The Car Monitor System

April 29, 2007

Joshua Rupiper

Tyler Andrews

James Dicke

Contents

1. Introduction

1.1 Purpose

The Car Monitor System's primary goal is to further connect the driver to the car. Using the system, a driver can easily obtain in-depth information about his or her car's operational history. This information, including mechanical and positional data, is accessible through a simple and easy-to-use web interface.

The average car has approximately thirty five microcontrollers, each managing some aspect of the vehicle's function [1]. Messages transmitted from these sensors are received and decoded by the vehicle's On-Board Diagnostics (OBD) bus. This information could then be obtained using a code reader, a tool which the majority of auto mechanics own and use regularly. Most drivers do not possess a code reader, making the data gathered by each sensor virtually inaccessible.

In most cars today, any data generated by the vehicle is temporarily recorded, and soon after is overwritten to make room for new data. No long-term logging mechanism exists to monitor measurements made by the vehicle. There are several benefits to acquiring captured vehicle data over a prolonged timeframe. These include the ability to diagnose long-term problems, identify trends, and monitor overall performance.

More specifically, engine data could be used to study and optimize a car's fuel efficiency. Car emissions could also be recorded, warning the driver of any serious problems. Various other types of data could be used to predict malfunctions, and together would contribute to the vehicle's overall safety and performance.

In addition to recording a vehicle's engine and operational parameters, extra sensors could be attached to the Car Monitor System to provide supplemental data. A

Global Positioning Satellite (GPS) receiver could record the car's location. The driver would then be provided with a frame of reference as to where the car was located when a particular event occurred, since he or she is more likely to remember the car's location rather than the exact time. This data could also be used to investigate the user's driving habits. For example, a teenager's parents would be able to determine where the car was located at a particular time. The parents would also be able to determine if their teen is indulging in unsafe driving habits, such as speeding, or failing to stop at a stop sign. A rangefinder could be used to monitor the distance to nearby cars to ensure that the teenager is maintaining a safe distance, particularly at high speeds. The rangefinder would also hint at current traffic conditions.

The Car Monitor System will be designed to be expandable, so that extra sensors can be easily added for further functionality. A separate logging mechanism could be used for each sensor and later transmitted to the web interface. This will allow for modern technologies and trends to be placed in cars after they are manufactured. In the same way that GPS device can be installed in a 1997 model vehicle, it will be possible to take advantage of newer innovations without having to replace an entire vehicle.

1.2 Originality

A number of vehicle monitoring and tracking systems are in commercial production today. However, there are clear advantages to using the Car Monitor System as opposed to these alternatives. First and foremost, the system is intended to be affordable for any user, and would not require monthly fees. Qualcomm's popular OmniTRACS system is capable of two-way satellite communication so that a vehicle's owner can track its position at all times [2]. The disadvantage is that set up costs range in

the thousands of dollars and a moderate monthly fee is charged for service. The Car Monitoring System is intended to cost far less than a thousand dollars, does not incur monthly fees, and can record both vehicle sensor information and GPS positioning. No commercially produced system has both of these features with such a low manufacturing and development cost.

2. System design specification

The primary function of the Car Monitor System will be to record information gathered by a series of sensors located on the vehicle. The data will then be transferred from localized storage to a database server, where it will be interpreted into a more meaningful form and made easily accessible to the user. The user could then view each recorded piece of data as indexed by a timestamp. For example, the location of the vehicle at any given time could be attained using GPS data. In total, the entire Car Monitor System is comprised of an onboard system with vehicle and GPS monitoring capabilities and a centralized server hosting a web-accessible user interface. A logical diagram of the Car Monitor System is shown in Figure 1.

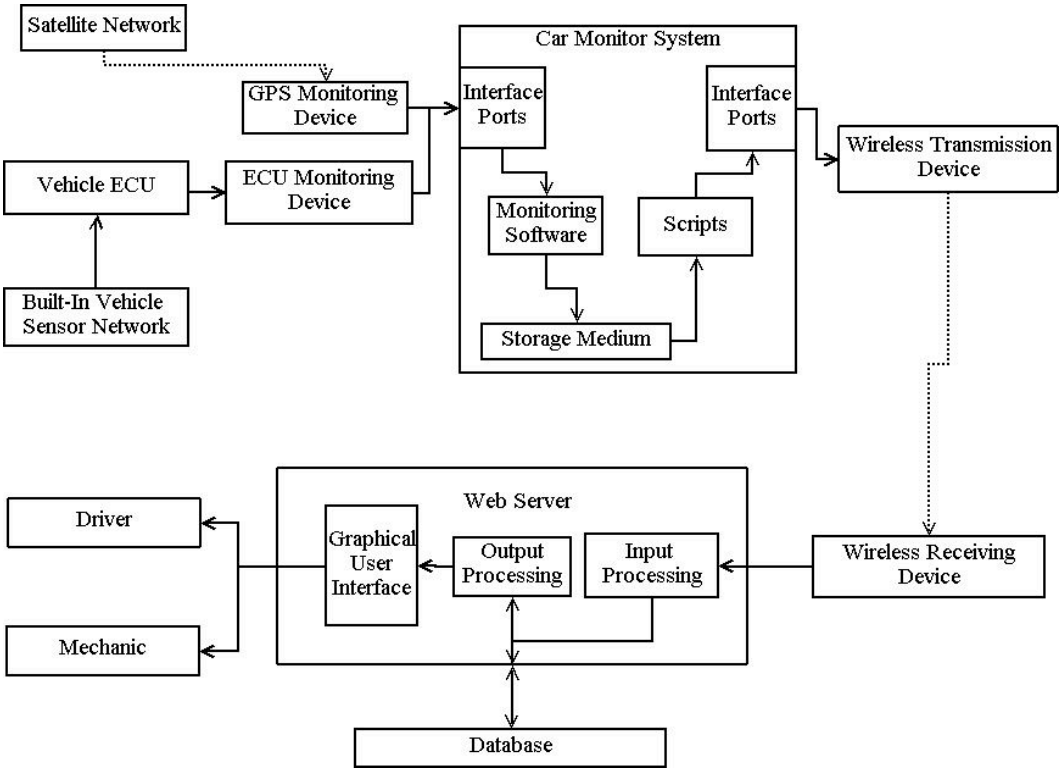

Figure 1: Logical design of the Car Monitor System

2.1 Onboard system

There are several requirements that must be fulfilled by the onboard system. The Car Monitor System will need to log and record information that will be viewable only by an online interface. Therefore, the user should not physically interact with the system once installed. . Also, it would be ideal for the onboard setup to be as small as possible, so that it could be positioned in a way that makes it nearly invisible to the driver. Power consumption is also a consideration. The system must not drain the automobile's battery supply at an unreasonable rate.

The onboard system itself must include functionality to interact with vehicular sensors, such as an OBD-II interface and GPS receiver. Any additional sensors added to the Car Monitor System would also need to be accessible. In order to store and transmit recorded data, the system would also need memory and the ability to interface with the Internet. Ideally, a specialized embedded system could be designed to meet these demands. However, a low-cost and low-power computer would definitely meet the provided specifications at the expense of an extended startup time and increased production cost.

2.2 Vehicle and GPS monitoring

The Car Monitor System's vehicular monitoring capabilities will require a direct connection to the OBD computer bus. Therefore, a physical interface will need to be attached to the onboard system. Information recorded from the car's OBD-II bus may include, but is not limited to, the vehicle's odometer, speedometer, engine RPM, throttle position, engine temperature, engine load, air intake, oxygen sensors, and diagnostic error

codes. This information must be made available to the onboard system by use of either a custom-designed or publicly available software suite.

For GPS tracking, a small and affordable receiver is required. The device will not require any user interaction, so the ideal GPS solution will not contain a viewable display. Also, the receiver should be as accurate as possible yet small enough to hide away with the onboard system.

2.3 Server and user interface

The server will be placed at a centralized location and will be the only aspect of the Car Monitor System needing maintenance. However, because the system will not require automatic updating, only a minimal amount of routine maintenance will be necessary. The server should be visible worldwide via the Internet.

The server's purpose is to store, organize, and present data to the user. For this reason, a database will be used to store all significant logged data. The database should merge smoothly with a scripting language so that the input data can be converted to a meaningful visible representation. It may also be necessary to merge multiple datasets and interpolate or extrapolate missing data. For example, a trip lasting several hours would contain too much data for a user to examine by hand. The server should identify key outliers and significant data points for easy accessibility.

3. System description

Each element of the Car Monitor System was carefully researched and chosen due to its compatibility with the design specification for the system. Specifically, the system components corresponding to the logical diagram are shown in Figure 2.

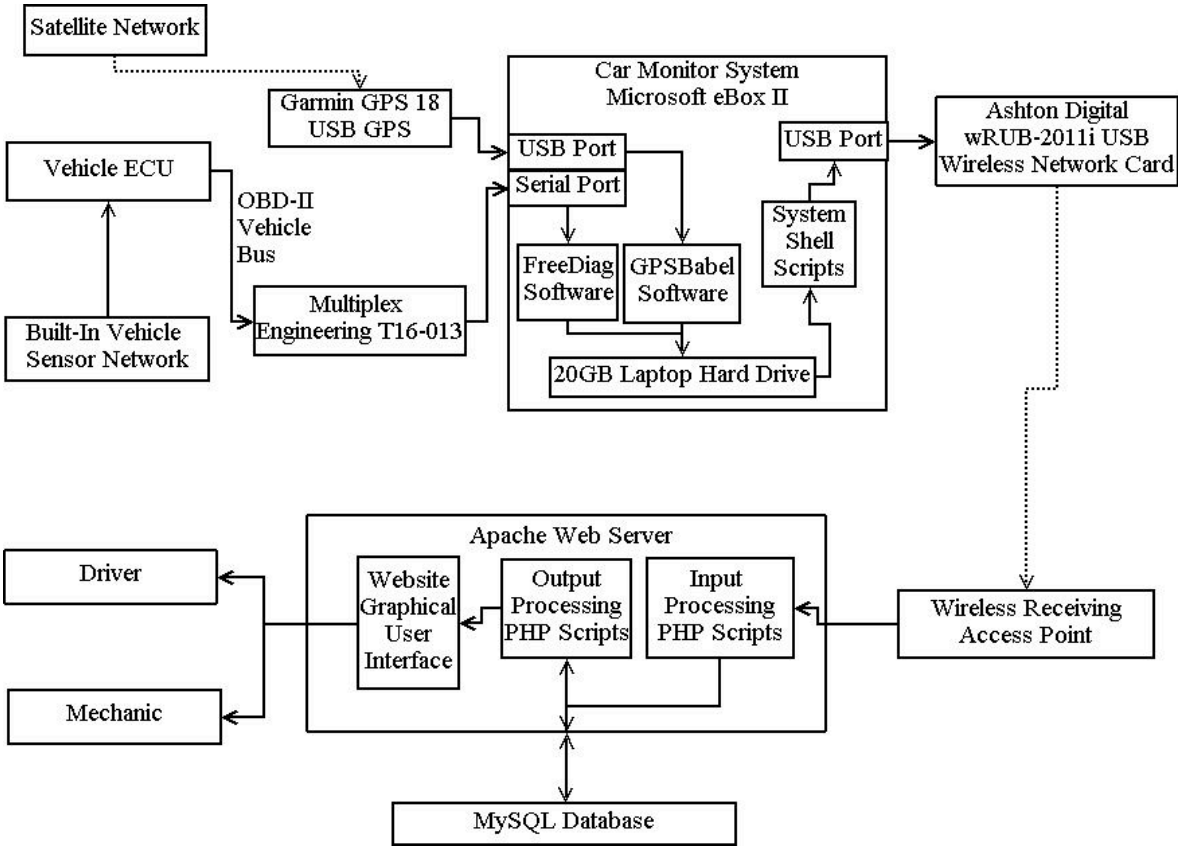

Figure 2: Design description of the Car Monitor System.

3.1 Onboard system

A Microsoft eBox II was used as the car-based platform on which to interface with the vehicle's OBD-II system and GPS receiver. The eBox has an x86-compatible 200 MHz processor, 128 MB RAM, built-in flash storage, video, audio, network adaptor, parallel port, serial port, and three USB ports [3]. An additional 2.5 inch 20 GB hard drive was also installed for greater capacity on the eBox's internal EIDE controller.

Physically, the device measures 5.2 x 2.5 x 4.4 inches and weighs approximately one pound. The small size of the platform allows for discrete placement within the vehicle. The entire onboard system, including the OBD-II and GPS interfaces, is shown in Figure 3.

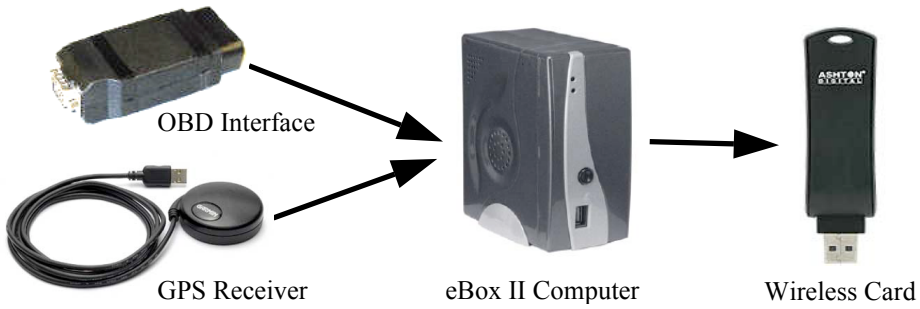

Figure 3: Physical onboard components of the Car Monitor System

Due to the fact that few programs and even fewer device drivers exist for the eBox's default Windows CE operating system, a lightweight Linux distribution, Demi-Sized Linux (DSL), was chosen and installed onto the eBox. This distribution of Linux is based on Debian, is open-source, and contains full Linux functionality while requiring less than 50 MB of space [4]. DSL has extremely low system requirements, which makes it ideal for use with the eBox platform. Also, because DSL supports Debian repositories, installation of numerous required software packages was significantly simpler, faster, and easier than it would have been using a custom operating system. In addition, Demi-Sized Linux is open-source and is thus free to use, adding no additional cost to the project.

The eBox's serial port is used to connect to the OBD-II interface. The USB ports were used to connect to both a wireless network interface card and to a GPS receiver. The wireless card being used is an Ashton Digital Air Dash WRUB-2011i. This device was chosen due to its Prism 2 chipset, which is known to be extremely reliable and stable in a Linux environment.

3.2 OBD-II interface description

While most drivers are familiar with the dreaded red Malfunction Indicator Light (MIL) and other basic indicators and meters on their dashboards, few are aware of the complex network of sensors and electronic controllers within their vehicle. During operation the vehicle's Engine Control Unit (ECU) continuously utilizes numerous sensors located within the vehicle to determine the most optimal configuration of timings and fuel mixings for the vehicle [5]. The ECU is also programmed to look for hundreds of vehicle problems, and in the event that one is detected it will store a detailed Diagnostic Trouble Code (DTC), archive a current copy of the vehicle's sensors at the time of the problem, and switch on an error light such as the MIL for the driver if the detected error merits the drivers attention [6].

Current vehicle computer systems are the result of the Environmental Protection Agency (EPA) tightening vehicle emissions requirements to reduce pollution. The EPA worked together with the Society of Automotive Engineers (SAE) to require that all vehicles sold in the U.S. adopt their control system standard titled OBD-II. All automotive manufacturers were required to abandon their previously incompatible emissions diagnostic protocols and converge on OBD-II beginning with vehicles sold in 1996 [7]. The single OBD-II protocol was adopted successfully, but the different manufacturers varied in their implementations and ended up creating five major incompatible underlying signal protocols. As time progressed, the manufacturers went beyond the initial OBD-II specifications and began to install additional sensors which expanded the interface's capabilities.

Despite the large number of signaling protocols, the physical OBD-II interface connector is exactly the same size on all vehicles and by requirement must be readily accessible to the driver. Numerous companies produce OBD-II scanning devices which have been traditionally used by mechanics, hobbyists, and racers to monitor details about their vehicle's emissions, performance, and DTCs. Modern OBD-II scanners commonly implement multiple signaling protocols and can be found as either standalone devices or as an interface requiring connection to a computer. The ECU is designed to handle a reasonable amount of scanning and polling, and doing so has no negative impact on the vehicle, even while moving.

3.3 OBD-II PID and DTC communication

The OBD-II protocol operates by first requiring the user to request specific parameters from the vehicle's onboard computer via a series of hexadecimal bytes sent over the OBD-II bus. The ECU issues a varying number of hexadecimal bytes in response to the request and these bytes must be either interpreted bitwise or converted to metric units through a series of predefined calculations. OBD-II operates in one of nine modes, the most common modes being numbers one and two. Each mode is broken up into Parameter Identification numbers (PID) assigned to represent standardized data within that mode. Mode one contains common vehicle parameters like speed, RPM, temperatures, while mode two contains the freeze-frame car parameters stored when the most recent DTC was issued. Since the actual sensors found on a given model and year of a vehicle will vary, the protocol must have a method of determining which sensors are present on any given vehicle. A special one-time initial command, PID 00 for mode one, is sent to the vehicle and the response indicates exactly which PIDs are supported within

the corresponding mode. A software application can then store this response and enter a loop requesting these supported values every few seconds. The application can either store and/or display the results. A few of the most common PIDs and their descriptions are provided in Table 1 for demonstration, but the full specification includes dozens of additional PIDs [8][9].

Mode (Hex)	**PID (Hex)**	**Description (BE = Value Returned is Bitwise Encoded)**
01	00	Returns all the PIDs supported on this vehicle (BE)
01	01	Returns number of DTCs, if MIL is active, and vehicle tests available (BE)
01	03	Current fuel system status (BE)
01	04	Calculated engine load value (%)
01	05	Temperature of engine coolant (degrees C)
01	0C	Engine RPMs
01	0D	Vehicle speed (km/h)
01	0F	Intake air temperature (degrees C)
02	02	Get the frozen engine parameters at the last trouble code (if applicable, BE)
03	--	Get all the saved DTCs (if applicable, BE)
04	--	Turn off the MIL and clear any saved DTCs

Table 1: OBD-II PIDs description.

As an example of a communication session between the user and the ECU over the OBD-II bus, the following sequence of events may occur. First, the user connects to the bus using the proper signaling protocol for the vehicle and initiates communication by sending the hexadecimal values 00 and 01 to request which PIDs the ECU supports. The ECU responds with a 4-byte hexadecimal response such as “be 3e b8 10” which is converted to the bits “1011 1110 0011 1110 1011 1000 0001 0000.” Each bit represents one of the first 32 PIDs (01 to 20 in hexadecimal) read from left to right where a one corresponds to a supported PID and zero corresponds to unsupported. In this case, we know that PIDs 01, 03, 04, 05, 06, 07, 0a, 0b, 0c, etc are supported and PIDs 02, 08, 09, 10, 0a, etc are not. The user now saves this first query and knows it can request any of the supported PIDs at a future time. To get the vehicle RPMs the user could send “01 0c” as

shown in table 1. If the ECU were to respond with the two bytes “19 5c,” the user could then perform the pre-defined calculation on the decimal equivalents: ((25*256)+92)/4 yielding a value of 1623 RPM.

The MIL light on a modern vehicle only indicates to the driver that a problem exists, but the Diagnostic Trouble Codes (DTCs) generated by the vehicle give a detailed report of the problem. Each DTC, such as “P0303”, is intended to be looked up in a reference manual or online to determine its description. The characters that comprise the DTC, from left to right, increase the specificity of the error. As you can see in Figure 4, the first character can be ‘P’ for a powertrain problem, ‘B’ for a body problem, ‘C’ for a chassis problem, or ‘U’ for an undefined problem. The second character is ‘0’ if it is a generic vehicle code common to all vehicles or ‘1’ if it is a manufacturer-specific error code. The third character can be 0-9 and indicates which part of the vehicle the error occurred (e.g. 3 is ignition system, 7 is transmission system, etc). The fourth and fifth characters work together to identify the details of the problem [6].

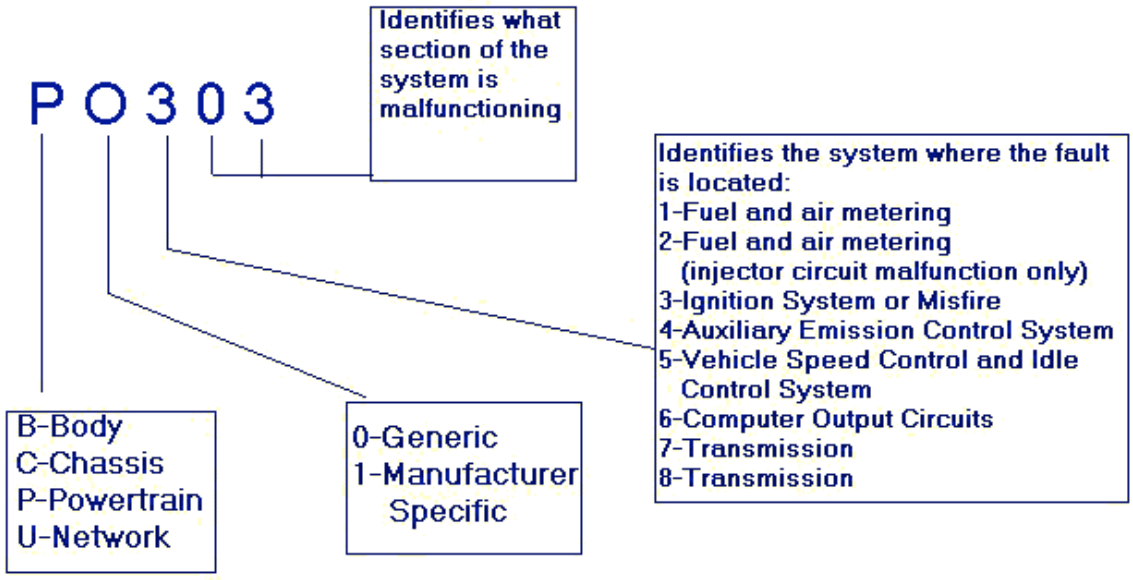

Figure 4: Translating the meaning of a diagnostic trouble code

The Car Monitor System implements the ability to detect that one or more DTCs have been generated and store all codes in the database. The current database implementation does not contain a listing of DTCs and their English descriptions. A future expansion to the database would be to include this comprehensive list of generic and manufacturer specific error codes where the manufacturer is matched with the user-provided value for the car. Special notifications could be issued through email upon DTC detection. Additionally, the DTC could be noted on the generated Keyhole Markup Language (KML) files with special colors or symbols, and the mechanic could receive selective data regarding the length of time each error has been present.

The following scenario is an example of receiving a DTC from a vehicle. DTCs are stored in mode 3 with no PID so the mode signal “03” should be sent to the car’s ECU. The car will respond with a hexadecimal string padded to a multiple of six bytes. Each DTC is two bytes long and bitwise interpreted from left to right. If the ECU’s answer is “0303 0000 0000” then only one DTC is stored, “0303”, which is converted into binary as “0000 0011 0000 0011”. The first two bits are “00” which together mean ‘P’. The third and fourth bits, “00”, conclude that the DTC is generic. Bits 5-8, 9-12, and 13-16 are direct binary interpretations into the digits 3, 0, and 3 [8]. The concatenated DTC is “P0303”, which when looked up in a comprehensive list of DTCs, translates to “Cylinder 3 Misfire Detected” [10].

3.3 OBD-II implementation

The physical OBD-II monitoring device purchased for the Car Monitor System was the T16-013 from Multiplex Engineering [11]. The device contains circuitry designed to interface with four out of the five major signaling protocols and converts the

OBD-II vehicle data to a standard serial (RS-232) port. It strikes an excellent balance between cost and functionality as it retains high compatibility and at $82 is at least 30% cheaper than similar devices from competitor companies supporting all five major protocols [12]. This device is part of the T16 family which is widely used among the open-source community and is supported by the Car Monitor System's OBD-II monitoring software.

FreeDiag is the open-source vehicle scanning software utilized for the Car Monitor System [13]. A key feature of the selection of this program is its widespread compatibility with several devices and OBD-II protocols. Additionally, FreeDiag appears to have the most mature code of any open-source OBD-II scanning project available and has been tested and proven to work with a large range of vehicles and scanning devices. It automatically handles the polling and logging of ECU data at approximately five second intervals. Example mode one output, which contains common parameters like speed, RPM, temperatures, etc is shown in Figure 5.

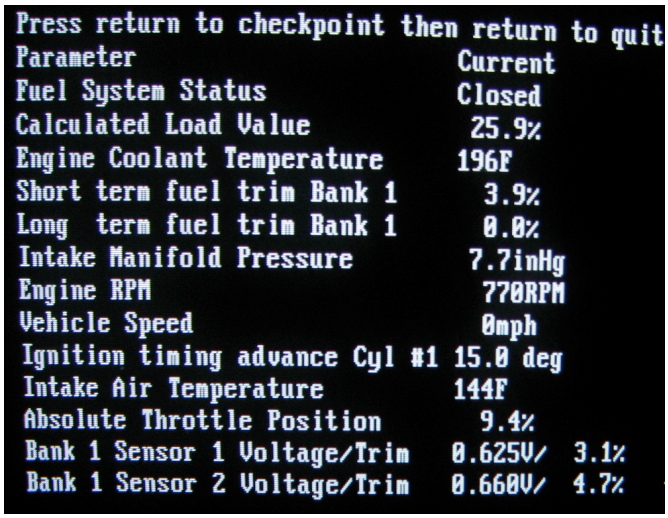

Figure 5: FreeDiag vehicle monitoring.

Another advantage to using FreeDiag is that it is also an open-source software package which adds nothing to the project cost, but still offers the advantages of customizability and compatibility. The ability to change the FreeDiag source code was critical as it allowed for the implementation of a more advanced and robust logging system. The program is currently set to create a new log file based on the current date and time whenever FreeDiag is initialized. Also, utilizing FreeDiag's built-in configuration scripting, custom scripts were created to make FreeDiag automatically begin scanning and monitoring when executed.

3.4 GPS description

A GPS receiver receives low-power radio signals broadcast by a constellation of 24 orbiting satellites [14]. In order to calculate its location, the GPS receivers must obtain the location of at least three satellites and the distance between each of these satellites and the device. The location of each satellite is determined by lookup in a digital almanac containing the location of each satellite. This simple lookup is possible due to the fact that each satellite travels in a predefined orbit. The distance between each satellite and the receiver is determined by multiplying the radio signal propagation speed by the delay in receiving the radio signal. Both the location of at least three satellites and the distance between each of these satellites rely heavily on having an extremely accurate clock.

In order to determine the precise time, the device first receives time signals from four or more satellites generated by each satellite's onboard atomic clock. Due to propagation delays, each of the received time signals will report a slightly different time. Using these minor variances in time and the signal propagation speed, namely the speed of light, the device calculates the correct time so that the signals align at a single point in

space. Once the location of each satellite and all relevant distances are known, GPS receivers then use three-dimensional trilateration in order to determine its exact position on the Earth's surface.

There are several problems that can reduce a GPS receiver's accuracy. Radio signal propagation speed can be affected by atmospheric conditions, altering the distance reported between the receiver and satellite. Radio signals may be also affected by large objects, such as mountains and skyscrapers, altering the radio signal speed. GPS satellites also may occasionally deviate from their predefined orbits, altering both the receiver's time and position calculations.

In order to correct these problems, many GPS receivers use differential GPS (DGPS). Stationary ground stations with confirmed locations receive signals from satellites, and compare their established GPS coordinates with a set of received coordinates to determine any fluctuations in each satellite's accuracy. The receiver stations then broadcast GPS signal corrections, which other GPS devices receive and factor into determining their position.

Once the device's position is determined, many GPS receivers are also capable of calculating the current heading and speed. These are calculated by comparing the current position and time to previously recorded position and time information.

3.5 GPS implementation

The GPS receiver selected for use with the Car Monitor System is the Garmin GPS 18 USB. The device is a 12-parallel-channel WAAS-enabled GPS receiver [15]. WAAS, or Wide Area Augmentation System, is a version of DGPS covering North America. A WAAS-enabled receiver is, on average, five times more accurate than a GPS

receiver without WAAS. Using WAAS, the GPS receiver can provide a position that is accurate to three meters 95 percent of the time.

The program being used to obtain GPS coordinates from the receiver is GPSBabel 1.3.3, an open-source multi-platform GPS data manipulation program. The original purpose of GPSBabel was to convert GPS data from one format to another [16]. However, additional interfaces have been added to GPSBabel so that it can retrieve GPS data directly from a connected GPS device and then convert the data to over one hundred different formats. The most common of the formats supported by GPSBabel include Microsoft Streets and Trips, Garmin MapSource, Google Earth, and Comma Separated Values (CSV). GPSBabel typically runs using a graphical user interface, but the program also supports operation from a command-line interface as well.

GPSBabel includes a feature for real-time tracking [17]. In order to initialize real-time tracking, the command shown in Figure 6 can be used to continually read from an attached USB Garmin device. The output format is set to KML format, which is primarily used by Google Earth.

gpsbabel –T –I garmin –f usb: -o kml –F filename.kml	
-T	Track
-I *inFormat*	Input format: Garmin proprietary format
-f *infile*	Input file: usb connection
-o *kml*	Output format: kml
-F *outfile*	Output file: filename.kml

Figure 6: Real-time tracking using GPSBabel.

The KML format complies with the XML standard, and uses a structured hierarchy of customizable tags. When a KML file is displayed in Google Earth, the result is a satellite map overlaid with constructs defined within the KML file. These structures include marks, paths, polygons, descriptions, and hyperlinks associated with each item.

3.6 Automated server interaction

An important feature of the Car Monitor System is its transparency to the user. The system should automatically determine it has entered the range of the home wireless network, verify the connection, and transmit data to the web server. The Car Monitor System does this through a series of shell scripts executing on the system continuously attempting to establish a connection to the preconfigured SSID of the wireless network. Upon connecting, an attempt is made to obtain an IP address automatically from a DHCP server and a series of ICMP pings is issued to the remote web server. If this is successful, a separate PHP script is executed that is responsible for removing any empty logs and uploading the GPS and OBD data. The PHP script utilizes HTTP POST to send the log files and removes them after transmission.

Future expansion to the scripting system would create a more robust and configurable environment on the eBox. A valuable addition would be either a symmetric-key or public-key based cryptography scheme between the eBox and web server using an encryption library such as MCrypt. This would permit secure data transfer independent of the wireless network's security level. A second useful feature would be a web-based management page where the user could change eBox system settings (e.g. public key, SSID, IP address configuration, etc) and these settings would be automatically downloaded and applied to the eBox. A third expansion could be a two-way communications system about log files received by the server to guard against partial transmissions. A server-generated list of uploaded logs and the number of records in each log would be matched against the logs on the eBox prior to file deletion, thereby guaranteeing the success of the previous transmission.

3.6 Server processing

The Car Monitor System's server side will be comprised of the following free and open-source applications: MySQL database, Apache web server, and scripts written in PHP. The combination of these three applications is a common backbone for many websites and servers. The Apache web server fetches web pages with exceptional speed. The MySQL database is feature-packed and is an excellent storehouse for all the car and GPS data. The PHP scripting language allows for powerful dynamically-generated web content with good documentation, and it contains a large predefined function library.

The MySQL database stores data about each driver or mechanic that interacts with the vehicle and their relationship to the vehicle as owner or mechanic. The database also contains a large number of DTCs, PIDs, and their descriptions. Log files have all relevant data stored as a trip associated with a vehicle and driver and are flagged when uploaded or transmitted. Each ECU reading is stored in a table of ECU logs which contain the measurements and their associated timestamps. Additional tables will be added to incorporate GPS logs and vehicle error code definitions.

The input script presently takes raw OBD-II signals from FreeDiag's output and parses out relevant vehicle information for direct storage in the database. Each vehicle log represents one trip and is given an optional name and is associated with the vehicle which recorded the data. The hexadecimal bytes are decoded based on their detected mode and PID, and the necessary calculations and conversions are performed to translate the parameter's value into its equivalent value in English units. The timestamp associated with each point in the dataset is then placed alongside the value in the database. This transformation is demonstrated in Figure 7.

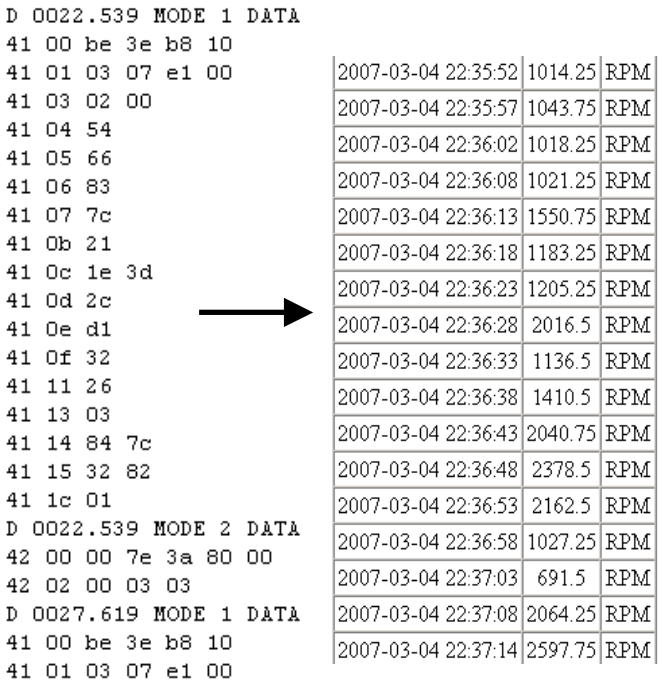

Figure 7: Parsing and converting PID data.

A similar input script will also be implemented for the GPS log files that reads in a KML file and inserts the coordinates and their corresponding timestamps into the database. The data remains in a raw state in the database until the output scripts determine which data is requested and at that time the merging of vehicle data, DTCs, and GPS data takes place. The output scripts may filter or perform interpolation on the data as it is prepared for output to the user.

The drivers and mechanics who visit the Car Monitor System website are currently presented with a “Trip Manager” where they can view any trips associated with any vehicle in the database. As shown in Figure 8, the user is given the option to view any of the basic vehicle parameters for a selected trip in a graph or tabular format.

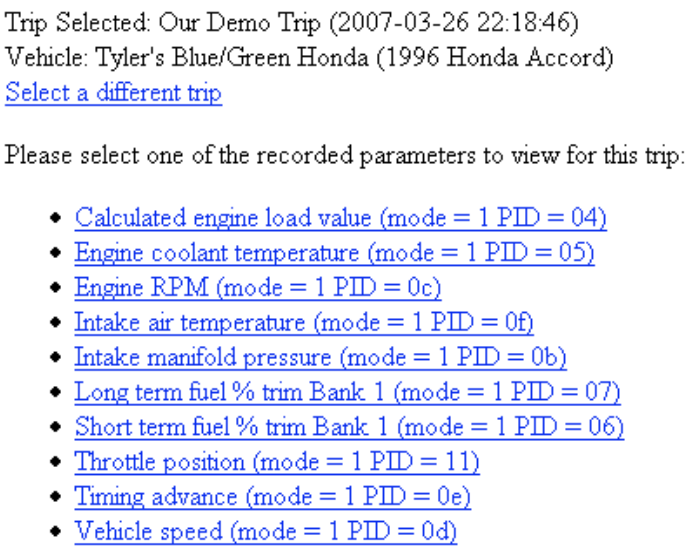

Figure 8: Trip manager parameter selection.

Future improvements to the website will incorporate the ability to see GPS data and obtain a KML file with route information that can be loaded into Google Earth for convenient replaying of the trip. Also, drivers and mechanics will be presented with separate password-protected interfaces limiting the mechanic to only vehicle information and restricting the GPS information which might be considered private to the driver.

4. Project development

A completed demonstration model of the Car Monitor System was successfully tested. This model incorporated the primary features required of the system and also met all basic specifications. The purpose of this section is to specifically outline the functionality of the demonstration model and also to discuss any observed shortcomings and possible extensions to the system.

4.1 Performance and functionality

Testing the functionality of the Car Monitor System is a fairly simple feat upon completion of the project. In order to test the system, a test drive from one point to another can be made. Several test drives have been made: some logging only OBD-II data and others logging only GPS data. A demo trip that incorporated both was also completed. This trip involved using both FreeDiag to monitor and log the OBD-II signal data and GPSBabel to record GPS coordinate data. The demo trip successfully showcased monitoring both position and engine parameters during a drive in the city.

Although simple testing is adequate to determine functionality, it is difficult to determine pertinent metrics of performance for the Car Monitor System. Despite this, several comparisons of the Car Monitor System were made, and several limitations of performance and reliability were determined.

During a demonstrational drive, a brief log of the vehicle's speed and location at regular intervals were manually recorded using a second GPS device. Once the drive was finished and the OBD-II and GPS logs uploaded, the Car Monitor System's results and the manual log were compared. The automatic logs were found to be consistent with the manual logs. Although it is difficult to determine exact discrepancies in accuracy, it is

apparent that the log is accurate to within the specifications of the system. Despite this, several additional performance issues complicate the performance of the Car Monitor System.

When the car's ignition is started, there is a delay before logging begins. This delay is due to the eBox's startup time, which is approximately one minute. During this time, no logs are generated, and the GPS and OBD-II information generated during that time period is lost. This delay in logging was found to be acceptable, due to the fact that many auto mechanics suggests letting a vehicle warm up for thirty seconds to a minute before driving. Also, at least thirty seconds are often required to drive the vehicle out of a parking lot or garage and onto the road.

If the eBox's startup time were to be decreased, limitations of GPS data acquisition times exist, in which the GPS receiver must acquire satellite signals to determine its position. The Garmin GPS 18 requires a "cold" startup time of forty five seconds, and this time is significantly lower if the device is "warm" or "hot". With the current startup time of the eBox, the GPS startup time is undetectable.

An additional metric of performance is the accuracy of the system's measurements. OBD-II information is entirely gathered by the vehicle's own sensor network, and any erroneous data gathered will be reported by the Car Monitor System. This is unavoidable, and confirming sensor data was beyond the scope and purpose of the Car Monitor System, whose goal was to simply log the information gathered by the vehicle's sensors, and not generate its own.

According to Garmin, coordinates obtained by the GPS 18 are accurate to within three meters using WAAS [15]. This degree of accuracy is certainly adequate for the Car

Monitor System, and the driver will be able to clearly determine the vehicle's position at any logged point.

In terms of reliability, the Car Monitor System must operate in any environmental condition existing in the inside of a car. This includes extreme hot and cold conditions. The eBox and GPS receiver were tested under such conditions, with no obvious immediate failures. Prolonged expose to either, particularly heat, may eventually cause damage to the eBox and the GPS receiver. No tests were made to investigate prolonged exposure.

4.2 Difficulties and shortcomings in design

A number of difficulties have appeared in designing and implementing the Car Monitor System. Many of these difficulties have been encountered and surpassed, while others may simply not have a feasible solution within the system's design.

One major hurdle to success was the lack of documentation for OBD-II interface software. It was often uncertain what each program was designed for and was capable of. Compatibility between specific OBD-II interfaces and each OBD program was also poorly documented.

Maintaining data integrity on the eBox is also a difficult task to accomplish. The eBox is powered using a power inverter, and draws its power from the vehicle's electric systems. When the car is shut off, the eBox also immediately loses power. If the eBox's hard drive is not safely shut down, errors may be created in the operating system, causing the system to entirely stop working, or portions of the generated log files may become corrupted. This is especially important in the case of a damaged Linux kernel. The entire

system may be unrecoverable, and the Car Monitor System would have to be reconfigured.

For this reason, a read-only operating system was considered. In such a system, multiple drives would be used to separate the location of the log files and operating system. The read-only operating system would be incorruptible, and only the logged data could be corrupted, which is considered an acceptable loss. Another option would be to use a backup battery to power the eBox for a short time after turning off the car. This would allow the system to perform a safe shutdown.

Another extension to the Car Monitor System would be to merge GPS and OBD-II logs into one KML file. Each GPS location would be associated with the most recently logged OBD-II reading. Expansive parsing and processing of both the OBD-II log file and original KML file could be completed by a PHP script in order to create a single merged KML file. Such a file, when viewed in Google Earth, would allow the user to view the entirety of the system's logged information in a single convenient format. One issue with this implementation is the difference in polling times between the two logs. GPS data is logged every second whereas OBD-II data is logged every five seconds. One simple solution would be to simply select one fifth of the GPS data points for viewing.

5. Overall analysis of implementation

The Car Monitor System is a complete working model. Although certain features and shortcomings in the design should be addressed before large-scale production, the system would not require any drastic modification. Nevertheless, it is worthwhile to consider both practicality and budget considerations for production.

5.1 Practicality and budget considerations

The intention of this project was to create a standalone system that didn't depend on any vehicular components except for compatibility with the OBD-II protocol. As mentioned previously, there are five OBD-II protocols in existence. Currently, the Car Monitor System is compatible with the SAE J1850 VPW standard used by Chrysler and European models, the SAE J1850 PWM, which is commonly found on Ford Motor Company vehicles, and the ISO-9141 standard utilized by most Asian vehicles [7]. This makes the Car Monitor System compatible with the vast majority of vehicles on the road. However, because basically all new model vehicles are utilizing the Controller Area Network (CAN) protocol for OBD-II communication, a production model of the Car Monitor System would likely need to be functional with the CAN interface. Nevertheless, the current version of the Car Monitor System would be practical for development and definitely fulfills almost all of specifications for the design.

In its current state, the Car Monitor System requires only a modest budget in order to purchase the necessary components. These will involve purchasing the components listed in Table 2.

Component	Approximate Cost	Production Cost
The eBox Platform	Provided by department	$120
The OBD-II Interface	$87	$80
GPS Receiver	$70	$60
Wireless USB Adapter	On loan	$20
2.5” Laptop Hard Drive	On loan	$20 (flash drive)
Total Cost:	**$157**	**$300**

Table 2: Components of the Car Monitor System with approximate cost.

As clearly illustrated by the table, the Car Monitor System is not overly expensive for the end user. The entire system could likely be put into production for under $300 per unit, and the production cost would undoubtedly decrease as each component is becoming more affordable every year.

Although this system is feasible for production, a much more practical system would involve using an embedded device instead of an eBox. This would drastically reduce the cost of the system, improve reliability, and decrease the overall size of the system. The main drawback to this implementation is that it would involve a relatively high development cost. This would be true due to the incorporation of an OBD-II interface, a GPS chipset, wireless integration, and a file system. Also, the Car Monitor System would be very difficult to modify from its original production state. Therefore, it would be costly to expand functionality in the future. A projected component cost table for a mass-produced embedded system utilizing all five OBD-II protocols is shown in Table 3.

Component	Approximate Production Cost
The Embedded Processor	$50
The OBD-II Interface	$80
GPS Receiver Chipset	$20
Wireless Network Chipset	$10
Flash Memory	$10
Total Cost:	**$170**

Table 2: The embedded Car Monitor System's approximate production cost

5.2 Conclusion

The Car Monitor System would be a valuable addition to many or most vehicles today. Car mechanics would obviously appreciate having a detailed log of all computer signals sent over a certain time period. Also, insurance agents could find certain data, such as where a car is normally driven and at what speeds, valuable. Furthermore, vehicle owners and drivers may find the Car Monitor System helpful in reminding them of routine car maintenance or possibly even find it useful for modeling the best route between home and a workplace. Not everyone wants to know more about their vehicle, but an extraordinary number of people do, especially those that regularly maintain and repair their own vehicles.

This project has fully realized the goals and specifications of the Car Monitor System. Although not completely ready for production, the system would require little modification for a low-scale commercial implementation. Furthermore, the demonstration model would definitely be a useful development base for a large-scale production scheme.

References

[1] Electronic Design. 2006 June 29. New Technologies Make Roads Safer... One Smart Car At A Time. http://www.elecdesign.com/Articles/Index.cfm?AD=1&AD=1&ArticleID=12862. Accessed 2006 Oct 28.

[2] Qualcomm. 2007. QWBS OmniTRACS Mobile Communications System. http://www.qualcomm.com/qwbs/solutions/prodserv/omnitracs.shtml. Accessed 2007 Feb 27.

[3] WindowsForDevices. 2006 June 16. Low-cost platform supports Windows CE contest. http://www.windowsfordevices.com/news/NS2983372021.html. Accessed 2006 Nov 14.

[4] Damn Small Linux. 2007 January 17. DSL information. http://www.damnsmalllinux.org/. Accessed 2007 Mar 28.

[5] Nice, Karim. 2001. "How Car Computers Work." http://electronics.howstuffworks.com/car-computer1.htm. Accessed 2007 Feb 21.

[6] OBD-Codes. OBD-II Trouble Codes. http://www.obd-codes.com/trouble_codes/. Accessed 2007 Mar 27.

[7] AutoTap. 2006. OBD-II Background Information. http://www.obdii.com/connector.html. Accessed 2006 Oct 14.

[8] Wikipedia. 2007 February 8. OBD-II PIDs. http://en.wikipedia.org/wiki/OBD-II_PIDs. Accessed 2007 Mar 16.

[9] OBD Diagnostics. 2007. OBD Mode 1 & Mode 2 PIDs. http://obddiagnostics.com/obdinfo/pids1-2.html. Accessed 2006 Nov 11.

[10] About: Auto Repair. 2003. OBD-II Codes. http://autorepair.about.com/cs/troubleshooting/l/bl_obd_3.htm. Accessed 2007 Apr 12

[11] Multiplex-Engineering. 2007 February. Interfaces. http://www.multiplex-engineering.com/interfaces.htm. Accessed 2006 Nov 11.

[12] ScanTool. 2007. http://www.scantool.net/products/index.php?cPath=8_6. Accessed 2007 Mar 30.

[13] FreeDiag. 2004 March. FreeDiag Vehicle Diagnostic Suite. http://FreeDiag.sourceforge.net/index.html. Accessed 2006 Nov 11.

[14] Brain, Marshall and Harris, Tom. 2000. “How GPS Receivers Work.” http://www.howstuffworks.com/gps.htm. Accessed 2007 Feb 21.

[15] Garmin. 2007. What is WAAS? http://www.garmin.com/aboutGPS/waas.html Accessed 2007 Mar 28.

[16] GPSBabel. 2006 November. GPSBabel Home. http://www.gpsbabel.org/. Accessed 2007 Feb 8.

[17] GPSBabel. 2007 January 17. Realtime tracking. http://www.gpsbabel.org/htmldoc-1.3.3/tracking.html. Accessed 2007 Mar 7.