*Note: The resultant pdf is located in the same file as the rest of the project and is called *out.pdf*.

For our data science final project, we decided to make an interface for the MoTec software for the FSAE team in order to make the process of interpreting data more streamlined for someone who is not unfamiliar with the software. Our code carries out the calculations for certain important features of the car (roll gradient, lateral load transfer, etc.) and outputs it to a PDF.

To begin, we start with preprocessing the data. The MoTec software outputs testing files into a csv which makes it relatively easy to read in the csv into a Pandas dataframe using the Pandas library's *read_csv* function. There are many things to clean up about this file. First, we have to select our sampling rate. In the end, we went with 200 Hz at the cost of additional data points since it exhibited much less noise in the dataset than using 1000 Hz (the default MoTec sampling rate). Then, we have to separate the file into three different parts: the notes included at the top of the file, the data in the file, and then the units and headers for the various sensors. Then, the actual data goes through more processing. It is run through a function that removes all of the quotes on the data and casts it as the correct type of data (float, or decimal). We then take this opportunity to keep track of all the sensors that fail to read a changing value. These sensors are added into a list and outputted to the final pdf with a warning that they may be broken. These sensors are then removed from the data set to avoid wasting memory and optimize processing times. The last step in my pre-processing of this dataset includes calibrating some of the sensors. For example, I set the damper positions to 0 at the start when the car's wheel speed sensors read 0. This allows us to perform our computations based off of the lin pot's initial reading which is not perfectly calibrated.

The first of my computations includes calculating **damper velocities**. To do this, I use the *gradient* function from the numpy library which calculates the difference between different steps of a Series over a known step value. Then, I use the Seaborn and matplotlib libraries to plot the damper velocity histograms. I also coded a **gating** function which will return a subset of the argument dataset that has absolute values above a given parameter in a given column. This helps filter out noise, and when we plot the damper velocity histograms with both the gated dataset and the non-gated dataset there is noticeable difference in the spread of the data.

The next major computation is calculating the **roll gradient**. First, we calculate the wheel travels for all four wheels by taking the respective damper positions over the spring installation ratio at each point in the dataset. Once we have wheel travel, we can find front and rear roll degrees by taking the arctan of the difference between the left and the right wheel over the track width in the front and the rear. Then, we can find the roll gradient by dividing that value over Ay or our G Force Lateral. While this gives us an instantaneous value of a roll gradient at each point in time, we want more of an overall characteristic of the car. Therefore, we use the statsmodels.api library for its *ols* function. This effectively returns the coefficient of an ordinary least squares regression which is analogous to the slope of a linear regression. We can also use the gating function in this scenario in order to make sure only significant data points are considered in the regression.

The last two computations are for the **lateral load transfer coefficients** and the **GG plot**. In the case of the LLT, I used the equation $LLT = \frac{(CG\ height)Ay}{Average\ track\ width}$. Using this, I was able to plot a graphical representation of the LLT during a skidpad run the team carried out in the Fall. The coefficients I received from the LLT were actually able to validate some empirical observations we noted. For example, we noticed that during a skidpad run, one of the wheels would lift off the ground. The maximum LLT coefficient I calculated was 0.508 which does indicate a wheel lifting while the minimum LLT coefficient was -0.491 which shows that the other wheel does not lift during the run, which confirms what the team noticed during testing. The coefficients do have slight errors (for example I believe that the coefficient should not be above .5), but this is most likely a result of noise in the dataset/sensor calibration. In the future, we will do more filtering to get more accurate results as well as look into more professional sensor calibration. For the GG plot, I simply plotted Ay vs Ax and produced a simple matplotlib graph.

After completing the computations, the part that ended up proving the most difficult was actually **exporting all of our data and charts to a PDF.** To do this, I wrote several functions that translated the code to HTML, this would allow me to encode the images and stylize the text. For example, I wrote a function that translated the graphs which were png images into base64 bytes and returned that as a URI in an img tag which would allow the graph image to be preserved within an HTML string. To do this, I saved the graphs to an io.BytesIO() object which serves as an in-memory buffer. I then moved back in the memory and read all the bytes out to a string, which encoded the image. I also encoded all of the outputs of the computation into other HTML strings and then pieced those together at the very end to form a proper, full HTML page. Then, using the library pdfkit, I am able to read the HTML string into a pdf in the same directory as my ipynb file. To carry out all of this, I had to learn HTML. I also tried out 3 different libraries to figure out how to encode the images and export them to a PDF.

In the future, there are a lot of changes and additions to be made in order to further streamline the code and make it run faster. For example, I had to use 10 different libraries to carry out the relatively straightforward functions of this code. I would like to simplify the number of libraries which would greatly increase the portability of the code as well as improve runtime. Furthermore, I would like to implement a class structure. Right now, this code is simply the underlying layer that interacts with the MoTec csv. It outputs a pdf for every csv that is inputted, but does not save variables in between each csv. Therefore, it cannot perform macro run-to-run comparisons. If it exists within an hierarchical structure, each instance of this code can maintain certain statistics such as maximum LLT and roll gradient which it can compare to other instances. I'd also like to make this code more flexible by allowing the user to input the path to the csv that it would like uploaded, which the code would then pull from wherever it is located on the computer and run computations upon that dataset. Lastly, I would like to expand the functionality of the code and perhaps look at more metrics. For example, the computations I performed are very heavy on the suspension side. If there is some analysis I can do for other systems such as the engine and etc., I can make different "modes" which will output a PDF with different metrics upon it depending on what the user inputs.

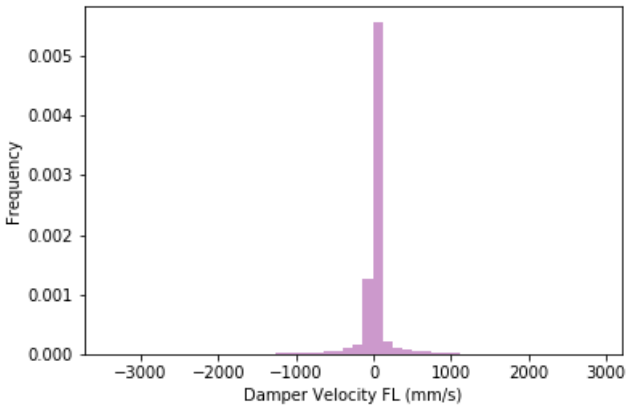
KMR 18-Mike-10/5/2018

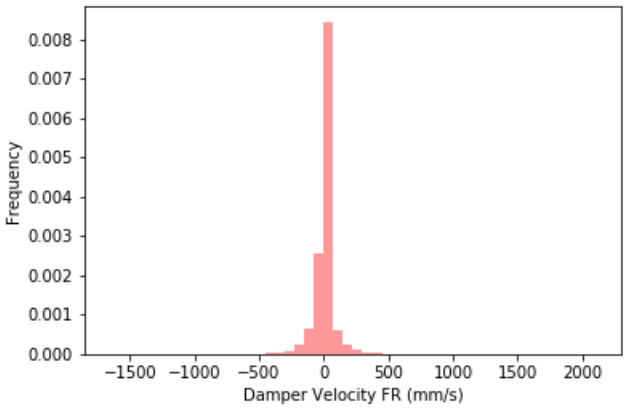
Venue: nan
Device: C185
Comments: rear rebound 2, all others 14
Log Time: 4:13:38 PM
Sample Rate: 200.000 Hz
Duration: 223.482 s
Range: Lap 7 (selection)
Beacon Markers: 65.207 71.098 76.751 83.118 88.839 102.891 108.582 114.129 119.941 125.549 139.056 144.756 150.505 156.655 162.439 175.366
Session: 3rd run
Origin Time: 1113.172 s
Start Time: 1113.172 s
End Time: 1336.654 s
Start Distance: 1 m
End Distance: 1200 m

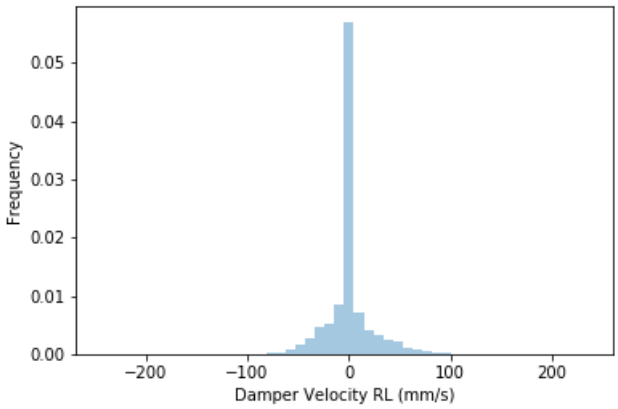
Warning: Possible Broken Sensors
Engine RPM, Throttle Pos, Manifold Pres, Air Temp Inlet, Engine Temp, Ground Speed, Lap Distance, Ign Advance, Lap Gain/Loss Running, Trip Distance, Lap Time Predicted, GPS Sats Used, GPS Altitude, GPS Latitude, GPS Longitude, GPS Heading, GPS Time, GPS Date, GPS Speed, Reference Lap Time, Drive Speed, Comms RS232-2 Diag, Wheel Speed FL, Engine Oil Temperature, Vehicle Speed, Corrected Speed, Brake Pad Temp FR

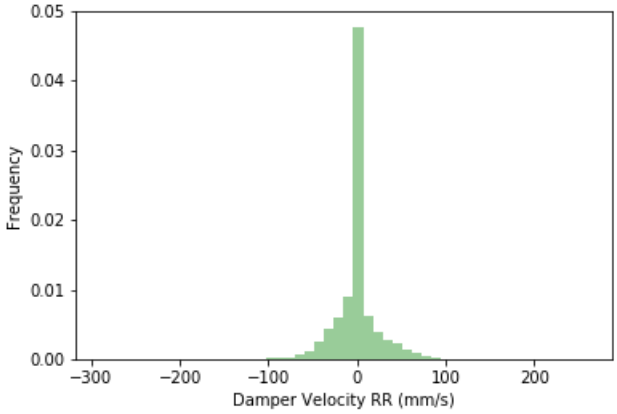
Damper Velocity Histograms

Non-gated

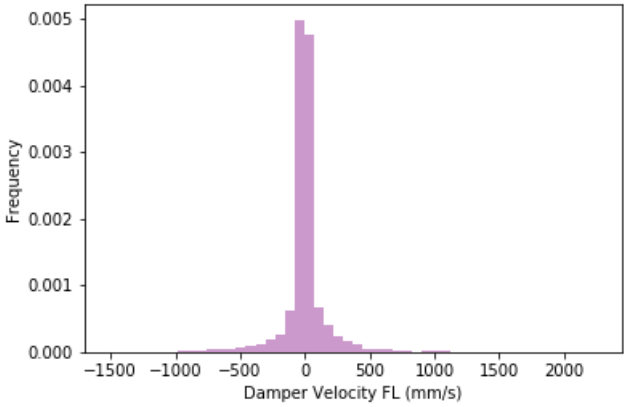


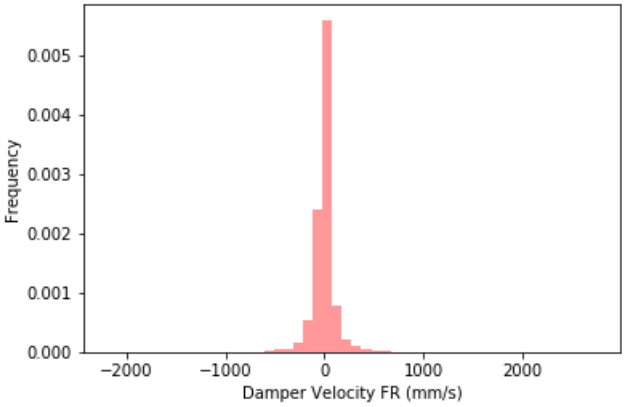


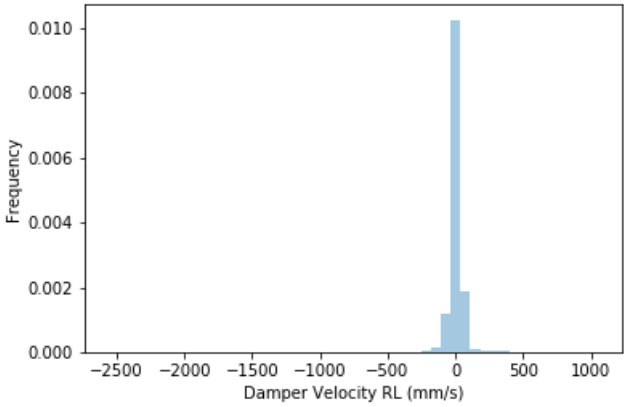


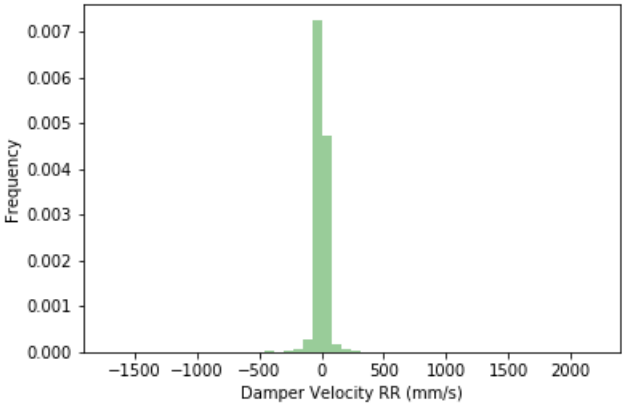


Gated at 1G









Roll Gradient

Non-gated

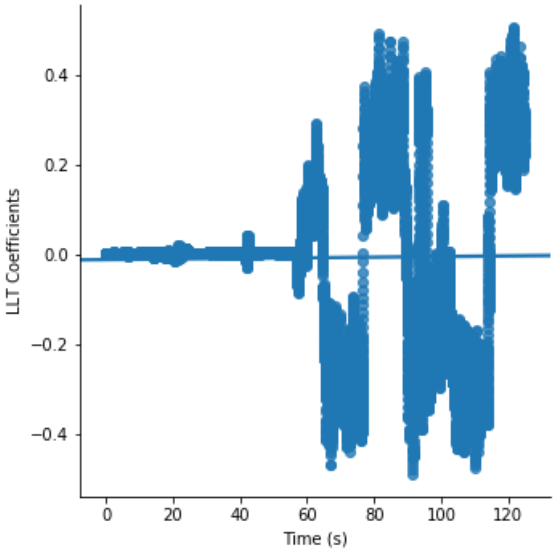
Front roll gradient: 0.7572052346332536
Rear roll gradient: 1.2551093276213101
Average roll gradient: 1.0061572811272819

Gated at 1G

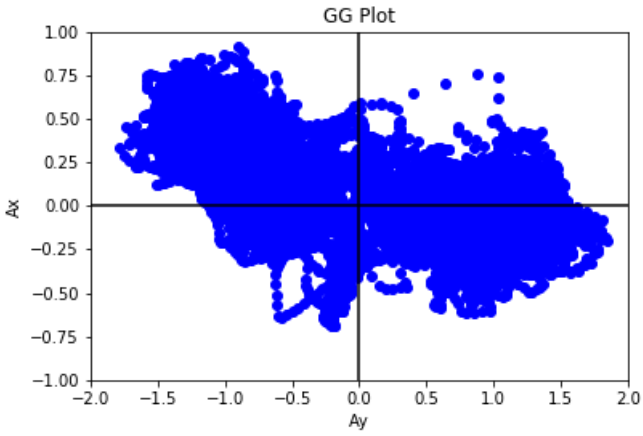
Gated front roll gradient: 0.7153134713343097
Gated rear roll gradient: 1.1912591935688923
Gated average roll gradient: 0.953286332451601

Lateral Load Transfer

Maximum Lateral Load Transfer: 0.5077659574468085
Minimum Lateral Load Transfer: -0.4912978723404255


LLT Coefficients
Time (s)
0.4
0.2
0.0
−0.2
−0.4
0
20
40
60
80
100
120

GG Plot


GG Plot
Ax
Ay
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0