UNIVERSITY OF ALASKA ANCHORAGE

CSCE A470

CAPSTONE PROJECT

Author:
Kristian Smith

Supervisor:

Prof. Randy Moulic, PhD

Anchorage AK, May 2015

Computer Science & Engineering Department
UNIVERSITY of ALASKA ANCHORAGE

Kristian Smith

kasmith11@alaska.edu

Version 5.3

***Abstract*— The basis for this research involves logging various sensor data from a large sample population of automobiles. The device necessary for such a task has to be capable of streaming On-Board Diagnostics and real-time GPS coordinates to a remote server for data logging. After discovering a multitude of web-based companies currently designing OBD tools, it became evident that the market was flooded with devices focused primarily on localized data logging versus remote logging for a large scale deployment. This led to a decision of compiling a device using the Arduino framework due to its affordability, support network, and overall adaptability. Using an Arduino Uno microprocessor, we were able to integrate GPS and GSM cellular shields for location and data streaming, respectively. An OBDII board was also obtained to access vehicle sensor data and other diagnostic information. The primary benefits of the proposed device include full access to any data available through OBDII/CAN protocols, and adaptability for on-device data processing and/or cellular transmission. With this tool, many future research opportunities arise including those in the fields of regional emissions analysis (our current focus), automated road condition notification, and adaptive maintenance strategies.**

Introduction

The ability to apply locality and time to vehicle data proves itself to be beneficial for numerous research applications. The current focus of our research is assessing the viability of standard platinum/ceramic oxygen sensors in cold-weather climates. Since, in newer vehicles, oxygen sensors determine the engine's air-fuel ratio, their proper operation is vital. Vehicles spend a period of time in an "Open Loop" air-fuel ratio programming until the oxygen sensors heat up to proper operating temperature of approximately 316 degrees Celsius. During this time, the Engine Control Unit uses a static ratio in which emissions of the vehicle are much greater. Newer oxygen sensor designs have built-in heating elements to bring them up to temperature quicker, but in a climate where temperatures frequently reach below 0 degrees Celsius, the heat-up time is still a relevant problem. There have been a number of IEEE papers regarding alternative oxygen sensor materials that are capable of sensing at extremely low temperatures [16]. By logging oxygen sensor data from a large sample of vehicles over the winter in Anchorage, AK, we will be able to calculate a predictable average for oxygen sensor heating times. Being able to pinpoint a location where certain vehicle conditions occur is the first step to adaptive maintenance strategies. Current automated maintenance/repair techniques are limited to a specific vehicle's internal network. Frequently, vehicles with mechanical or electrical failures only display the symptoms under certain conditions, therefore the vehicle will, in general, show these same symptoms provided the same environmental conditions are met. These variables could include road condition, ambient air temperature, vehicle speed, cornering angle, etc. Due to these factors being foremost locality based, the symptoms of such failures tend to exhibit themselves repeatedly in the same physical location when driving. Much of this environmental data can be collected from the vehicle at the moment of the incident. With this information, it becomes simpler to analyze the specific failure and its cause. The ability to access data from a sample population of vehicles could prove itself useful to a vehicle manufacturer as well. By logging data from new vehicles, the manufacturer has the capability of recognizing potential recall needs long before an incident occurs. Aside from reducing liability, having this kind of information could greatly reduce necessary research and development for future model years. Another possible use for location specific vehicle data monitoring would be in regard to road conditions. By using real-time data gathered from the vehicle paired with GPS coordinates, areas of adverse road conditions could be determined automatically. For instance, a vehicle enters a corner at a specific steering angle and speed, but the GPS coordinates along the actual path of travel show that the desired cornering angle was not achieved, it could be concluded that the vehicle's front wheels were sliding (under-steering). Depending on the speed and attempted angle of the corner, this may mean that

the roadway is abnormally slick. This condition can be more easily determined if the vehicle is travelling on a straight roadway as it only requires taking the difference in the vehicle speed parameter and GPS calculated speed.

In order to obtain and process the required data for these research topics, a device has been assembled containing the following elements:

- Universal microprocessor
- GPS transceiver able to transfer coordinates to a microprocessor
- GSM cellular device able to communicate with a microprocessor and a remote server
- OBD II device able to communicate with the microprocessor and the vehicle Engine Control Unit (ECU)

(*Actual devices chosen are specified later in the document.*)

Since the 1996 model year, automobiles in the United States have been required to have installed an "On-Board Diagnostics" port (OBD II). Many other countries have implemented similar requirements such as EOBD for European countries, JOBD for vehicles sold in Japan and ADR (Australian Design Rule) for those vehicles sold in Australia. This requires vehicle manufacturers to provide on-demand access to diagnostic information such as Diagnostic Trouble Codes (DTC) and various sensor readings. Due to this standardization, not only dealership mechanics have the ability to access this information, but anyone with an OBDII scan tool. Many forms of these devices are available in the aftermarket: DTC scanners, PC scan tools, and printed circuit boards (PCB) containing necessary OBDII transmission protocols. While DTC scanners can be useful tools when diagnosing powertrain malfunctions and PC scan tools provide an added graphical representation of sensor data, the versatility of a PCB with pinned serial inputs and outputs allows control by a microcontroller versus direct user input in order to gather vehicle data autonomously.

Proposed Device

For the purpose of simplicity and adaptability, Arduino was chosen as the main platform for the research. Arduino is an open-source microprocessor prototyping platform with plug-and-play hardware options. This provides a proven programming structure (derivate of C programming language) with plenty of support available, not to mention a uniform form factor for creating a reasonably compact device for testing. Below is an outline of the various boards used.

{Arduino Uno SMD Microprocessor}

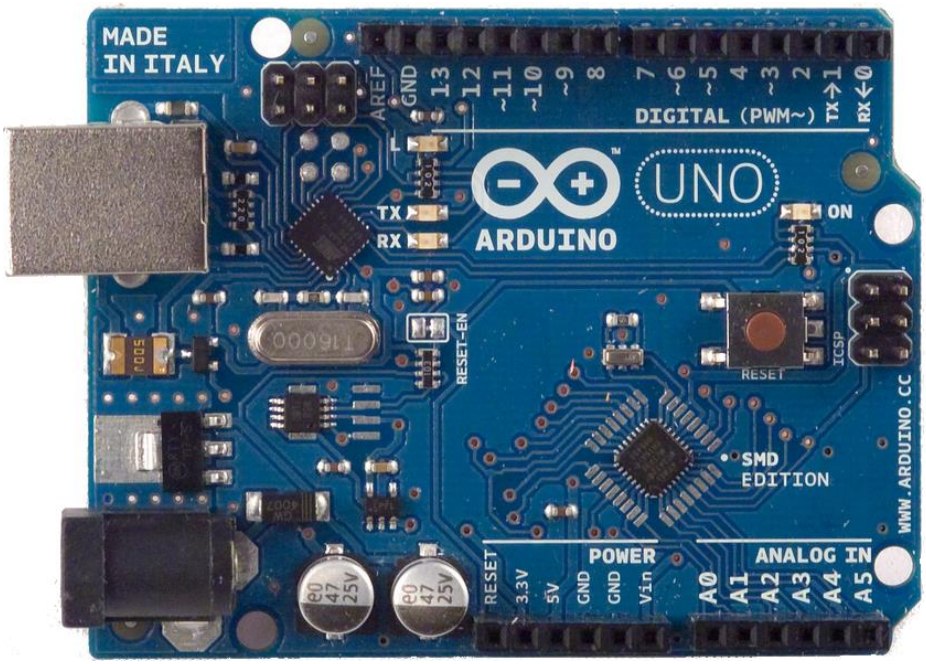


Figure 1: Arduino Uno Microprocessor Board

Based on the ATmega328 microprocessor, the Arduino Uno board is one of the most popular Arduino boards currently available. The Uno contains 6 analog inputs, 14 digital input/output terminals, a flash memory of 32KB, and a clock speed of 16MHz. It has a USB Type B communication port which can be used to program the device from a PC using the Arduino Integrated Development Environment (IDE), as well as transmitting runtime-acquired data back to the PC for viewing through the "Serial Monitor" also found in in the Arduino IDE. This device will maintain the setup and required programming loop for all three boards to which it will be connected.

The reason for choosing the Arduino Uno was its pure simplicity. The programming structure is a subset of "C/C++," of which there is much aftermarket equipment available. Due to their longevity and these devices are relatively slow compared to some of the ARM architecture devices on the market, but in exchange they take up very little energy. This can be quite beneficial if one decides to power it off the vehicle's battery while the car isn't running in order to use for a passive anti-theft device.

{SIM900 GSM/GPRS Shield IComsat from ITEAD Studio}

Figure 2: ITEAD GSM Shield [13]

The SIM900 GSM/GPRS Shield is a prebuilt cellular shield based on the SIM900 Quad-band GSM/GPRS module designed specifically for the Arduino platform. This board stacks directly onto the Arduino Uno board. Many features the SIM900 is capable of are unnecessary for scope of the proposed device requirements including 3.5mm headphone and microphone ports, General Purpose Input/Output (GPIO) pin-outs, and extra Transmit/Receive pins (Tx/Rx). This board will be tasked with transmitting vehicle and GPS data to a server for storage or computation. Due to this fact, it requires a data activated GSM-based carrier's SIM card which will be placed in the SIM card latch underneath the shield.

{GPS Shield from ITEAD Studio}

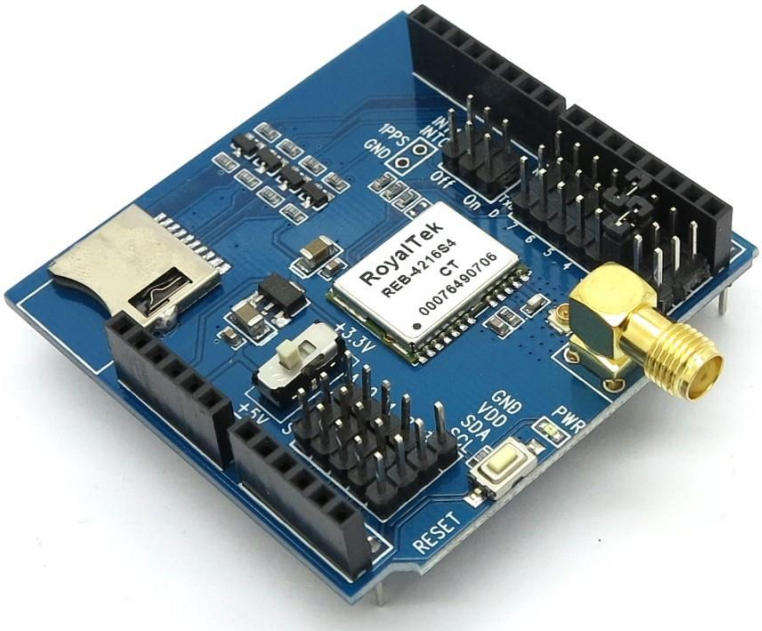


Figure 3: ITEAD GPS Shield [14]

This GPS shield from ITEAD Studio is based on the GlobalSat EB-365 GPS Module. The unit is stackable to the Arduino Uno as well as the GSM/GPRS Shield. This board has the ability to continuously poll GPS coordinates and store them as strings to a MicroSD card or send them directly back to a microcontroller by way of available Tx pin. Forms of data accessible from the GPS Shield are shown below.

- GPGGA-Global Positioning System Fix Data
- GPGLL- Geographic Position, Latitude/Longitude and Time
- GPGSA- GPS Dilution of Precision and Active Satellites
- GPRMC- Recommended Minimum Specific GPS Transit Data
- GPVTG- Track Made Good and Ground Speed

{SparkFun OBD-II-UART Board}

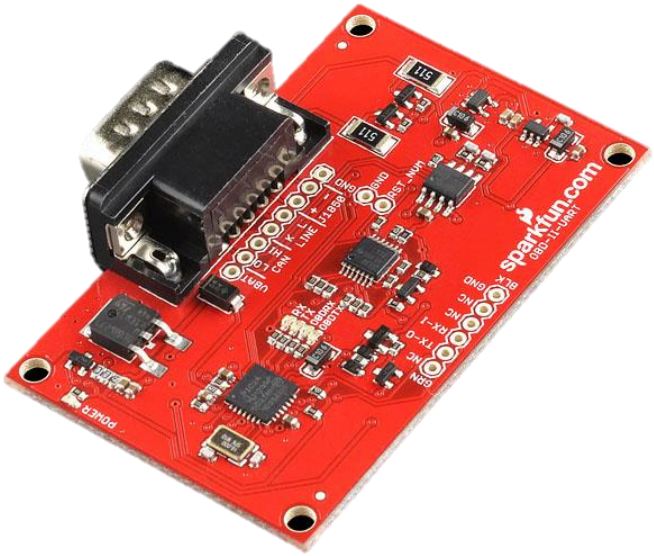


Figure 4: SparkFun OBDII UART Board

This board, designed by SparkFun Electronics, is based off of the STN1110 OBDII to UART (Universal Asynchronous Receiver/Transmitter) interpreter chip. Constructed with a 16-bit processor core, the STN1110 offers more features and better performance than any other ELM327 compatible integrated circuit (most OBDII scanning devices utilize the ELM327 chip). This board is capable of interpreting eight different ISO and SAE standard OBD protocols as well as Controller Area Network (CAN) protocols. The CAN Bus is generally connected to many more peripheral devices such as in-dash navigation, door locks, stereo, etc. Unfortunately the CAN-Bus was standardized on recent vehicle models, so that functionality will become more useful as the years progress. The OBD-II-UART also comes with a secure boot-loader for easy upgrading of firmware. The 9-pin serial connection point attaches to a DLC (Diagnostic Link Connector) conversion cable which plugs into the vehicle's DLC port. The board is powered by the vehicle when connected and the only other pins required for basic operation are the transmit/receive (TX-0/RX-1) and Ground (GND) pins. Once connected, a hexadecimal Parameter ID (PID) request can be sent to the RX-1 pin and the board will echo back the requested PID along with the corresponding data via the TX-0 pin.

{Device Setup}

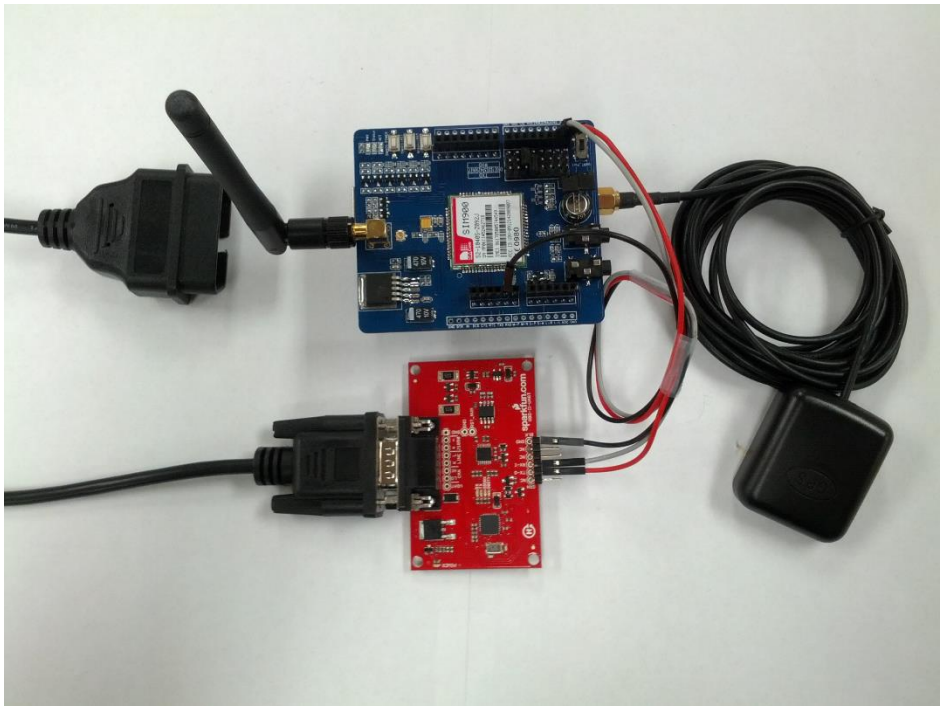


Figure 5: OBD Datalogger Prototype (Above View)

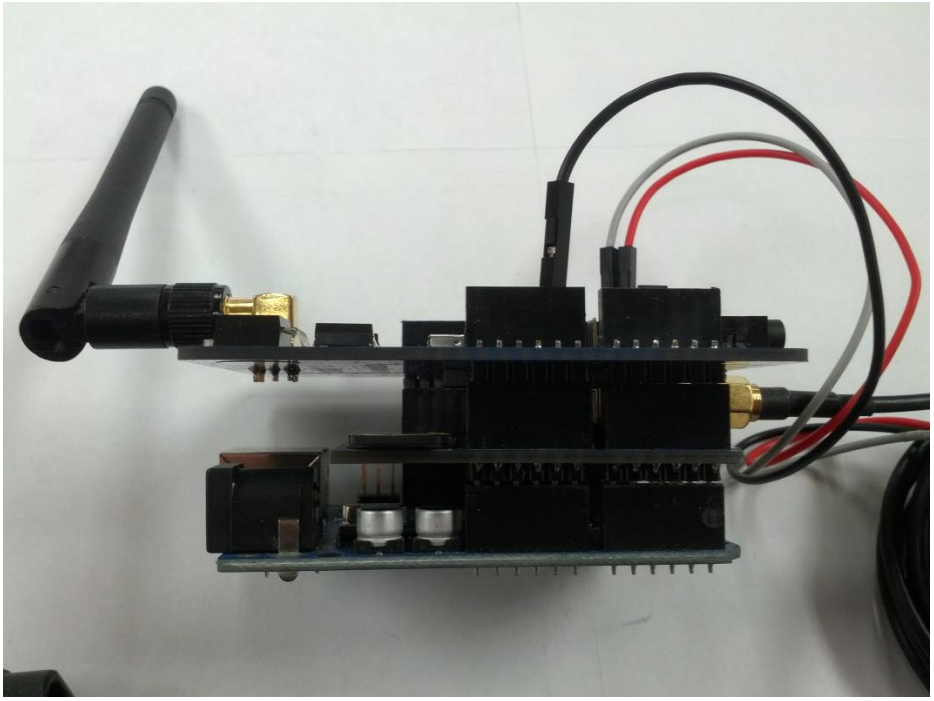


Figure 6: OBD Datalogger Prototype (Side View)

Minimal setup is required using these components. The cellular and GPS shields must be stacked vertically via the header pins to the Arduino Uno. The OBDII-UART board will need to be connected to the common ground as well as to one set of transmit and receive header pins. Finally, the GPS and cellular antennas should then be affixed to the device. An example of the assembled device can be seen in Fig. 1. Once these physical connections have been made, the device is ready to be programmed using the Arduino IDE. A setup function establishes transmission protocols and baud rates for all connections between boards as well as external communications. Additionally, the main loop running on the Arduino is the location from which all processes are directed.

Programming

The simplicity of the proposed device is that it is completely operated from the programming of the Arduino Uno. The Arduino is coded from a derivative of C/C++. Since the basic code for each board is licensed in **GPL 2.0**, it makes sense to continue utilizing this licensing structure throughout the project and just add on to what others have designed. There is also a free downloadable IDE for Arduino that allows for simple compiling, debugging, and code uploading. In general, there will need to be one header file for each communication board (GPS, OBD, and GSM) for best organization. Each board needs to initialize a baud rate before any communication with the Arduino occurs. The industry standard is 9600 bits/sec so that is what will be utilized throughout the project. Once the communication has been initialized between the supplementary board and the microprocessor, there are just simple serial commands being sent to provide the desired information.

{GPS}

As far as the GPS board is concerned, it automatically requests different data of the GPS Shield including coordinates, relative speed, and time at a consistent rate of 2 Hz. All that needs to be done is to parse through the data that the board is constantly streaming to receive the specific data needed for the particular application. Below is a sample serial output of the data that arrives from the GPS Shield (only post-processing being carriage returns).

$GPGGA,184354.000,6111.4591,N,14949.4067,W,1,07,1.2,59.0,M,9.6,M,,0000*4C

$GPGLL,6111.4591,N,14949.4067,W,184354.000,A*23

$GPGSA,A,3,29,23,03,09,16,10,26,,,,,,1.9,1.2,1.5*3F

$GPRMC,184354.000,A,6111.4591,N,14949.4067,W,0.00,,220415,,*0A

$GPVTG,,T,,M,0.00,N,0.0,K*7E

As you can see, both the GPGGA and the GPGLL provide the latitude and longitude desirable for the research. When the device hasn't yet acquired satellite activity, the coordinates and other fields show up blank and only the comma separators appear. The easiest way to deal with a lack of service is to just use the last known coordinates seeing as, when the vehicle is moving, it gathers signal rather quickly.

{OBDII}

The SparkFun OBDII Board also receives commands from the Arduino Uno, but in this case, the commands are based on Parameter Identifiers (PID's) that the vehicle's computer is able to decode and generate a response from. The most beneficial code structure is having the Arduino loop a request for data from the OBDII board and when it hits a condition, proceed with operations utilizing the other componentry. The complication arises with the fact that most of the parameters require some sort of post-processing technique. An example of this is RPM which arrives from the vehicle ECU in a division of 4. This is one of the more simple examples, whereas others may arrive in hexadecimal or other encoding. Luckily, there was already an implementation of the Arduino programming to utilize on this board. It contains an 'OBD' object that gets instantiated within the Arduino main program and contains higher level functions, so that there is little post-processing to be done for about 20 built-in parameter ID's.

{SIM900 Cellular}

The programming for the cellular shield relies particularly on 'AT' commands and delays. The delays are required to ensure proper network communication before continuing on to another process. The following snippet of code is what is being used in the program to send an SMS with the GPS coordinates and OBDII data to a number specified.

```
SIM900.print("AT+CMGF=1\r");                                  // AT command to send SMS message
delay(100);
SIM900.println("AT + CMGS = \"+18005555555\"");               // recipient's mobile number, in international format
delay(100);
SIM900.println("RPM = " + String(value) + " \nGPS Coordinates = " + dataString);      // message to send
delay(100);
SIM900.println((char)26);                                    // End AT command with a ^Z, ASCII code 26
delay(100);
SIM900.println();
delay(5000);                                                 // give module time to send SMS
SIM900power();                                               // turn off module
```

The cellular shield will hold up the functionality of the device as a whole if it cannot connect to the cell network. This means that no data acquisition can begin until the SIM900 is able to communicate with the GSM network associated to the SIM card.

In figure 9 on the following page, you can see the program flow of the code designed for the project. This is an implementation for text message alerts. First is the setup function which initiates the communication standards for each board. After that, the main program loop begins. When the device sees a specific condition based on OBDII data, it will save that parameter information and move on to gathering GPS coordinates. As long as GPS coordinates are available, the GPS shield sends one character to be saved at a time and concatenates it to a string. If the device is unable to gather GPS signal, that function will be bypassed. The next step is to set up the SMS message. How this function works can be seen above. This process repeats indefinitely as per the design of the Arduino programming structure.

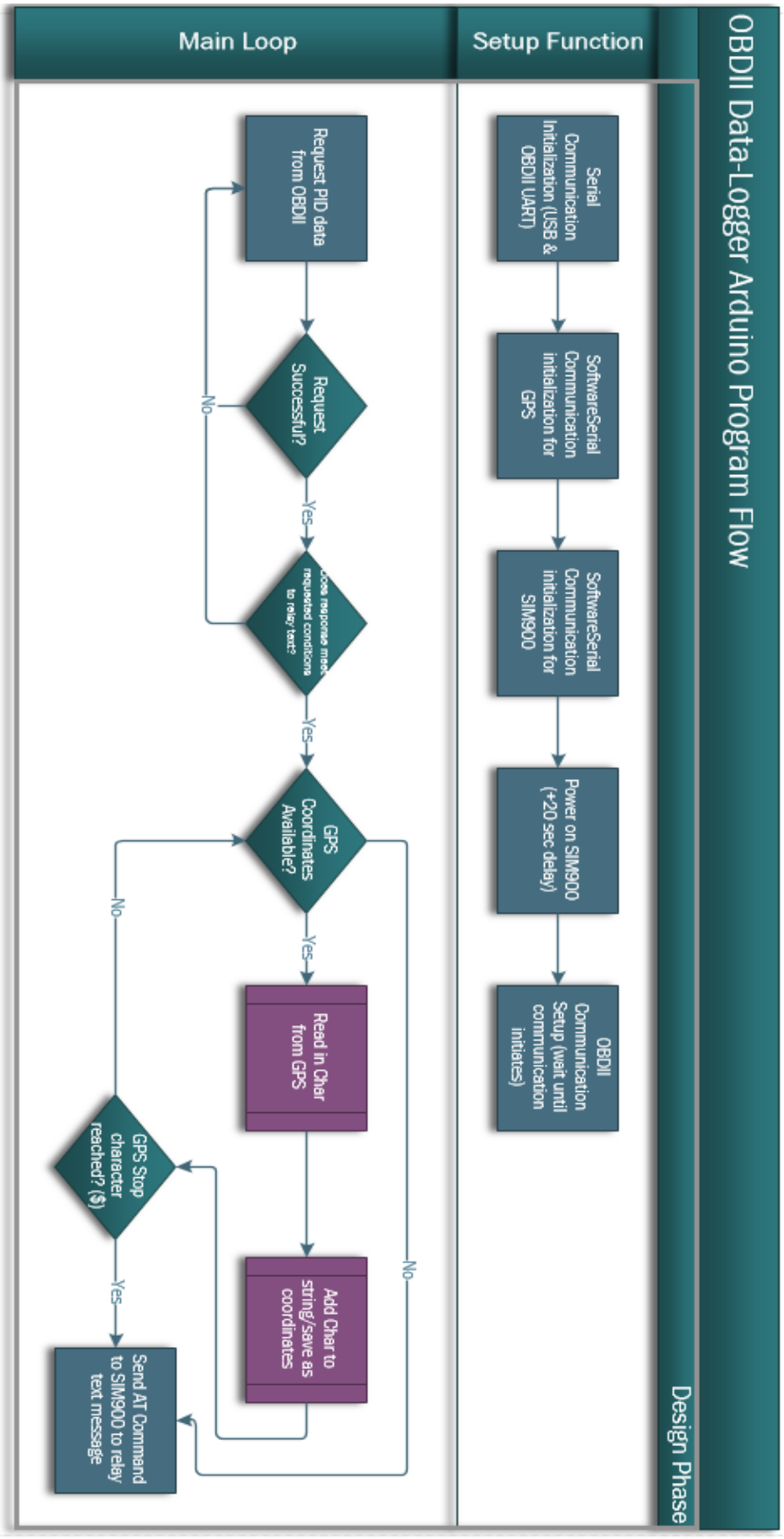


Figure 9: Program Flow

Related Work

For our research, we required an inexpensive device with all the requirements shown on this table. It is evident that the only realistically affordable options were the Bluetooth scanners and the Arduino platform device. The downfall of the Bluetooth device being that it requires a smartphone for each vehicle, as well as the fact that it relies on a Bluetooth signal versus a wired connection to the processor. A number of other projects have arisen in the last few years in the field of OBDII loggers. Most of these have been Arduino microcontroller-based, which allows for programmable automation. One specific article of interest titled "The ultimate GPS and OBD-II data logger based on Arduino MEGA," [11] highlighted a method for building an OBD II scanning device that also logged data to Google Earth for data referencing based on location. Unfortunately, the design was geared toward monitoring one vehicle and rather than sending data live through a cellular network, as the device is only set up to save the acquired data to a memory card for later access.

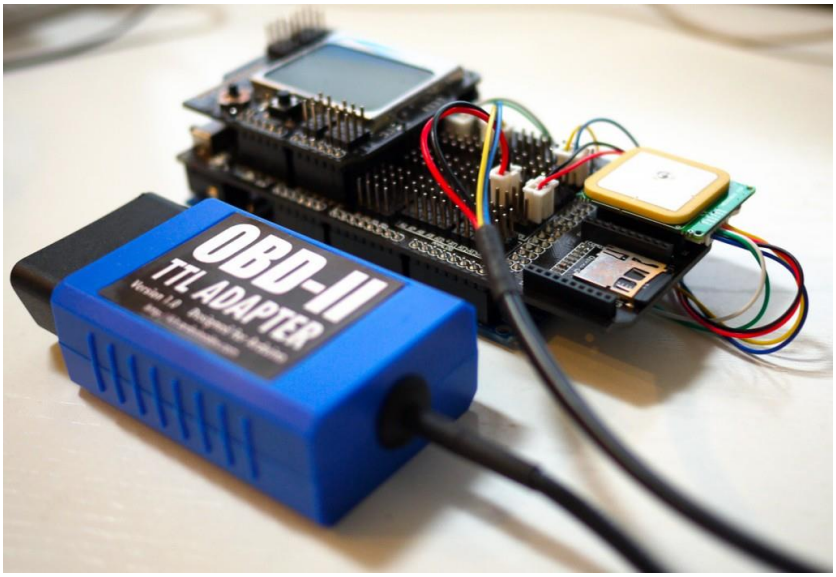


Figure 8: *"The ultimate GPS and OBD-II data logger based on Arduino MEGA"*

A group of students from our University (University of Alaska Anchorage) worked on a comparable project until the end of spring 2012. Their research project, entitled "CANOPNR," [1] similarly dealt with logging vehicle/GPS data to a remote server. Their team had deployed logging devices into multiple vehicles and had begun tabulating data using a web-based application. Their device utilized the Arduino framework, but it focused specifically on CAN-Bus (Controller Area Network Bus) protocol for data

transmission with the Engine Control Unit. Since the CAN-Bus protocol wasn't required in US vehicles until 2008 and OBDII was required from 1996 onward, it seemed necessary to design a device that could also communicate with OBDII vehicles.

Research was carried out in attempt to discover similarly designed devices currently on the market. There are a number of devices on the market, but as is made apparent from the graph, it is difficult to find a device that is capable of OBDII communication, cellular communication, GPS all while being a standalone device. The ones that did fit this build where just too expensive for the everyday user, and likely too expensive for most businesses to see it as a viable use of funds. The devices that came closest were the Bluetooth OBD scanning devices. In theory, one could design Android or iOS applications in conjunction with the onboard GPS and cellular capabilities of the phone to mimic the functionality of the Arduino device designed for this research. The only problem is that you are then relying on yet another wireless signal, Bluetooth. For reliability and simplicity, it made more sense to use a microprocessor.
There are a number of OBDII logging devices on the market, a few of which can be seen in Table 1 below.

Table 1: Market Cost Comparison Chart

Device	Stand-Alone	GSM-Enabled	Programmable	GPS	Data Logger	CAN	Approx. Cost (USD)
Bluetooth OBD Scanners		√	√	√	√		$150-$500
PC Scan Tools	√					√	$30-$400
Hem Data Mini Logger	√			√	√		$800-$1500
Car Chip Pro	√			√	√		$305
Data Scout					√	√	$500
DashDyno SPD ProPack	√				√	√	$320
IOSiX OBD-II Datalogger	√	Opt.		Opt.	√		$700
Arduino/GSM/GPS/OBDII Device	√	√	√	√	√	√	$160

Applications

The next step for the project is to deploy these devices on several test vehicles and initialize data acquisition. Beginning with basic OBD parameters, a database schema can be designed which is capable of organizing data by Vehicle Identification Number (an accessible OBD parameter), time, and location. Each device will be able to send constant or time-interval data using the cell network to automatically populate the database on the remote server. Once reliable data acquisition is achieved from the deployed devices, other forms of research can begin.

{Low Temperature Oxygen Sensor Analysis}

Being able to monitor emissions components on a greater scale than just one vehicle at a time allows us to better understand different concerns pertaining to a variety of environmental variables, especially climate. In an area such as Anchorage, AK, the temperatures are below freezing point (0°C) for 30-40 percent of the year. At these temperatures, emissions components take much longer to heat up to their proper operating temperature, which leads to increased engine emissions. A vehicle's oxygen sensors and catalytic converter(s) generally will not function properly until the vehicle reaches a standard operating temperature (70-100°C) [17]. Until this occurs, the vehicle operates on an "Open Loop." This means that the air-fuel ratio is set by the Engine Control Unit (ECU) and with no assistance from the oxygen sensors. In order to ensure safe engine operation, the ECU defaults to "rich," meaning the engine runs more fuel than likely necessary, which leads to increased emissions. Once the oxygen sensors(s) heat up, the vehicle will begin to operate in a "Closed Loop" where it utilizes the pre-catalyst oxygen sensor(s) to inform the ECU of any unspent fuel in the exhaust for adjustment of the air-fuel ratio. The "Closed Loop" functionality reduces emissions, increases fuel efficiency, and increases performance.

There is related IEEE paper regarding monitoring the degradation of oxygen sensors over time by use of On-Board Diagnostics [4]. It discusses an analysis technique for analyzing the operation of an oxygen sensor throughout its lifespan. Adding to this research the effects of climate on oxygen sensor performance may shed light upon how environmental conditions may also adversely affect these sensors. Another IEEE paper was located that focuses on the design of an oxygen sensor which operates at extremely low temperatures [16].

The purpose of researching this functionality is to discover the time differential between "Open Loop" operations of vehicles in cold climates versus mild/warmer climates. If the differential proves itself to be substantial, it may pave the way for more advanced emissions components in such cold climates. Also, it would be very important to know if there is an external temperature limit at which a vehicle will never reach "Closed Loop" operations.

We plan to deploy devices in a number of vehicles to begin logging the necessary oxygen sensor data in the winter of 2013/2014. The data we will be collecting includes oxygen sensor voltages and lambda ratings, ambient air temperature, as well as catalyst temperatures. With this data, the viability of modern oxygen sensors for cold climates could be assessed statistically.

{Adaptive Maintenance}

As previously discussed, having the ability to analyze specific conditions contributing to a mechanical or electrical failure in an automobile assists in locating the failure and possibly preventing it from happening in the future. On a small scale device deployment, it's likely that this analysis will only benefit the vehicle from which the data originated. This being said, if it achieved a large enough scale to acquire data from vehicles sharing like components, new modes of examining the performance of specific systems in an automobile would manifest themselves. Each vehicle goes through a different wear cycle due to variance of conditions and driving styles. A vehicle manufacturer could analyze long-term OBDII data provided by a number of vehicles (same model) to determine whether certain parts are underengineered for the purpose of the consumer. This information could potentially be used to provide recalls proactively rather than reactively.

{Road Condition Assessment}

Automated road condition notification is another possible use for the OBDII /GPS data acquired. If adverse roadway conditions can be perceived by the driver, then it's likely that the vehicle that has sensors able to detect that same condition. As discussed in the introduction, wheel slip can be calculated using steering angle, vehicle perceived speed, and GPS speed. This could potentially help inform other drivers on the roadway when determining a route for travel. In the same respect, this could be a simple routing algorithm extension for routing an autonomous vehicle.

The basis for the "CANOPNR" research [1] was partially to assess slippery road conditions. The primary difference being their use of the CAN Bus. The Controller Area Network is capable of accessing other forms of data, including the status of the traction control system. Due to the access of these advanced systems, discovering a wheel slip condition requires no calculations whatsoever.

Aside from detecting slick road surfaces, the information gathered from the proposed devices could also potentially detect uneven roads. For instance, if the roadway is rutted from overuse, a vehicle driving on it would usually exhibit a “pull” to the left or right depending on its location in the rut. This can also be detected automatically using GPS and steering angle.

Results

Before the device was completed, a test was carried out regarding the data acquisition speed of the Sparkfun OBDII UART board in relation to different vehicles. This seemed prudent seeing as there are different protocols that the OBDII port operates on depending on the vehicle manufacturer.

Table 2: OBDII Data Acquisition Speeds

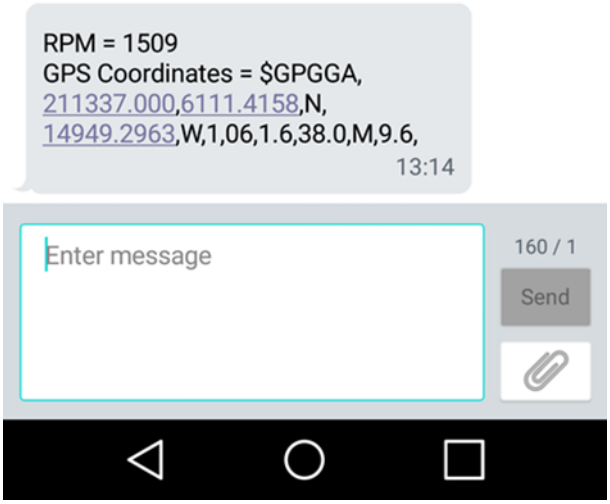
Model Year	Make/Model	Data Points/Second
N/A	ECUSim 5100	38.0
1996	Ford Bronco	5.5
1998	Oldsmobile Bravada	19.6
1999	Dodge Ram 1500	22.0
2000	Ford F350	20.9
2001	Chevrolet Tahoe	26.4
2001	Mazda B3000	25.7
2003	Saturn L200	26.8
2004	Chevrolet Avalanche	26.2
2004	Chevrolet 2500 HD	24.5
2004	Chevrolet Astro	22.8
2005	Jaguar X-Type	4.2
2005	Honda Accord	4.5

After accessing these data speeds, it became apparent that the j1850 protocol utilized by General Motors and The Ford Motor Company maintains the highest acquisition rates at an approximate 23 data points per second. For most research and monitoring applications, this will be much more than sufficient in

providing precision data. On the other hand, vehicles supporting the ISO 9141/14230 such as the Honda and Jaguar on the list were only capable of 4-5 data points per second. That may be a large disparity, but even with an acquisition rate of 4 Hz, that is still twice the speed of the GPS coordinates.

Figure 9 below shows a view of the final implementation functionality via the text sent to an Android cell phone. The test program was designed to send a text when the engine went over 1500 RPM. The first line shows revolutions per minute, whereas the next lines display GPS coordinates of the location where that condition was met.

Figure 9: Final Implementation Functionality



Conclusions

The initial goal of the project was to design and implement an OBDII data-logging device that is capable of collecting GPS data and utilize the cellular network to access the data. Overall, this goal was achieved. There are some future possible additions to the implementation that would add substantial value to the system for fleet management or research uses.

One helpful addition would be a server setup for data storage and processing. The ability for a user to be able to access the server via web-interface GUI would be the difference between the device being a tool for programmers or a tool for anyone. The future goal is to have an apache web-server along with a

MySQL server for logging the data into tables. This would all be programmed with a PHP user interface so that the end user is able to pick a specific piece of data and display it in table format or export it to CSV. On top of these functionalities, it would be quite beneficial to plot GPS coordinates on a map so that the user can view the geographical location of particular incidents as well as driving route mapping.

Another feature that would also be useful is a text response functionality. If the Arduino was constantly listening for text requests via the SIM900 shield. This would allow one to reprogram the functionality of the device without physically being connected to the device, but even possibly states or countries away. With the device polling for incoming texts, one could format a message with what either a request for text data or for the server to start or stop logging data.

This project was very helpful in providing a basic understanding of the OBDII protocols and how they are utilized. Although it is a standardized protocol, there are many differences from manufacturer to manufacturer. The different protocols do provide for different forms of data and at different speeds. Another take-away from this research project is always keep mobile connectivity in mind for wireless devices. The most important issue for a device that relies heavily on GPS and cellular connectivity is where your dead spots are and how to enhance signal in areas of low connectivity.

Most importantly, the goal set was achieved. A low-power consumption device was designed capable of collecting OBDII data from vehicles, collect GPS data, and stream said data over a cellular network. There is still work in setting up an automated server in order to make it a viable solution for fleet maintenance or further research use, but now the framework has been set.

References

[1] Enriquez, D.J.; Bautista, A.; Field, P.; Sun-il Kim; Jensen, S.; Ali, M.; Miller, J.; , "CANOPNR: CAN-OBD programmable-expandable network-enabled reader for real-time tracking of slippery road conditions using vehicular parameters," Intelligent Transportation Systems (ITSC), 2012 15th International IEEE Conference on , vol., no., pp.260-264, 16-19 Sept. 2012

[2] Zaldivar, J.; Calafate, C.T.; Cano, J.C.; Manzoni, P.; , "Providing accident detection in vehicular networks through OBD-II devices and Android-based smartphones," Local Computer Networks (LCN), 2011 IEEE 36th Conference on , vol., no., pp.813-819, 4-7 Oct. 2011

[3] Hu Jie; Yan Fuwu; Tian Jing; Wang Pan; Cao Kai; , "Developing PC-Based Automobile Diagnostic System Based on OBD System," Power and Energy Engineering Conference (APPEEC), 2010 Asia-Pacific , vol., no., pp.1-5, 28-31 March 2010

[4] Wang Dongliang; Huang Kaisheng; Liu Wei; Lin Zhihua; Wang Yinhui; , "OBD System Oxygen Sensor Degradation Monitoring and Mechanism Analysis," Measuring Technology and Mechatronics Automation (ICMTMA), 2011 Third International Conference on , vol.2, no., pp.740-744, 6-7 Jan. 2011

[5] Lin, C. E.; Shiao, Y.-S.; Li, C.-C.; Yang, S.-H.; Lin, S.-H.; Lin, C.-Y.; , "Real-Time Remote Onboard Diagnostics Using Embedded GPRS Surveillance Technology," Vehicular Technology, IEEE Transactions on , vol.56, no.3, pp.1108-1118, May 2007

[6] Yufeng Chen; Zhengtao Xiang; Wei Jian; Weirong Jiang; , "Design and implementation of multi-source vehicular information monitoring system in real time," Automation and Logistics, 2009. ICAL '09. IEEE International Conference on , vol., no., pp.1771-1775, 5-7 Aug. 2009

[7] Shi-Huang Chen; Jhing-Fa Wang; YuRu Wei; Shang, J.; Shao-Yu Kao; , "The Implementation of Real-Time On-line Vehicle Diagnostics and Early Fault Estimation System," Genetic and Evolutionary Computing (ICGEC), 2011 Fifth International Conference on , vol., no., pp.13-16, Aug. 29 2011-Sept. 1 2011

[8] Wang Quanqi; Wang Jian; Wang Yanyan; , "Design of vehicle bus data acquisition and fault diagnosis system," Consumer Electronics, Communications and Networks (CECNet), 2011 International Conference on , vol., no., pp.245-248, 16-18 April 2011

[9] Lin, C.E.; Chih-Chi Li; Sung-Huan Yang; Shun-Hua Lin; Chun-Yi Lin; , "Development of On-Line Diagnostics and Real Time Early Warning System for Vehicles," Sensors for Industry Conference, 2005 , vol., no., pp.45-51, 8-10 Feb. 2005

[10] Namburu, S.M.; Wilcutts, M.; Chigusa, S.; Liu Qiao; Kihoon Choi; Pattipati, K.; , "Systematic Data-Driven Approach to Real-Time Fault Detection and Diagnosis in Automotive Engines," Autotestcon, 2006 IEEE , vol., no., pp.59-65, 18-21 Sept. 2006

[11] Stanley. (2012, June 12). {*The ultimate GPS and OBD-II data logger based on Arduino MEGA*} [Online]. http://www.arduinodev.com/ultimate-gps-obd-data-logger-displayer/ [12] Zaldivar, J.; Calafate, C.T.; Cano, J.C.; Manzoni, P.; , "Providing accident detection in vehicular networks through OBD-II devices and Android-based smartphones," Local Computer Networks (LCN), 2011 IEEE 36th Conference on , vol., no., pp.813-819, 4-7 Oct. 2011

[12] (2012). {*OBD II-UART*} [Online]. Available: https://www.sparkfun.com/products/9555

[13] (2011, May 10). {*SIM900 GSM/GPRS Shield IComsat Preview*} [Online]. Available: http://blog.iteadstudio.com/sim900-gprs-shield-icomsat-preview/

[14] (2013). {*ITEAD GPS SHIELD*} [Online]. Available:http://imall.iteadstudio.com/development-platform/arduino/shields/im120417017.html

[15] (2012). {*Arduino Uno*} [Online]. Available: http://arduino.cc/en/Main/arduinoBoardUno

[16] Hu, Y.; Tan, O.K.; Cao, W.; Zhu, W.; , "Fabrication and characterization of SrTiO3 oxygen sensor operating at very low temperature," Sensors, 2003. Proceedings of IEEE, vol.2, no., pp. 1305- 1309 Vol.2, 22-24 Oct. 2003

[17] B. Hollembeak, "Open Loop vs Closed Loop," in {it\ Shop Manual for Automotive Fuels and Emissions,} Clifton Park, New York: Thomas Delmar Learning, 2005, ch. 6, pp. 176-177.

[18] (2012). {*OBD Mini Logger*} [Online]. Available: http://hemdata.com/products/dawn/obd-mini-logger

[19] (2013). {*Car Chip Fleet Pro*} [Online]. Available: http://www.carchip.com/Products/8246.as

[20] (2013).{*Data Scout OBD-II Data Logger*} [Online]. Available: http://www.deltaforcetuning.com/Data-Scout-data-logging-OBD-wireless-p/da-9000.htm

[21] } (2013). {*DashDyno SPD ProPack*} [Online]. Available:http://www.auterraweb.com/dashdynopropack.html

[22] (2010). {*IOSiX OBD-II Datalogger*} [Online]. Available:http://caflor.net/datalogger.html

Appendix

```
//Arduino Code
/*************************************************************************
* Arduino Implementation for OBD-II UART/I2C Adapter
* Distributed under GPL v2.0
* (C)2014-2015 Kristian Smith <kasmith11@alaska.edu>
*************************************************************************/
#include <SPI.h>
#include <Wire.h>
#include <OBD.h>
#include <SoftwareSerial.h>

const int chipSelect = 10;
SoftwareSerial GPS(2,3);
SoftwareSerial SIM900(4,5);

COBD obd;

void setup()
{//Cell
  Serial.begin(9600);
  SIM900.begin(19200);
  GPS.begin(9600);
  SIM900power();
  delay(20000);  // give time to log on to network.

  //OBD
  // we'll use the debug LED as output
  pinMode(13, OUTPUT);
  // start communication with OBD-II adapter
  obd.begin();
  // initiate OBD-II connection until success
  while (!obd.init());
}

void SIM900power()  // software equivalent of pressing the GSM shield "power" button
{
  digitalWrite(9, HIGH);
  delay(1000);
  digitalWrite(9, LOW);
  delay(5000);
}
/*void sendSMS()
{
  SIM900.print("AT+CMGF=1\r");                                    // AT command to send SMS message
  delay(100);
  SIM900.println("AT + CMGS = \"+19076027732\"");                       // recipient's mobile number, in international
       format
  delay(100);
  SIM900.println("Hello, world. This is a text message from an Arduino Uno.");       // message to send
  delay(100);
  SIM900.println((char)26);                    // End AT command with a ^Z, ASCII code 26
  delay(100);
  SIM900.println();
  delay(5000);                                // give module time to send SMS
  SIM900power();                               // turn off module
}*/
```

```
void loop()
{
 int value;

 // GPS ---make a string for assembling the data to log:
 char index = 0;
 char temp = 0;
 String dataString = "";
 //end GPS
 //OBD
 // save engine RPM in variable 'value', return true on success
 if (obd.read(PID_RPM, value)) {
  // light on LED on Arduino board when the RPM exceeds 1500
  //digitalWrite(13, value > 1500 ? HIGH : LOW);
  if (1){
  //END OBD
    //GPS
    while(GPS.available())
  {
   temp = GPS.read();
   dataString += String(temp);
   index++;
   if(String(temp) == "$")
     Serial.println("");
   if(index>200){
     break;
   }
  }//END GPS

  //SIM900
  SIM900.print("AT+CMGF=1\r");                                    // AT command to send SMS message
  delay(100);
  SIM900.println("AT + CMGS = \"+12533502780\"");                       // recipient's mobile number, in international
      format
  delay(100);
  SIM900.println("Relative Throttle Pos = " + String(value) + " \nGPS Coordinates = " + dataString);       // message to
      send
  delay(100);
  SIM900.println((char)26);                  // End AT command with a ^Z, ASCII code 26
  delay(100);
  SIM900.println();
  delay(5000);                          // give module time to send SMS
  SIM900power();                          // turn off module
  //END SIM900
 }

 }


}
```