversion de datos entrantes en porcentajes
visuales para el analisis estatico. //////////
      private void AnalisisTiempoRealEstatico (int[] datos) {
            System.out.println("entro analisis time real");
            try {
                  ////////// Porcentaje cilindro 1 //////////
            int valor1 = datos[0];
                  if (valor1 < 45){
                        int porcentaje1 = 100;
                        InputStream is1 =
getAssets().open("cil_"+porcentaje1+".png");

      Cilindro1Estatico.setImageDrawable(Drawable.createFromStream(is1, null));

                        txtDatoPresion1Ok.setVisibility(View.INVISIBLE);
                        txtDatoPresion1Half.setVisibility(View.INVISIBLE);
                        txtDatoPresion1Bad.setVisibility(View.INVISIBLE);

                  }else if(valor1 > 73){
                        int porcentaje1 = 0;
                        InputStream is1 =
getAssets().open("cil_"+porcentaje1+".png");

      Cilindro1Estatico.setImageDrawable(Drawable.createFromStream(is1, null));

                        txtDatoPresion1Ok.setVisibility(View.INVISIBLE);
                        txtDatoPresion1Half.setVisibility(View.INVISIBLE);
                        txtDatoPresion1Bad.setVisibility(View.INVISIBLE);

                  }
```

```
                        double fx1 = ((p1*((valor1)^2))+(p2*valor1)+p3);
                        int porcentaje1 = (int)fx1;
                        System.out.println("porcentaje1:" + porcentaje1);
                        InputStream is1 = getAssets().open("cil_" +
porcentaje1 + ".png");

      Cilindro1Estatico.setImageDrawable(Drawable.createFromStream(is1, null));
                        String DatoEstatico1 = String.valueOf(porcentaje1);
                        txtDato1Estatico.setText(DatoEstatico1);

                  //////// Porcentaje cilindro 2 //////////
                  int valor2 = datos[1];
                  if (valor2 < 45){
                        int porcentaje2 = 100;
                        InputStream is2 =
getAssets().open("cil_"+porcentaje2+".png");

      Cilindro2Estatico.setImageDrawable(Drawable.createFromStream(is2, null));

                        txtDatoPresion2Ok.setVisibility(View.INVISIBLE);
                        txtDatoPresion2Half.setVisibility(View.INVISIBLE);
                        txtDatoPresion2Bad.setVisibility(View.INVISIBLE);

                  }else if(valor2 > 73){
                        int porcentaje2 =0;
                        InputStream is2 =
getAssets().open("cil_"+porcentaje2+".png");

      Cilindro2Estatico.setImageDrawable(Drawable.createFromStream(is2, null));

                        txtDatoPresion2Ok.setVisibility(View.INVISIBLE);
                        txtDatoPresion2Half.setVisibility(View.INVISIBLE);
                        txtDatoPresion2Bad.setVisibility(View.INVISIBLE);
                  }
                        double fx2 = ((p1*((valor2)^2))+(p2*valor2)+p3);
                        int porcentaje2 = (int)fx2;
                        System.out.println("porcentaje2:" + porcentaje2);
                        InputStream is2 = getAssets().open("cil_" +
porcentaje2 + ".png");

      Cilindro2Estatico.setImageDrawable(Drawable.createFromStream(is2, null));
                        String DatoEstatico2 = String.valueOf(porcentaje2);
                        txtDato2Estatico.setText(DatoEstatico2);

                  ////// Porcentaje cilindro 3 //////////
                  int valor3 = datos[2];
                  if (valor3 < 45){
                        int porcentaje3 = 100;
                        InputStream is3 = getAssets().open("cil_" +
porcentaje3 + ".png");

      Cilindro3Estatico.setImageDrawable(Drawable.createFromStream(is3, null));

                        txtDatoPresion3Ok.setVisibility(View.INVISIBLE);
```

```
                        txtDatoPresion3Half.setVisibility(View.INVISIBLE);
                        txtDatoPresion3Bad.setVisibility(View.INVISIBLE);

                  }else if(valor3 > 73){
                        int porcentaje3 = 0;
                        InputStream is3 = getAssets().open("cil_" +
porcentaje3 + ".png");

      Cilindro3Estatico.setImageDrawable(Drawable.createFromStream(is3, null));

                        txtDatoPresion3Ok.setVisibility(View.INVISIBLE);
                        txtDatoPresion3Half.setVisibility(View.INVISIBLE);
                        txtDatoPresion3Bad.setVisibility(View.INVISIBLE);
                  }
                        double fx3 = ((p1*((valor3)^2))+(p2*valor3)+p3);
                        int porcentaje3 = (int)fx3;
                        System.out.println("porcentaje3:" + porcentaje3);
                        InputStream is3 = getAssets().open("cil_" +
porcentaje3 + ".png");

      Cilindro3Estatico.setImageDrawable(Drawable.createFromStream(is3, null));
                        String DatoEstatico3 = String.valueOf(porcentaje3);
                        txtDato3Estatico.setText(DatoEstatico3);

                  ////// Porcentaje cilindro 4 //////////
                  int valor4 = datos[2];
                  double fx4 = ((p1*((valor4)^2))+(p2*valor4)+p3);


                  if (valor4 < 45){
                        int porcentaje4 =100;
                        InputStream is4 = getAssets().open("cil_" +
porcentaje4 + ".png");

      Cilindro4Estatico.setImageDrawable(Drawable.createFromStream(is4, null));

                        txtDatoPresion4Ok.setVisibility(View.INVISIBLE);
                        txtDatoPresion4Half.setVisibility(View.INVISIBLE);
                        txtDatoPresion4Bad.setVisibility(View.INVISIBLE);

                  }else if(valor4 > 73){
                        int porcentaje4 =0;
                        InputStream is4 = getAssets().open("cil_" +
porcentaje4 + ".png");

      Cilindro4Estatico.setImageDrawable(Drawable.createFromStream(is4, null));

                        txtDatoPresion4Ok.setVisibility(View.INVISIBLE);
                        txtDatoPresion4Half.setVisibility(View.INVISIBLE);
                        txtDatoPresion4Bad.setVisibility(View.INVISIBLE);
                  }

                        int porcentaje4 = (int)fx4;
                        System.out.println("porcentaje4:" + porcentaje4);
```

```
                        InputStream is4 = getAssets().open("cil_" +
porcentaje4 + ".png");

      Cilindro4Estatico.setImageDrawable(Drawable.createFromStream(is4, null));
                        String DatoEstatico4 = String.valueOf(porcentaje4);
                        txtDato4Estatico.setText(DatoEstatico4);

      ////////////////////////////////////////////////////////////////////
//////////////////////////


                        double PCil1 = ((pUno * (porcentaje1^2)) +
(pDos*porcentaje1));
                        int PresionCil1 = (int)PCil1;
                        String DatoPresion1 = String.valueOf(PresionCil1);
                        txtPresionDato1.setText(DatoPresion1);

                        double PCil2 = ((pUno * (porcentaje2^2)) +
(pDos*porcentaje2));
                        int PresionCil2 = (int)PCil2;
                        String DatoPresion2 = String.valueOf(PresionCil2);
                        txtPresionDato2.setText(DatoPresion2);

                        double PCil3 = ((pUno * (porcentaje3^2)) +
(pDos*porcentaje3));
                        int PresionCil3 = (int)PCil3;
                        String DatoPresion3 = String.valueOf(PresionCil3);
                        txtPresionDato3.setText(DatoPresion3);

                        double PCil4 = ((pUno * (porcentaje4^2)) +
(pDos*porcentaje4));
                        int PresionCil4 = (int)PCil4;
                        String DatoPresion4 = String.valueOf(PresionCil4);
                        txtPresionDato4.setText(DatoPresion4);

                        File ruta_sd =
Environment.getExternalStorageDirectory();
                        File f = new File(ruta_sd.getAbsolutePath(),
"ProToTyPe.txt");

                        OutputStreamWriter fout = new OutputStreamWriter(new
FileOutputStream(f));

                        fout.write("ANALISIS DEL MOTOR" + "\n " + "\n"
                                    + "# CILINDRO" + "       " + "EFICIENCIA
(%)" + "     " + " PRESÓN (PSI) " + "\n" + "\n"
                                    + "Cilindro 1" + "            " +
porcentaje1 +" %" + "            " + PresionCil1 + " PSI" + "
" + "\n"
                                    + "Cilindro 2" + "            " +
porcentaje2 +" %" + "            " + PresionCil2 + " PSI" + "
" + "\n"
```

```
                                        + "Cilindro 3" + "            " +
porcentaje3 +" %" + "            " + PresionCil3 + " PSI" + "
" + "\n"
                                        + "Cilindro 4" + "            " +
porcentaje4 +" %" + "            " + PresionCil4 + " PSI" + "
" + "\n"
                                        );
                              fout.close();

                        } catch (IOException e) {
                              e.printStackTrace();
                              System.out.println("Error al guardar Datos");
                        }


                  ////////////////////////////////////Progress Bar CILINDRO 1
//////////////////////

                  malo1 = (ProgressBar) findViewById(R.id.progressBar0Estatico); //
Para tomar el progressBar adecuado.
                  medio1 = (ProgressBar) findViewById(R.id.progressBar1Estatico); //
Para tomar el progressBar adecuado.
                  bueno1 = (ProgressBar) findViewById(R.id.progressBar2Estatico); //
Para tomar el progressBar adecuado.

                  malo1.setMax(100); // Colocar un maximo de 100 al ProgressBar.
                  medio1.setMax(100); // Colocar un maximo de 100 al ProgressBar.
                  bueno1.setMax(100); // Colocar un maximo de 100 al ProgressBar.

                  malo1.setProgress(0); // Coloca como minimo de 0 al ProgressBar.
                  medio1.setProgress(0);// Coloca como minimo de 0 al ProgressBar.
                  bueno1.setProgress(0);// Coloca como minimo de 0 al ProgressBar.

                  ////////// Inicia la operacion en segundo plano para mostrar el
valor en la barra de progreso. //////////

                  int a = datos[0];
                  double fxA = ((p1*((a)^2))+(p2*a)+p3);
                  int i1 = (int)fxA;

                  if (i1 < 0){
                        try{
                        InputStream is1 = getAssets().open("cil_0.png");

            Cilindro1Estatico.setImageDrawable(Drawable.createFromStream(is1, null));
                        //////// Para las Presiones. //////////
                        txtDatoPresion1Bad.setVisibility(View.INVISIBLE);
                        txtDatoPresion1Ok.setVisibility(View.INVISIBLE);
                        txtDatoPresion1Half.setVisibility(View.INVISIBLE);

                        txt1Presion0.setVisibility(View.VISIBLE);
                        txt1Presion115.setVisibility(View.INVISIBLE);
                        }catch(IOException e){}

                  } else if (i1 < 75) {
```

```
            ////////// Para las barras de progreso. //////////
            malo1.setProgress(i1);
            malo1.setVisibility(View.VISIBLE);
            medio1.setVisibility(View.GONE);
            bueno1.setVisibility(View.GONE);

            //////// Para las Presiones. //////////
            double Pp1Bad = ((pUno*((i1)^2))+(pDos * (i1)));
            int Presion1Bad = (int)Pp1Bad;
            String DatoPresion1Bad = String.valueOf(Presion1Bad);
            txtDatoPresion1Bad.setText(DatoPresion1Bad + " PSI");

            txtDatoPresion1Bad.setVisibility(View.VISIBLE);
            txtDatoPresion1Ok.setVisibility(View.INVISIBLE);
            txtDatoPresion1Half.setVisibility(View.INVISIBLE);
            txt1Presion0.setVisibility(View.INVISIBLE);
            txt1Presion115.setVisibility(View.INVISIBLE);

        } else if (i1 < 90) {
            medio1.setProgress(i1);
            medio1.setVisibility(View.VISIBLE);
            malo1.setVisibility(View.GONE);
            bueno1.setVisibility(View.GONE);

            //////// Para las Presiones. //////////
            double Pp1Half = ((pUno*((i1)^2))+(pDos * (i1)));
            int Presion1Half = (int)Pp1Half;
            String DatoPresion1Half = String.valueOf(Presion1Half);
            txtDatoPresion1Half.setText(DatoPresion1Half + " PSI");

            txtDatoPresion1Half.setVisibility(View.VISIBLE);
            txtDatoPresion1Bad.setVisibility(View.INVISIBLE);
            txtDatoPresion1Ok.setVisibility(View.INVISIBLE);
            txt1Presion0.setVisibility(View.INVISIBLE);
            txt1Presion115.setVisibility(View.INVISIBLE);


        } else if (i1 <= 100) {
            bueno1.setProgress(i1);
            bueno1.setVisibility(View.VISIBLE);
            malo1.setVisibility(View.GONE);
            medio1.setVisibility(View.GONE);

            //////// Para las Presiones. //////////
            double Pp1Ok = ((pUno*((i1)^2))+(pDos * (i1)));
            int Presion1Ok = (int)Pp1Ok;
            String DatoPresion1Ok = String.valueOf(Presion1Ok);
            txtDatoPresion1Ok.setText(DatoPresion1Ok + " PSI");

            txtDatoPresion1Ok.setVisibility(View.VISIBLE);
            txtDatoPresion1Half.setVisibility(View.INVISIBLE);
            txtDatoPresion1Bad.setVisibility(View.INVISIBLE);
            txt1Presion0.setVisibility(View.INVISIBLE);
            txt1Presion115.setVisibility(View.INVISIBLE);
```

```
            }else if (i1 > 100) {
                  txt1Presion0.setVisibility(View.INVISIBLE);
                  txt1Presion115.setVisibility(View.VISIBLE);
            }

            ////////////////////////////////////Progress Bar CILINDRO 2


            /////////Inicia la operacion en segundo plano para mostrar el
valor en la barra de progreso. /////////
            int b = datos[1];
            double fxB = ((p1*((b)^2))+(p2*b)+p3);
            int i2 = (int)fxB;

            if (i2 < 0){

                  try{
                  InputStream is1 = getAssets().open("cil_0.png");

      Cilindro1Estatico.setImageDrawable(Drawable.createFromStream(is1, null));
                  //////// Para las Presiones. /////////
                  txtDatoPresion2Bad.setVisibility(View.INVISIBLE);
                  txtDatoPresion2Ok.setVisibility(View.INVISIBLE);
                  txtDatoPresion2Half.setVisibility(View.INVISIBLE);

                  txt2Presion0.setVisibility(View.VISIBLE);
                  txt2Presion115.setVisibility(View.INVISIBLE);
                  }catch (IOException e){}

            } else if (i2 < 75) {
                  malo2.setProgress(i2);
                  malo2.setVisibility(View.VISIBLE);
                  medio2.setVisibility(View.GONE);
                  bueno2.setVisibility(View.GONE);

                  //////// Para las Presiones. /////////
                  double Pp2Bad = ((pUno*((i2)^2))+(pDos * (i2)));
                  int Presion2Bad = (int)Pp2Bad;
                  String DatoPresion2Bad = String.valueOf(Presion2Bad);
                  txtDatoPresion2Bad.setText(DatoPresion2Bad + " PSI");

                  txtDatoPresion2Bad.setVisibility(View.VISIBLE);
                  txtDatoPresion2Ok.setVisibility(View.INVISIBLE);
                  txtDatoPresion2Half.setVisibility(View.INVISIBLE);
                  txt2Presion0.setVisibility(View.INVISIBLE);
                  txt2Presion115.setVisibility(View.INVISIBLE);

            } else if (i2 < 90) {
                  medio2.setProgress(i2);
                  medio2.setVisibility(View.VISIBLE);
                  malo2.setVisibility(View.GONE);
                  bueno2.setVisibility(View.GONE);

                  //////// Para las Presiones. /////////
                  double Pp2Half = ((pUno*((i2)^2))+(pDos * (i2)));
```

```
                int Presion2Half = (int)Pp2Half;
                String DatoPresion2Half = String.valueOf(Presion2Half);
                txtDatoPresion2Half.setText(DatoPresion2Half + " PSI");

                txtDatoPresion2Half.setVisibility(View.VISIBLE);
                txtDatoPresion2Bad.setVisibility(View.INVISIBLE);
                txtDatoPresion2Ok.setVisibility(View.INVISIBLE);
                txt2Presion0.setVisibility(View.INVISIBLE);
                txt2Presion115.setVisibility(View.INVISIBLE);

            } else if (i2 <= 100) {
                bueno2.setProgress(i2);
                bueno2.setVisibility(View.VISIBLE);
                malo2.setVisibility(View.GONE);
                medio2.setVisibility(View.GONE);

                //////// Para las Presiones. //////////
                double Pp2Ok = ((pUno*((i2)^2))+(pDos * (i2)));
                int Presion2Ok = (int)Pp2Ok;
                String DatoPresion2Ok = String.valueOf(Presion2Ok);
                txtDatoPresion2Ok.setText(DatoPresion2Ok + " PSI");

                txtDatoPresion2Ok.setVisibility(View.VISIBLE);
                txtDatoPresion2Half.setVisibility(View.INVISIBLE);
                txtDatoPresion2Bad.setVisibility(View.INVISIBLE);
                txt2Presion0.setVisibility(View.INVISIBLE);
                txt2Presion115.setVisibility(View.INVISIBLE);
            }else if (i2 > 100) {
                txt2Presion0.setVisibility(View.INVISIBLE);
                txt2Presion115.setVisibility(View.VISIBLE);
            }

            ////////////////////////////////////Progress Bar CILINDRO 3

                // progressBar

            ////////// Inicia la operacion en segundo plano para mostrar el
valor en la barra de progreso. //////////
            int c = datos[2];
            double fxC = ((p1*((c)^2))+(p2*c)+p3);
            int i3 = (int)fxC;


            if (i3 < 0){
                //////// Para las Presiones. //////////

                txtDatoPresion3Bad.setVisibility(View.INVISIBLE);
                txtDatoPresion3Ok.setVisibility(View.INVISIBLE);
                txtDatoPresion3Half.setVisibility(View.INVISIBLE);

                txt3Presion0.setVisibility(View.VISIBLE);
                txt3Presion115.setVisibility(View.INVISIBLE);

            }else if (i3 < 75) {
                malo3.setProgress(i3);
```

```
        malo3.setVisibility(View.VISIBLE);
        medio3.setVisibility(View.GONE);
        bueno3.setVisibility(View.GONE);

        //////// Para las Presiones. //////////
        double Pp3Bad = ((pUno*((i3)^2))+(pDos * (i3)));
        int Presion3Bad = (int)Pp3Bad;
        String DatoPresion3Bad = String.valueOf(Presion3Bad);
        txtDatoPresion3Bad.setText(DatoPresion3Bad + " PSI");

        txtDatoPresion3Bad.setVisibility(View.VISIBLE);
        txtDatoPresion3Ok.setVisibility(View.INVISIBLE);
        txtDatoPresion3Half.setVisibility(View.INVISIBLE);
        txt3Presion0.setVisibility(View.INVISIBLE);
        txt3Presion115.setVisibility(View.INVISIBLE);

    } else if (i3 < 90) {
        medio3.setProgress(i3);
        medio3.setVisibility(View.VISIBLE);
        malo3.setVisibility(View.GONE);
        bueno3.setVisibility(View.GONE);

        //////// Para las Presiones. //////////
        double Pp3Half = ((pUno*((i3)^2))+(pDos * (i3)));
        int Presion3Half = (int)Pp3Half;
        String DatoPresion3Half = String.valueOf(Presion3Half);
        txtDatoPresion3Half.setText(DatoPresion3Half + " PSI");

        txtDatoPresion3Half.setVisibility(View.VISIBLE);
        txtDatoPresion3Bad.setVisibility(View.INVISIBLE);
        txtDatoPresion3Ok.setVisibility(View.INVISIBLE);
        txt3Presion0.setVisibility(View.INVISIBLE);
        txt3Presion115.setVisibility(View.INVISIBLE);

    } else if (i3 <= 100) {
        bueno3.setProgress(i3);
        bueno3.setVisibility(View.VISIBLE);
        malo3.setVisibility(View.GONE);
        medio3.setVisibility(View.GONE);

        //////// Para las Presiones. //////////
        double Pp3Ok = ((pUno*((i3)^2))+(pDos * (i3)));
        int Presion3Ok = (int)Pp3Ok;
        String DatoPresion3Ok = String.valueOf(Presion3Ok);
        txtDatoPresion3Ok.setText(DatoPresion3Ok + " PSI");

        txtDatoPresion3Ok.setVisibility(View.VISIBLE);
        txtDatoPresion3Half.setVisibility(View.INVISIBLE);
        txtDatoPresion3Bad.setVisibility(View.INVISIBLE);
        txt3Presion0.setVisibility(View.INVISIBLE);
        txt3Presion115.setVisibility(View.INVISIBLE);
    } else if (i3 > 100) {
        txt3Presion0.setVisibility(View.INVISIBLE);
        txt3Presion115.setVisibility(View.VISIBLE);
    }
```

```
            ////////////////////////////////////Progress Bar CILINDRO 4

                        // progressBar

            ////////// Inicia la operacion en segundo plano para mostrar el
valor en la barra de progreso. //////////
            int d = datos[3];
            double fxD = ((p1*((d)^2))+(p2*d)+p3);

            int i4 = (int)fxD;

            if (i4 < 0){
                  //////// Para las Presiones. //////////
                  txtDatoPresion4Bad.setVisibility(View.INVISIBLE);
                  txtDatoPresion4Ok.setVisibility(View.INVISIBLE);
                  txtDatoPresion4Half.setVisibility(View.INVISIBLE);

                  txt4Presion0.setVisibility(View.VISIBLE);
                  txt4Presion115.setVisibility(View.INVISIBLE);

            }else if (i4 < 75) {
                  malo4.setProgress(i4);
                  malo4.setVisibility(View.VISIBLE);
                  medio4.setVisibility(View.GONE);
                  bueno4.setVisibility(View.GONE);

                  ///////// Presion Clilindro 3. //////////
                  double Pp4Bad = ((pUno*((i4)^2))+(pDos * (i4)));
                  int Presion4Bad = (int)Pp4Bad;
                  String DatoPresion4Bad = String.valueOf(Presion4Bad);
                  txtDatoPresion4Bad.setText(DatoPresion4Bad + " PSI");

                  txtDatoPresion4Bad.setVisibility(View.VISIBLE);
                  txtDatoPresion4Ok.setVisibility(View.INVISIBLE);
                  txtDatoPresion4Half.setVisibility(View.INVISIBLE);
                  txt4Presion0.setVisibility(View.INVISIBLE);
                  txt4Presion115.setVisibility(View.INVISIBLE);

            } else if (i4 < 90) {
                  medio4.setProgress(i4);
                  medio4.setVisibility(View.VISIBLE);
                  malo4.setVisibility(View.GONE);
                  bueno4.setVisibility(View.GONE);

                  double Pp4Half = ((pUno*((i4)^2))+(pDos * (i4)));
                  int Presion4Half = (int)Pp4Half;
                  String DatoPresion4Half = String.valueOf(Presion4Half);
                  txtDatoPresion4Half.setText(DatoPresion4Half + " PSI");

                  txtDatoPresion4Half.setVisibility(View.VISIBLE);
                  txtDatoPresion4Bad.setVisibility(View.INVISIBLE);
                  txtDatoPresion4Ok.setVisibility(View.INVISIBLE);
                  txt4Presion0.setVisibility(View.INVISIBLE);
                  txt4Presion115.setVisibility(View.INVISIBLE);
```

```
            } else if (i4 <= 100) {
                  bueno4.setProgress(i4);
                  bueno4.setVisibility(View.VISIBLE);
                  malo4.setVisibility(View.GONE);
                  medio4.setVisibility(View.GONE);

                  double Pp4Ok = ((pUno*((i4)^2))+(pDos * (i4)));
                  int Presion4Ok = (int)Pp4Ok;
                  String DatoPresion4Ok = String.valueOf(Presion4Ok);
                  txtDatoPresion4Ok.setText(DatoPresion4Ok + " PSI");

                  txtDatoPresion4Ok.setVisibility(View.VISIBLE);
                  txtDatoPresion4Half.setVisibility(View.INVISIBLE);
                  txtDatoPresion4Bad.setVisibility(View.INVISIBLE);
                  txt4Presion0.setVisibility(View.INVISIBLE);
                  txt4Presion115.setVisibility(View.INVISIBLE);
            } else if (i4 > 100) {
                  txt4Presion0.setVisibility(View.INVISIBLE);
                  txt4Presion115.setVisibility(View.VISIBLE);
            }
////////////////////////////////////////////////////////////////////////////
      }

      /////////Realiza la conversion de datos entrantes en porcentajes
visuales para el analisis Dinamico. //////////
      private void AnalisisTiempoRealDinamico (int[] datos) {
            System.out.println("entro analisis time real");

            try {
                  ////////// Porcentaje cilindro 1 //////////
            int valor1Din = datos[0];
                  if (valor1Din < 45){
                        int porcentaje1Din = 25;
                        InputStream is1Din =
getAssets().open("cildin_"+porcentaje1Din+".png");

      cilindro1Dinamico.setImageDrawable(Drawable.createFromStream(is1Din,
null));

                        txtDatoPresion1OkDin.setVisibility(View.INVISIBLE);
                        txtDatoPresion1HalfDin.setVisibility(View.INVISIBLE);
                        txtDatoPresion1BadDin.setVisibility(View.INVISIBLE);


                  }else if(valor1Din > 73){
                        int porcentaje1Din = 0;
                        InputStream is1Din =
getAssets().open("cildin_"+porcentaje1Din+".png");

      cilindro1Dinamico.setImageDrawable(Drawable.createFromStream(is1Din,
null));

                        txtDatoPresion1OkDin.setVisibility(View.INVISIBLE);
                        txtDatoPresion1HalfDin.setVisibility(View.INVISIBLE);
```

```
                        txtDatoPresion1BadDin.setVisibility(View.INVISIBLE);


                    } else {
                        double fx1Din =
((p1Din*((valor1Din)^2))+(p2Din*valor1Din)+p3Din);
                        int porcentaje1Din = (int)fx1Din;
                        System.out.println("porcentaje1:" + porcentaje1Din);
                        InputStream is1Din = getAssets().open("cildin_" +
porcentaje1Din + ".png");

      cilindro1Dinamico.setImageDrawable(Drawable.createFromStream(is1Din,
null));
                    }

                    //////// Porcentaje cilindro 2 //////////
                    int valor2Din = datos[1];
                    if (valor2Din < 45){
                        int porcentaje2Din = 25;
                        InputStream is2Din =
getAssets().open("cildin_"+porcentaje2Din+".png");

      cilindro2Dinamico.setImageDrawable(Drawable.createFromStream(is2Din,
null));

                        txtDatoPresion2OkDin.setVisibility(View.INVISIBLE);
                        txtDatoPresion2HalfDin.setVisibility(View.INVISIBLE);
                        txtDatoPresion2BadDin.setVisibility(View.INVISIBLE);

                    }else if(valor2Din > 73){
                        int porcentaje2Din =0;
                        InputStream is2Din =
getAssets().open("cildin_"+porcentaje2Din+".png");

      cilindro2Dinamico.setImageDrawable(Drawable.createFromStream(is2Din,
null));

                        txtDatoPresion2OkDin.setVisibility(View.INVISIBLE);
                        txtDatoPresion2HalfDin.setVisibility(View.INVISIBLE);
                        txtDatoPresion2BadDin.setVisibility(View.INVISIBLE);

                    } else{
                        double fx2Din =
((p1Din*((valor2Din)^2))+(p2Din*valor2Din)+p3Din);
                        int porcentaje2Din = (int)fx2Din;
                        System.out.println("porcentaje2:" + porcentaje2Din);
                        InputStream is2Din = getAssets().open("cildin_" +
porcentaje2Din + ".png");

      cilindro2Dinamico.setImageDrawable(Drawable.createFromStream(is2Din,
null));

                    }

                    ////// Porcentaje cilindro 3 //////////
```

```
                    int valor3Din = datos[2];
                    if (valor3Din < 45){
                          int porcentaje3Din = 25;
                          InputStream is3Din = getAssets().open("cildin_" +
porcentaje3Din + ".png");

      cilindro3Dinamico.setImageDrawable(Drawable.createFromStream(is3Din,
null));

                          txtDatoPresion3OkDin.setVisibility(View.INVISIBLE);
                          txtDatoPresion3HalfDin.setVisibility(View.INVISIBLE);
                          txtDatoPresion3BadDin.setVisibility(View.INVISIBLE);

                    }else if(valor3Din > 73){
                          int porcentaje3Din = 0;
                          InputStream is3Din = getAssets().open("cil_" +
porcentaje3Din + ".png");

      cilindro3Dinamico.setImageDrawable(Drawable.createFromStream(is3Din,
null));

                          txtDatoPresion3OkDin.setVisibility(View.INVISIBLE);
                          txtDatoPresion3HalfDin.setVisibility(View.INVISIBLE);
                          txtDatoPresion3BadDin.setVisibility(View.INVISIBLE);
                    } else {
                          double fx3Din =
((p1Din*((valor3Din)^2))+(p2Din*valor3Din)+p3Din);
                          int porcentaje3Din = (int)fx3Din;
                          System.out.println("porcentaje3:" + porcentaje3Din);
                          InputStream is3Din = getAssets().open("cildin_" +
porcentaje3Din + ".png");

      cilindro3Dinamico.setImageDrawable(Drawable.createFromStream(is3Din,
null));

                    }

                    ////// Porcentaje cilindro 4 //////////
                    int valor4Din = datos[3];

                    if (valor4Din < 45){
                          int porcentaje4Din =25;
                          InputStream is4Din = getAssets().open("cildin_" +
porcentaje4Din + ".png");

      cilindro4Dinamico.setImageDrawable(Drawable.createFromStream(is4Din,
null));

                          txtDatoPresion4OkDin.setVisibility(View.INVISIBLE);
                          txtDatoPresion4HalfDin.setVisibility(View.INVISIBLE);
                          txtDatoPresion4BadDin.setVisibility(View.INVISIBLE);

                    }else if(valor4Din > 73){
                          int porcentaje4Din =0;
```

```
                        InputStream is4Din = getAssets().open("cildin_" +
porcentaje4Din + ".png");

      cilindro4Dinamico.setImageDrawable(Drawable.createFromStream(is4Din,
null));

                        txtDatoPresion4OkDin.setVisibility(View.INVISIBLE);
                        txtDatoPresion4HalfDin.setVisibility(View.INVISIBLE);
                        txtDatoPresion4BadDin.setVisibility(View.INVISIBLE);

                  }else{
                        double fx4Din =
((p1Din*((valor4Din)^2))+(p2Din*valor4Din)+p3Din);
                        int porcentaje4Din = (int)fx4Din;
                        System.out.println("porcentaje4:" + porcentaje4Din);
                        InputStream is4Din = getAssets().open("cildin_" +
porcentaje4Din + ".png");

      cilindro4Dinamico.setImageDrawable(Drawable.createFromStream(is4Din,
null));

                  }

            } catch (IOException e) {
                  e.printStackTrace();
                  System.out.println("Error en la LECTURA");
            }

            ////////////////////////////////////Progress Bar CILINDRO 1
//////////////////////

            malo1Din = (ProgressBar) findViewById(R.id.progressBar0Dinamico);
// Para tomar el progressBar adecuado.
            medio1Din = (ProgressBar) findViewById(R.id.progressBar1Dinamico);
// Para tomar el progressBar adecuado.
            bueno1Din = (ProgressBar) findViewById(R.id.progressBar2Dinamico);
// Para tomar el progressBar adecuado.

            malo1Din.setMax(25); // Colocar un maximo de 100 al ProgressBar.
            medio1Din.setMax(25); // Colocar un maximo de 100 al ProgressBar.
            bueno1Din.setMax(25); // Colocar un maximo de 100 al ProgressBar.

            malo1Din.setProgress(0); // Coloca como minimo de 0 al ProgressBar.
            medio1Din.setProgress(0);// Coloca como minimo de 0 al ProgressBar.
            bueno1Din.setProgress(0);// Coloca como minimo de 0 al ProgressBar.

            ////////// Inicia la operacion en segundo plano para mostrar el
valor en la barra de progreso. //////////

            int aDin = datos[0];
            double fxADin = ((p1Din*((aDin)^2))+(p2Din*aDin)+p3Din);
            int i1Din = (int)fxADin;

            if (i1Din < 0){
                  //////// Para las Presiones. //////////
```

```
                    txtDatoPresion1BadDin.setVisibility(View.INVISIBLE);
                    txtDatoPresion1OkDin.setVisibility(View.INVISIBLE);
                    txtDatoPresion1HalfDin.setVisibility(View.INVISIBLE);

                    txt1Presion0Din.setVisibility(View.VISIBLE);
                    txt1Presion115Din.setVisibility(View.INVISIBLE);

              } else if (i1Din < 20) {
                    ////////// Para las barras de progreso. //////////
                    malo1Din.setProgress(i1Din);
                    malo1Din.setVisibility(View.VISIBLE);
                    medio1Din.setVisibility(View.GONE);
                    bueno1Din.setVisibility(View.GONE);

                    //////// Para las Presiones. //////////
                    int i11Din = (i1Din * 4);
                    double Pp1BadDin = ((ppBad1*((i11Din)^2))+(ppBad2 *
(i11Din))+ppBad3);
                    int Presion1BadDin = (int)Pp1BadDin;
                    String DatoPresion1BadDin = String.valueOf(Presion1BadDin);
                    txtDatoPresion1BadDin.setText(DatoPresion1BadDin + " PSI");

                    txtDatoPresion1BadDin.setVisibility(View.VISIBLE);
                    txtDatoPresion1OkDin.setVisibility(View.INVISIBLE);
                    txtDatoPresion1HalfDin.setVisibility(View.INVISIBLE);
                    txt1Presion0Din.setVisibility(View.INVISIBLE);
                    txt1Presion115Din.setVisibility(View.INVISIBLE);

              } else if (i1Din < 23) {
                    medio1Din.setProgress(i1Din);
                    medio1Din.setVisibility(View.VISIBLE);
                    malo1Din.setVisibility(View.GONE);
                    bueno1Din.setVisibility(View.GONE);

                    //////// Para las Presiones. //////////
                    int i11Din = (i1Din * 4);
                    double Pp1HalfDin = ((ppHalf1*((i11Din^2)))+(ppHalf2 *
(i11Din))+ppHalf3);
                    int Presion1HalfDin = (int)Pp1HalfDin;
                    String DatoPresion1HalfDin = String.valueOf(Presion1HalfDin);
                    txtDatoPresion1HalfDin.setText(DatoPresion1HalfDin + " PSI");

                    txtDatoPresion1HalfDin.setVisibility(View.VISIBLE);
                    txtDatoPresion1BadDin.setVisibility(View.INVISIBLE);
                    txtDatoPresion1OkDin.setVisibility(View.INVISIBLE);
                    txt1Presion0Din.setVisibility(View.INVISIBLE);
                    txt1Presion115Din.setVisibility(View.INVISIBLE);


              } else if (i1Din <= 25) {
                    bueno1Din.setProgress(i1Din);
                    bueno1Din.setVisibility(View.VISIBLE);
                    malo1Din.setVisibility(View.GONE);
                    medio1Din.setVisibility(View.GONE);
```

```
                    //////// Para las Presiones. //////////
                    int i11Din = (i1Din * 4);
                    double Pp1OkDin = (((ppOk1*((i11Din)^2)))+(ppOk2 *
(i11Din))+ppOk3);
                    int Presion1OkDin = (int)Pp1OkDin;
                    String DatoPresion1OkDin = String.valueOf(Presion1OkDin);
                    txtDatoPresion1OkDin.setText(DatoPresion1OkDin + " PSI");

                    txtDatoPresion1OkDin.setVisibility(View.VISIBLE);
                    txtDatoPresion1HalfDin.setVisibility(View.INVISIBLE);
                    txtDatoPresion1BadDin.setVisibility(View.INVISIBLE);
                    txt1Presion0Din.setVisibility(View.INVISIBLE);
                    txt1Presion115Din.setVisibility(View.INVISIBLE);

             }else if (i1Din > 25) {
                    txt1Presion0Din.setVisibility(View.INVISIBLE);
                    txt1Presion115Din.setVisibility(View.VISIBLE);
             }
////////////////////////////////////////////////////////////////////////////

             ////////////////Progress Bar CILINDRO 2

             malo2Din = (ProgressBar) findViewById(R.id.progressBar3Dinamico);
// Para tomar el progressBar adecuado.
             medio2Din = (ProgressBar) findViewById(R.id.progressBar4Dinamico);
// Para tomar el progressBar adecuado.
             bueno2Din = (ProgressBar) findViewById(R.id.progressBar5Dinamico);
// Para tomar el progressBar adecuado.

             malo2Din.setMax(25); // Colocar un maximo de 100 al ProgressBar.
             medio2Din.setMax(25); // Colocar un maximo de 100 al ProgressBar.
             bueno2Din.setMax(25); // Colocar un maximo de 100 al ProgressBar.

             malo2Din.setProgress(0); // Coloca como minimo de 0 al ProgressBar.
             medio2Din.setProgress(0);// Coloca como minimo de 0 al ProgressBar.
             bueno2Din.setProgress(0);// Coloca como minimo de 0 al ProgressBar.



             //////////Inicia la operacion en segundo plano para mostrar el
valor en la barra de progreso. //////////
             int bDin = datos[1];
             double fxBDin = ((p1Din*((bDin)^2))+(p2Din*bDin)+p3Din);
             int i2Din = (int)fxBDin;

             if (i2Din < 0){
                    //////// Para las Presiones. //////////
                    txtDatoPresion2BadDin.setVisibility(View.INVISIBLE);
                    txtDatoPresion2OkDin.setVisibility(View.INVISIBLE);
                    txtDatoPresion2HalfDin.setVisibility(View.INVISIBLE);

                    txt2Presion0Din.setVisibility(View.VISIBLE);
                    txt2Presion115Din.setVisibility(View.INVISIBLE);

             } else if (i2Din < 20) {
```

```
                    malo2Din.setProgress(i2Din);
                    malo2Din.setVisibility(View.VISIBLE);
                    medio2Din.setVisibility(View.GONE);
                    bueno2Din.setVisibility(View.GONE);

                    //////// Para las Presiones. //////////
                    int i22Din = (i2Din * 4);
                    double Pp2BadDin = ((ppBad1*((i22Din)^2))+(ppBad2 *
(i22Din))+ppBad3);
                    int Presion2BadDin = (int)Pp2BadDin;
                    String DatoPresion2BadDin = String.valueOf(Presion2BadDin);
                    txtDatoPresion2BadDin.setText(DatoPresion2BadDin + " PSI");

                    txtDatoPresion2BadDin.setVisibility(View.VISIBLE);
                    txtDatoPresion2OkDin.setVisibility(View.INVISIBLE);
                    txtDatoPresion2HalfDin.setVisibility(View.INVISIBLE);
                    txt2Presion0Din.setVisibility(View.INVISIBLE);
                    txt2Presion115Din.setVisibility(View.INVISIBLE);

              } else if (i2Din < 23) {
                    medio2Din.setProgress(i2Din);
                    medio2Din.setVisibility(View.VISIBLE);
                    malo2Din.setVisibility(View.GONE);
                    bueno2Din.setVisibility(View.GONE);

                    //////// Para las Presiones. //////////
                    int i22Din = (i2Din * 4);
                    double Pp2HalfDin = ((ppHalf1*((i22Din)^2))+(ppHalf2 *
(i22Din))+ppHalf3);
                    int Presion2HalfDin = (int)Pp2HalfDin;
                    String DatoPresion2HalfDin = String.valueOf(Presion2HalfDin);
                    txtDatoPresion2HalfDin.setText(DatoPresion2HalfDin + " PSI");

                    txtDatoPresion2HalfDin.setVisibility(View.VISIBLE);
                    txtDatoPresion2BadDin.setVisibility(View.INVISIBLE);
                    txtDatoPresion2OkDin.setVisibility(View.INVISIBLE);
                    txt2Presion0Din.setVisibility(View.INVISIBLE);
                    txt2Presion115Din.setVisibility(View.INVISIBLE);

              } else if (i2Din <= 25) {
                    bueno2Din.setProgress(i2Din);
                    bueno2Din.setVisibility(View.VISIBLE);
                    malo2Din.setVisibility(View.GONE);
                    medio2Din.setVisibility(View.GONE);

                    //////// Para las Presiones. //////////
                    int i22Din = (i2Din * 4);
                    double Pp2OkDin = ((ppOk1*((i22Din)^2))+(ppOk2 *
(i22Din))+ppOk3);
                    int Presion2OkDin = (int)Pp2OkDin;
                    String DatoPresion2OkDin = String.valueOf(Presion2OkDin);
                    txtDatoPresion2OkDin.setText(DatoPresion2OkDin + " PSI");

                    txtDatoPresion2OkDin.setVisibility(View.VISIBLE);
                    txtDatoPresion2HalfDin.setVisibility(View.INVISIBLE);
```

```
                txtDatoPresion2BadDin.setVisibility(View.INVISIBLE);
                txt2Presion0Din.setVisibility(View.INVISIBLE);
                txt2Presion115Din.setVisibility(View.INVISIBLE);
            }else if (i2Din > 25) {
                txt2Presion0Din.setVisibility(View.INVISIBLE);
                txt2Presion115Din.setVisibility(View.VISIBLE);
            }

            ////////////////////////////////////Progress Bar CILINDRO 3

            malo3Din = (ProgressBar) findViewById(R.id.progressBar6Dinamico);
// Para tomar el progressBar adecuado.
            medio3Din = (ProgressBar) findViewById(R.id.progressBar7Dinamico);
// Para tomar el progressBar adecuado.
            bueno3Din = (ProgressBar) findViewById(R.id.progressBar8Dinamico);
// Para tomar el progressBar adecuado.

                                                        // progressBar


            //////////Inicia la operacion en segundo plano para mostrar el
valor en la barra de progreso. //////////
            int cDin = datos[2];
            double fxCDin = ((p1Din*((cDin)^2))+(p2Din*cDin)+p3Din);
            int i3Din = (int)fxCDin;

            if (i3Din < 0){
                ///////// Para las Presiones. //////////
                txtDatoPresion3BadDin.setVisibility(View.INVISIBLE);
                txtDatoPresion3OkDin.setVisibility(View.INVISIBLE);
                txtDatoPresion3HalfDin.setVisibility(View.INVISIBLE);

                txt3Presion0Din.setVisibility(View.VISIBLE);
                txt3Presion115Din.setVisibility(View.INVISIBLE);

            } else if (i3Din < 20) {
                malo3Din.setProgress(i3Din);
                malo3Din.setVisibility(View.VISIBLE);
                medio3Din.setVisibility(View.GONE);
                bueno3Din.setVisibility(View.GONE);

                ///////// Para las Presiones. //////////
                int i33Din = (i3Din * 4);
                double Pp3BadDin = ((ppBad1*((i33Din)^2))+(ppBad2 *
(i33Din))+ppBad3);
                int Presion3BadDin = (int)Pp3BadDin;
                String DatoPresion3BadDin = String.valueOf(Presion3BadDin);
                txtDatoPresion3BadDin.setText(DatoPresion3BadDin + " PSI");

                txtDatoPresion3BadDin.setVisibility(View.VISIBLE);
                txtDatoPresion3OkDin.setVisibility(View.INVISIBLE);
                txtDatoPresion3HalfDin.setVisibility(View.INVISIBLE);
                txt3Presion0Din.setVisibility(View.INVISIBLE);
                txt3Presion115Din.setVisibility(View.INVISIBLE);
```

```
            } else if (i3Din < 23) {
                  medio3Din.setProgress(i3Din);
                  medio3Din.setVisibility(View.VISIBLE);
                  malo3Din.setVisibility(View.GONE);
                  bueno3Din.setVisibility(View.GONE);

                  //////// Para las Presiones. //////////
                  int i33Din = (i3Din * 4);
                  double Pp3HalfDin = ((ppHalf1*((i33Din)^2))+(ppHalf2 *
(i33Din))+ppHalf3);
                  int Presion3HalfDin = (int)Pp3HalfDin;
                  String DatoPresion3HalfDin = String.valueOf(Presion3HalfDin);
                  txtDatoPresion3HalfDin.setText(DatoPresion3HalfDin + " PSI");

                  txtDatoPresion3HalfDin.setVisibility(View.VISIBLE);
                  txtDatoPresion3BadDin.setVisibility(View.INVISIBLE);
                  txtDatoPresion3OkDin.setVisibility(View.INVISIBLE);
                  txt3Presion0Din.setVisibility(View.INVISIBLE);
                  txt3Presion115Din.setVisibility(View.INVISIBLE);

            } else if (i3Din <= 25) {
                  bueno3Din.setProgress(i3Din);
                  bueno3Din.setVisibility(View.VISIBLE);
                  malo3Din.setVisibility(View.GONE);
                  medio3Din.setVisibility(View.GONE);

                  //////// Para las Presiones. //////////
                  int i33Din = (i3Din * 4);
                  double Pp3OkDin = ((ppOk1*((i33Din)^2))+(ppOk2 *
(i33Din))+ppOk3);
                  int Presion3OkDin = (int)Pp3OkDin;
                  String DatoPresion3OkDin = String.valueOf(Presion3OkDin);
                  txtDatoPresion3OkDin.setText(DatoPresion3OkDin + " PSI");

                  txtDatoPresion3OkDin.setVisibility(View.VISIBLE);
                  txtDatoPresion3HalfDin.setVisibility(View.INVISIBLE);
                  txtDatoPresion3BadDin.setVisibility(View.INVISIBLE);
                  txt3Presion0Din.setVisibility(View.INVISIBLE);
                  txt3Presion115Din.setVisibility(View.INVISIBLE);
            }else if (i3Din > 25) {
                  txt3Presion0Din.setVisibility(View.INVISIBLE);
                  txt3Presion115Din.setVisibility(View.VISIBLE);
            }
            ////////////////////////////////////Progress Bar CILINDRO 4
//////////////////////
            malo4Din = (ProgressBar) findViewById(R.id.progressBar9Dinamico);
// Para tomar el progressBar adecuado.
            medio4Din = (ProgressBar) findViewById(R.id.progressBar10Dinamico);
// Para tomar el progressBar adecuado.
            bueno4Din = (ProgressBar) findViewById(R.id.progressBar11Dinamico);
// Para tomar el progressBar adecuado.

            malo4Din.setMax(25); // Colocar un maximo de 100 al ProgressBar.
            medio4Din.setMax(25); // Colocar un maximo de 100 al ProgressBar.
            bueno4Din.setMax(25); // Colocar un maximo de 100 al ProgressBar.
```

```
            malo4Din.setProgress(0); // Coloca como minimo de 0 al ProgressBar.
            medio4Din.setProgress(0);// Coloca como minimo de 0 al ProgressBar.
            bueno4Din.setProgress(0);// Coloca como minimo de 0 al ProgressBar.

            // progressBar


            //////////Inicia la operacion en segundo plano para mostrar el
valor en la barra de progreso. //////////
            int dDin = datos[3];
            double fxDDin = ((p1Din*((dDin)^2))+(p2Din*dDin)+p3Din);
            int i4Din = (int)fxDDin;

            if (i4Din < 0){
                  //////// Para las Presiones. //////////
                  txtDatoPresion4BadDin.setVisibility(View.INVISIBLE);
                  txtDatoPresion4OkDin.setVisibility(View.INVISIBLE);
                  txtDatoPresion4HalfDin.setVisibility(View.INVISIBLE);

                  txt4Presion0Din.setVisibility(View.VISIBLE);
                  txt4Presion115Din.setVisibility(View.INVISIBLE);

            } else if (i4Din < 20) {
                  malo4Din.setProgress(i4Din);
                  malo4Din.setVisibility(View.VISIBLE);
                  medio4Din.setVisibility(View.GONE);
                  bueno4Din.setVisibility(View.GONE);

                  //////// Para las Presiones. //////////
                  int i44Din = (i4Din * 4);
                  double Pp4BadDin = ((ppBad1*((i44Din)^2))+(ppBad2 *
(i44Din))+ppBad3);
                  int Presion4BadDin = (int)Pp4BadDin;
                  String DatoPresion4BadDin = String.valueOf(Presion4BadDin);
                  txtDatoPresion4BadDin.setText(DatoPresion4BadDin + " PSI");

                  txtDatoPresion4BadDin.setVisibility(View.VISIBLE);
                  txtDatoPresion4OkDin.setVisibility(View.INVISIBLE);
                  txtDatoPresion4HalfDin.setVisibility(View.INVISIBLE);
                  txt4Presion0Din.setVisibility(View.INVISIBLE);
                  txt4Presion115Din.setVisibility(View.INVISIBLE);

            } else if (i4Din < 23) {
                  medio4Din.setProgress(i4Din);
                  medio4Din.setVisibility(View.VISIBLE);
                  malo4Din.setVisibility(View.GONE);
                  bueno4Din.setVisibility(View.GONE);

                  //////// Para las Presiones. //////////
                  int i44Din = (i4Din * 4);
                  double Pp4HalfDin = ((ppHalf1*((i44Din)^2))+(ppHalf2 *
(i44Din))+ppHalf3);
                  int Presion4HalfDin = (int)Pp4HalfDin;
                  String DatoPresion4HalfDin = String.valueOf(Presion4HalfDin);
```

```
                txtDatoPresion4HalfDin.setText(DatoPresion4HalfDin + " PSI");

                txtDatoPresion4HalfDin.setVisibility(View.VISIBLE);
                txtDatoPresion4BadDin.setVisibility(View.INVISIBLE);
                txtDatoPresion4OkDin.setVisibility(View.INVISIBLE);
                txt4Presion0Din.setVisibility(View.INVISIBLE);
                txt4Presion115Din.setVisibility(View.INVISIBLE);

            } else if (i4Din <= 25) {
                bueno4Din.setProgress(i4Din);
                bueno4Din.setVisibility(View.VISIBLE);
                malo4Din.setVisibility(View.GONE);
                medio4Din.setVisibility(View.GONE);

                //////// Para las Presiones. //////////
                int i44Din = (i4Din * 4);
                double Pp4OkDin = ((ppOk1*((i44Din)^2))+(ppOk2 *
(i44Din))+ppOk3);
                int Presion4OkDin = (int)Pp4OkDin;
                String DatoPresion4OkDin = String.valueOf(Presion4OkDin);
                txtDatoPresion4OkDin.setText(DatoPresion4OkDin + " PSI");

                txtDatoPresion4OkDin.setVisibility(View.VISIBLE);
                txtDatoPresion4HalfDin.setVisibility(View.INVISIBLE);
                txtDatoPresion4BadDin.setVisibility(View.INVISIBLE);
                txt4Presion0Din.setVisibility(View.INVISIBLE);
                txt4Presion115Din.setVisibility(View.INVISIBLE);
            }else if (i4Din > 25) {
                txt4Presion0Din.setVisibility(View.INVISIBLE);
                txt4Presion115Din.setVisibility(View.VISIBLE);
            }
        }

        /////////// Para tener la pantalla de resultados. ///////////
        public void Resultados (View v){
            Intent result = new Intent(this,
ResultadosActivityPrototype.class);
            ////////// Datos porcentuales. //////////
            result.putExtra("Dato1EstaticoOK", txtDato1EstaticoOK.getText());
            result.putExtra("Dato1EstaticoHALF",
txtDato1EstaticoHALF.getText());
            result.putExtra("Dato1EstaticoBAD", txtDato1EstaticoBAD.getText());

//          result.putExtra("Dato2Estatico", txtDato2Estatico.getText());
            result.putExtra("Dato3Estatico", txtDato3Estatico.getText());
            result.putExtra("Dato4Estatico", txtDato4Estatico.getText());

            ////////// Datos de presion. //////////
            result.putExtra("PresionCil1", txtPresionDato1.getText());
            result.putExtra("PresionCil2", txtPresionDato2.getText());
            result.putExtra("PresionCil3", txtPresionDato3.getText());
            result.putExtra("PresionCil4", txtPresionDato4.getText());

            startActivity(result);
        }}
```