Original research / Artículo original / Pesquisa original - Tipo 1

Implementation of an OBD-II Diagnostics Tool over CAN-BUS with Arduino

Armando Rodríguez Rodríguez / arrodriguez@upr.edu.cu

José Raúl Vento Álvarez / vento@upr.edu.cu

Ricardo Inouye Rodriguez / richard@upr.edu.cu

Universidad de Pinar del Río, Cuba

ABSTRACT From its origin, the main objective of the OBD [On Board Diagnostics] standard has been the control of the gases emitted by the vehicles and its corresponding effects in the environment. This project implements a system based on the OBD-II protocol over a CAN [Controller Area Network] bus, which allows the visualization of variables in real-time and the performing of a diagnosis of the vehicle state showing the operating, failure, and energy consumption codes. The on board diagnosis systems allow to retrieve the stored failure codes together with a large number of variables –important for the diagnosis– such as speed, fuel level, and CO2 [Carbon Dioxide] emissions in real-time. We implemented an OBD-II system located at the end of the test vehicle (a bus) –which corresponds to the scanner or diagnosis unit– in an Arduino Mega 2560 development board connected to a CAN transceiver-controlled composed module. The scanner has a USB connection that eases the graphical visualization of data in a PC through an interface created in LabVIEW.

KEYWORDS OBD-II; CAN; Arduino; ECU.

Implementación de una herramienta para diagnóstico OBD-II sobre CAN-BUS con Arduino

RESUMEN Desde su surgimiento, el objetivo fundamental del estándar OBD [On Board Diagnostics] ha sido el control de los gases emitidos por los automóviles y de sus efectos en el medio ambiente. Este proyecto implementa un sistema basado en el protocolo OBD-II sobre un bus CAN [Controller Area Network], que permite visualizar variables en tiempo real y realizar un diagnóstico del estado del automóvil que muestra los códigos de funcionamiento, falla y rendimiento energético. Los sistemas de diagnóstico abordo permiten conocer los códigos de fallo almacenados y un gran número de variables de especial relevancia, como la velocidad, el nivel de combustible y el nivel de emisión de dióxido de carbono, en tiempo real. Se implementó un sistema OBD-II centrado en el extremo del bus, que corresponde al escáner o unidad de diagnóstico, en una placa Arduino Mega 2560 conectada a un módulo compuesto transceiver-controller CAN. El scanner posee una conexión USB que facilita visualizar los datos recuperados de forma versátil en una PC a través de una interfaz gráfica creada en LabVIEW™.

PALABRAS CLAVE OBD-II; CAN; Arduino; ECU.

Implementação de uma ferramenta para diagnóstico OBD-II em CAN-BUS com Arduino

RESUMO Desde a sua criação, o objetivo fundamental do padrão OBD [On Board Diagnóstics] tem sido o controle dos gases de escape em automóveis e seus efeitos sobre o meio ambiente. Este projeto implementa um sistema baseado no protocolo OBD-II sobre um CAN [Controller Area Network] BUS, que permite visualizar variáveis em tempo real e realizar um diagnóstico do estado do automóvel que mostre os códigos de operação, falha e eficiência energética. Os sistemas de diagnóstico a bordo permitem conhecer os códigos de falhas armazenadas e um grande número de variáveis de especial relevância, como a velocidade, o nível de combustível e o nível de emissões de dióxido de carbono, em tempo real. Foi implementado um sistema OBD-II centrado no extremo do BUS correspondente ao scanner ou unidade de diagnóstico, em uma placa Arduino Mega 2560 conectada a um módulo composto transceiver-controller CAN. O scanner possui uma conexão USB que facilita a visualização dos dados recuperados de forma versátil em um PC através de uma interface gráfica criada no LabVIEW™.

PALAVRAS-CHAVE OBD-II; CAN; Arduino; ECU.

Received / Recepción: Jan. 4, 2018 - Accepted / Aceptación: Feb. 2, 2018 *doi:10.18046/syt.v16i45.2747*

I. Introducción

The current Cuban scenario shows an accelerated expansion of the automotive sector associated to the increasing in tourism (non-heavy vehicles) and the amount of trucking vehicles in the roads. In most of the cases, cars in these two divisions have advanced technological features, which include onboard intelligent systems, failure detection, and performance analysis systems including energetic efficiency and environmental effects.

These advances have allowed to simplify the vehicle repairing tasks, seeking to detect potential failures that might damage it, and to centralize the variables visualization in real-time; nowadays, these values can be read from the ECU [Electronic Control Unit] and there are several examples of commercial systems providing the ability to perform vehicle state diagnostics (xTool, 2017).

Some efforts have been developed in the last decade leveraging the applications research related with OBD-II [On Board Diagnostics II], implemented over a CAN bus embedded into the onboard diagnostics systems in automobiles (Simbaña, Caiza, Chávez, & López, 2016; Rayo 2009).

The present project implements an onboard diagnostics system based on the OBD-II protocol over a CAN bus that will allow both the real-time variables visualization and the diagnostics performance for the vehicle state. Our proposal shows the stored trouble codes and it allows their deletion after solved. The management of this process if performed through a graphical interface developed in LabVIEW.

II. OBD-II Systems

The OBD-II system started to be mandatory in every new vehicle in the United States from 1996, looking for monitoring components that might affect the control system of the gas emission and to measure real-time parameters such as temperature, pressure, and speed. When the OBD-II system detects a problem through its transductors, a warning light is turned on in the instrument panel, alerting the driver of the problem. Furthermore, it stores the information of the detected failures in the computer memory, which eases the technical personnel to find problems and correct them (Meseguer, 2013).

A. Components of the OBD System

The main components of the OBD-II system are: the ECU, also known as the vehicle computer; the transductors, which send the data towards the ECU; the MIL [Malfunction Indicator Lamp], located in the instrument

I. Introducción

El escenario cubano actual presenta un acelerado desarrollo del transporte automotor asociado al crecimiento del turismo y al manejo de cargas por carretera. En la mayoría de los casos, el parque disponible de vehículos en estas dos esferas es de avanzada tecnología, que incluye sistemas inteligentes abordo y sistemas de diagnóstico de fallas y análisis de funcionamiento, que incluyen eficiencia energética y efectos medioambientales.

Estos avances han permitido simplificar la reparación del automóvil, detectar fallos potenciales que pudieran dañarlo aún más y simplificar y centralizar la visualización de variables en tiempo real, que ahora pueden ser leídas directamente desde el ordenador central o Unidad de Control Electrónico [ECU, Electronic Control Unit].

En la actualidad son varios los ejemplos de sistemas comerciales que brindan la posibilidad de realizar diagnósticos del estado del automóvil (xTool, 2017).

En la última década se han desarrollado varios esfuerzos en la investigación de aplicaciones relacionadas con OBD-II [On Board Diagnostics II] implementadas sobre un bus CAN incorporado a los sistemas de diagnosis de abordo en vehículos automotrices (Simbaña, Caiza, Chávez, & López, 2016; Rayo 2009).

El presente proyecto implementa un sistema de diagnóstico abordo basado en el protocolo de diagnosis OBD-II sobre un CAN bus que permitirá, tanto visualizar variables en tiempo real, como realizar un diagnóstico del estado del automóvil , que muestre los códigos de falla almacenados y permita borrarlos una vez reparados. La gestión de este proceso se realiza a través de una interfaz gráfica creada empleando el software LabVIEW™.

II. Sistemas OBD-II

El sistema OBD-II se comenzó a utilizar de forma obligatoria en los nuevos automóviles en los Estados Unidos de América desde 1996, con el objetivo de monitorear los componentes que afecten el sistema de control de las emisiones de gases contaminantes y medir parámetros en tiempo real, tales como: temperaturas, presiones y velocidades. Cuando el sistema OBD-II detecta algún problema a través de los transductores, se enciende una luz de advertencia en el tablero, que alerta al conductor de la falla existente. Además, guarda la información sobre las fallas detectadas en la memoria de la computadora del automóvil, lo que facilita al técnico automotriz encontrar los problemas para corregirlos posteriormente (Meseguer, 2013).

A. Componentes del sistema OBD

Los componentes del sistema OBD-II son: la ECU, conocida como la computadora del automóvil; los transductores, encargados de enviar los datos hacia la ECU; la luz indicadora de fallas [MIL, Malfunction Indicator Light], ubicada en el tablero; y el conector de diagnóstico [DLC, Data Link Connector], que sirve de interfaz entre la ECU y los dispositivos de diagnóstico automotriz.

Computadora del automóvil

La ECU es la computadora del automóvil y su función principal es obtener y manejar los datos provenientes de todos los

transductores del motor del automóvil. Es un dispositivo que se encuentra generalmente debajo del tablero en la parte del conductor (Cervantes & Espinosa, 2010).

Transductores del automóvil

Los transductores son los dispositivos encargados de monitorear de forma continua el funcionamiento y operación del motor del automóvil (García, 2015). Las medidas que pueden obtener los transductores del motor incluyen: el número de revoluciones por minuto, la temperatura del líquido refrigerante, la presión absoluta del colector de admisión, la presión barométrica, la temperatura del aire de admisión, la posición del acelerador y la velocidad del automóvil.

En la **Tabla 1** se presentan algunos transductores del motor del automóvil que permiten obtener las medidas mencionadas y que hacen que la ECU pueda determinar la cantidad de combustible, el punto de ignición y otros parámetros necesarios para evaluar las condiciones del automóvil (Zabler, 2002).

Table 1. Transductors of the vehicles / Transductores del motor del automóvil

Sensor	Measurement
Crankshaft Position [CKP]	Revolutions per minute
	Revoluciones por minuto (rpm)
Engine Coolant Temperature [ECT]	Temperature of the engine cooling liquid
	Temperatura del refrigerante del motor
Manifold Absolute Pressure [MAP]	Absolute pressure of the manifold
	Presión absoluta del colector de admisión
Barometer	Measuring barometric pressure
	Presión barométrica
Intake Air Temperature [IAT]	Temperature of the intake air
	Temperatura del aire de admisión
Throttle Position Sensor [TPS]	Position of the accelerator (throttle)
	Posición del acelerador
Vehicle Speed Sensor [VSS]	Speed of the vehicle
	Velocidad del automóvil

Luz indicadora de fallas

La MIL es utilizada por el sistema OBD-II y se enciende cuando los transductores del motor detectan un problema en el automóvil, su propósito es alertar al conductor acerca de la necesidad de realizar un mantenimiento del mismo.

Data Link Conector

El DLC es una interfaz con forma trapezoidal de 16 pines basado en el estándar SAE J1962, que se ubica bajo el tablero, generalmente en el lado del conductor. Sirve como interfaz de acceso y recuperación de datos desde la ECU hacia un equipo de diagnóstico (escáner automotriz) (McCord, 2011). En la **Tabla 2** se muestra la descripción de los 16 pines de este conector.

B. Modos de medición

El sistema OBD-II utiliza nueve modos de medición, cada uno de ellos permite el acceso a los datos de la ECU del automóvil (ver **Tabla 3**). Para solicitar datos de un automóvil es necesaria la utilización de códigos PID [Parameter Identification]. Cada PID está relacionado con una

panel; and the DLC [Data Link Connector], which is an interface between the ECU and the devices for diagnostics.

Computer in the Vehicle

The ECU is the computer in the vehicle and its main function is to collect and process data coming from all the engine transductors. It is a device generally located under the instrument panel in the driving seat (Cervantes & Espinosa, 2010).

Transductors

These are the devices in charge of continuously monitor the engine operation (García, 2015). Data collected by the transductors include: the number of revolutions per minute [RPM], temperature of the cooling liquid, manifold absolute pressure, barometric pressure, admission air temperature, position of the gas pedal, and speed.

In **Table 1** we present some engine transductors in charge of gathering the mentioned variables and allowing the ECU to determine the fuel amount, ignition point, and other necessary parameters to assess the current vehicle conditions (Zabler, 2002).

Malfunction indicator lamp

The MIL is used by the OBD-II system to indicate a problem in the engine, detected by the transductors. Its purpose is to alert the driver about the need to perform a maintenance for the vehicle.

Data Link Connector

It is a trapezoid-shaped interface with 16 pins based on the SAE J1962 standard, located under the instrument panel, generally in the driver side. Its function is to be an access and data collecting interface from the ECU towards a diagnostics equipment (automotive scanner) (McCord, 2011). In **Table 2**, we show a brief description of each DLC pin.

B. Measuring Modes

The OBD-II system uses 9 measuring modes, each one of them allows the access to the ECU data in the vehicle (see **Table 3**). In order to request data, it is necessary to use PID [Parameter Identification] codes. Each PID is related with a specific measurement of the modes 1 and 2 of the OBD-II system. For instance, if the real-time datum of the vehicle speed is requested, the mode 1 should be chosen and the PID "0D" has to be used.

C. ECU Communication with External Devices

Within the OBD-II system, the communication protocols allow to establish a bidirectional communication to

Table 2. Description of the DLC pins / Descripción de los pines del DLC

Pin	Features	Pin	Features
1	Manufacturer internal usage Uso del fabricante	9	Manufacturer internal usage Uso del fabricante
2	J1850 VPM & PWM positive bus Bus (+) J1850 VPM y PWM	10	J1850 negative bus *Bus (-) J1850*
3	Manufacturer internal usage Uso del fabricante	11	Manufacturer internal usage *Uso del fabricante*
4	Ground (chassis) Tierra (chasis)	12	Manufacturer internal usage *Uso del fabricante*
5	Ground signal Señal de Tierra	13	Manufacturer internal usage *Uso del fabricante*
6	High CAN data bus (J-2284) Bus de datos CAN alto (J-2284)	14	Low CAN data bus (J-2284) *Bus de datos CAN bajo (J-2284)*
7	K ISO 9141-2 line Línea K ISO 9141-2	15	L ISO 9141-2 line *Línea L ISO 9141-2*
8	Manufacturer internal usage Uso del fabricante	16	Battery voltage *Voltaje de batería*

share information between a diagnostics tool (automotive scanner) and the vehicle ECU.

The supported communication protocols for the OBD-II system include the SAE J1850 PWM (pulse width modulation at 41.6 kbps), SAE J1850 (variable pulse width at 10.4 – 41.6 kbps), ISO 9141-2 (asynchronous serial communication at 10.4 kbaud), ISO

medida específica de los modos 1 y 2 del sistema OBD-II. Por ejemplo, si se desea solicitar el dato en tiempo real de la velocidad del automóvil, se debe ingresar al modo 1 y utilizar el PID 0D.

C. Comunicación de la ECU con dispositivos externos

En el sistema OBD-II, los protocolos de comunicación permiten establecer comunicación e intercambiar mensaje de forma bidireccional, entre una herramienta de diagnóstico (escáner automotriz) y la ECU del automóvil.

Los protocolos de comunicación soportados por el sistema OBD-II incluyen el SAE J1850 PWM (modulación por ancho de pulso a 41.6 Kbps), el SAE J1850 VPW (ancho de pulso variable a 10.4 - 41.6 Kbps), el ISO 9141-2 (comunicación serial asincrónica a 10.4 Kbaud), el ISO 14230 KWP (comunicación serial asincrónica hasta 10.4 Kbaud), y el ISO 15765 CAN (250 - 500 Kbps).

III. Propuesta de arquitectura de la solución implementada

Las herramientas de diagnóstico OBD-II son las encargadas de gestionar el proceso de chequeo de desperfectos técnicos en los automóviles; forman parte de la evolución del sistema electrónico del vehículo con la incorporación de sistemas destinados a la mejora del rendimiento y a la comodidad y seguridad de los usuarios.

En esta investigación se plantea la implementación de un scanner OBD-II sobre un bus CAN, en una plataforma Arduino, y su monitoreo en una PC [Personal Computer] a través de LabVIEW™. La interfaz eléctrica encargada de manipular el

Table 3. Measuring modes supported by the OBD-II system / Modos de medición soportados por el Sistema OBD-II

Mode	Features
01	Collecting updated data: it allows the real-time access to the ECU inputs and outputs. *Obtención de datos actualizados: permite el acceso en tiempo real a los valores de salidas y entradas de la ECU.*
02	Access to frozen data frames: the ECU takes a sample of the values related with the emissions at the exact moment when a failure arises. *Acceso a cuadro de datos congelados: la ECU toma una muestra de los valores relacionados con las emisiones en el momento exacto de ocurrir un fallo.*
03	Gathering of the failure codes: it allows to extract all the DTC [Data Trouble Codes] stored in the ECU memory. *Obtención de los códigos de falla: permite extraer de la memoria de la ECU todos los códigos de fallo [DTC, Data Trouble Code] almacenados.*
04	Code erasing and failure in the stored values: it allows to delete all the stored codes in the ECU, including the DTC and the saved data frame. *Borrado de códigos de falla y valores almacenados: permite borrar todos los códigos almacenados en la ECU, incluyendo los DTC y el cuadro de datos grabado.*
05	Tests results in the oxygen transductors: it allows the access to the test results performed to the oxygen transductors. *Resultado de las pruebas de los transductores de oxígeno: permite el acceso a los resultados de las pruebas realizadas a los transductores de oxígeno.*
06	Tests results of other transductors: results of the diagnostics in components not submitted to constant surveillance *Resultados de las pruebas de otros transductores: resultado del diagnóstico en componentes que no están sujetos a vigilancia constante.*
07	Pending failure codes sampling: it allows to read all the pending DTC from the ECU memory. *Muestra de códigos de falla pendientes: permite leer de la memoria de la ECU todos los DTC pendientes.*
08	Components operating control: it permits the execution of tests in the actuators. *Control de funcionamiento de componentes: permite realizar pruebas en los actuadores.*
09	Vehicle information: it allows to request the VIN [Vehicle Identification Number] *Información del automóvil: permite solicitar el número de identificación del automóvil [VIN, Vehicle Identification Number].*

bus CAN está definida por el shield (placa de expansión) CAN para Arduino, que pertenece a la factoría de Seeed Studio. En la **Figura 1** se describe el sistema desarrollado.

El diagrama general del sistema diseñado, como se aprecia, está dividido en dos, la etapa de adquisición de datos (scanner) y la interfaz gráfica para la visualización y monitoreo de parámetros.

Adicionalmente, se cuenta con una herramienta portátil profesional de lectura de códigos OBD-II, como vía de comprobación de los resultados de la puesta en práctica del sistema. Además, se ha simulado, en otra plataforma Arduino, un módulo equivalente a la computadora del auto, el cual provee un comportamiento aproximado a la realidad.

A. Entornos de desarrollo

En el proceso de diseño del sistema se usan varios entornos de desarrollo diferentes: uno de alto nivel, en el cual se implementa la interfaz gráfica y, al mismo tiempo, la parte de procesamiento de los datos adquiridos a través del scanner OBD-II; otro, destinado a la programación de los elementos de hardware, en este caso la plataforma Arduino que actúa como adaptador USB-CAN.

La aplicación en alto nivel se desarrolló empleando el software LabVIEW™ 2011. Para la programación del hardware se empleó el entorno de desarrollo integrado o IDE de Arduino en versión 1.8.2.

LabVIEW™ 2011

Su nombre representa el acrónimo de Banco de pruebas de Laboratorio para Ingeniería de Instrumentación Virtual [Laboratory Virtual Instrumentation Engineering Workbench]. Es una plataforma y entorno de desarrollo para diseñar sistemas, con un lenguaje de programación visual gráfico.

Se recomienda para sistemas hardware y software de pruebas, control y diseño, simulado o real y embebido, pues acelera la productividad. Usa lenguaje G, donde la G indica que es lenguaje gráfico.

IDE de Arduino v1.8.2

Arduino necesita de un programa externo, ejecutado en otro ordenador, para poder escribir programas para la placa. Este software es lo que llamamos Arduino IDE [Integrated Development Environment] o Entorno de Desarrollo Integrado en español.

Arduino IDE es muy sencillo y está basado en Processing, un lenguaje de programación y entorno de desarrollo integrado de código abierto, basado en Java, que sirve como medio para la enseñanza y producción de proyectos multimedia e interactivos de diseño digital.

Para usarlo el procedimiento es el siguiente: se escribe un programa en el IDE y se carga en el Arduino; el programa se ejecutará automáticamente en la placa.

La estructura básica del lenguaje de programación de Arduino es bastante simple y se compone de al menos dos partes o funciones (setup y loop), que encierran bloques que contienen declaraciones, estamentos o instrucciones.

La función setup se encarga de recoger la configuración, es la primera función ejecutada dentro del programa y se ejecuta una única vez. Se utiliza, principalmente, para inicializar los

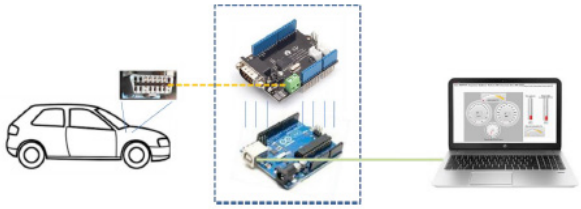

Figure 1. General diagram of the system / Esquema general del sistema

14230 KWP (asynchronous serial communication up to 10.4 kbaud), and ISO 15765 CAN (250 – 500 kbps).

III. Architecture Proposal for the Implemented Solution

The OBD-II diagnostics tools are the ones in charge of managing the checking process of technical failures in automobiles; they are part of the evolution in the vehicle electronic system with the incorporation of systems destined to improve the performance, comfort, and safety of the end users.

In this research paper, we propose to implement an OBD-II scanner over a CAN bus and using the Arduino platform by monitoring in a PC through LabVIEW. The interface in charge of controlling the CAN bus is defined by the CAN shield (expansion board) for Arduino, which is manufactured by Seed Studio. **Figure 1** presents the developed system.

The general diagram of the developed system is divided in two parts: the data gathering stage (scanner) and the graphical interface to visualize and monitor several parameters.

Additionally, we have a professional portable tool to read OBD-II codes as a way to check the results of our system. Besides, we have simulated an equivalent ECU or car computer –in another Arduino board–, which provides an approximate behavior to the reality.

A. Development Environments

Within the system design process, we used several development environments: a high level one –it hosts the graphical interface and, at the same time, the processing of the data collected from the OBD-II scanner–, and the other environment dedicated to program hardware elements, i.e., the Arduino platform acting as USB-CAN adapter.

The application in the high level was developed using the LabVIEW 2011 software. We employed the Arduino IDE [Integrated Development Environment] version 1.8.2 to program the hardware.

LabVIEW™ 2011

Its name represents the acronym for Laboratory Virtual Instrumentation Engineering Workbench. It is a plat-

form and environment to design systems with a graphical programming language; it is recommended for hardware and software testing and its corresponding control and design with real or embedded simulation, since it can speed up the productivity. It uses G language, where the G stands for using a graphical language.

Arduino v1.8.2 IDE

Arduino needs an external software –executed in a PC– to write programs in the development board. This software is commonly called Arduino IDE; it is relatively simple and it is based on Processing. This latter is an open source programming language and IDE based in Java; generally used as a medium to teach multimedia and interactive digital design projects.

In order to use the Arduino IDE, the following procedure is required: a program is coded in the IDE and it is loaded into the Arduino board. The program will execute immediately within it.

The basic structure of the Arduino programming language is simple and it is composed of at least two parts or functions ("setup" and "loop"), entailing code blocks that contain declarations, statements, or instructions.

The "setup" function is in charge of collecting the configuration, it is the first function running in the program and it is executed only once. It is mainly used to initialize the operating modes in the input/output [I/O] pins and to configure the serial communication; although it has many other additional utilities.

The "loop" function has the code to be continuously executed (input reading outputs triggering, etc.). This function is the core of all Arduino program and it is the one performing most of the load.

Arduino UNO R3 Board

Arduino UNO is the electronic board based on the ATmega328P microprocessor. It has 14 digital pins for input/output (6 of them with PWM features), 6 analog entries, a 16 MHz quartz crystal, a USB connection, a DC power outlet, a ICSP header, and a restart button (see **Figure 2**). The "UNO" name ("one" in Spanish and Italian) was chosen to commemorate the launch of the Arduino IDE 1.0. UNO is the first in a series of Arduino USB boards and the reference model for the platform.

We chose this board because the electric interface to control the CAN bus (previously described) is designed for total compatibility with Arduino UNO, i.e., it has the same dimensions of the board such as the pinout of the

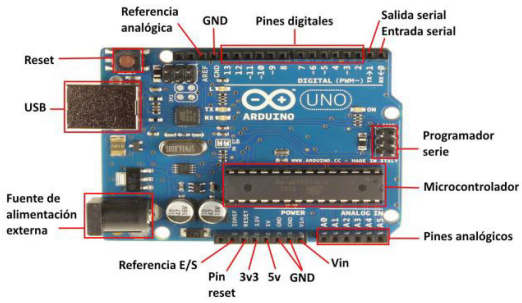

Figure 2. Arduino UNO R3 board / Placa Arduino UNO R3 ("Getting...", 2016)

modos de trabajo de los pines E/S y para configurar la comunicación serie, aunque tiene muchas utilidades adicionales.

La función loop, también conocida como función bucle, contiene el código que se ejecutará continuamente (lectura de entradas, activación de salidas, etc.). Esta función forma el núcleo de todos los programas de Arduino y es la que realiza la mayor parte del trabajo.

Placa Arduino UNO R3

Arduino UNO es una placa electrónica basada en el ATmega328P. Cuenta con catorce pines digitales de entrada/salida (de los cuales seis se pueden utilizar como salidas PWM), seis entradas analógicas, un cristal de cuarzo de 16 MHz, una conexión USB, un conector de alimentación, una cabecera ICSP y un botón de reinicio (véase la **Figura** 2). El nombre "UNO" fue elegido para conmemorar el lanzamiento del entorno de programación integrado Arduino (IDE) 1.0. UNO es el primero de una serie de placas Arduino USB y el modelo de referencia para la plataforma Arduino.

La selección de esta es que la interfaz eléctrica empleada para controlar el bus CAN (descrita posteriormente) está diseñada para una total compatibilidad con Arduino UNO, o sea, posee las mismas dimensiones de la placa, así como el pinout del controlador CAN para la comunicación vía SPI con la misma, descrita en la librería que la acompaña.

Seeed CAN bus shield para Arduino

CAN es uno de los protocolos de comunicación bus más usados debido a su largo alcance, su velocidad de comunicación y su alta fiabilidad. Se encuentra comúnmente en máquinas de control y en el bus de diagnóstico automotriz. La placa CAN bus Shield dota de conectividad CAN a la placa Arduino, para ello cuenta con el controlador CAN MCP2515, con interfaz SPI, encargado del preprocesamiento de los mensajes CAN, y con un transceiver CAN MCP2551, que maneja la interfaz eléctrica del bus. Sus principales características son:

- implementa CAN 2.0B a velocidades de hasta 1 Mbps;
- cuenta con una interfaz SPI de hasta 10 MHz;
- soporta tramas estándar (11 bits), extendidas (29 bits) y tramas remotas;
- cuenta con dos buffers de recepción, para el almacenamiento de mensajes con prioridad, un conector industrial estándar de 9 pines sub-D y dos indicadores LED;
- su voltaje de operación es de 5V, sus dimensiones 68x53 mm, y su peso 50 g.

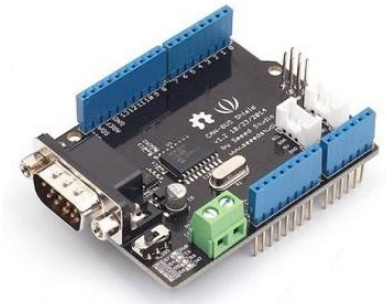

Figure 3. CAN bus Shield / Escudo del CAN bus ("Shield...", n.d)

Existen otras alternativas de implementación del bus CAN, como es el caso de la placa de expansión CAN de Sparkfun, pero ella no es totalmente compatible con Arduino UNO, por lo que es necesario modificar las librerías para establecer la comunicación entre ambas placas. Otro caso es la placa desarrollada por Microchip, OLIMEX PIC32-EMZ64, la cual implementa bus CAN; no obstante, su precio es elevado.

Simulador de computadora del vehículo

Es una representación a la cual se le han acoplado sensores y circuitos eléctricos que emulan parámetros como la velocidad, la aceleración y otros de tipo térmico, tales como la temperatura ambiente y la del refrigerante del motor. La **Figura 3** describe de forma general este subsistema.

Este simulador soporta varios modos de operación de los contemplados en el estándar OBD-II como son: el flujo de datos en tiempo real, la lectura de códigos de fallas y la barra de códigos de fallas.

Flujo de datos en tiempo real (modo 01)

Se pueden apreciar parámetros como la velocidad, la aceleración, la posición del pedal del acelerador, varios valores de temperatura, el estado del indicador MIL, el nivel de combustible y estadísticas relacionadas a la presencia de fallos en el motor, entre otros.

Lectura de códigos de fallas (DTC) (modo 03)

Se han programado dos códigos de error siguiendo los requisitos de la estructura del mensaje OBD, para la respuesta a una petición en modo 03. Adicionalmente, se incorporó un LED para simular el indicador MIL de la pizarra del auto, que se activa al detectar la presencia de fallos; se genera al presionar un botón, también implementado y conectado a la interrupción 1 de la placa Arduino UNO.

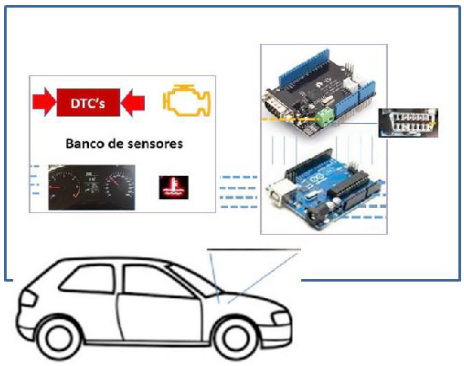

Figure 4. ECU simulator / Simulador ECU

CAN controller for the SPI communication. This is documented in the corresponding library.

Seeed CAN Bus Shield for Arduino

CAN is one of the most employed bus communication protocols due to its relatively long reach, communication speed, and high reliability. It is commonly implemented in control machines and in automotive diagnostics buses. The CAN bus shield board provides CAN connectivity to the Arduino by using the CAN MCP2515 controller with SPI interface. This latter is in charge of controlling the pre-processing of the CAN messages and it also has a CAN MCP2551 transceiver handling the bus electric interface.

Its main features are:

- It implements CAN 2.0B at speeds up to 1 Mbps;
- it has a SPI interface capable to operate at 10 MHz;
- it supports standard (11 bits), extended (29 bits), and remote frames;
- it has two receiving buffers for storing priority messages, a 9-pin sub-D industrial standard, and 2 LED indicators;
- its operating voltage is 5 V, its dimensions are 68 x 53 mm, and its weight is only 50 g.

There are other alternatives to implement the CAN bus, such as the case of the Sparkfun CAN expansion, but it is not completely compatible with Arduino UNO; a modification in the libraries to establish communication between both devices is required. Another example is the OLIMEX PIC32-EMZ64 board developed by Microchip, which implements CAN buses; however, its price is higher than the chosen one.

Vehicle Computer Simulator

It is a representation of an ECU with connected sensors and electric circuits emulating parameters such as speed, acceleration, and others (room temperature and engine coolant temperature). **Figure 4** describes this subsystem in a general way.

This simulator supports several operation models implemented in the OBD-II standard such as real-time data flow, DTC reading, and failure code bar.

Real-Time Data Flow (mode 01)

Parameters such as speed, acceleration, position of the gas pedal, several temperature values, MIL state, fuel level, and statistics related to failures in the engine, among others, can be measured.

DTC Reading (mode 03)

Two data trouble codes have been programmed following the requirements of the OBD message structure to reply a request in mode 03. Additionally, we added an LED to simulate the MIL indicator of the vehicle board –which is activated when a failure is detected–. This action of detecting a failure is triggered by pressing a button implemented and connected to the interruption 1 of the Arduino UNO.

The programmed trouble codes were:

- P0217, temperature excess in the engine; and
- P0143, low voltage in the oxygen sensor circuit (Bank 1, sensor 3).

Erase Trouble Codes (mode 04)

This mode erases the trouble codes and generates a positive response message for the operation mode 04 if the deletion was successful. At the same time, it turns off the LED indicating failures (MIL simulator). We added a shield to provide CAN connectivity and enable the OBD diagnostics.

B. Diagnostics Tool

The developed system was developed in an Arduino Mega 2560 board and its corresponding expansion card, providing CAN connectivity and allowing the connection with CAN networks.

We have programmed it to comply with the initial objectives in the OBD-II standard, such as real-time parameters measurement, trouble codes related with failures in the vehicle, and ability to delete the data in the computer once the maintenance in the vehicle solved the issues ("CANOBD2...", 2017).

Developed Scanning Software

The program in the Arduino Mega board was born with the objective to emulate an OBD-II diagnostics interface within it. The scanning software is in charge of making requests through the CAN bus and through the Arduino expansion board handling that data bus. The main program implements the following functionalities:

- read and process the received commands from the PC related to the diagnostics requests;
- elaborate CAN frames as per the previous commands and related with the 01, 03, and 04 operation modes of the OBD-II standard; and
- receive response messages through the CAN bus and elaborate a frame containing the requested variables by the PC software.

Los códigos de error programados son:

- P0217, exceso de temperatura en el motor; y
- P0143, bajo voltaje en el circuito de sensor de oxígeno (Banco 1 Sensor 3).

Borrar códigos de fallas (modo 04)

Borra los códigos de error y genera un mensaje de respuesta positivo para el modo de operación 04, si fue exitoso el proceso de borrado, al mismo tiempo apaga el indicador de fallos luminoso. Se le adicionó un shield para dotarlo de conectividad CAN y así posibilitar el diagnóstico OBD.

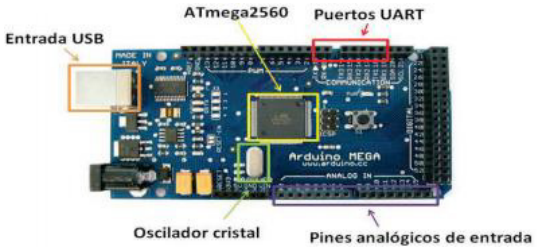

Figure 5. Arduino Mega 2560 R3 ("Arduino…", n.d)

B. Herramienta de diagnóstico

El sistema desarrollado consta de una placa Arduino Mega 2560 y la tarjeta de expansión que la dota de conectividad CAN, de ahí que sea posible la interconexión con una red de este tipo. Se ha programado de tal forma, para dar cumplimiento a los objetivos iniciales sobre el diagnóstico OBD-II, tales como lectura de parámetros en tiempo real y de códigos de error relacionados con fallas y borrar los registros de estos en la computadora una vez solucionados a nivel de mantenimiento mecánico del automóvil.

Software desarrollado para el scanner

El programa presente en esta placa se ha concebido con el objetivo de emular una interfaz de diagnóstico OBD-II sobre una placa Arduino. Este se software se encarga de realizar peticiones a través del bus CAN mediante la placa de expansión para Arduino que maneja este bus de datos.

El programa principal implementa estas funcionalidades:

- leer e interpretar los comandos recibidos desde la PC relacionados con las peticiones de diagnóstico;
- elaborar tramas CAN de acuerdo con los comandos anteriores y relacionados con los modos de operación 01, 03, 04 del estándar OBD-II; y
- recibir mensajes de respuesta a través del bus CAN y elaborar una trama que contenga las variables encuestadas por el software en la PC.

El bus de datos CAN dentro del vehículo contiene múltiples módulos interconectados, cada uno de ellos se representa por una dirección o ID. Inmediatamente después de conectar el scanner y empezar a recibir los datos, se percibe que llegan muchos mensajes con identificadores diferentes a los esperados. En el caso del diagnóstico, se deben enviar con el ID predefinido para la encuesta 0x7DF y se espera recibir respuestas identificadas por 0x7E8 o 0x7E9, que corresponden a los módulos más importantes dentro del sistema (ISO 15765-4). Por

Table, 4. Filtering and masking of CAN messages / Filtrado y enmascaramiento de mensajes CAN

Item	Buffer 0	Buffer 0
ID *Identificación*	000 1110 1000 (0x0E8)	111 1110 1000 (0x7E8)
Mask *Máscara*	111 1111 1111 (0x3FF)	111 1111 1111 (0x3FF)
AND *Y*	000 1110 1000	111 1110 1000
Filter *Filtro*	111 1110 1000 (0x7E8)	111 1110 1000 (0x7E8)
Do they match? *¿Coinciden?*	No	Yes

lo tanto, se decide aplicar, dentro de esta función, el filtrado y enmascaramiento de mensajes, para así asegurar la recepción de los mensajes de diagnóstico solamente. Las funciones que permiten implementar esto son init_Mask () y init_Filt ().

En la **Tabla 4** se presenta un ejemplo de cómo funciona el método de filtrado y enmascaramiento de mensajes por ID, dentro del protocolo de comunicaciones CAN.

El segmento de código ejecutado en la función loop permite recibir los comandos desde la interfaz gráfica en la PC y efectuar las peticiones de diagnóstico hacia la ECU del automóvil. Una vez recibidos los mensajes, los procesa y los devuelve a la PC para se visualice en la pantalla.

Las peticiones se hacen atendiendo a los datos recibidos. Por ejemplo, si el comando recibido contiene el ID de la ventana principal en la interfaz gráfica, entonces se construye un mensaje OBD-II para encuestar los parámetros relacionados con los indicadores que están en esa ventana, con excepción de las peticiones de borrado de los DTC, las cuales tienes un comando diferente, debido a que solo se realizan bajo voluntad del usuario.

Las peticiones se conforman de acuerdo con el formato de la trama OBD-II para peticiones de diagnóstico, rellenando el res-

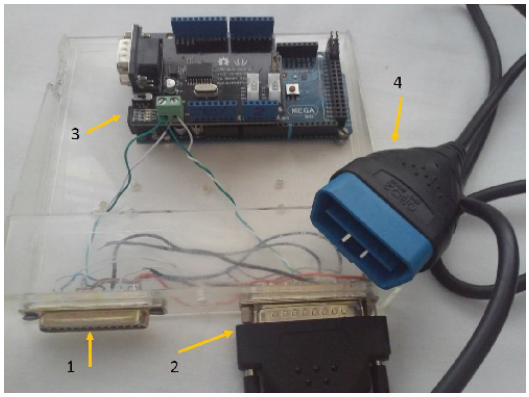

Figure 8. Implemented scanner: (1) CAN interface to diagnostics the ECU simulator; (2) CAN interface to connect with the vehicle; (3) Arduino Mega 2560 board and CAN bus shield as processing unit; (4) OBD-II (DLC) connector / Scanner implementado: (1) interfaz CAN para diagnosticar el mulador de ECU; (2) interfaz CAN para conectar al vehículo; (3) placa Arduino Mega 2560 y CAN bus shield como unidad de procesamiento; (4) conector OBD-II (DLC)1

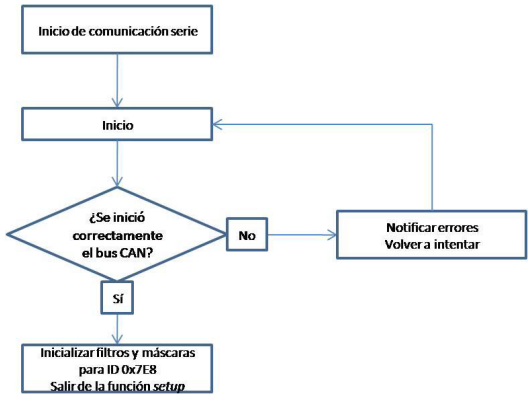

Figure 6. Setup function flow / Flujo de la función de configuración

The CAN data bus inside the vehicle has multiple interconnected modules, each one of them is represented by an address or ID. Immediately after the scanner is plugged in and it starts receiving data, we perceived that many messages with different IDs to the expected ones arrived. For the diagnostics case, the system must send the 0x7DF ID and the expected responses must arrive with 0x7E8 and 0x7E9 IDs, corresponding to the most important modules in the system (ISO 15765-4).

For that reason, we decided to apply the message filtering and masking in these functions to ensure the reception of diagnostics messages only. The functions that allow to implement this filter are init_Mask () and init_Filt ().

Table 4 presents an example of the message ID filtering and masking operation method inside the CAN communications protocol.

The executed code in the "loop" function allows to receive in the PC the commands from the graphical interface, it also provides diagnostics requests to the car ECU. Once the messages are received, it processes them and returns the information to the PC to get a broader visualization.

The requests are made considering the received data. For instance, if the received command has the ID of the main

Figure 9. Application main window / Ventana principal de la aplicación

window in the graphical interface, then an OBD-II message is built to survey the related parameters with the indicators in that window, except the DTC deleting requests. These latter have a different command, since they are only user-triggered. **Figure 6** shows the set up function flow.

The requests are formed as per the format in the OBD-II frame for diagnostics requests, i.e., by filling the rest of the "data" field with the value 0x55 (01010101 in binary) to maintain the bit synchrony (**Figure 7**).

C. Graphical User Interface

As mentioned, the graphical interface was developed using the LabVIEW software. The computer acts as a final stage in the processing of the gathered data, which are displayed in a comfortable and pleasant environment; this eases the interaction with the system. From here, it is possible to monitor several parameters in the vehicle distributed in several menu options. **Figure 8** presents details from the implemented scanner; **Figure 9** the application main window; and **Figure 10** the program flow chart**.**

The application is capable to read through the USB port the data from the Arduino board, framed with the information relative to the desired parameters to visualize. In the top section, it is possible to observe several tabs which ease the access to the menus enabled to execute the tests and actions in the automobile.

At boot time, the application executes –only once– each element in the block diagram as per the hierarchy

to de los octetos del campo Datos con el valor 0x55 (01010101 en binario) para no perder la sincronía de bits (**Figura 7**).

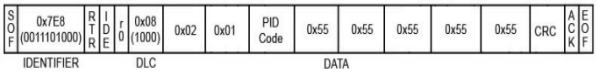

Figure 7. Structure of OBD-II request mode 01 / Estructura de petición OBD-II en modo 01

C. Interfaz gráfica de usuario

Como se mencionó, la interfaz gráfica se desarrolló empleando el software LabVIEW™. La PC actúa como etapa final en el procesamiento de los datos obtenidos, los cuales se muestran en un ambiente cómodo y agradable, lo que facilita la interacción con el sistema. Desde aquí es posible monitorear diferentes parámetros presentes en el automóvil distribuidos en varios menús. En la **Figura 8** se presentan detalles del *scanner* implementado; En la **Figura 9** la ventana principal de la aplicación; y en la **Figura 10** el diagrama de flujo del programa**.**

La aplicación es capaz de leer, a través de un puerto USB de la PC, los datos provenientes de la placa Arduino, entramados de tal forma y con la información relativa a los parámetros que se quieran visualizar. En la parte superior se observan varias pestañas que facilitan el acceso a los menús habilitados para ejecutar las pruebas y acciones sobre el automóvil.

Al iniciar la ejecución de la aplicación, LabVIEW™ se ejecuta cada elemento en el diagrama de bloques, según la jerarquía descrita en el esquema de flujo, una sola vez, posteriormente se detiene. Por lo tanto, se encapsula todo el diagrama dentro de un ciclo While, asegurando así su continuidad hasta tanto no ocurra un error o se pulse el botón "Desconectar" desde el panel frontal. Ambas condiciones determinan la ejecución del programa. Completar cada ciclo de ejecución demora alrededor de 100 ms, por lo que podemos afirmar

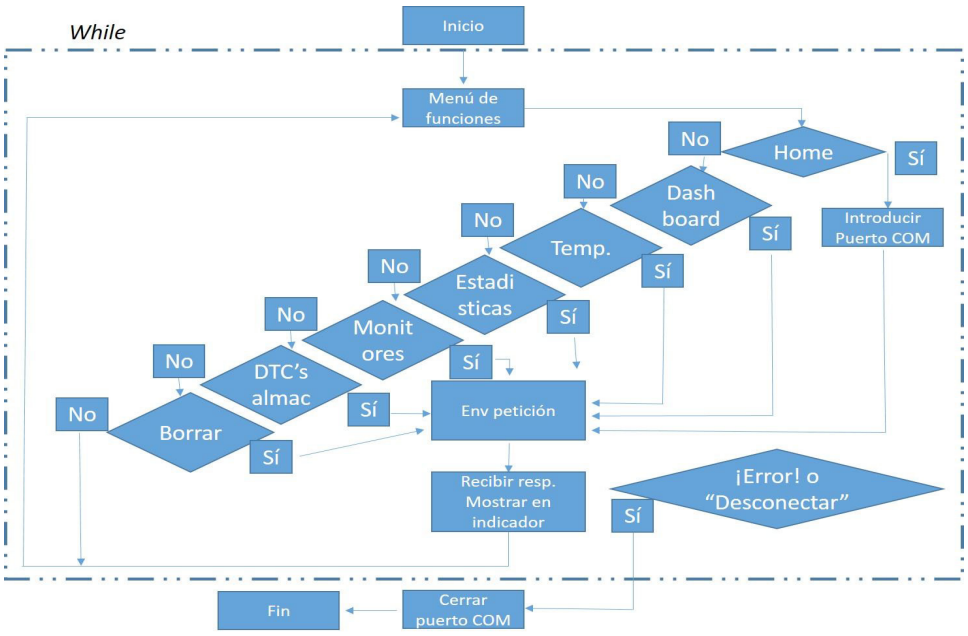

Figure 10. Program flow in LabVIEW / Flujo del programa en LabVIEW

una correcta lectura de los datos desde el automóvil y una aproximación cercana al tiempo real.

IV. Conclusiones

Con la realización de este trabajo se logró dar cumplimiento al objetivo principal: desarrollar un sistema de diagnóstico abordo basado en el protocolo de diagnosis OBD-II sobre un bus CAN. Se desarrolló una herramienta para el diagnóstico en automóviles, el monitoreo de parámetros técnicos y su visualización mediante una interfaz gráfica de usuario en tiempo real. Esta funcionalidad no está disponible en la mayoría de los sistemas profesionales que se distribuyen. Se implementó un simulador de la computadora de abordo de un vehículo y su diagnóstico sobre CAN bus.

Como trabajo futuro, el equipo del proyecto enfrentará las tareas siguientes:

- implementar el resto de los protocolos de comunicación contemplados dentro del estándar ODB-II, en aras de robustecer las prestaciones del dispositivo, sin comprometer la viabilidad del proyecto;
- implementar una interfaz inalámbrica de comunicación entre el scanner desarrollado y la PC, ya sea a través de Bluetooth, Wi-Fi o XBee; y
- continuar la profundización de las nuevas tecnologías para incorporar los requerimientos actuales del estándar OBD de tercera generación. S&T

described in the flow scheme and it stops. Hence, all the diagram is encapsulated in a "while" cycle, ensuring its continuity if errors do not arise or the "unplug" button is not pressed from the front panel. Both conditions determine the program execution and complete each execution cycle takes about 100 ms. For this reason, we can ensure a correct data reading from the automobile and a near approximation to real-time conditions.

IV. Conclusions

With this proposal, we were able to comply with the general objective: to develop an on-board diagnostics system based in the OBD-II protocol over a CAN bus. We developed a tool for vehicle diagnostics, monitoring of technical parameters, and its corresponding real-time visualization through a graphical user interface. This functionality is not available in most of the professional systems commercially available. We also implemented a simulator of the vehicle on-board computer and its diagnostics over CAN bus.

As future works, the project team will face the following tasks:

- Implement the remaining communication protocols in the OBD-II standard to enhance the robustness of the device without compromising the project viability;
- implement a wireless communication interface between the developed scanner and the PC, either via Bluetooth, Wi-Fi, or XBee; and
- continue the research in new technologies to incorporate the current requirements of the third generation OBD standard. S&T

References / Referencias

Arduino Mega 2560 R3. (n.d). Retrieved from: arduino.cl/arduino-mega-2560/

CanOBD2® Diagnostic Tool™Part#3100e: OBD2 Diagnostic Tool™. (2017). Retrieved from: https://www.innova.com/en-US/Product/Detail/3100e

Cervantes, A. & Espinosa, S. (2010). *Escáner automotriz de pantalla táctil* [tesis]. Instituto Politécnico Nacional: México.

García, A. (2015). *Diseño de una red CAN bus con Arduino* [tesis]. Universidad Politécnica de Navarra: Pamplona, España.

Getting Started with Arduino and Genuino UNO. (2016). Retrieved from: https://www.arduino.cc/en/Guide/ArduinoUno

ISO 9141-2:1994 - Road vehicles — Diagnostic systems — Part 2: CARB requirements for interchange of digital information. Geneva, Switzerland: ISO

ISO 14230-2:2016 - Road vehicles — Diagnostic communication over K-Line (DoK-Line) — Part 2: Data link layer. Geneva, Switzerland: ISO

ISO 15765-4:2016 - Road vehicles: Diagnostics on Controller Area Network (CAN). Part 4: Requirements for emission-related systems. Geneva, Switzerland: ISO

McCord, K. (2011). *Automotive diagnostic systems*. North Branch, MN: CarTech.

Meseguer, J. E. (2013). *Caracterización de los estilos de conducción mediante smartphones, dispositivos OBD-II y redes neuronales* [tesis de maestría]. Universidad Politécnica de Valencia: España.

Rayo M. O. (2009). *Diseño y realización de un sistema on board diagnostics (OBD-II)* [trabajo de final de curso]. Universidad Politécnica de Cataluña: Barcelona, España.

Shield introduction. (n.d)Retrieved from: http://wiki.seeed.cc/Shield

Simbaña, W., Caiza, J., Chávez, D., & López, G. (2016). Diseño e implementación de un sistema de monitoreo remoto del motor de un vehículo basado en OBD-II y la plataforma Arduino. *Revista Politécnica, 37*(1). Retrieved from: http://www.revistapolitecnica.epn.edu.ec/ojs2/index.php/revista_politecnica2/article/view/573/pdf

xTool [Web site]. Retrieved from: www.x-tool.org

Zabler, E. (2002). *Los sensores en el automóvil*. Stuttgart, Alemania: Robert Bosch GMB.

CURRICULUM VITAE

Armando Rodríguez Rodríguez ITelecommunications and Electronics Engineer from the Universidad de Pinar del Río [UPR] "Hermanos Saiz Montes de Oca" (Cuba, 2017). Since 2017, he is a professor at the Telecommunications and Electronics Department - Technical Sciences Faculty – UPR, imparting maters related with analogical electronics and power supplies. His research lines are: process automation, demotic control systems and optical fiber sensors / Graduado de Ingeniero en Telecomunicaciones y Electrónica en la Universidad de Pinar del Río [UPR] "Hermanos Saiz Montes de Oca" (Cuba, 2017). Es profesor del Departamento de Telecomunicaciones y Electrónica de la Facultad de Ciencias Técnicas de la Universidad de Pinar del Río desde 2017. Ha impartido asignaturas relacionadas con la electrónica analógica y fuentes de Alimentación. Investiga en automatización de procesos , sistemas de control demótico y sensores de fibra óptica.

José Raúl Vento Álvarez Telecommunications Engineer from the Instituto Superior Politécnico "José Antonio Echeverría" -CUJAE (Havana, Cuba - 1982); He has a Master's degree in Telecommunications Networks, and a and a Ph.D in Telecommunications from the Universidad Politécnica de Madrid (España, 1996 y 1998, respectively). Since 1990, he is professor at the Telecommunications and Electronics Department - Technical Sciences Faculty – Universidad de Pinar del Río (Cuba), imparting maters related with telecommunications' networks, telematics and optical communications, and he has been grade project tutor (Engineering, and Master and Ph.D in Telecommunications). His research interest are optical fiber sensors and Internet of Things [IoT] / Ingeniero en Telecomunicaciones del Instituto Superior Politécnico "José Antonio Echeverría" - CUJAE (La Habana, 1982); Máster en Redes de Telecomunicaciones (1996) y Doctor Ingeniero en Telecomunicación (1998) de la Universidad Politécnica de Madrid (España). Profesor del Departamento de Telecomunicaciones y Electrónica de la Facultad de Ciencias Técnicas de la Universidad de Pinar del Rio desde 1990. Ha impartido las asignaturas relacionadas con redes de telecomunicaciones, telemática y comunicaciones ópticas, y ha sido tutor de varios proyectos de pregrado en Ingeniería, y maestría y doctorado en Telecomunicaciones. Investiga en sensores de fibra óptica y sistemas de control en Internet de las Cosas [IoT].

Ricardo Inouye Rodríguez Informatics Engineer (2006) and Master in Forestry Sciences (2010). He is an assistant professor at the Informatics Department - Universidad de Pinar del Río [UPR] "Hermanos Saiz Montes de Oca". Currently he occupies the position as Main Teacher of Academical Year y presides the Provincial Council of the Unión de Informáticos de Cuba (Pinar del Rio). His areas of interest are: development of java applications, Web programming and programming for mobile dispositives / Ingeniero en Informática (2006) y Máster en Ciencias Forestales (2010). Es profesor asistente del Departamento de Informática de la Universidad de Pinar del Río [UPR] "Hermanos Saiz Montes de Oca". Actualmente ocupa el cargo de Profesor Principal de Año Académico y es Presidente del Consejo Provincial de la Unión de Informáticos de Cuba (Pinar del Río). Sus áreas de interés profesional son: el desarrollo de aplicaciones en JAVA, la programación web y la programación para dispositivos móviles.