ECE 399 Report

Spring Semester 2024

4/20/24

Summit Koegel

**Project Description/Statement**

I did data analysis with the controls team on Ecar and Ccar for all the various sensors on the car. I mapped this data and created visualizations out of it.

**Design Specifications**

The original intentions for this project were to write a few scripts in Python that would take data from sensors on Ecar and Ccar and create visualizations as needed from this data. You would have been able to give the system a dataset in a CSV or MDF format and it would in turn give you a graph of the data you were attempting to map. The system should be simple enough to use by any teammate without a background in computer science but with general knowledge of the sensors and signals on the car.

**Summary**

This project aimed to create user-friendly graphical representations of sensor data from the cars. The project took a knowledge of the programming language Python and Pandas data frames and we were looking for a way for the team to input a dataset and have the program output some means of software that could plot the signals they were looking for.

A screenshot of a computer

Description automatically generatedWith the help of Surya, we wrote what we called a data converter for the signals. The purpose of the data converter was to create a mapping between the signal names recognized in the DBC files and signal names we knew the team would be able to understand. A picture of the data converter is shown in Figure 1. The next step was to create a Python script that could read testing data(in the form of a CSV or MDF file) and output a data frame that only had the signals from the data converter we wanted. I discovered a small roadblock here when testing the script because I was given data that didn’t have the signals I was testing for. This led to me believe that my script was wrong when, in fact, I just needed to try a more up-to-date MDF file. Once the parsing script was completed, my teammate, Akash Deepak, joined in helping me with creating a piece of software that could visualize the data. Before that, we would be able to calculate some summary statistics from recent days of testing. This meant taking the data and normalizing the

**Figure 1. Image of data converter with columns for Category, Signals, Nomenclature, Units, and Signal Priority**

time scale so that we would see sensor readings at a timestamp of the least common denominator of time-reading intervals. After that, I was able to calculate whatever statistics were needed: amount of time the engine was above 1000 RPM, duration of time in HV, average speed, etc.It is difficult to automate this process because depending on the summary statistics needed you might have to manipulate the data in a slightly different way.

In creating the visualizations, we wanted to find a way to automate the process so that anyone who wanted to plot some signals could just enter their specifications into our software and give it a dataset and it would generate the visualization they were looking for. We started by using a library for plotting called Matplotlib which is widely used and makes clean fast running graphs. The problem was that if we wanted to make the application user-friendly then we would have to teach people how to input data into the code and select the columns they were looking to plot. We decided on using a software called Plotly which has a built-in interface allowing you to host a webpage that contains your visualization software. This would allow us to create user-interface components like buttons, drag-and-drop menus, and checkboxes in an HTML/JavaScript adjacent language. This made the software far more visually appealing and allowed us to host it on a server if need be. In the meantime, anyone could run the application and it could be hosted on their local machine. This began a long and arduous process of writing code to combine the two frameworks, tweaking the specifications of the layout to make it function and look aesthetically pleasing, and debugging incorrect attempts.

Plotly does not always integrate seamlessly with other frameworks or programming languages, and it was sometimes difficult to pass data from one function to another without hindering the flow of the interactivity of the webpage. We often had to find ways to work around these limitations by creating buffers between server calls or re-populating data frames continuously throughout the code. We decided, at the minimum, that the team would need to specify what car they were working on, what dataset they wanted to graph, and what signals needed to be graphed. It was important to us to create a graph that could auto-generate axes titles and their corresponding scaled axes’ values. An image of the graph and the interactivity of the software can be found in Figures 2 and 3. Plotly’s framework helped in making it easy to create graph manipulation/resizing tools that the team could use to zoom in or cut out certain pieces of data. We added this functionality so the team could zoom in on any piece of data and see the exact X and Y values. Lastly, we wanted the interface to include the data that was being plotted in case someone wanted to see a pattern of data instead of just a few points at a time. An image of this table can be found in Figure 4. The functionality of the graph meets the standard of the intention set for this project, but we quickly learned that not all data that we are graphing is clean and linear. It would be helpful in the future to add the ability to create different types of graphs such as scatter plots so you could more easily plot wider, more spread-out data or data that fluctuates in both axes quite frequently.

Our hope is that this project can help with various testing endeavors and shine a light on issues that the team is having with sensors or other components of the cars. The software should be able to aid in making improvements a more simple and efficient process. We will continue to refine the project as needed in order to make it as user-friendly and comprehensive as we can.

A screenshot of a computer

Description automatically generated

**Figure 2. Filters for the user to select the specifications of the dataset and signal names they want to plot**

**A graph showing a line

Description automatically generated**

**Figure 3. A graph showing Engine RPM over time and expanding on a specific point**

**A screenshot of a computer

Description automatically generated**

**Figure 4. A data table showing the parsed and refactored dataset that was fed into the program with pagination for easy traversing**

**References**

1. “Mdfreader.” PyPI, pypi.org/project/mdfreader/. Accessed 29 January 2024.
2. “Plotly.” Plotly Python Graphing Library, plotly.com/python/. Accessed 12 March 2024.