**ECE 3220 Final Project – Spring 2017**

**Max Houck and Marshall Lindsay**

**FSAE Tire Temperature Visualization**

**Abstract**

We have developed a program for the Mizzou Racing Formula SAE team to allow them to visualize their logged tire temperature data, allowing it to be analyzed by a human eye much easier. A simple graphics package and several data filtering and smoothing algorithms were implemented to allow viewing both logged data and real time telemetry. The objective of this project was to develop a base platform within the scope of this course that may be expanded upon in the future with more advanced features.

**Introduction**

Formula SAE is a collegiate engineering design project in which students design and manufacture a quarter scale formula style racecar to compete in both static and dynamic events. A very important part of the design is using data to validate engineering decisions. The car has already been equipped with tire temperature sensors, using three infrared sensors per wheel to measure the temperature gradient across different points of the tire. This data is extremely useful, although it can be difficult to make sense of when viewed as raw numbers or in line graphs. The goal of this project is to create a base for a simple, extendable interface that can be used to process and visualize this data in order to make it easier to understand.



*Mizzou Racing’s 2016 Competition Car*

**Background**

Data acquisition is a very important part of engineering. It allows us to validate our previous designs and also improve new designs. Tire temperature data in particular is important to the suspension design of the vehicle, as it shows how efficiently the tread of the tire is being used. Temperatures should be even across the entire tread, and not too hot or cold on one particular edge of the tire. An existing system has already been designed using AVR microcontrollers and infrared temperature sensors to measure the temperature of 3 points across the inside, middle, and outside of the tire tread. These sensors are fitted to the car and then the data is recorded using the cars datalogging system. Once the data has been acquired, analysis must be performed in able to interpret it and make changes based upon the results.



*Formula 1 Tire Temperature Cameras*

Although there are a couple high end racecar data analysis software packages already in existence, these are extremely expensive often not very customizable. By developing our own platform for visualizing tire temperature data, it will allow us to meet the needs of the team and expand upon our team in the future, if we want to add more data channels to visualize along with the tire temperatures. The program will also be extendable, so if for example a more advanced temperature sensor that reads more than 3 points on the tire is used, the program can be easily modified to support this. This would give an advantage over purchasing a software package in that we can customize this to fit our changing needs.

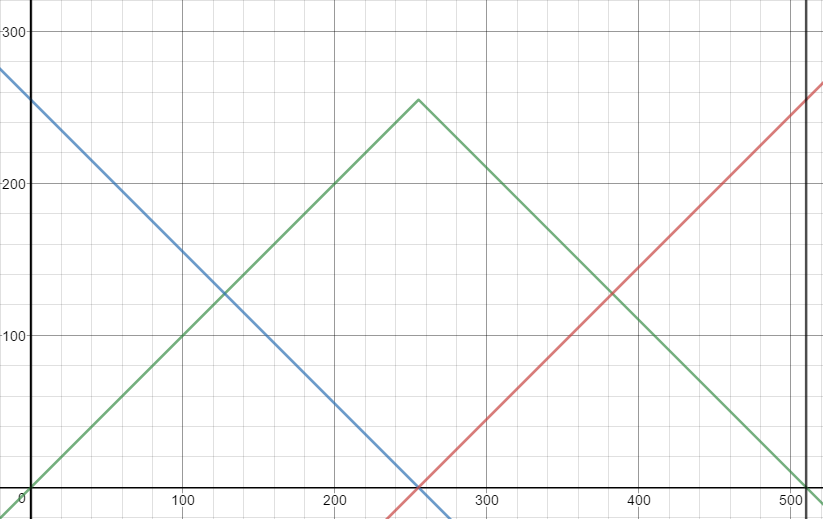
**Implementation**

The goals of the project were to create a simple visualization package, filter and smooth incoming data, and start developing a real-time telemetry viewing system. The focus was on elements of the project that were within the scope of this course in C++. We started by creating a UML diagram of the code, which is shown below. This UML diagram changed and grew as the code was written and the project was restructured, but the diagram helped to keep us organized and keep the bigger picture in mind.

*Uml diagram*

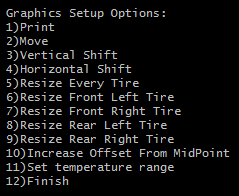
The car and tire classes were nested within each other, not utilizing inheritance but rather the program would create a car which contained four tire objects, which each contained three sections of data. This kept all of the data easily organized and accessible. By allowing the user to define the names of the files where data is stored just once and loading it in, any other methods we implement can simply grab the data they need from the car object that was created.

A simple command line graphics library, windows.h was utilized to create the graphics for the program. Not a lot of time was spent on this, as it was not part of the scope of this course, but it was intended to be a base foundation on which the program may be expanded in the future. The data processing and analysis algorithms may be reused with a more extensive graphics application, perhaps featuring an in-car video or other data plots. The color gradients for the graphics were calculated using linear formulas to modify the RGB values of the colors displayed. By sweeping the colors up and down, full values of red, green, blue, and all colors in between are achieved and displayed nicely to mesh together. A simple linear interpolation between the 3 data points on each tire was used to calculate the gradient to be displayed in between them. This logic was used to view both the logged data and the real time input.



*510 RGB Gradient*

A menu system was implemented in order to allow the user to change the way the graphics looked. When running the program on different devices with varying screen sizes and terminal windows, there wasn’t one setting that worked everywhere. We implemented a simple menu system that allowed the user to change variables that affected the way the graphics were displayed.



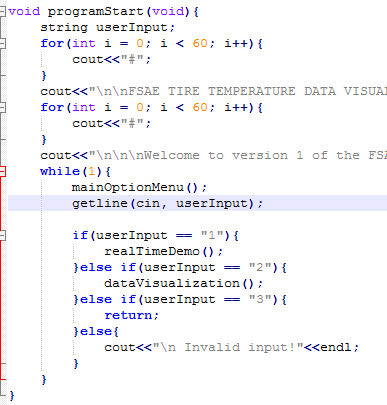
*Graphics settings menu*

The real time interface was constructed as a basis for a more advanced system that may be implemented in the future. This system will eventually need to consist of a full wireless system to interact with the car as it drives, but for now we focused on taking the data, filtering it, and displaying it. It was very simple to set up an Arduino using existing serial libraries and I2C code to simulate this data for our purposes. Algorithms were developed to filter out ‘bad’ data. One of these strategies involved creating a simple, resizable buffer to compare incoming data against previous values. If there was a huge jump, this would be caught as erroneous data and the filter would smooth it out a bit. However, if the huge jump continued for more iterations, the algorithm would eventually allow it to ‘pass through’ the filter and be displayed on the graphics. We began developing another method to smooth the data, comparing residuals to a best fit line and smoothing based on these results. One option that may be implemented in the future is allowing the user to define what type and how aggressive they want their filtering to be based on the incoming data.

*Picture of Arduino and sensor setup*

**Experiments and Results**

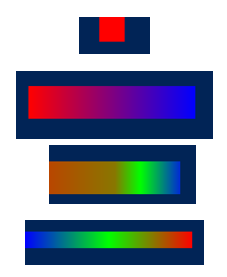
We ran experiments with many different error cases and data inputs. Exception handling was implemented heavily throughout the program to catch all different error cases and handle them accordingly. Erroneous user inputs were caught and corrected using very simple try catch blocks throughout the program, as seen in the example below. This made the program more user friendly and robust. All possible different user inputs were tested, and were handled without flaw by the exception handling blocks.



*Handling errors in user input with try-catch blocks*

In addition to user input, we also ran many different trials with the input data. Error handling was implemented for these files and many different cases with invalid formats, strange numbers, and different filename issues. These cases were handled at the appropriate point in the code and the program responded correctly.

The graphics were experimented with a lot, spending a lot of time making sure they would look right in all situations. Shown below are a couple examples of different gradient techniques we tried using as the program was developed. In the end we decided to use a “510 Gradient” which consists of 510 colors as the red, green, and blue values are varied in succession.



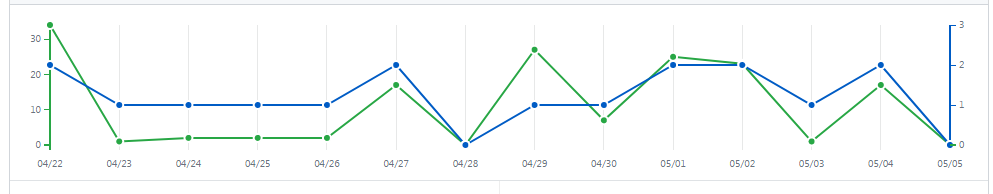
*Iterations of color gradients*

We also spent some time making sure the graphics could be displayed on different systems. This led to us creating the menu system to resize them, so no matter the size of your terminal window or screen resolution the graphics blocks could be resized to fit the users needs.

**Discussions and Conclusions**

As with most projects, the layout we had in mind changed several times as we developed the project. The UML diagrams helped us to stay on the same page as to what we were implementing. It is always a challenge to envision the best way to organize your program before you have it written, and although the diagrams helped with this we still ended up a little disorganized in the end. With bigger projects we may encounter in the real world these will be very important as the scope of the project and the amount of people involved grow.

Github was used extensively throughout the development of the project. This was both of our group members first time using GitHub as a collaboration tool, and it was an excellent experience. The use of frequent commits allowed us to roll back the code in case a bug was introduced or we needed to undo some changes. The logs and diff tools also allowed both of us to see what the other was working on, where they added, deleted, and changed code, and what still needed to be done. Github was also a good tool for keeping the project synced between multiple computers and making sure we were always working with the most up-to-date version of the project.



*Github activity*

Overall, we were happy with the way our program progressed. It is a very good platform for future development that the Formula SAE team may work on in the future. A few really cool things can still be implemented. One of these could be pulling data directly from a database. This would make it a lot easier to handle the vast amount of data that we acquire every time we test the car. Some of the other parameters that are logged may also be displayed alongside the tire temperature charts. Another future development could be real time telemetry. We have developed a solid base to process and display the incoming data, and some sort of wireless system would need to be developed to feed the data into our program.

This project was a very good exercise in real C++ development. We encountered many new concepts and challenges along the way, and had to work them out to get the project done. We successfully used GitHub to stay organized and work collaboratively. We are also very proud that we created something with real motivation behind it that will definitely be utilized in the future and most likely continue to be developed.