UCD Urban Institute Ireland

Working Paper Series

2007

Description of the Development of the LabVIEW Instantaneous Emissions Estimation Software for use in the Urban Environment Project Air Quality Work Package

Edward Casey, Donal Lennon, William Smith, David Timoney

UCD UII 07/01

UCD Urban Institute Ireland

University College Dublin

www.ucd.ie/uii

UCD Urban Institute Ireland

WORKING PAPER SERIES

2007

Description of the Development of the LabVIEW Instantaneous Emissions Estimation Software for use in the Urban Environment Project Air Quality Work Package

Edward Casey, Donal Lennon, William Smith, David Timoney

University College Dublin

Working Paper Series: UCD Urban Institute Ireland

ISSN: 1649-3613

UCD Urban Institute Ireland
University College Dublin,
Clonskeagh Drive,
Clonskeagh,
Dublin 14
Ireland

This working paper can be downloaded as a PDF at
www.ucd.ie/uii/

Description of the Development of the LabVIEW Instantaneous Emissions Estimation Software for use in the Urban Environment Project Air Quality Work Package

Edward Casey[a,b,c,*], Donal Lennon[a,b] William Smith[a,c], David Timoney [a,c]
[a]Urban Environment Project Air Quality Research Group
[b]UCD Urban Institute Ireland
[c]UCD School of Electrical Electronic & Mechanical Engineering

1 Introduction

This Working Paper describes the development of the current LabVIEW software programme used to extract *On-Board Diagnostic* (OBD) signals from vehicles. LabVIEW is a graphical programming language based on G-coding [3] and provides a simple environment for data collection and display. The current software version (Version 2.1) receives inputs from the vehicle via the OBD connector and displays the parsed and manipulated values of engine load, engine speed, coolant temperature and vehicle speed on a purpose built graphical user interface. Table 1 describes the various versions of the software package and explains which parameters were used in each one, such that the current working version was achieved.

The purpose of this document is to outline the processes involved in achieving this system and to describe its use in service.

2 The structure of On-Board Diagnostic Signals

On-board Diagnostics is a system whereby certain pieces of information about the operation of a vehicle, either past or current, can be extracted from the vehicle's *Engine Control Unit* (ECU). For the system employed in this Paper, the physical connection between the vehicle's control unit and the user interface, stored on a laptop, is a serial cable connected to an *ElmScan Scantool*. The extracted data is then processed using a specially designed piece of software.

All On-Board Diagnostic signals, either requests or responses, are coded in hexadecimal format. This is a compressed numerical system where decimal values are represented by a combination of figures and characters. The minimum value in hexadecimal is `00` (decimal value of 0) and the maximum is `FF` (decimal value of 255), resulting in 256 (2^8) possible numeric decimal values. These hexadecimal values are then converted to decimal *bytes* for further processing.

All request codes are described in the SAE J1979 Standard [7]. The service `$01` is the most commonly used, since this returns the current powertrain diagnostic data which relates to emissions outputs. The general structure of a request is to use Service `$01` followed by the requested *Parameter Identifier* (PID). For example, requesting information about engine speed takes the form `$01 0C`. The response to such a request is preceded by the code `$41` such that a full response concerning engine speed is `$41 0C` xx_1 xx_2, where xx_1 is the hexadecimal value corresponding to response byte A and xx_2 is the hexadecimal value corresponding to byte B. Although certain parameters, such as PID `00` and `01` can contain more than two bytes, all the parameters processed in this programme contain just one or two bytes, namely A and B.

For complete processing of the data, the hexadecimal byte values must be converted to decimal values, since this is the data type most easily manipulated in LabVIEW. While a large number of version schemes are published, the LabVIEW environment

*Corresponding Author: Edward Casey.
E: edward.casey@ucd.ie. T: 00 353 1 7162699.
UCD Urban Institute Ireland Working Paper Series (07/01), ISSN 1649-3613. 21112007

Table 1: Development of the LabVIEW software.

Ver.	Parameters	Notes
1.1	`0C, 11, 05, 0D`	This version was never tested in a vehicle and was used only as a means to test the parsing functions. A save data function was also tested. The bilinear interpolation VI was also fully implemented from this version, following earlier sub-tests and review of previous versions. Note that PID `11` was mistakenly used instead of PID `04`.
1.2	`0C, 11, 05, 0D`	This version saw initial development of the integration and and differentiation of the vehicle speed parameter for determining distance covered and acceleration respectively.
1.3	`0C, 11, 05, 0D, 5E`	This version involved an initial parsing test for the fuel usage parameter, PID `5E`. Initial tests were successful in terms of data parsing.
1.4	`0C, 11, 05 , 0D, 5E`	Minor developments were made in the graphical user interface layout to incorporate the display of the new parameter, PID `5E`.
1.5	`0C, 11, 05 , 0D, 5E`	Data flow was modified to allow the user to specify the fuel type and the save to file location prior to testing. This system never worked particularly successfully.
1.6	`0C, 11, 05 , 0D, 5E`	A minor addition to the case structure controlling which emissions maps are being used, such that the user can see which ones are in use.
1.7	`0C, 11, 05 , 0D, 5E 5C`	The engine oil temperature parameter, PID `5C`, was included in this version. Parsing was successful.
1.8	`0C, 11, 05 , 0D, 5E 2F, 05, 5C`	This was the first major vehicle test and highlighted the problems with a number of parameters, PIDs `2F, 05, 5C`, which displayed the `No data` warning as a response to each request.
1.9	`0C, 04, 05, 0D`	Following on from the previous test, a feedback loop and shift register were included to ensure a continuous signal. This was included on account of the fact that no data was displayed when the parameter was not being parsed.
1.10	`0C, 04, 05, 0D`	A save to file implementation for all parameters was included in this version. The graphical user interface was also modified to take the reduction of parameters into account. These became the standard parameters for the 'complete' version.
1.11	`0C, 04, 05, 0D`	An integration scheme was included to allow for the display of instantaneous and aggregated emissions data. The graphical user interface was updated accordingly.
1.11.2	`0C, 04, 05, 0D`	This version used the Maximum Torque Curve implementation to determine load. The MathScript node was incorporated in a Sub-VI.
2.0.1	`0C, 04, 05, 0D`	A generic implementation of the MathScript Sub-VI is used in this version
2.1	`0C, 04, 05, 0D`	The MathScript implementation for load calculation was removed and the simplified Maximum Torque Value was used. The graphical user interface was modified accordingly.

Figure 1: The engine control unit is connected through a serial cable to the scantool and a USB cable takes data to the laptop.

includes a conversion function, making it particularly suited to OBD based programming.

3 Parameter Selection

Despite the fact that a huge variety of possible parameters exists within the SAE J1979 standard [7], not all of these parameters are actually implemented in vehicles. This is particularly true of European vehicles, where the implementation dates for OBD compatibility are considerably later than for those in the United States.

The current version of the programme has tried to take advantage of newly introduced or implemented parameters while still building on previous incarnations of the system. Obviously, certain parameters were retained due to their importance. The selected and tested parameters are discussed in more detail below.

One of the first decisions made was that, in the absence of modern, type specific emissions maps, a set of previously obtained Volvo maps would be used [4, 6]. These maps are two dimensional arrays which describe the quantity of a pollutant species which is evolved in units of grams per second (g s^{-1}). The columns of each array describe the engine speed (rpm) while the rows describe the engine load (Nm). Clearly, to make use of these tables, both engine speed and engine load must be included in the programme. The Engine Speed parameter (PID `0C`) is easily extractable and parsing is relatively straightforward. This process is described in more detail in Section 4.1 on page 4.

Vehicle load, however, proved to be significantly more difficult:

Extraction of load type data can be achieved using a variety of PIDs within the OBD environment. The simplest form would be to extract the Engine Reference Torque (PID `63`) at the beginning of the experimentation. This value describes the one-hundred percent reference value for all indicated engine torque parameters and is defined only once [7].

Using this value for engine reference torque, it would have been possible to use parameter `62`, Actual engine Percentage Torque, to scale the Engine Reference Torque appropriately. The output from PID `62` is a single byte and parsing results in a percentage value. Simple multiplication would result in a value for the current instantaneous torque value.

Unfortunately, implementation of this torque determination protocol did not yield results in the test vehicle. Although the test vehicle is a modern vehicle (2006, VW, Polo, 1.2 petrol), the required PIDs are not implemented on the OBD ECU.

Two alternative parameters exist for load calculation but both return just percentage values of the calculated (PID `04`) or absolute (PID `43`) load.

The Calculated Load Value (PID `04`) is the parameter of choice, since the calculation scheme defined is valid for both positive ignitions (petrol) and compression ignition (diesel) engines. Two percentage load calculation schemes are presented. A possibly useful characteristic of this parameter is that a value of 100% load is achieved at WOT conditions for any engine type, at any altitude, for any given temperature and at any value of engine speed [7].

The calculation schemes are described by the following equations from the SAE 1979 standard [7]:

$$\%_{\texttt{load}} = \frac{A_c}{A_p} \times \frac{P_{baro}}{29.92} \times \sqrt{\frac{298}{T_{amb} + 273}} \qquad (1)$$

where A_c is the current airflow and A_p is the peak airflow measured at Wide Open Throttle at Standard Temperature and Pressure as a function of engine speed; or

$$\%_{\texttt{load}} = \frac{T_c}{T_p} \times \frac{P_{baro}}{29.92} \times \sqrt{\frac{298}{T_{amb} + 273}} \qquad (2)$$

where T_c is the current engine torque and T_p is the peak engine torque at Standard Temperature and Pressure as a function of engine speed. For both calculation schemes, P_{baro} is the barometric pressure and T_{amb} is the ambient temperature in °C.

With respect to the Absolute Load Parameter (PID 43), only spark ignition engines are required to support this PID. This means that PID 04 or an equivalent would have to be implemented for diesel vehicles anyway. To this end, it was considered that the best approach would be to use PID 04 and come up with a method of determining the absolute load value, in newton metres (Nm), and using the percentage load as a scaling factor. The methods examined to allow for this are discussed in more detail in Section 4.2.

Traditional methods of developing driving cycles have relied almost exclusively on determining the drive pattern from vehicle speed-time traces. The benefit of this particular piece of software is that both the speed-time trace and the traces of engine load and engine speed can be determined. Thus, it is possible to relate vehicle speed back to engine load and engine speed without the need for cumbersome gearbox algorithms.

Vehicle speed can be determined from OBD PID 0D. Inclusion of this parameter is not essential, since the use of a global positioning system during experimentation will also show vehicle speed. However, the processing of vehicle speed, as described in Section 4.3, allows for determination of the total distance travelled and the acceleration and deceleration of the vehicle. The acceleration data is useful in terms of determining the aggressiveness of the driving while the total distance travelled could be used as an input to the cold start module. Full details of the parsing and processing of the vehicle speed data is described later.

A number of temperature measurements can be obtained from OBD PIDs. The most useful of these is probably the engine coolant temperature (PID 05). The use of this parameter allows for the inclusion of the cold start excess emissions module. Although this module is not fully implemented in the current version of the software, it will be included in future versions. The parsing of the temperature parameter is included in the software and is described in Section 4.4.

4 Data Parsing, Scaling & Manipulation

Despite the vast array of possible parameters that can be extracted from the OBD system there are only a handful of parsing and scaling functions. This greatly reduces the amount of processing required and the quantity of code needed to achieve a result.

As described earlier, all requests for real-time data are preceded by the code $01 and the parameter identifier. A relatively small number of identifiers are used in this programme.

The specific parsing, scaling and manipulation of each of the sought parameters is dealt with below.

4.1 Engine Speed (PID 0C)

As described earlier, engine speed is one of the two fundamental parameters required for the determination of real-time pollutant emissions estimates. In order to describe the parsing and scaling function employed in the LabVIEW programme, consider the following.

Using the request $01 0C, a reply of $41 0C 1F 39 is returned. The first element of the data parsing is to examine the response for a common string token. For each reply about the engine speed, this common string token will be $41 0C. Using the *Match Pattern* function within LabVIEW, it is possible to strip off the common string token and retain just the data bytes 1F and 39, denoted xx_1 and xx_2 respectively. In order to logically separate the data bytes, it is necessary to examine the reduced or stripped string, namely 1F 39 and split it into separate tokens. By using two *Scan String for Tokens* functions in parallel, one with a token offset of three characters, the two bytes are successfully separated.

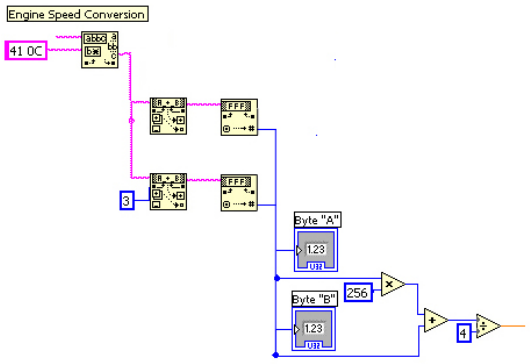

Figure 2: Data parsing and scaling function for engine speed.

The next step is to convert the hexadecimal values, 1F and 39, to decimal value using the *Hexadecimal String to Number* function. This results in two double-type values which can be manipulated using standard mathematical structures. The scaling function used to convert the two decimal values to a 'real' revolutions per minute value is given in the

Table 2: Description of the final parameters used. Taken from SAE JA1979 [7].

Parameter Name	PID	Data Bytes	Maximum	Minimum	Units
Engine Speed	0C	A, B	0	65,536	rpm
Vehicle Speed	0D	A	0	255	km hr^{-1}
Engine Load	04	A	0	100	%
Coolant Temperature	05	A	-40	+215	°C

SAE J1979 Standard [7]. Mathematically, it is given by

$$RPM = \frac{(\mathtt{A} \times 256) + \mathtt{B}}{4} \tag{3}$$

where A and B are the decimal values of the two bytes. Thus for hexadecimal values of 1F and 39, the corresponding decimal values are 31 and 57. Applying Equation 3 yields an engine speed value of 1998 rpm.

A further manipulative step, with regard to engine speed and pollutant emissions calculations, will be discussed later in this document.

4.2 Engine Load (PID 04)

Although multiple engine load determination schemes exist within the scope of OBD-II parameters, PID 04 was selected as the load determinant since it functions for both petrol and diesel engines and works for both naturally aspirated and boosted engines.

The principal difficulty with this particular calculation for load is the fact that the result is a percentage of the peak torque available during normal, fault-free driving [7] as opposed to an absolute value for load in newton metres (Nm).

It is best to consider a further example to clarify the parsing scheme. In response to the request $01 04, a reply with a single byte is returned, such as $41 04 58. As with the engine speed, the hexadecimal string is stripped to just the single byte. In this particular case, the byte need not be further parsed but simply converted to decimal, with hexadecimal value 58 corresponding to a decimal value of 88.

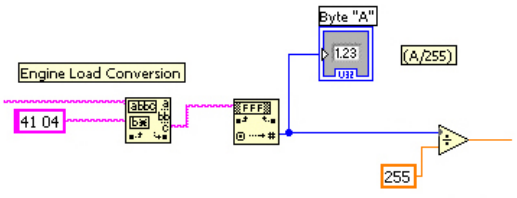

Figure 3: Data parsing an scaling of engine load.

This decimal value is then scaled according to

$$\%_{load} = \frac{\mathtt{A} \times 100}{255} \tag{4}$$

Continuing the example, this results in a percentage load value of 34.5%. This however, is not particularly useful on its own. A method for determining the absolute load value, in Newton metres is required. A number of possiblilities were examined during the development and validation stages.

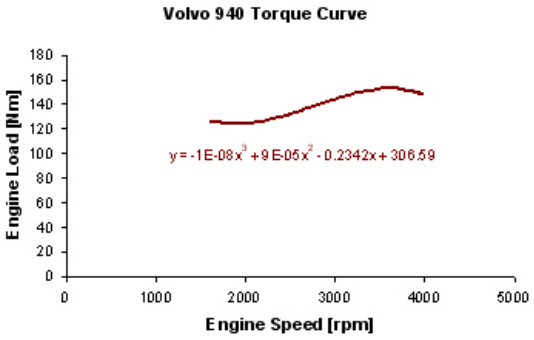

Figure 4: Torque curve for the Volvo 940. This digitised plot was used to determine the maximum load value for parsing.

Consider the graph shown in Figure 4. This is the torque curve for the Volvo 940. Maximum torque is defined as a function of engine speed. The equation of the line was entered into a LabVIEW MathScript Node. Using the value for engine speed, determined according to Equation 4 above, the maximum available torque for any given engine speed can be determined. It was, however, found that the speed of calculation was quite long, of the order of tenths of seconds to seconds. Also, such a scheme requires the user to digitise a graph, determine the equation of the spline and define the coefficients to be used by the programme. This implementation (Version 2.0.1) describes an equation of the forth order, given by

$$T_{max} = ax^4 + bx^3 + cx^2 + dx + e \tag{5}$$

It is not expected that any equation would be of greater order. Thus, only the required co-efficient would be entered in the frontend, as illustrated in Figure 5.

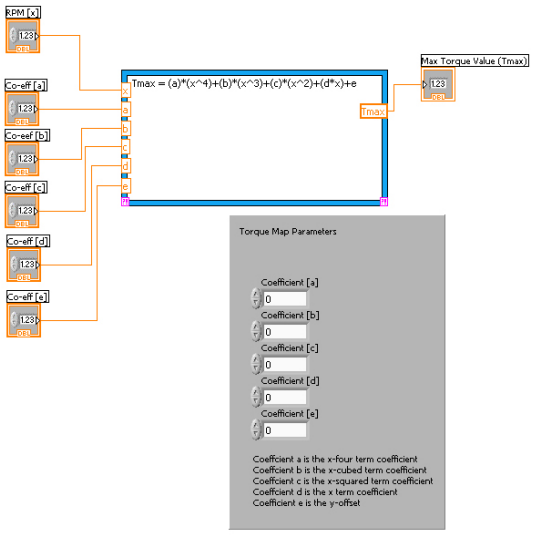

Figure 5: An implementation of the maximum torque curve method as employed in Version 2.0.1. The cumbersome nature makes it unwieldy as does the slow calculation using the MathScript Node.

This is a cumbersome process and is open to significant levels of human error. Also, it does require access to the maximum torque curves of each of the vehicles under test.

On closer inspection, the difference between the maximum and minimum peak torque is quite small and it should be possible to use the peak torque at rated rpm for a given engine as an approximation of the torque value. This negates the need for complex manipulation of the peak torque curve. The user simply enters a single value of peak torque and this value is multiplied by the percentage of peak torque extracted form the OBD-II system. Further details are given in Section 9.1

The resulting value of torque will, therefore, lie between 0 and the maximum rated torque for the engine under test.

As with engine speed, a further manipulation of the engine load value is required and will be discussed later.

4.3 Vehicle Speed (PID `0D`)

Although vehicle speed will be determined independently by the global positioning system available through the Built Environment Laboratory of the Urban Institute, the vehicle speed parameter is also extracted by the LabVIEW programme. The same parsing scheme is employed in this case as for engine load, since a request, of the form `$01 0D`, returns a response containing just one byte, such as `$41 0D 3E`.

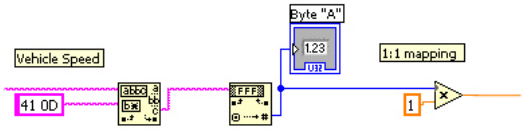

Figure 6: Data parsing and scaling function for vehicle speed. Notice the direct one-to-one mapping of the extracted byte value and the vehicle speed.

Thus, it is not necessary to scan the string for a token. This particular parameter also uses a direct relationship between the converted hexadecimal value and the actual speed of the vehicle. Thus, a hexadecimal value of `3E` corresponds to a decimal value of 62 which represents 62 kilometres per hour (km hr^{-1}). Mathematically, this may be interpreted as

$$v = \texttt{A} \tag{6}$$

From an engineering point of view, the standard unit of metres per second (m s^{-1}) is more useful. To this end, a conversion scheme is introduced. Mathematically, this may be described as:

$$v\ [\text{m s}^{-1}] = \frac{v\ [\text{km hr}^{-1}]}{3.6} \tag{7}$$

Having converted the value to m s^{-1}, the *Time Domain Maths* function was employed twice to carry out two different mathematical functions. The integration of the speed signal results in a figure for total distance travelled in metres. Similarly, differentiation of the signal results in a value for the instantaneous acceleration of the vehicle. Both parameters are useful in terms of determining the aggressiveness of the driving (acceleration) and in implementing the cold start factor calculation (distance). A full implementation of cold start has not been completed but will be based on both distance travelled and engine coolant temperature.

4.4 Engine Coolant Temperature (PID `05`)

Although the cold start emissions module has not been fully implemented in this version of the programme, the coolant temperature will be used to determine the cut-off point for cold start excess emissions. Further to a review of work carried out in other institutions [2, 5, 8], it was decided that the combination of coolant temperature and total distance travelled would be deciding factors in the cut-off point. Further investigation of these parameters

and the implementation of the cold start excess factor calculations will be examined in the near future.

Work in previous versions of the software currently under discussion [6] did use the engine coolant temperature as the determinant for cold start emissions. The engine coolant temperature determination scheme is described below and is included in this version of the programme.

The request for snap-shot data is given by `$01 05` and a reply takes the the form `$41 05 68`, for example. Since only one byte of data is returned, the parsing scheme is the same as for vehicle speed or percentage engine load.

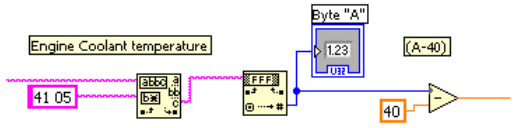

Figure 7: Data parsing and scaling function for engine coolant temperature.

Each byte corresponds to a 1°C change in temperature with a -40°C offset. Mathematically, this is described by:

$$T_{coolant} = \mathtt{A} - 40 \tag{8}$$

such that for the example value mentioned above, the hexadecimal value of `68` corresponds to a decimal value of 104 and, with the offset, results in a coolant temperature of 64°C.

5 Instantaneous Emissions Estimation Scheme

In order to determine an instantaneous emissions quantity, described in grams per second (g s^{-1}), a relationship between pollutant output and engine load and speed has been devised. Previous versions of the software have employed the used of emissions maps based on the Volvo 940. These emissions maps are two dimensional arrays describing the pollutant emissions as a function of both engine load and engine speed. The engine speed is described by the columns of the table and is graduated in revolutions per minute (rpm); the engine load is described by the rows and is graduated in newton metres (Nm).

The first processing task was to ensure the compatibility of the emissions maps and the LabVIEW package:

Each map was read into software using the *Read From File* express virtual instrument (Express VI). Using this VI, it is possible to define the table boundaries, such that header rows and descriptor columns are removed and just the data remains. This in turn led to a new problem, that of defining which column and row represented which engine operating condition.

In all, eight emissions maps exist. Each of carbon monoxide (CO), hydrocarbons (HC), oxides of nitrogen (NO_x), particulate matter (PM) is defined for both petrol and diesel. For any given experiment, either petrol or diesel maps are used and this is defined by the user. Discrimination is achieved using a *Case Structure* where there are eight inputs, four each from petrol and diesel, but only four outputs. The cases are either diesel (true) or petrol (false). A switch is used to ensure the correct connection of the terminals, as shown in Figure 8.

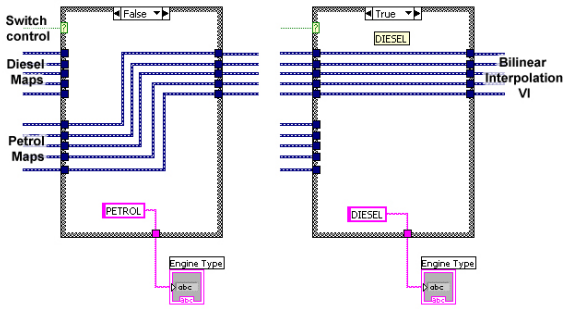

Figure 8: Discrimination between petrol and diesel emissions maps is achieved using a simple switch. In the 'false' position, only the petrol maps are read in, while in the 'true' position, the diesel maps are passed to the Bilinear Interpolation VI.

The two dimensional nature of the emissions maps necessitated the use of a *Bilinear Interpolation* VI. This function takes three arguments, namely a column identifier, a row identifier and the data to be interpolated. Calculation of these identifiers is described below. Data is passed from the emissions maps as dynamic data and is converted to double integer type data.

The engine speed values in the emissions maps lie between 1000 rpm and 5500 rpm with increments of 250 rpm, giving a total of 19 columns. In the case of the engine load, the minimum value was 0 Nm and the maximum was 185 Nm, rising in increments of 5 Nm, resulting in 38 rows. Since the bilinear interpolation VI does not recognise absolute values, such as an engine speed of 2000 rpm and 20 Nm load, but rather absolute cell co-ordinates in terms

of column and row number, a new scheme had to be devised for calculating the emissions output. The system proposed by Deery was deemed to be the best [4].

In this scheme, each value of engine speed is allocated a column number, such that 1000 rpm corresponds to column 0 (the first index) and 5500 rpm corresponds to column 18. This leads to a direct linear relationship between engine load and column index. The resulting equation of the line could them be used to convert the engine speed to a column index for use by the bilinear interpolation VI. The last problem is the fact that a zero engine speed value does not exist. To this end, a new column was included and all the values were assumed to be zero emissions at 0 load. This meant that the indices had to be shifted by 1 and also that a new equation for the 0 rpm to 1000 rpm cell had to be devised.

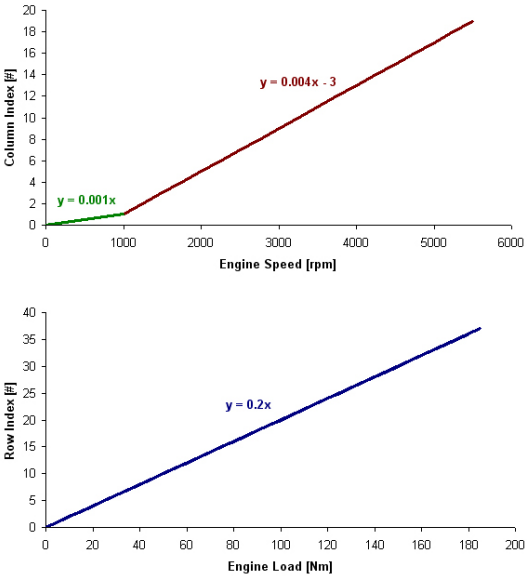

Figure 9: Graphical representation of the conversion equations used for (top) engine speed and (bottom) engine load for the bilinear interpolation VI.

An identical system is used for the load rows, where the zero load corresponds to row 0 and 185 Nm corresponds to index 37. No further manipulation was required.

The LabVIEW implementation for both the conversion of rpm and Nm values to cell co-ordinates are shown in Figure 9. In the case of the engine speed implementation, it was necessary to include the conditional structure so that the correct interpolation formula could be used for the given engine speed ranges. This ensures that for engine speed values between 0 rpm and 999 rpm, the equation:

$$y = 0.001x \tag{9}$$

is used while for values n the range 1000 rpm to 5500 rpm,

$$y = 0.004x - 3 \tag{10}$$

is used. This is shown diagrammatically in Figure 10.

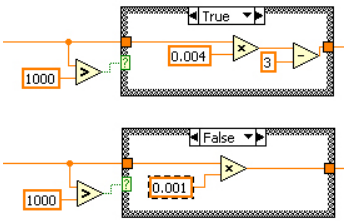

Figure 10: Implementation of the calculation scheme to determine the column identifier for both the range 0-999 rpm (above) and 1000-5500 rpm (below).

In all, five interpolations are implemented to allow for calculations of each of the four pollutant species and the fuel consumption (FC). The resulting output values are displayed on-screen both in graphical form and as a number. the values of instantaneous emissions are them passed to a Time Domain Maths VI and integrated, such that a total emissions value can e determined for the trip so far.

6 Communicating With On-Board Diagnostic Devices Using LabVIEW

Communication with the OBD ECUs is achieved using the *Serial VISA Connection* in LabVIEW. This system allows the user to specify which *VISA Resource* they wish to use. Such resources normally take the form of a COM port, such as `COM1` or an LPT port, such as `LPT1`. For the purposes of this programme, a COM connection is used. The VISA establishes a connection with the OBD device and then outputs the data extracted from the OBD. The VISA connection works in two directions. Firstly, a request is sent to the OBD ECU for a parameter, such as engine speed, then the request is processed and a reply is sent back. This reply is then printed to screen and passed to the main programme for parsing, scaling and manipulation as described in Section 4. Once the programme is terminated, the VISA connection is closed.

The VISA requests are constructed automatically via the *Queue Sequencer*. This creates a queue of hexadecimal requests which are then passed to the VISA resource, processed and returned. The requests can be timed manually, although times below 250 ms have resulted in incomplete response data.

The VISA and Queue Sequencer act in the same way as the HyperTerminal described in Section 8, only in an automated manner.

Although the previous versions of this programme used the *Stacked Sequence Structure*, this version uses the *Framed Sequence*. This ensures that execution of the programme takes place in a predefined sequence. The user can therefore select whether the vehicle under test is a petrol or diesel vehicle and provisions can be made to ensure that the data is saved to the correct location in memory before the engine is started. Full details of these features are discussed in more detail in Section 7. It also ensures that the VISA connection is opened before data parsing takes place, ensuring that invalid data is not saved.

7 Saving and Storing of Data

All experimental data is saved to an `.lvm` or `.tdm` file format and is readable by Microsoft Excel for post-processing. All gathered parameters and their associated manipulations are stored. In order to save time, the saved data is sent to a pre-defined data file. The name and location of the data file are displayed on the GUI. The *Strip Path* VI is used to split the file pathname into the name and the path for ease of viewing on the control panel

The *Write to Measurement File* VI can be configured before the test by accessing the block diagram. The dialogue box allows the user to define various parameters, such as the location data is to be saved to, the type of data stored, what time stamps are to be used and whether data should overwrite previous data files. In this version of the software, each new test is saved to new data file, numbered sequentially.

8 Establishing a Communications Link

Although earlier versions of the ElmScan used a direct serial connection, the ElmScan 5-USB uses a serial emulation via a USB port. To establish a link with the vehicle, a number of steps must be carried out.

The physical equipment consists of the ElmScan scan tool, an OBD-II to serial connector and a USB-A to USB-B connector. The OBD-II plug contains slots for up to 16 pins, although only 9 pins are used for this system. These 9 pins still allow communication with five common OBD protocols.

The first task was to locate the *Diagnostic Link Connector* (DLC) within the car. This is a 16 pin D-shaped socket located within the passenger compartment of the vehicle. The OBD-II socket is inserted into the connector and the serial plug is connected to the scan tool. At this point, the power LED lights up and system cycles through the red and yellow Tx and Rx LEDs. Power is supplied to the scan tool from the vehicle's battery via pin 16 of the OBD connection. The scan tool was then connected to the laptop using the USB cable. In order for the USB hub to recognise the serial connection, a set of drivers must be installed to establish a *Virtual COM Port* (VCP). These drivers are supplied with the scan tool.

Having established the physical link with the car, the next step was to ensure that the vehicle and the laptop were actually communicating in a coherent manner. Installation of drivers was verified by opening the System panel within Control Panel in Windows. The Device Manager within the Hardware tab was used to search for the new device and the COM port number corresponding to the USB Serial Device was noted.

The Windows HyperTerminal was used to establish the communications link between the vehicles OBD system and the laptop. This system allows the user to specify the COM port and its properties. The COM port specific to the USB Serial Device was specified and a baud rate (the number of bits transferred per second) of 9600, a data bit value (the maximum number of data bits in each message) of 8, a parity value of N and a stop bit of 1 were also specified, in accordance with the ElmSacn manufacturer's specification [1]. Flow control was governed by the hardware. Having carried out these tasks, the HyperTerminal displayed the ElmScan prompt line. At this point, the user entered the request `$0100` and a reply of four bits is returned. The specific values are not necessarily of any significant importance but the fact that a reply, in the format `$41 00 AA BB CC DD` is returned means that a successful communication link has been opened with the vehicle. At this point, the HyperTerminal was closed and operational control was passed to the LabVIEW software package.

9 Graphical User Interface Development

One of the attractive features of the LabVIEW environment is the fact that a user friendly front end can easily be developed using the standard palettes. It means that all acquired and calculated parameters can be shown to the user in an easy to understand format.

The frontend is divided using a tab structure, not unlike a standard Windows interface. Most users should be familiar with such a system. The first element of the tabs structure gives the user the ability to have a clear overview of the current operating conditions.
All elements of the front end are retrieved from various palettes and are dropped and manipulated on screen. Thus, sophisticated front ends can be developed quickly. Such frontends also afford the user the ability to both control the data flow and view retrieved data.

9.1 Initialisation

Before running the system, it is possible to select the fuel type used, either diesel or petrol. The switch activates a case structure terminal within the programme and allows either the petrol emissions maps or the diesel emissions maps to be read by the bilinear interpolation VIs. The COM port in use can also be selected such that communication can start immediately. The currently sought parameter is also displayed and the request and responses are displayed in the *Reply Received* text box. It is also possible for the user to specify the speed at which the parameters are sought and refreshed, either by using the scroll bars or by typing the value into the box provided. A standard error handled is used in this programme and the associated indicators are shown in this tab. The current file name for the saved data is also displayed, along with an LED which indicates that saving is occurring. A single value for torque is also entered by the user. This value should be the maximum rated torque for the engine and is scaled during the data extraction process by the Calculated Load Value (PID 04).

9.2 Times

The Times tab displays the current date and time and also shows the current duration of the test. This value is important for the cold start function which is to be implemented at a later date. It is also used by the Time Domain functions for integration and differentiation of the vehicle speed signals.

9.3 Virtual Dashboard

The Virtual Dashboard is used to display the current values of the various engine based parameters extracted from the OBD system. This allows the user and the driver to make visual inspections of the data and ensure that no major discrepancies exist between the 'real' display and the 'virtual' display. The parameters shown are coolant temperature, in thermometer style, engine speed and vehicle speed, all shown on dials. The vehicle speed is also described in both kilometres per hour (km hr^{-1}) and metres per second (m s^{-1}). The load value is initially expressed as percentage and then scaled against the maximum rated torque to yield a load value in newton metres (Nm). Values for total distance covered (km and m) and acceleration (m s^{-2}) are also shown and help to describe the type of driving currently under consideration.

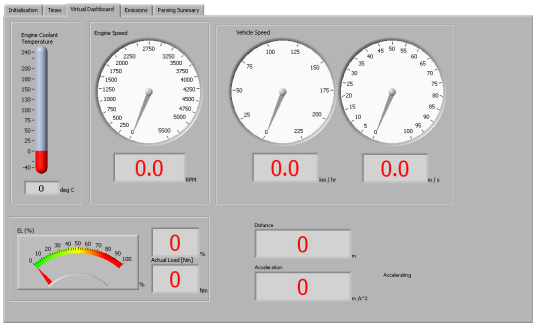

Figure 11: The final version of the graphical user interface (GUI) for the Virtual Dashboard.

9.4 Emissions

This tab displays a graph of the current and past emissions for the duration of the journey. A cumulative value and an instantaneous value for each pollutant species, namely carbon monoxide (CO), hydrocarbons (HC), oxides of nitrogen (NO_x), particulate matter (PM), as well as the current fuel consumption (FC) are all displayed. A parsing check display is also shown. This describes the current fuel type and the row and column index being used for the calculations.

9.5 Parsing Summary

In an attempt to quantify errors or logical mistakes in the programming, a parsing summary tab exists in the current version. This allows the user to quickly view all the hexadecimal, decimal and scaled values on one screen.

10 Further Work

Although the current version of the software does function well, there are a number of refinements that could be made.

A full implementation of the cold start emissions scaling factors is needed. This should be based on more than just the coolant temperature, as was case in earlier versions of the software. Examinations are ongoing on how best to implement this. Other institutions, such as INRETS in France, have been working on similar proposals.

It would be useful to have a higher refresh rate than the current system. It is hoped that this can be resolved in the near future. Discussions are ongoing with National Instruments about this. If the refresh rate can be increased, a clearer picture of the drive cycle can be described, without the need for complex post processing using filters and so on.

Although the system is capable of determining the speed of the vehicle, the inclusion of GPS type data will be useful. It would also be possible to include a visualisation module, using such data as might be available from Google Maps, to allow the user to see where the vehicle is. This may also be useful for post processing and correlation work at a later date. Work in this field is ongoing.

11 Acknowledgments

The authors would like to acknowledge the support of the whole Urban Environment Project Group and the financial support offered by the Environmental Protection Agency (EPA) and the National Development Plan (NDP).

Edward Casey would also like to thank Darragh Clabby (UCD) and Declan Deery (UCD) for their help with the LabVIEW programming. Thanks also to Sue Murphy (UII) and Mike Brennan (UEP) for their help with the OBD software in-vehicle tests during development.

References

[1] ElmScan 5 USB Quick Start Guide. PDF available from www.scantool.net.

[2] André, M. The ARTEMIS European driving cycles for measuring car pollutantt emissions. *Science of the Total Environment 334 – 335* (2004), 73 – 84.

[3] Bishop, R. H. *Learning With Labview 8.* Pearson Prentice Hall, 2007.

[4] Deery, D. Estimation of On-Road Vehicle Fuel Consumption and Emissions Using Analysis of On-Board Diagnostic Signals. Bachelors of Engineering Degree Dissertation, UCD School of Electrical, Electronic & Mechanical Engineering, 2007.

[5] Liu, Z., Li, L., and Deng, B. Cold start characteristics at low temperatures based on the first firing cycle in and LPG engine. *Energy Conservation and Management 48* (2007), 395 – 404.

[6] McGahan, P. Estimation of On-Road Vehicle Fuel Consumption and Emissions Using Analysis of On-Board Diagnostic Signals. Bachelors of Engineering Degree Dissertation, UCD School of Electrical, Electronic & Mechanical Engineering, 2006.

[7] SAE International. Surface Vehicle Standard J1979: E/E Diagnostic Test Modes. Standards document, Society of Automotive Engineers, 2007.

[8] Weilenmann, M., Soltic, P., saxer, C., Forss, A.-M., and Heeb, N. Regulated and nonregulated diesel and gasoline cold start emissions at different temperatures. *Atmospheric environment 39* (2005), 2433 – 2441.