A PORTABLE SOLUTION FOR ON-SITE ANALYSIS AND VISUALIZATION OF RACE CAR TELEMETRY DATA

Christopher Backhaus, Kyle Boyer, Safwan Elmadani, Paul Houston, Sean Ruckle

Advisor: Dr. Michael Marcellin

The University of Arizona

ABSTRACT

The University of Arizona Baja Racing Team competes annually in a grueling off-road racing competition designed to test the durability of each team's vehicle. For the last several years, we have been creating and improving upon a telemetry system for the car in order to provide live data and analysis to the driver and pit crew during races, as well as to inform the design of future vehicles. This year, we have created a portable system consisting of a high-performance computer running a custom software package in a ruggedized case with a variety of networking gear. The software is built around a modular, multithreaded analysis engine and can perform live and retrospective analysis on data received from multiple sources, the results of which can be displayed using the built-in GUI or accessed via web interface.

INTRODUCTION

BACKGROUND

Every year, the University of Arizona Baja racing team joins a host of other schools from around the world to compete in a grueling three-day competition consisting of multiple challenges culminating with a four-hour off-road endurance race. Our vehicle is equipped with a telemetry system that transmits raw data like GPS coordinates, velocity, acceleration experienced by suspension members, wheel and gear RPMs, etc. back to our pit crew.

MOTIVATION

In previous years, the team used their personal laptops or a Raspberry Pi to visualize and store telemetry data received during the endurance race. This solution was fragile and took a significant amount of time to set up. It also limited the amount of analysis that could be done in

near real-time. To solve these problems, we designed our own system and custom software with the goal of having an all-in-one solution for collecting, analyzing, and visualizing data that the team can continue to use for years to come.

The system is durable enough to be used year after year at the competitions the team participates in and powerful enough to run demanding analyses in near real-time. The software is written with an emphasis on expandability, meaning that the team can focus on producing and displaying useful data with the software rather than redesigning it.

HARDWARE

To build a rugged system we decided to start with a commercial IP67 case and then mount the hardware that we wanted inside of it. A monitor is mounted behind a protective polycarbonate cover in the lid of the case, minimizing the amount of setup that is required for the system. Networking gear, including a dual-band access point, gigabit router, and 900 MHz radio, is located behind the monitor. The remainder of the hardware is mounted under another polycarbonate faceplate in the lower half of the case with a full water cooling setup to help the high-performance CPU and GPU to run efficiently in hot environments. We chose to use a high-speed solid state drive over a mechanical drive as it allows data to be stored and retrieved quickly and also is far more resistant to mechanical shock. When the case is open, a removable Bluetooth keyboard with a trackpad and a USB hub are accessible. The system also includes an IMU and GPS to detect if the case is open and to provide a reference point for measuring the distance between vehicles and the pit.

In the process of building the system, we learned many things about the decisions we made regarding the hardware and have found a few things we would change if we were to do this again. The first of which is that we chose to use a small form factor desktop case to mount our computer hardware inside the larger case thinking this would make it easier. However, we had to modify the case to such a degree that it would have been easier and more cost efficient to just build mounts from scratch. The other hardware design we would change is the way we mounted the keyboard and other peripherals to the plastic cover in the bottom of the case. We tried both hot glue and epoxy but the surface was both painted and highly flexible, which caused the adhesives to peel off. If we were to build this again, we would replace the polycarbonate with opaque acrylic, and instead of painting it we would use a solvent to chemically weld the pieces together.

Figure 1: Partially Assembled System

SOFTWARE

The purpose of the software is to receive, store, analyze, and display data in an adaptable and modular fashion. The software, which is written in C++, is divided into several core modules: a GUI, an analysis engine, a database connector, and a data interface. All of the modules, except the GUI, are connected in a top-level class, which allows the software to run in a headless mode. This architecture also allows modules to be easily replaced both to upgrade and to alter the functionality of the program.

We started the software design by setting design goals. The core goal was that the system must receive, display, and store data. We expanded our goals by integrating design considerations such as flexibility, modularity, and expandability. To make the system flexible in receiving data, it needed to receive live data over a serial bus or data from a database. The modularity was built on a standardized communication interface between subsystems. The analysis engine needed to be designed in a way that made it easily expandable while keeping the system as near to real-time as possible, while operating in a non-real time operating environment (i.e. Windows 10).

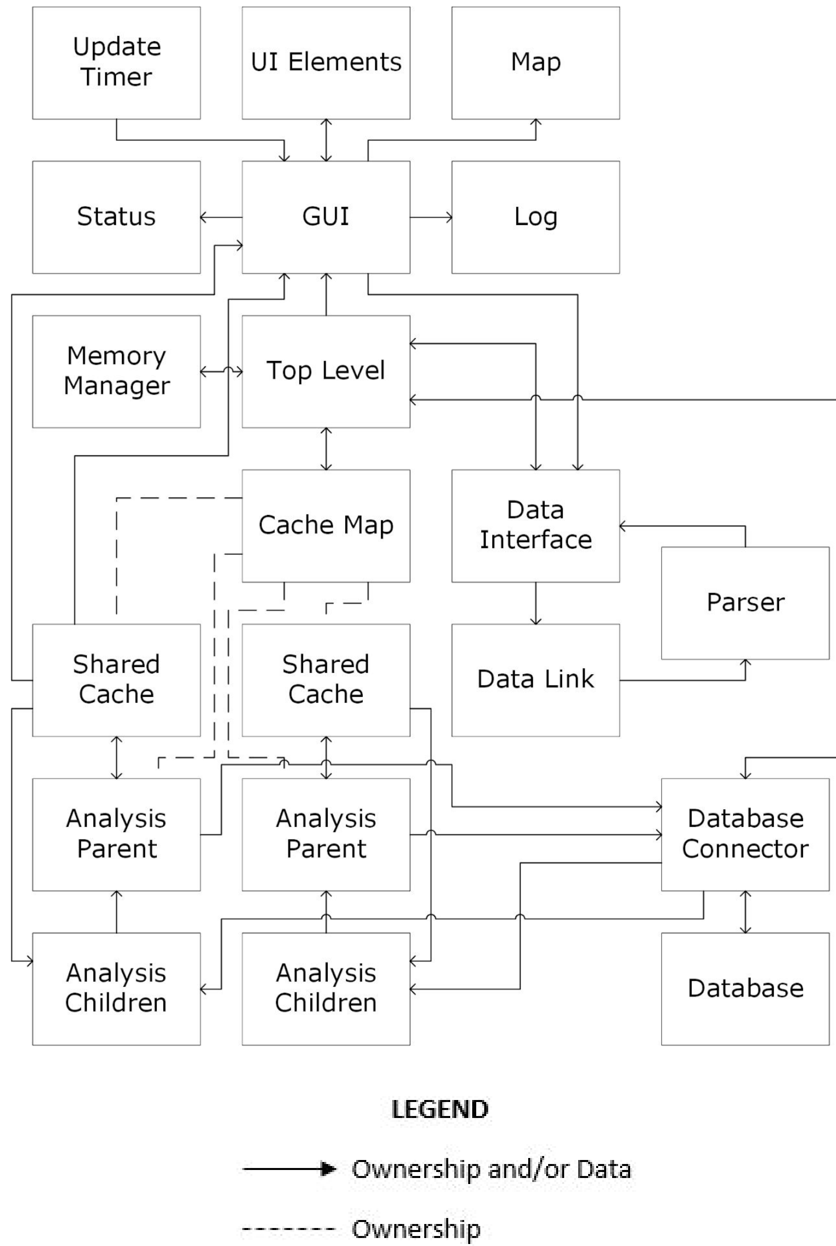

Figure 2: Software Block Diagram.

DATA INTERFACE

Data enters the system as a string of key-value pairs via a serial buffer. The string is checked, parsed, and converted into a custom data storage object. The data interface and data storage objects support arbitrary keys, making adding new fields to transmissions easier. The data interface also handles sending telemetry commands back to the vehicle.

One problem we faced with this in the past is that occasionally we would receive half a transmission, or two mixed together. This caused issues with the program trying to parse

incomplete data or duplicate keys. In order to mitigate this, we added two requirements to the data transmission structure: We require that the first key be a sequence number which must be increasing between messages and that the second key be the total number of keys contained in the transmission. These two requirements allow us to check if a transmission is garbled in transit and ignore it if so.

DATABASE CONNECTOR

The software interfaces with a MySQL database to store all received and analyzed data. This provides a convenient, secure, and expandable way to store data. Data can be simultaneously retrieved if it is needed for analysis and playback. We organized the database so that each vehicle has its own table. New tables are dynamically created each time a new vehicle ID is encountered. Data is stored in each table using date and time as primary keys with the requirement that multiple data points can not have the same timestamp in a table. We did this because early on in testing we were using a looping set of data and when we added playback to the system it caused major problems if repeating timestamps were encountered. The use of a database will allow data be accessible through a web interface for future software expansion.

SHARED CACHE

After being parsed and stored in a database, data is moved to a segment of shared memory that is safely accessible by multiple threads, or a shared cache. This allows the GUI and analysis engine to use the same data concurrently with minimal copying. It also reduces the delay between data entering the system and being ready to display. One cache is maintained for each active vehicle. The cache map module provides a mapping between vehicle IDs and caches.

The shared cache was one of the most challenging software modules to implement due to a lack of experience with C++ synchronization tools. We encountered frustrating ownership issues with the standard library's *shared_lock* and *shared_mutex* types that we used to implement the shared cache. Careful reading and interpretation of the documentation eventually led us to the solution.

ANALYSIS ENGINE

The analysis engine dynamically spawns threads to handle all configured analyses for each vehicle. For each vehicle, the analysis is performed in stages based on dependence on prior analyses. At each stage, multiple analysis threads process the data in parallel and pass the changes as transactional updates back to a parent object that aggregates the changes and applies them to a copy of the original data. When the updated copy is ready, a pointer swap is performed to update the shared cache, thereby reducing the time during which the shared cache must be locked. By keeping this gap as short as possible, we are able to reduce the time between stages and make data available to the GUI before all analyses are done, without the risk of race conditions.

Adding a new analysis is accomplished by deriving a class, overwriting one function to perform the analysis, and enabling it in a configuration file. We have written analyses to convert data from raw integer values to proper values, calculate the distance between the system and vehicles, and detect simple anomalies. Anomalies are defined on a case-by-case basis, such as brake pressure decreasing over time or the car being upside down. While we did not focus on writing analyses, we hope that future team members will use the software as a platform to implement more complex filters and analyses.

GUI

The GUI contains a set of movable, scalable UI elements including a map that displays the position and heading of each vehicle, status widgets for each car, a playback manager, and graphs that display data of the user's choice. One of the main hurdles we faced while creating the GUI was eliminating flicker. Flicker is generally caused by the screen being erased every time something is drawn to the screen. For instance, when displaying the GPS location of the car on a map, every time the position changed the entire map would be erased and redrawn. To remove flicker we found it best to suppress those erase operations and to double buffer our GUI elements.

One of the other problems we ran into was performance. The overall goal of this system was to display near real-time data to users and because of that performance was a constant concern. The way our analysis engine works and we receive data means that we won't always get new data to display for every type of data. Rather than fight the code, we take advantage of this fact and only re-render the parts of the GUI that have new data to display.

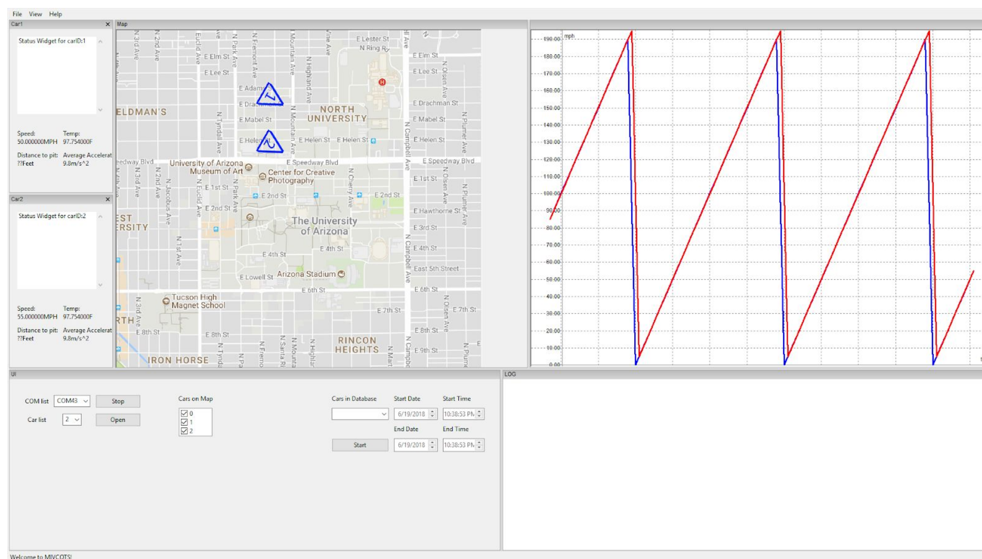

Figure 3: GUI with Simulated Data

LESSONS LEARNED

As with any project, there are some things we would do differently if we were to go back to the beginning with the benefit of hindsight. We learned early on the benefits of CAD modeling but learned much later the benefits of verifying those models with actual measurements prior to machining parts. The saying “measure twice, cut once” comes to mind. We also learned that an on-off switch is extremely valuable on RF transceivers, in case one wishes to turn on the computer without having to attach several antennas to the outside of the box. In the event that a coaxial cable is stepped on and pulled out of the corresponding antenna, it is also a good idea to have regular inspections to go along with the on-off switch, so that one can use said switch before the reflections off of the mismatched line burn out the transmitter. Luckily, we also learned (and reaped) the benefits of planning for the worst-case scenario, and had a backup copy of the telemetry data stored to an onboard SD card on the vehicle, in case the wireless link had any difficulties. Thus, we were able to recover nearly all of the race data and successfully import it into our software through a virtual null modem cable to simulate the data stream originally sent by the car.

CONCLUSION

The system is now ready for use by the University of Arizona's Baja Racing team. This was an ambitious project and we learned multitudes of lessons about planning, designing, building, and implementing a project with both complex hardware and software considerations. Many aspects of the design could have been executed better. However, we feel that the system is both useful in its current state and sufficiently futureproof to remain so for some time yet. The hardware is durable and powerful and the software is expandable and modular so both can be used for years before an update or redesign will be necessary.