The Black Box: A Data Acquisition and Telemetry System for the MIT FSAE Electric Race Car

David Gomez

***Abstract* — A critical part of the engineering design process is the collection and analysis of testing data to verify assumptions and find areas for improvement. Unfortunately, test data can sometimes be difficult to collect depending on what is being designed. In the case of something like the MIT FSAE Electric race car this is certainly true. In the past, whatever real data the FSAE team had came purely from bench top tests but the challenge of collecting data from a large fast moving, and electrically noisy vehicle proved challenging. With the Black Box telemetry system this issue will cease to be a problem. The Black Box enables high latency and reliability recording of vehicle data onto an internal SD card. The black box also allows even faster response to incoming data by implementing a system to wirelessly transmit several key data parameters live and also wirelessly transfer logged information to a trackside computer, eliminating the need to take apart electronics enclosures just to get the testing data from one run.**

I. System Overview

Of critical importance for data logging in the FSAE car is the CAN bus. CAN is a robust data transmission protocol that is used extensively in the automotive industry. Every device on the car communicates using CAN meaning that almost every relevant piece of information about the car can be collected by logging the CAN bus data. This is actually one of the primary functions of the Black Box and was one of the main considerations in many aspects of the design. The vehicle's CAN bus operates at a maximum rate of 500 kBd/s which pushed the limits of some past logging attempts that had some slow serial bottlenecks or inefficient SD card writing. For this telemetry system, a Teensy 3.6 was chosen as the MCU partially because of some of it's very favorable CAN and data logging related features. The Teensy 3.6 incorporates two integrated CAN controllers

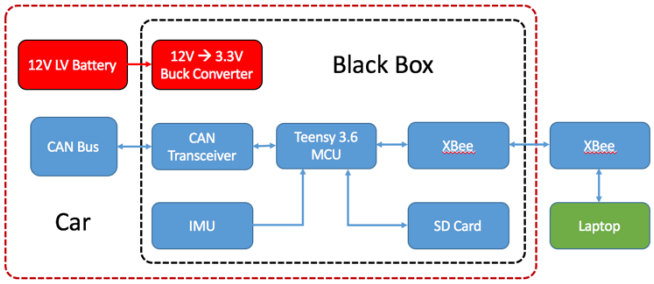

Fig. 1. High level overview of the Telemetry system. The red blocks are power components. The blue blocks are low voltage digital.

which enable it to maintain two independent CAN bus connections with only a single external transceiver, many other microcontrollers require the addition of a transceiver and controller. The Teensy 3.6 also has a built in 4 bit SD interface to connect with an SD card. This interface improves on the standard SPI interface by using four parallel data lines instead of just one. That combines with the blazing 180 MHz processor enables the Teensy 3.6 to reach write speeds exceeding 15 MB/s, which allows the Teensy to easily log every message on the bus and still have overhead for other tasks.

The SD card is the primary data logging device in this system; however, in discussions with team mates it became evident that there was interest in having access to some data in real time. For this reason, a wireless XBee radio was also included. The XBee is essentially a wireless serial cable that enables long range wireless communication between two devices. The latency is somewhat low, only around 100 kBd/s, however, only a few specific data signals and error messages need to be received live so this does not cause any issue.

The XBee in the Black Box communicates with another XBee which is connected to a laptop which will be placed near the track the car is driving on. The laptop is running a custom python program (which will be explained in more detail later) that enables incoming live data to be visualized and also data stored on the SD card to be parsed. The XBee also allows the additional functionality of downloading testing files stored on the SD card wirelessly without having to physically connect the SD card to the computer. This is useful as the FSAE car must be waterproof so all electronics are typically stored in well sealed and covered enclosures which can be a pain to access and open.

II. Hardware Overview

A. *Power*

The only source of power for electronics on the FSAE car comes from a 12 volt battery used to power all the low voltage electronics. As all the digital electronics in the Black Box require 3.3 volts to function, conversion was necessary. Because of the large gap in voltage, a linear regulator would have been unpleasantly inefficient so a buck converter was used instead. The only downside to the buck converter is that it is significantly more complicated to implement requiring more

components and board space as well as careful placement of components. During testing on a proto-board, the buck converter caused issues with the microcontrollers ability to proper reset its self during turn on due to ringing caused by switching on the 3.3 volt. This problem will likely go away once the design is properly implemented on a PCB.

B. *Teensy 3.6*

At the heart of the Black Box lies the Teensy 3.6. The Teensy 3.6 is fantastic for this application as it's dual CAN controllers and integrated SD card reduce the component count needed. It's 32 bit 180 MHz ARM processor has more than enough speed to manage both data logging, processing, and transfer. The Teensy 3.6 also has an integrated real time clock which is useful for creating timestamps to tag the data collected. In figure 2, note that the coin cell battery connected to the Teensy is used to power the real time clock

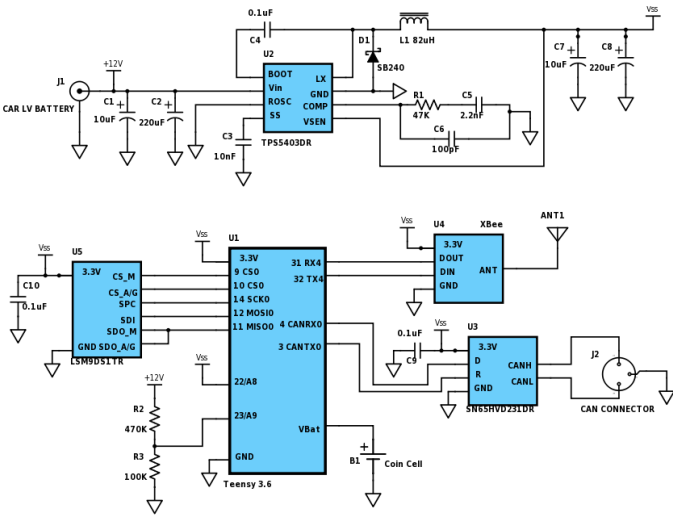

Fig. 2. Schematic of the electrical system.

even when there is no power being applied to the board. This prevents the current time from being lost. Also in figure 2 note the connections on pin 22 and 23. These are both analog inputs used to measure and log both the LV battery battery voltage and 3.3 volt Black Box voltage.

C. *CAN*

As previously stated, the Teensy 3.6 has two dedicated CAN controllers. These devices are responsible for taking raw data and placing it into CAN data frames. They also handle other aspects of CAN communication such as message priorities and masking. Previous microcontrollers used on the FSAE car required a dedicated external CAN controller chip; however, this need is eliminated by the Teensy 3.6. Both systems still require a CAN transceiver though. The CAN transceiver is responsible for converting the CAN data frame from digital logic levels to the differential signal protocol needed for proper CAN communication. The TI SN65HCD231DR was selected for this task as it can operate with out extra external components and is also capable of operating on a 1 mega baud per second CAN bus, allowing for future improvements in performance.

D. *XBee*

The XBee selected for this application is the XBee-Pro S1. This device has a theoretical range of 1 mile and can communicate serial information at a maximum rate of 115,200 kBd/s; however, in practice a rate of 111,111 kBd/s must be used. This is because the XBee only has an internal master clock at 16 MHz and therefore can only generate serial data transmissions at integer multiples of this master clock. 111,111 kBd/s is the closest integer multiple to 115,200 kBd/s that dividing down a 16 MHz can give; however, the much faster clocks in both the Teensy 3.6 and USB-Serial adapter connected to the laptop are capable of communicating at almost exactly 115,200 kBd/s. This mismatch in speed can cause a lot of errors so the best solution is to require the faster devices to match the actual speed of the XBee.

E. *IMU*

The Black Box also incorporates a LSM9DS1TR which is a 9 degree of freedom inertial measurement unit. This is a sensor that incorporates a 3 axis accelerometer, gyroscope, and magnetometer enabling a variety of useful motion based measurements to be made and logged such as lateral acceleration. The LSM9DS1TR communicates with the Teensy over an SPI interface. The sensor has separate outputs for the accelerometer/gyroscope and magnetometer which are selected between via two different chip select pins.

III. Software Overview

The software exists in two main parts, C++ code that runs on the Teensy and Python code that runs on the trackside laptop.

A. *Black Box Software*

The code that runs on the Black Box has one primary purpose: log every CAN message received on the bus. Once messages are received, the Black Box is also responsible for determining if a message should also be instantaneously transmitted wirelessly for live data which it does by comparing the CAN ID of the message to a list of messages that should be sent live. The Black Box is also capable of excetuing commands sent from the trackside laptop. This is primarily used to allow the laptop to signal the Black Box to dump the data from a file over the wireless channel.

The code that runs on the Black Box is intentionally limted in the processing it does on the data. The CAN messages are logged in their raw form, only a timestamp is appended to them. To save space, at the top of each data file the Black Box

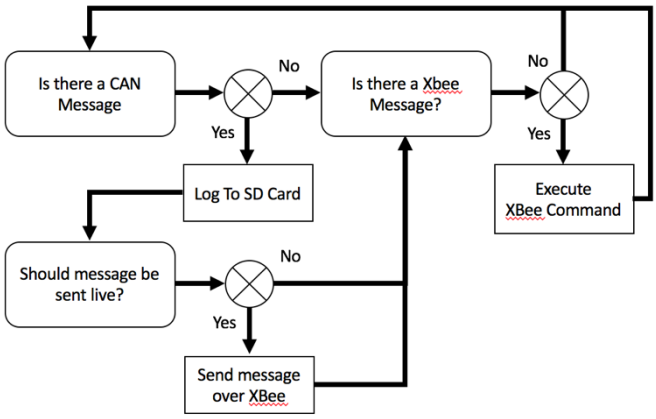

Fig. 3. A flowchart of the Black Box software's main control loop.

makes, a 4-byte number representing the current unix time from the real time clock is saved. Each subsequent CAN message logged begins with 29 bits that represent the time in milliseconds since the unix time at the start of the data file. The following 11 bits are used to store the CAN ID of the message. The next 8 bytes are then reserved for the payload of the CAN message. One CAN message take 13 bytes to log. Even though some messages have data and IDs that could be stored in fewer bits this fixed width encoding is used to simplify the process of parsing the data and to eliminate the need for any special position marking characters.

B. Laptop Software

In order to parse the raw data collected by the Back Box, I have created a GUI interface in Python that allows a user to parse raw data files that are either on an SD card connected to the computer or downloaded from the Black Box wirelessly into CSV files that identify the meaning and value of the data.

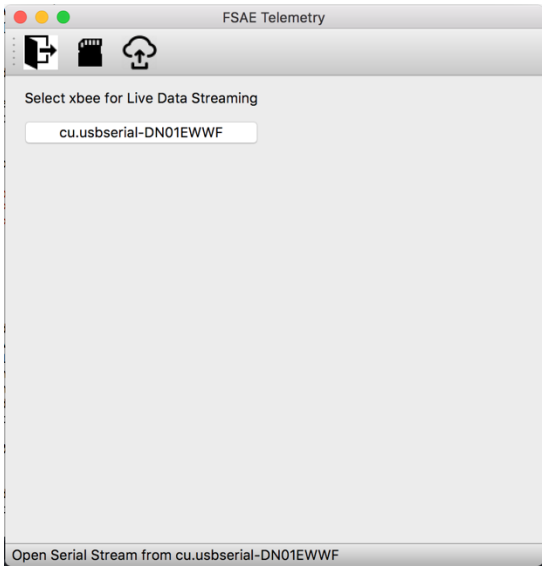

Fig. 4. The main window of the GUI interface. The top bar contains buttons to parse data from either an SD card or to download data from the Black Box. The center area is populated with buttons that open a menu to view live data generated and sent to the laptop.

The software automatically generates a separate CSV file for each CAN message type which enables easy plotting of the data. To select files for parsing, two different methods are used. The SD card button opens up a standard file selection dialog that allows the user to see the contents of the SD card and then select the file they want. When the user selects the download button, the laptop XBee sends a command to the Black Box to

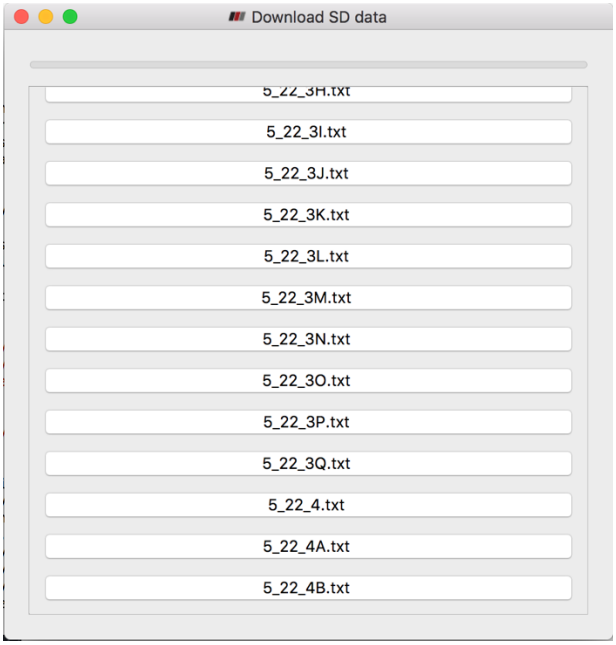

Fig. 5. An example of the file listing from the Black Box. The file names are created in the format MONTH_DAY_HOUR of the time of creation. If a file of the same name already exists, a letter is appended.

return a list of all the file names stored on the device. The file names are then displayed to the user as shown in figure 5. The user can now select a desired data file at which point the Black Box will be triggered to dump the file's data to the laptop XBee. The data is parsed and saved to CSV files in the same way as the data from an SD card is.

The GUI also manages an interface that allows the user to see live data visualizations of whatever data has been selected. Only CAN messages that have been specified in the Black Box code will be sent live. There are also parameters to limit the rate at which some pieces of data are sent as they do not need to be view live at their full temporal resolution.

Separate from the GUI interface but also important to the project as a whole is the CAN spec parsing tool. The FSAE team keeps a text document that outlines all of the messages that should be seen on the CAN bus. The documents also specifies other information such as the endianness, CAN ID, name of the data, position of the data in the payload, and other useful parameters. In an effort to make adding the ability to log new future messages as simple as possible the software has no hardcoded functions to parse out what data is what based on the CAN IDs, instead several python dictionaries are stored in a separate file that enable the software to look up the name and proper way to parse any CAN message received. Because typing out this lengthy dictionary would be a pain

and likely result in errors, another python script was made that is able to parse the FSAE CAN specification text document and automatically generate the necessary python dictionaries. When the fact that the Black Box itself only logs raw CAN messages without parsing them, this means that in order to add and log a new CAN message, the only thing that needs to be done is add the parameters of the message to the FSAE CAN specification text document, a trivial task that even someone with no software background (i.e. the many mechanical engineers on the FSAE team) can do.

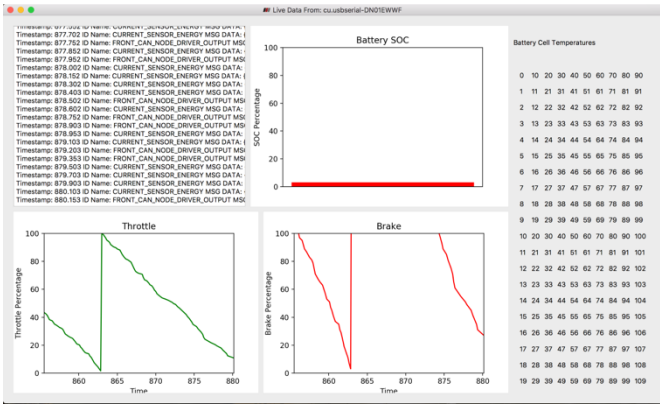

Fig. 6. The live data viewing interface displaying simulated CAN data.

IV. Conclusion

Unfortunately, due to a damaged motor controller preventing the car from running, the Black Box was not able to be tested on the actual car before the frame needed to be sent out for powder coating and this document needed to be turned in. It was, however, connected to the stationary car electronics in the shop and was able to log messages although the data was not very interesting.

The car will be functional again soon though and the Black Box will be ready to log data and be further improved with feedback from the team during real testing.

This telemetry system is a great step in the right direction for the FSAE team and should enable some helpful debugging support for this year's car and intelligent design information for next years' car.

The system also serves as a good base for future team members to build off of. The hardware is more than capable of meeting the needs of the team for years to come and the software is extremely extensible and will be able to be quickly updated to handle unpredicted needs.

As always there is room for improvement though. A PCB layout is an obvious step that I unfortunely did not have time to complete. Higher power and bandwidth wireless radios also exist that could improve upon the XBee. There is also a limitless number of ways the software could be improved to better visualize and parse the data the Black Box produces as well.