Modelling of an Electrical-Mechanical Damper System

ENGG 4220

Interdisciplinary Mechanical Engineering Design

Winter 2019

Hari Simha PhD, PEng

Lindsay Burton - 0882880

Orion Miller - 0939412

Ben Tory-Pratt - 0897555

Austin Van Rossum - 0710438

Table of Contents

1- Introduction & Background

1.1 - Introduction

The following report details the design, methods of testing and results of a non-linear dampener testing apparatus. This testing apparatus was inherited from the work of a previous student group and it has been refined and improved through this project. The apparatus as seen consists of a dampener connected to a velocity input. The velocity input consists of a scotch-yoke mechanism driven by a 12 Volt motor. The scotch-yoke mechanism provides a sinusoidal velocity input which will give the varying velocity input that is required to test the responsiveness of the non-linear dampening mechanism. The motor and sensors are controlled through a motor controller and Arduinos, while the data collected through the sensors is catalogued real-time through computers connected to the Arduinos.

The changes that were made from the original apparatus consist of a new base, new strain gauge and some refinements to existing components. The new base was manufactured so as to provide a sound structure for the upright tubing to connect into. The base will also provide a foundation to absorb vibration and prevent any secondary movement caused by the operation of the motor. The new strain gauge was selected as it provided a more suitable weight range and thus would provide more accurate measurements than the strain gauge used by the previous group. Other various components were adjusted for ease of usability and accommodation of new sensors and configurations. Most notably, a plexiglass box was added to house the wiring, Arduinos, LEDs, motor controller and switches.

The apparatus incorporates two main physics concepts; the electrical control system and the mechanical inputs to the dampener. The electrical system directly controls the speed of the motor which in turn influences the velocity inputs to the dampener. The sensors then measure the force reactions and displacement. The mechanical and electrical systems are inseparable as they work together to simulate various velocity conditions that the dampener may experience when used in practical applications.

1.2 – Background Theory and Concept

The Ohlins TTX25 dampers being tested are used for shock absorption in FSAE race cars. The damping of the shock plays a significant role in tuning the response of the vehicle to expected conditions. Shown below is a diagram of a quarter car model, the standard way of modeling a suspension system.

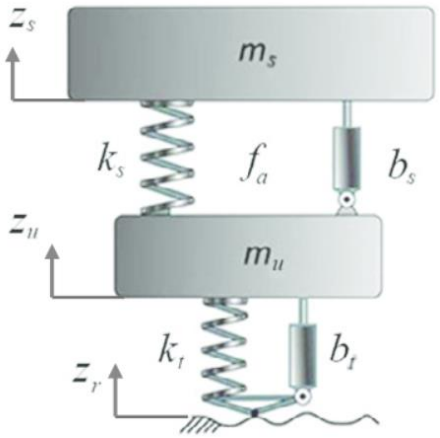


Figure 1: Quarter Car Suspension Model

The two masses are the sprung and unsprung masses. The sprung mass is one quarter of all the vehicle mass which is *supported* by the suspension, i.e. the chassis, engine, driver, whereas the unsprung mass includes the mass *controlled* by the suspension, i.e. the wheel, upright, and hub. Components which connect these two masses, such as control arms, suspension linkages, and driveshafts can be treated as being partly sprung and partly unsprung. The most accurate method for doing this is to look at the center of mass of one of these components and see how much displacement it has relative to the wheel assembly during suspension travel. For example, if during an upward bump motion, the wheel has a displacement of 25 mm and the center of mass of an a-arm moves 15 mm upward, that component contributes 60 % of its mass to the unsprung mass, and the remainder to sprung mass.

The spring and damper with the “t” subscript are the stiffness and damping associated with the tire. These are dependent on factors like the tire construction and pressure. The spring and damper with the “s” subscript are the stiffness and damping of the shock. These are typically termed as the “wheel rate” and “wheel damping” respectively, as the amount of resistance provided at the wheel against normal forces in the contact patch is dependent not only on the k and b (or c) coefficients of the shock but also the motion ratio. The motion ratio quantifies the amount of linear displacement that occurs in the damper for a given displacement of the wheel; this is determined by the suspension geometry and varies throughout the range of travel.

The damping coefficient of the shock absorber in this system is the main variable of interest for this project. Through conducting force vs velocity tests, some light can be shed on how to appropriately set up the dampers for given race situations. In general, the design of dampers for passenger vehicles is biased towards low damping rates, as this favours driver comfort. In race cars, damping rates are higher, usually somewhere around 50-70 % of critical, to maximize the tractive potential of the tires. However, finding the ideal balance is difficult, as the other relevant parameters of the vehicle must be carefully accounted for, as well as the driving conditions, such as whether the road is smooth or rough, hot or cold, dry or wet. There are many more considerations that can quickly complicate this process, such as whether to optimize the suspension setup for the roll motion that occurs during cornering at the expense of response during pitching motions (from longitudinal acceleration) or vice versa.

To assist in this design process, it is first necessary to have a thorough understanding of the damper's range of capability, so that the valve adjustment required for a given response is known. The two force vs. velocity plots provided from Ohlins, the manufacturer, can be used to derive a model.

The dampers are 4-way adjustable, with valves for low-speed compression, high-speed compression, low-speed rebound, and high-speed rebound. The numbers seen on each plot above indicate different adjustments used, with the numbers representing the different adjustment values, given in the same order as just stated. Low speed adjustments are taken in "clicks" from fully closed (1/6 of a revolution), meaning that higher numbers have less damping, whereas high speed adjustments are taken in turns (full revolutions) from the fully closed position, so higher numbers mean more damping.

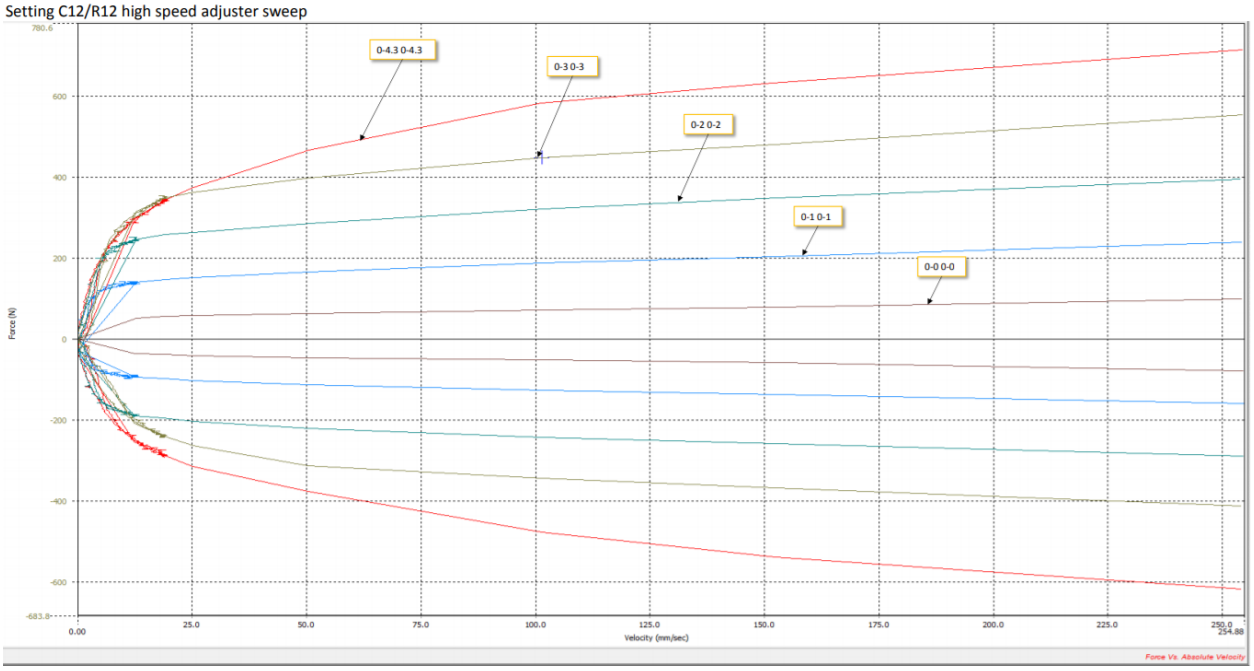


Figure 2: Ohlins TTX25 Force vs. Velocity, High Speed Adjustment

In these figures, Force is plotted on the vertical axis, with positive values for compression and negative values for rebound or extension. Velocity ranges from approximately 0 to 250 mm/s. In the high-speed adjustment plot above, the low speed damping is kept constant at its highest value. It can be seen that in general, changing the high-speed damping seems to limit the height the curve can reach, although each curve has a similar initial shape.

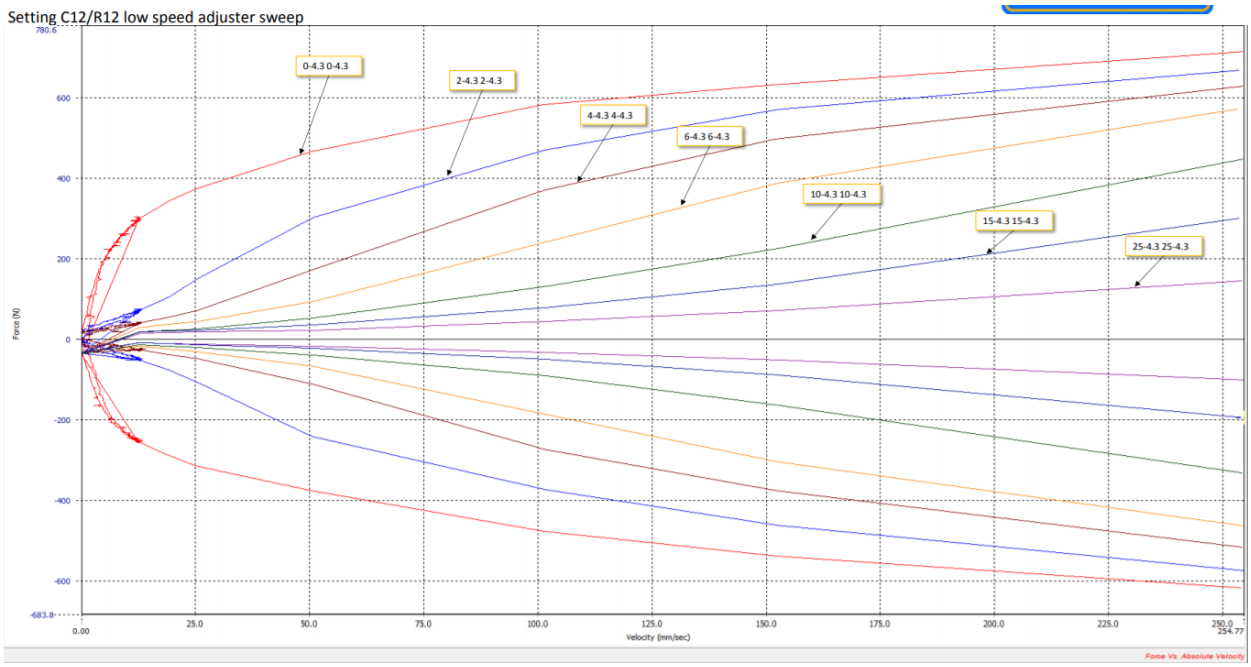


Figure 3: Ohlins TTX25 Force vs. Velocity, Low Speed Adjustment

In this low-speed plot, the high-speed damping is kept constant at its highest value. It is clearly apparent that this form of adjustment influences the response differently. This adjustment greatly changes the behaviour through the low velocity region, with less effect on the final value reached at maximum velocity. The highest damping provides the red curve seen, which has a negative rate of change in slope, however by the time the value is reduced to the damping of the green line, the curve has a positive rate of change in slope.

Using these plots, a model can be constructed to detail the change in response associated with different adjustments. The common parameters used to characterize this information are:

- R_{CE}: Ratio between compression and extension forces for a given sweep
- C_D: Damping coefficient (mean initial slope of compression and extension curves)
- λ: Non-linearity coefficient

This is a simple but effective representation of the system. Using this empirical approach will allow the model to have a high degree of accuracy, while avoiding the need to develop analysis/simulation of the physical fluid interactions from first principles, a process which would be very time intensive and require highly specialized knowledge.

2- Detailed Design

2.1 - Apparatus Design and Programming

The design of this device was composed of two main sections. The mechanical design was required to hold the damper, translate the rotational motion of the motor into linear motion on the damper, and withstand all applied stresses. The electrical system was required to control the motor input and obtain readings for the desired force and velocity outputs.

2.2 - Mechanical Design

Much of the mechanical system was previously fabricated as part of a prior project. It was comprised of a metal base block, a scotch yoke mechanism, two vertical beams, a horizontal block connecting the beams, and links to connect the damper to the rest of the assembly. This design allowed for the effective accomplishment of the project goals, with the minimum number of required parts and machining. All components were stored, disassembled in the race team shop. After inspection of the components, and conferring with the previous designers, it was determined that one major improvement was required. The base plate upon which the motor and cam are mounted was determined to be too thin based on observations from previous trial runs. Deflection in the baseplate could reduce the force recorded in the load cell at the top of the assembly. To address this issue, the base plate was replaced with a 2 ¾" thick aluminum block. The block was faced, and the required holes drilled, before it was introduced into the assembly. This component can be seen in figure 5 below.

Figure 5: Mounting Block after Being Machined

Following this adjustment, the rest of the assembly was put together, as shown in figure 6 below.

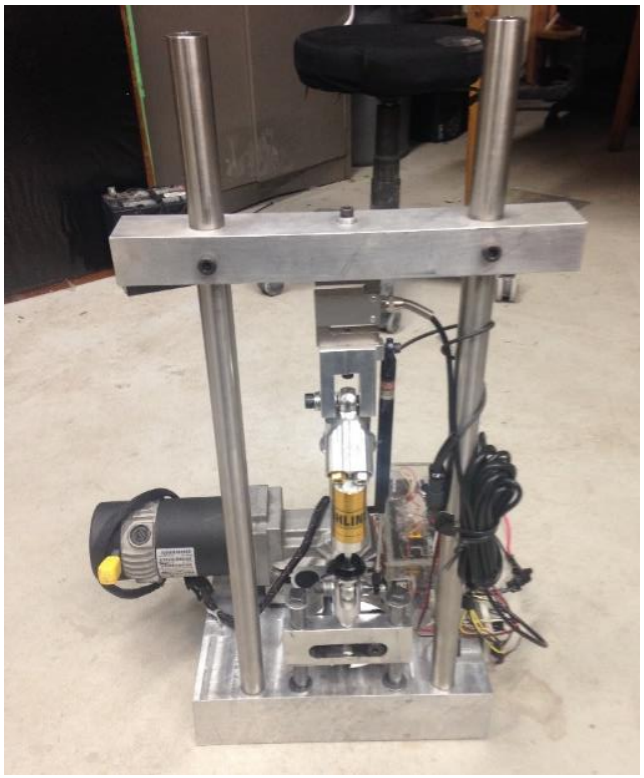


Figure 4: Views of Completed Device

2.3 - Structural Analysis

Structural analysis was conducted on a solid model of the apparatus in order to assess the feasibility of the structure. Since this project was inherited from a previous group, it was important to ensure the design of the structure would be able to handle and stresses exerted by the dampener. The additional modifications and additions that were completed as per the scope of this project could also be verified and justified through solid model analysis. The results of the simulations can be seen in figures 5 to 8 below.

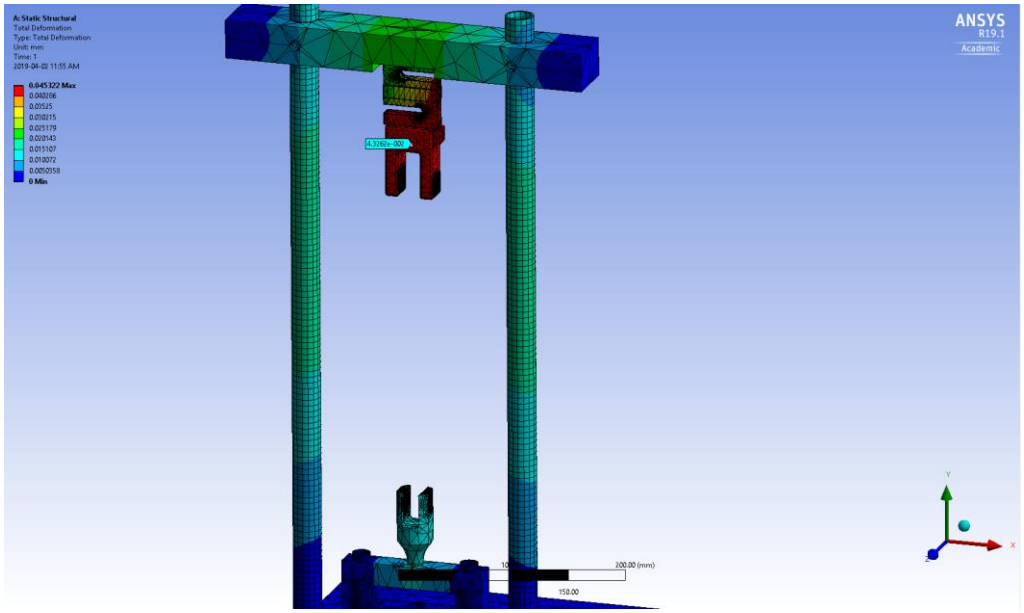


Figure 5: Deformation with 750N applied bearing force (compression)

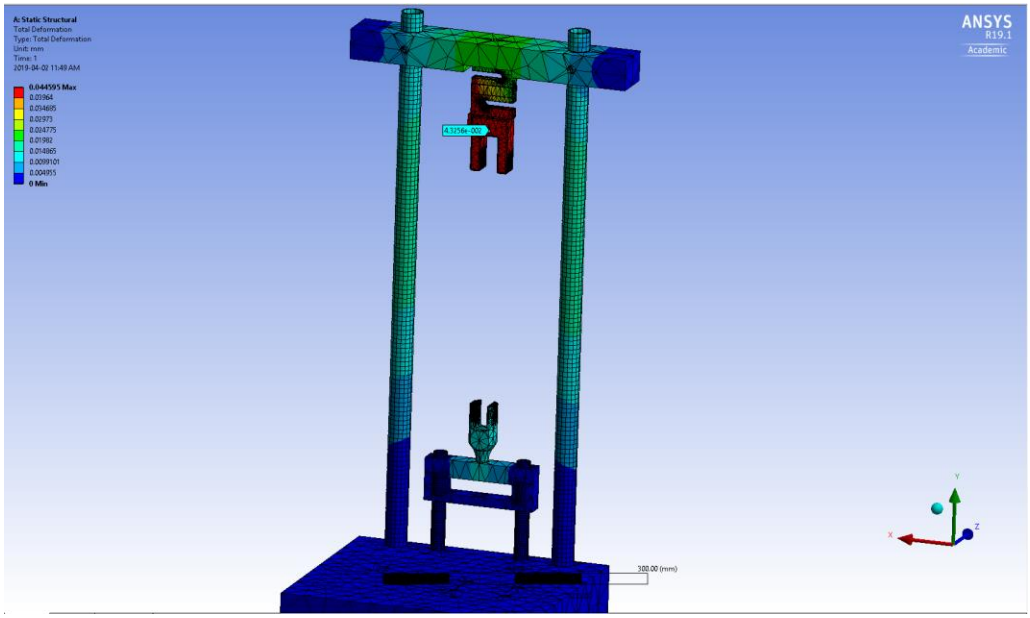


Figure 6: Deformation with 750N applied bearing force (tension)

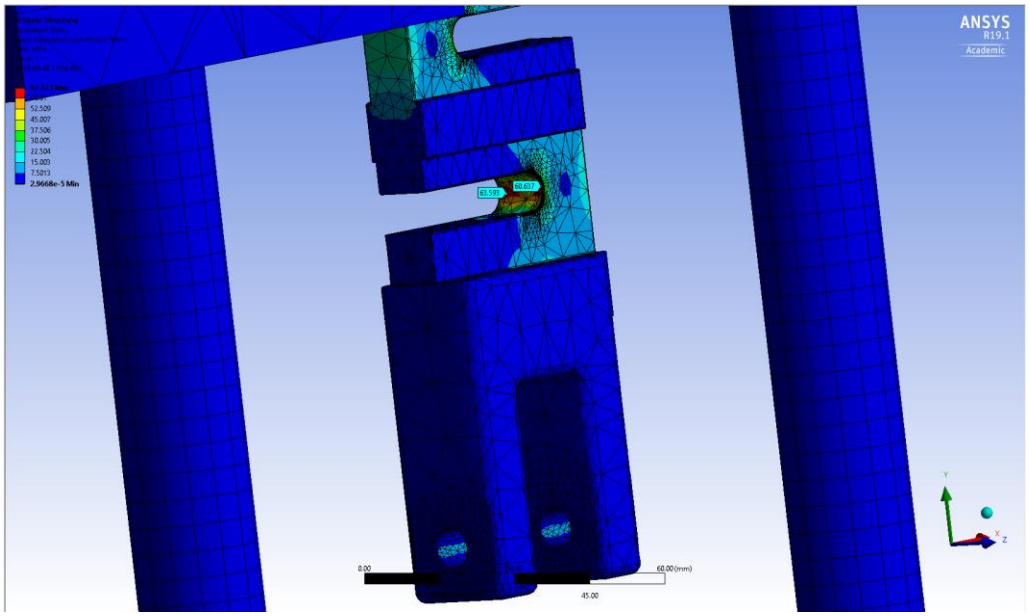


Figure 7: Equivalent Von-Mises Stress with 750N applied bearing force (compression)

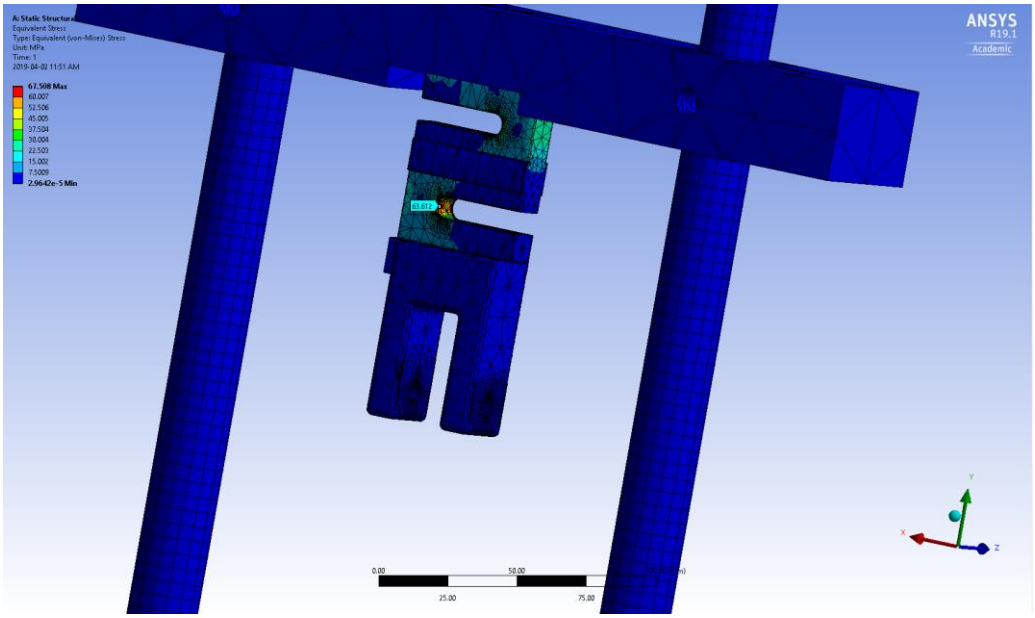


Figure 8: Equivalent Von-Misses Stress with 750N applied force (tension)

A standard mesh size was used with refinements around points of interest and stress concentrations. A maximum of 750N bearing stress was applied to the contact points between the apparatus and the dampener. Both compressive and tensile scenarios were assessed. The Deformation and Von-Mises solutions were considered as these will give the best indication of the usability of the apparatus since yield stresses should not be exceeded and any deformations in the apparatus will skew sensor results.

The preliminary structural analysis does not indicate any issues with the structural nature of the apparatus therefore the construction and modifications to the apparatus are sufficient to complete the damper testing.

2.4 - Electrical Design

In order to control the motor input and track the displacement of the cam and damping force, an electrical system was required. The main components of this system were: 2 Arduino microcontrollers, a 30 volt motor driver, a 100kg rated S-type load cell, a weight sensor module, a 100mm linear potentiometer, two switches, and four LEDs. These components were also available, disassembled, with the rest of the components from the previous project, however a circuit design and code had to be created. A complete circuit diagram of the electrical system can be seen in figure 9 below.

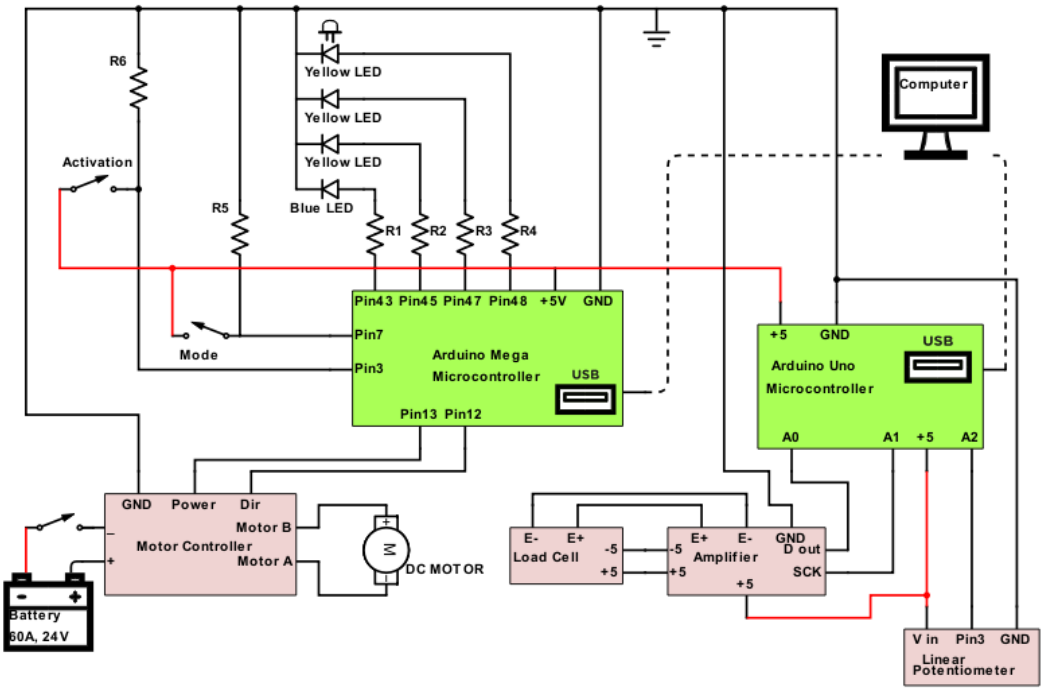


Figure 9: Circuit Diagram

The objective of the set up was to obtain load cell and potentiometer readings from the dynamometer for various motor input velocities and damper settings. To accomplish this, the Arduino mega was used in conjunction with the motor controller, switches, and LEDs to provide live manual control of the motor input speed. The set up was programmed such that the push switch had to be pushed down to begin speeding up the motor. As the motor speeds up, its speed was indicated by the LEDs displayed on top of the electrical box. If no LEDs were lit, meant the motor was stationary, one LED indicated a low speed range, two a medium low speed range, three a medium high-speed range, and four a high to maximum speed range. Once the desired speed was reached, the push button could be pressed and returned to its initial state to maintain a constant motor velocity. If a decrease in speed was desired, the push button could be pushed down, and the mode switch held down, until the desired speed was reached. The Arduino code for this motor control can be seen in figure 10 below.

```
const int Onswitch = 8;
const int Modeswitch = 9;
const int LEDpfour = 7;
const int LEDpthree = 6;
const int LEDpone = 5;
const int LEDptwo = 4;
const int pwm = 10;
const int dir = 3;

int OnSwitch = 0;
int ModeSwitch = 0;

int LEDone = 0;
int LEDtwo = 0;
int LEDthree = 0;
int LEDfour= 0;

int value = 0;

void setup() {
  Serial.begin(9600);

  pinMode(pwm,OUTPUT);
  pinMode(dir,OUTPUT);

  pinMode(Onswitch, INPUT_PULLUP);
  pinMode(Modeswitch, INPUT_PULLUP);

  pinMode(LEDpone, OUTPUT);
  pinMode(LEDptwo, OUTPUT);
  pinMode(LEDpthree, OUTPUT);
  pinMode(LEDpfour, OUTPUT);

  digitalWrite(dir, HIGH);
}

void loop() {
  OnSwitch = digitalRead(Onswitch);
  Serial.println(OnSwitch);

  if (OnSwitch == HIGH) {
    Serial.println("Altering Speed");
      delay(100);
    ModeSwitch = digitalRead(Modeswitch);

    if (ModeSwitch == 1){
      value++;
    }

      else{
        value--;
      delay(200);
      }

     if(value>255) {
        value= 255;
     }
      else if(value<0){
       value= 0;
      }

    else if (value>0 && value<60){
      Serial.println("Low Speed");
      digitalWrite(LEDpone, HIGH);
      digitalWrite(LEDptwo, LOW);
      digitalWrite(LEDpthree,LOW);
      digitalWrite(LEDpfour, LOW);
    }
    else if (value>49 && value<120){
      Serial.println("Medium Low Speed");
      digitalWrite(LEDpone, HIGH);
      digitalWrite(LEDptwo, HIGH);
      digitalWrite(LEDpthree,LOW);
      digitalWrite(LEDpfour, LOW);
    }
    else if (value>99 && value<180){
      Serial.println("Medium High Speed");
      digitalWrite(LEDpone, HIGH);
      digitalWrite(LEDptwo, HIGH);
      digitalWrite(LEDpthree,HIGH);
      digitalWrite(LEDpfour, LOW);
    }
    else if (value>149 && value<255){
      Serial.println("High Speed Speed");
      digitalWrite(LEDpone, HIGH);
      digitalWrite(LEDptwo, HIGH);
      digitalWrite(LEDpthree,HIGH);
      digitalWrite(LEDpfour, HIGH);
    }
}
  else{
    value=value;
    Serial.println("Constant Speed");
    delay(500);
  }
  analogWrite(pwm, value);
  Serial.println(value);
  delay(500);
  }
```

Figure 10: Arduino Motor Control Code

In order to obtain the measurements, sensor values also had to be recorded. To do this, the Arduino Genuino Uno was used in combination with the S-type load cell, the weight sensor module, and the linear potentiometer to obtain the desired readings. To do this, a loop of code was composed that read in the sensor values, calibrated them, and output them to the serial monitor, where they were exported to a text file. The Arduino code for this sensor integration can be found in figure 10 below.

```
#include <Hx711.h>

Hx711 scale(A0, A1);
int val;
int calval;
int Potval;
void setup() {
  Serial.begin(9600);
}

void loop() {
  val = scale.getGram();

  //Serial.println (val);
  Potval = analogRead(A3);
  Serial.println((Potval-500)*0.1*100);
  //Serial.println(val);


  if (val > (.1) && val < 10){
    Serial.println(val*29.422+6.802);
  }
  else if (val < -4000) {
    calval = ( 8420- val*(-1))*29.422+6.802;
    Serial.println(calval);
  }
  else if (val > (-0.1) && val < 0.1){
    Serial.println("0");
  }
  else {
    calval = val*29.422+6.802;
    Serial.println(calval);

  }

  delay(10);



}
```

Figure 11: Sensor control Arduino Code

3- Modelling Details

3.1 - MATLAB Model and Test Results

The goal of the modelling component of this project has been to allow analysis of theoretical and tested force vs velocity curves, throughout the full range of valve adjustment. The behaviour of the damper at any of the damper settings can now be understood, while the manufacturer plots only provide data in set increments of the damper valves. The theoretical vs. tested values can also be compared to validate the accuracy of the original supplied data, with the potential to test all four dampers of the Gryphon Racing car to ensure that they are a properly “matched set” and within tolerance. To do this, first the manufacturer plots were digitized so that data sets could be imported into Matlab. The structure of the program is as follows (full code available in Appendix C):

The data sets are first imported in to the program. A series of filtering and sorting commands are done to separate out the data into appropriate compression and rebound bins for each adjustment level. These individual data sets, referred to as sweeps throughout the program, are then concatenated into a master data set. The starts and ends of each individual sweep within this set are defined, as well as each of their associated adjustment conditions. In this manner, the data can now be easily worked with using loops. A simple one-term exponential equation is fitted to each curve, using a loop which

cycles through each sweep of the data set. From these equations, the previously mentioned modeling parameters Cd, Lambda, and Rce are derived.

The test data is then imported into the program. Firstly, all the variables of the test are plotted vs. time to give a quick glimpse of what occurred during the test. This includes the displacement and velocity of the damper, as well as the damper force vs time. Damper velocity is found by taking the difference between each adjacent displacement data points and dividing by the sampling frequency specified in the Arduino code.

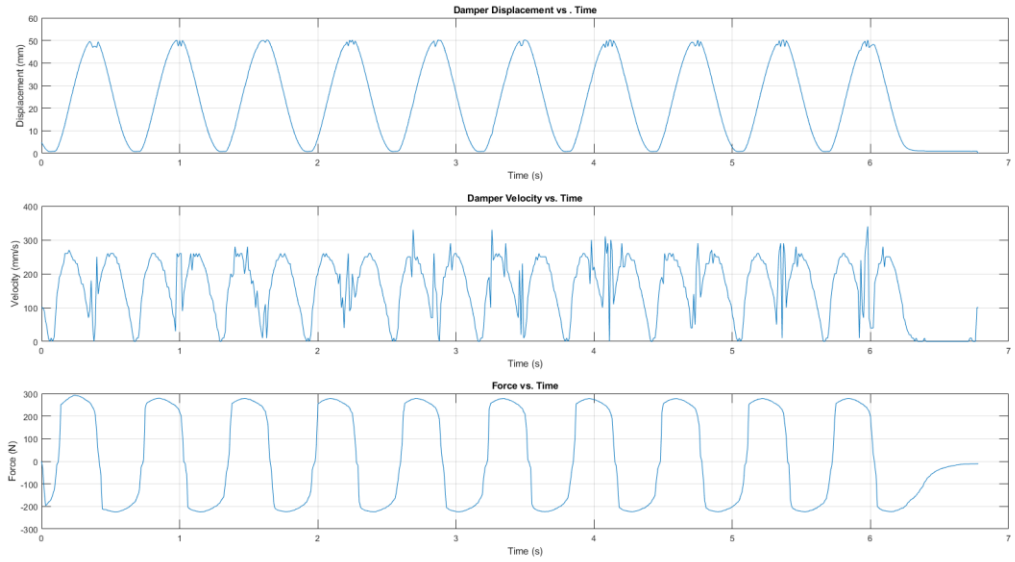


Figure 12: Displacement, Velocity and Force Plots vs. Time

The user is asked in the command line whether they would like to work with the low-speed or high-speed variation model, and what the specific damper settings of their test were. The suitable values from the theoretical model are then used to construct an expected force vs velocity response for the test data. The modeled curve and test data are then plotted together as seen below. For all following graphs, force is plotted in Newtons, and velocity is given in mm per second.

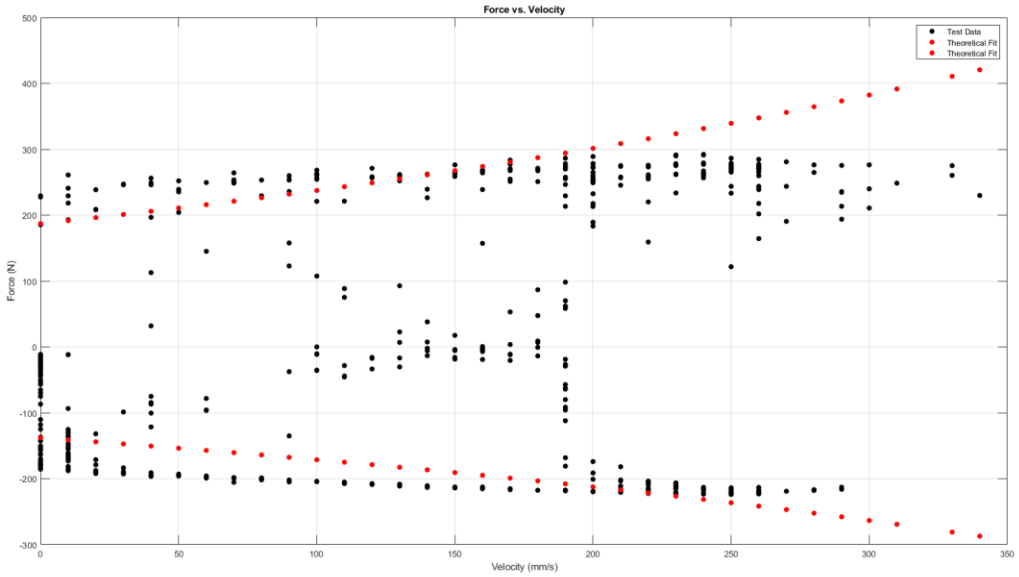


Figure 13: Experimental Data plotted with Mathematical Model

Lastly, the original OEM data is plotted. This is done with the intention that as the device continues to be used in the future, test data can be plotted against the original curves to verify whether the apparatus can reproduce them.

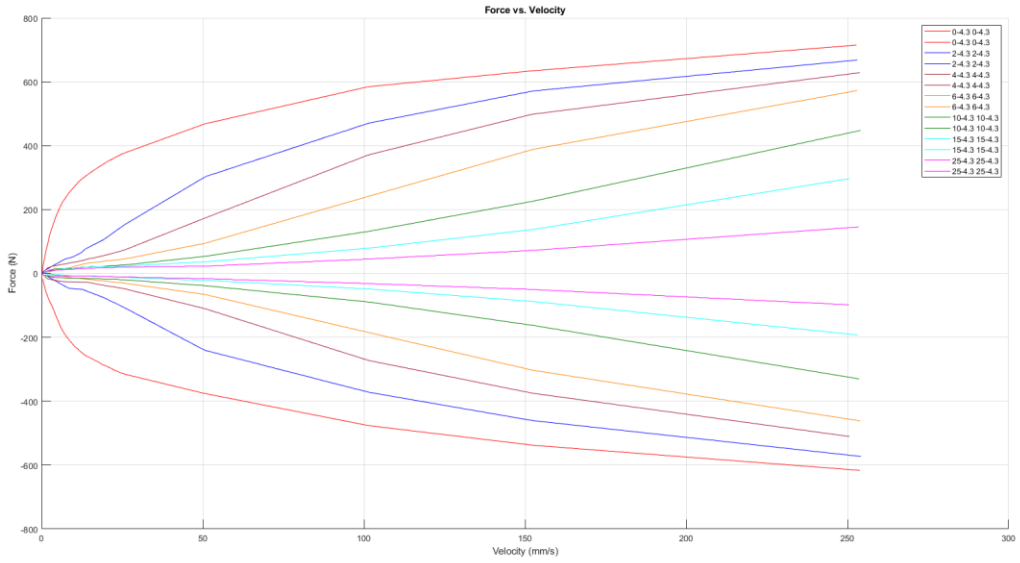


Figure 14: Reproduced OEM plot

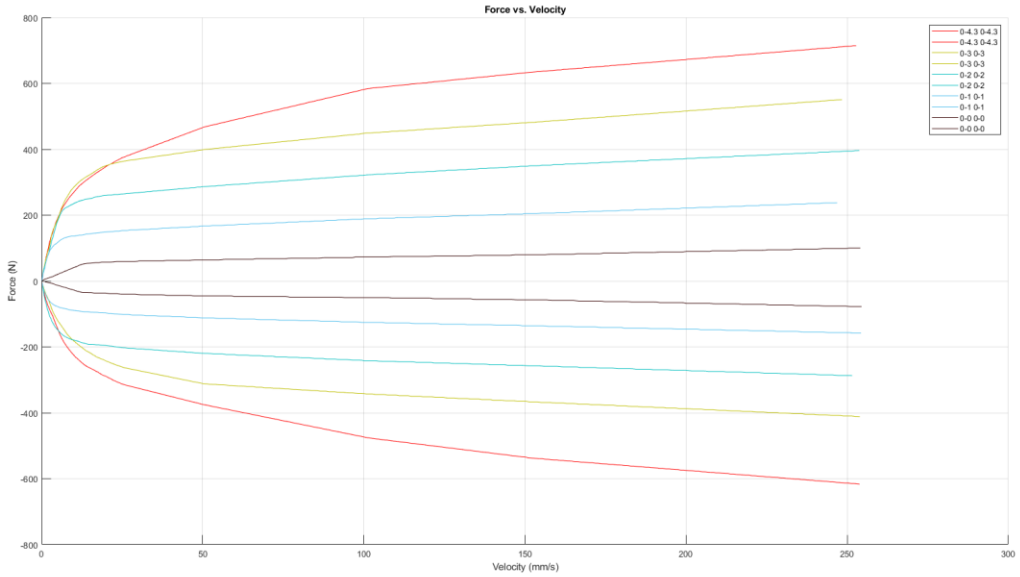


Figure 15: Reproduced OEM plot

Throughout the program, some capability has also been written in to deal with temperature sensor data from the damper. Inferring relationships about how the viscosity and damping force change with temperature is however quite advanced for the current scope of the project, so this sensor was not ultimately implemented. More refining to the accuracy of the curve fitting should first be carried out, ideally using the "lsqnonlin" function, so that the model can be improved before attempting to make conclusions about the more minor effects seen with temperature variation.

4- Testing and Design Evaluation

4.1 - Discussion

The prototype is effective in creating simulated velocity input scenarios and it is effective in collecting data when error is considered. The velocity inputs resulted in expected behaviour from the apparatus as modelled through mathematical modelling. In order to test the effectiveness of the mathematical model, dampener settings were chosen so that the force-velocity curve would fall between the established force-velocity curves seen in the OEM plots. A discussion of the accuracy of results and effectiveness of the model follows below.

The displacement/ velocity plots in figure 13 show the data obtained from the linear potentiometer. The displacement curve was obtained directly from the calibrated sensor data and the velocity was obtained by taking the derivative of the displacement data. Some noise and small variations in the displacement plot can be seen at the peeks of the sinusoidal curve. These small variations can be attributed to various sources of error, discussed below. Any variations or disturbances in the displacement data are amplified by the derivative, thus showing greater noise in the velocity plot. When this noise is ignored, as it can be attributed to the mathematical processing, the velocity follows an oscillating, sinusoidal pattern which is to be expected as a scotch-yoke mechanism outputs a sinusoidal velocity pattern. This varying velocity was used to replicate the OEM plot by sampling various velocity

values at various points in time and matching them to points on the force data occurring at the came time.

The force plot seen in figure 13 was obtained directly from the calibrated strain gauge data. This force plot does not follow the shape of a sinusoidal function; however, it does show a repeating, oscillating pattern which it to be expected as the velocity input followed an oscillating pattern. The pattern does not follow is sinusoidal shape due to the non-linearity of the damper. When the velocity reaches its inflection point (the velocity vector changes direction) and it begins to speed up, the damper will exert its highest resistance in opposition to the change in velocity. The force rapidly increases until the maximum velocity is reached, and then the force slightly levels out before as the velocity begins to slow. This can be attributed to the dissipation of the force within the dampener due to the propertied of the fluid within the damper as well as the design of the dampener itself. The next rapid change in force occurs when the velocity begins to speed up again as the vector changes once more. This pattern continues at the oscillating velocity input repeats.

The theoretical OEM data was fit with an exponential function as this provided the most appropriate line of best fit. The top, red trend line represents the compression response of the dampener and the bottom, red trend line represents the rebound response. This exponential function provides an accurate representation of the OEM data at lower velocity inputs; however, the model loses accuracy at higher velocities. As see in figure 14, the experimental data (in black) follows the trend of the theoretical exponential curves (in red) up until a velocity of approximately 200 mm/second is reached. After this point the theoretical model loses accuracy. When compared to the OEM plots in figures 11 and 12, the experimental data does follow the high-speed trends exhibited in the OEM plots, therefore the data can be considered as a valid reflection of the behaviour of the system. The points that fall between the two dominant trend lines can be attributed to error within the data sampling. When the force-velocity pots were created, periodic sampling of the force and velocity data was obtained, and this sampling could have captured some of the noise present from the amplification due to the derivative being applied in the velocity plot. These points will then cause inaccurate reflections of the velocities at those points, thus causing outliers to occur within the force-velocity plot.

Some other sources of error can cause some of the irregularities present in the data. The data outliers present in the figure 14 can also be attributed to error within the strain gauge. There were consistent technical difficulties in calibrating the strain gauge as it consistently exhibited odd readings when the dampener was in the rebound state (the strain gauge experienced tensile force). Further mathematical calibration and “if statements” were applied in order to combat these irregularities. In addition, a small amount of testing was completed within the timeline of this project. In order to establish strong force-velocity correlation curves, extensive testing is required in order for definitive experimental trends to occur.

The displacement, velocity and force graphs do not exhibit completely smooth sinusoids. This can be attributed to slight movement of components of the apparatus. Some bolts could not be completely tightened due to the nature of the tolerancing within our manufacturing capabilities. In some cases, there were slight clearance fits between fasteners and holes to aid in ease of assembly and these clearance fits could have introduced unwanted vibrations in to the system. The scotch yoke pin also experienced some friction in the slot, and was not perfectly fitted, leaving some room for slight movement and stuttering perpendicular to the intended sliding direction. This slight shifting in the

apparatus is reflected in the graphs by the discrepancies in the smooth displacement, velocity and force curves in figure 13.

4.2 - Future Changes and Recommendations

Some improvements that that could be implemented during future testing to help mitigate some of the sources of error could be: potentially employing more complex numerical methods to create more sophisticated functions to better model the manufacturer provided damper plots in the high velocity regions, using tighter tolerances in the manufacturing all of the structural components, pins and fasteners used secure and transfer force between the motor, the shock, the strain gauge, and frame, to reduce vibrations and unintended motions which may introduce error into the strain gauge reading readings, and exploring alternate mechanisms for converting the rotational motion of the motor shaft into linear translation could be tested as a method of reducing unintended jerk movements in the damper system. The slot and pin components may not have been rigid and/or made to tight enough tolerances to effectively translate 100 percent of the rotational motion of the motor into linear translational motion without incurring losses and stuttering at the top and bottom of the motion cycle, where the apparatus transitioned between compression and tension. A different mechanism, such as a four-bar linkage, may provide a smoother linear translation motion cycle, without any sudden accelerations or jerks, producing a more consistent force curve.

4.3 – Conclusion and Closing Comments

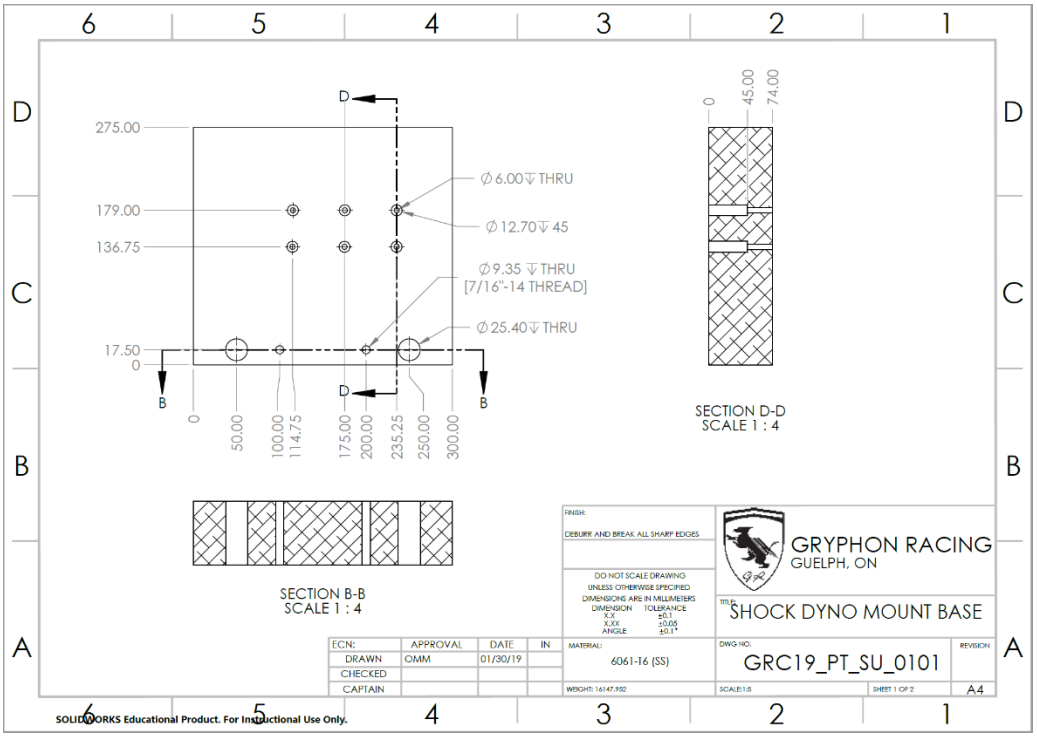
When accounting for mechanical errors, the model was able to provide an approximation of the force values expected in the test. The model was most accurate in the low-velocity tension, low-velocity compression, and high-velocity compression regions, but lost some accuracy in the high velocity-tension region. As seen in figure 14, test data falls to below 60 percent of the expected theoretical force value.

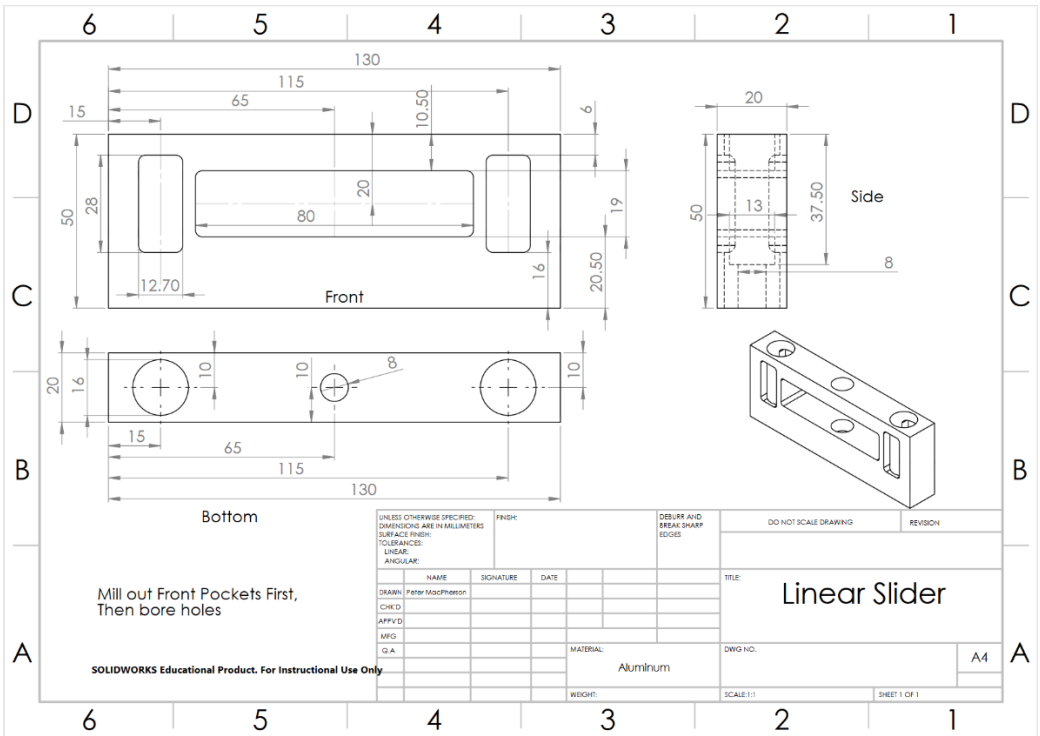
The apparatus itself was able to successfully perform all required tests on the damper, allowing sufficient data to be collected in order to analyze the model over a wide range of speed values, and compare it to the corresponding manufacturer data. The electrical components of the apparatus; the series of microcontrollers, the integrated circuitry, the electrical sensors, the user controls, the software, and the electrical motor were all successfully incorporated into the system, allowing for safe and consistent control of the testing device, and for accurate data collection. The mechanical components; the frame, linkages, fasteners and damper itself were all successful as well, as no mechanical parts experienced any failures, unexpected or otherwise. The only potential future mechanical modifications recommended to the system structure are to increase precision and reduce error.

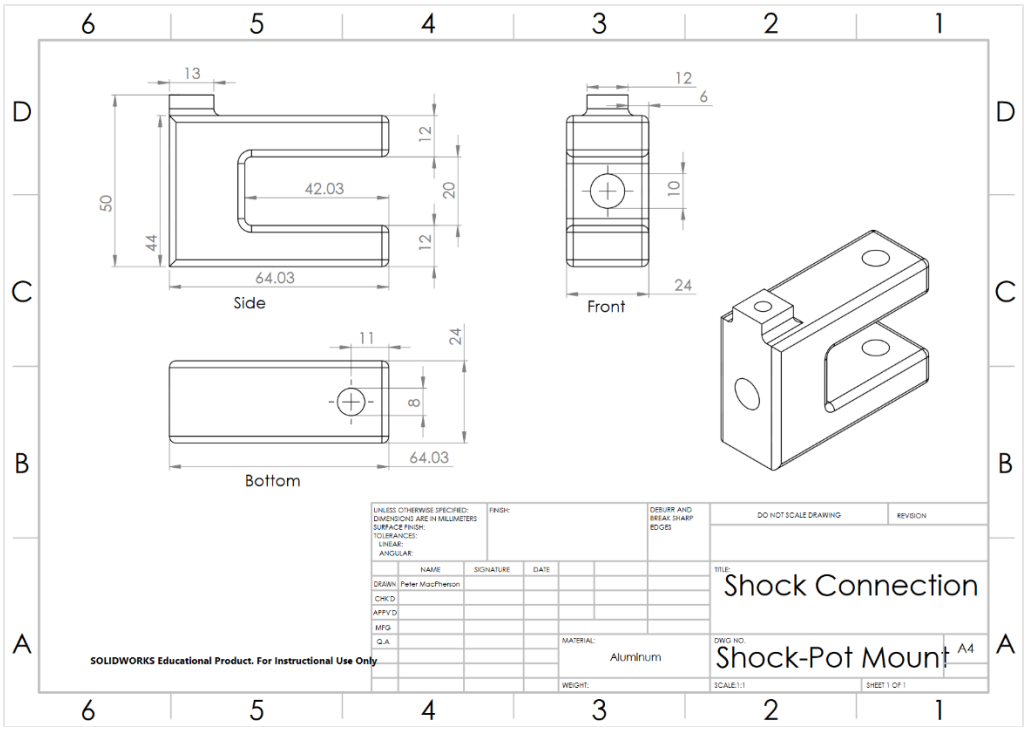
Finally, the initial modifications made to the apparatus to address the issues faced by the previous student groups were all successful as well. The deflections in the baseplate were corrected in its replacement, and the new strain gauge that was used in the updated design was properly suited to the ranges of forces encountered in the test and did not encounter any forces outside of its suitable range.

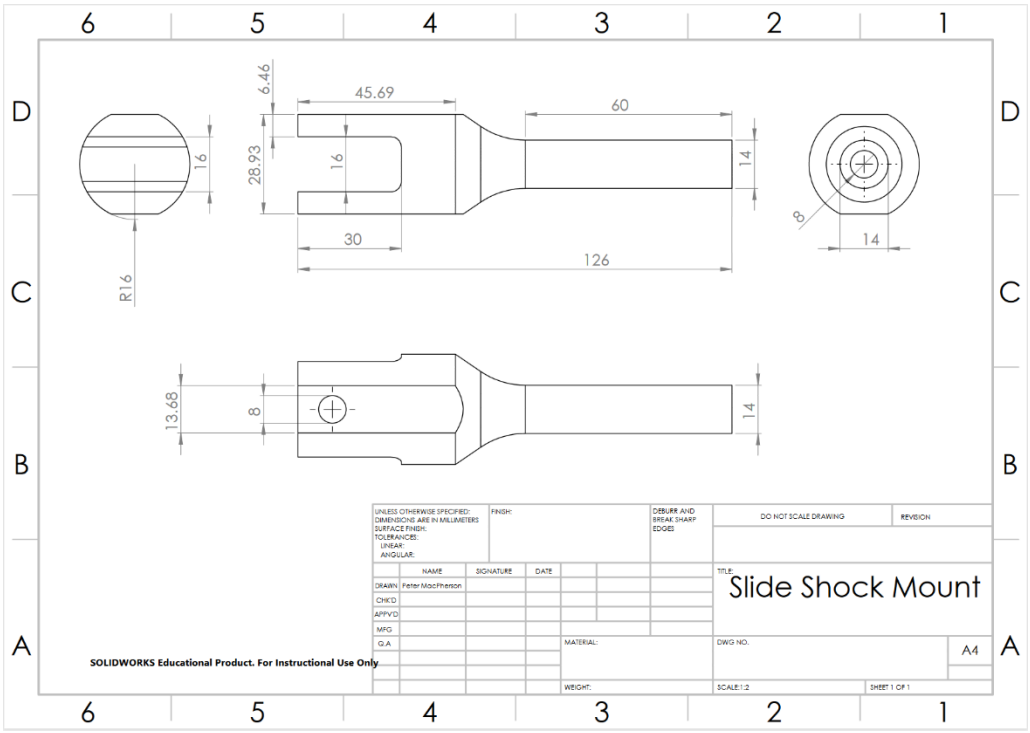
5- Appendices

5.1 – Appendix A – Drawings






13
12
6
50
42.03
20
12
44
12
10
64.03
24
Side
Front
11
24
8
64.03
Bottom
Shock Connection
Shock-Pot Mount
Aluminum
A4
SOLIDWORKS Educational Product. For Instructional Use Only


6.46
45.69
60
16
28.93
16
14
30
126
R16
8
14
13.68
8
14
Slide Shock Mount
A4
SOLIDWORKS Educational Product. For Instructional Use Only

5.2 - Appendix B – Bill of Materials

The materials and components listed blow represent the components that have been manufactured/obtained as part of the scope of this project and report. All other materials and components within the apparatus were previously existing from the previous student group's work. The aluminum used to manufacture the components was supplied by the race team, thus no costs were incurred by this team.

Component	**Material**	**Cost**
Base	Aluminum	$0
Linear Slider	Aluminum	$0
Shock and Strain Gauge Connection	Aluminum	$0
Slide Shock Mount	Aluminum	$0
Strain Gauge	100kg rated gauge	Supplied by Race Team
12V Battery leads (2 leads)	Gauge 4	$33.88

5.3 - Appendix C – Matlab Code

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Lindsay Burton, Orion Miller, Ben Tory-Pratt, Austin Van Rossum

% OHLINS TTX25 DAMPER MODELER

% March 2019

% References:

%https://deepblue.lib.umich.edu/bitstream/handle/2027.42/49574/proj25_report.pdf?seque
nce=2

%Known Issues:

%Legend labels on the graphs are shown duplicated. I don't know of a way these can

%be removed without making major changes to the way the data is

%plotted/organized.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clearvars

clc

tic

cd(uigetdir); % Select directory of data

OhlinsData0_0_0_0 = load('0-0 0-0.csv'); %Import in individual data sets

OhlinsData0_1_0_1 = load('0-1 0-1.csv');

OhlinsData0_2_0_2 = load('0-2 0-2.csv');

OhlinsData0_3_0_3 = load('0-3 0-3.csv');

OhlinsData0_43_0_43 = load('0-4.3 0-4.3.csv'); %Used for both low and hi speed

OhlinsData2_43_2_43 = load('2-4.3 2-4.3.csv');

OhlinsData4_43_4_43 = load('4-4.3 4-4.3.csv');

OhlinsData6_43_6_43 = load('6-4.3 6-4.3.csv');

OhlinsData10_43_10_43 = load('10-4.3 10-4.3.csv');

OhlinsData15_43_15_43 = load('15-4.3 15-4.3.csv');
```

```
OhlinsData25_43_25_43 = load('25-4.3 25-4.3.csv');



index0_0_0_0 = OhlinsData0_0_0_0(:,2)>0; %Use logical indexing to split each into two
files

%separate compression and rebound data

OhlinsData0_0_0_0Comp = OhlinsData0_0_0_0(index0_0_0_0,1:2);

OhlinsData0_0_0_0Reb = OhlinsData0_0_0_0(~index0_0_0_0,1:2);

%Order force vals from lowest to highest (fixes a quirk in the data)

OhlinsData0_0_0_0Comp = sort(OhlinsData0_0_0_0Comp);

OhlinsData0_0_0_0Reb = sort(abs(OhlinsData0_0_0_0Reb));

%Make the comp and rebound sets have equal lengths (some have a slight

%difference)

OhlinsData0_0_0_0Reb(:,2) = -OhlinsData0_0_0_0Reb(:,2);


%Repeat this procedure for all other data sets

index0_1_0_1 = OhlinsData0_1_0_1(:,2)>0;

OhlinsData0_1_0_1Comp = OhlinsData0_1_0_1(index0_1_0_1,1:2);

OhlinsData0_1_0_1Reb = OhlinsData0_1_0_1(~index0_1_0_1,1:2);

OhlinsData0_1_0_1Comp = sort(OhlinsData0_1_0_1Comp);

OhlinsData0_1_0_1Reb = sort(abs(OhlinsData0_1_0_1Reb));

OhlinsData0_1_0_1Reb(:,2) = -OhlinsData0_1_0_1Reb(:,2);

OhlinsData0_1_0_1Comp = OhlinsData0_1_0_1Comp(1:286,:);


index0_2_0_2 = OhlinsData0_2_0_2(:,2)>0;

OhlinsData0_2_0_2Comp = OhlinsData0_2_0_2(index0_2_0_2,1:2);

OhlinsData0_2_0_2Reb = OhlinsData0_2_0_2(~index0_2_0_2,1:2);

OhlinsData0_2_0_2Comp = sort(OhlinsData0_2_0_2Comp);

OhlinsData0_2_0_2Reb = sort(abs(OhlinsData0_2_0_2Reb));

OhlinsData0_2_0_2Reb(:,2) = -OhlinsData0_2_0_2Reb(:,2);

OhlinsData0_2_0_2Reb = OhlinsData0_2_0_2Reb(1:290,:);


index0_3_0_3 = OhlinsData0_3_0_3(:,2)>0;

OhlinsData0_3_0_3Comp = OhlinsData0_3_0_3(index0_3_0_3,1:2);

OhlinsData0_3_0_3Reb = OhlinsData0_3_0_3(~index0_3_0_3,1:2);

OhlinsData0_3_0_3Comp = sort(OhlinsData0_3_0_3Comp);
```

```
OhlinsData0_3_0_3Reb = sort(abs(OhlinsData0_3_0_3Reb));
OhlinsData0_3_0_3Reb(:,2) = -OhlinsData0_3_0_3Reb(:,2);
OhlinsData0_3_0_3Comp = OhlinsData0_3_0_3Comp(1:277,:);

index0_43_0_43 = OhlinsData0_43_0_43(:,2)>0;
OhlinsData0_43_0_43Comp = OhlinsData0_43_0_43(index0_43_0_43,1:2);
OhlinsData0_43_0_43Reb = OhlinsData0_43_0_43(~index0_43_0_43,1:2);
OhlinsData0_43_0_43Comp = sort(OhlinsData0_43_0_43Comp);
OhlinsData0_43_0_43Reb = sort(abs(OhlinsData0_43_0_43Reb));
OhlinsData0_43_0_43Reb(:,2) = -OhlinsData0_43_0_43Reb(:,2);
OhlinsData0_43_0_43Comp = OhlinsData0_43_0_43Comp(1:280,:);

index2_43_2_43 = OhlinsData2_43_2_43(:,2)>0;
OhlinsData2_43_2_43Comp = OhlinsData2_43_2_43(index2_43_2_43,1:2);
OhlinsData2_43_2_43Reb = OhlinsData2_43_2_43(~index2_43_2_43,1:2);
OhlinsData2_43_2_43Comp = sort(OhlinsData2_43_2_43Comp);
OhlinsData2_43_2_43Reb = sort(abs(OhlinsData2_43_2_43Reb));
OhlinsData2_43_2_43Reb(:,2) = -OhlinsData2_43_2_43Reb(:,2);

index4_43_4_43 = OhlinsData4_43_4_43(:,2)>0;
OhlinsData4_43_4_43Comp = OhlinsData4_43_4_43(index4_43_4_43,1:2);
OhlinsData4_43_4_43Reb = OhlinsData4_43_4_43(~index4_43_4_43,1:2);
OhlinsData4_43_4_43Comp = sort(OhlinsData4_43_4_43Comp);
OhlinsData4_43_4_43Reb = sort(abs(OhlinsData4_43_4_43Reb));
OhlinsData4_43_4_43Reb(:,2) = -OhlinsData4_43_4_43Reb(:,2);
OhlinsData4_43_4_43Reb = OhlinsData4_43_4_43Reb(1:266,:);

index6_43_6_43 = OhlinsData6_43_6_43(:,2)>0;
OhlinsData6_43_6_43Comp = OhlinsData6_43_6_43(index6_43_6_43,1:2);
OhlinsData6_43_6_43Reb = OhlinsData6_43_6_43(~index6_43_6_43,1:2);
OhlinsData6_43_6_43Comp = sort(OhlinsData6_43_6_43Comp);
OhlinsData6_43_6_43Reb = sort(abs(OhlinsData6_43_6_43Reb));
OhlinsData6_43_6_43Reb(:,2) = -OhlinsData6_43_6_43Reb(:,2);

index10_43_10_43 = OhlinsData10_43_10_43(:,2)>0;
OhlinsData10_43_10_43Comp = OhlinsData10_43_10_43(index10_43_10_43,1:2);
```

```
OhlinsData10_43_10_43Reb = OhlinsData10_43_10_43(~index10_43_10_43,1:2);
OhlinsData10_43_10_43Comp = sort(OhlinsData10_43_10_43Comp);
OhlinsData10_43_10_43Reb = sort(abs(OhlinsData10_43_10_43Reb));
OhlinsData10_43_10_43Reb(:,2) = -OhlinsData10_43_10_43Reb(:,2);

index15_43_15_43 = OhlinsData15_43_15_43(:,2)>0;
OhlinsData15_43_15_43Comp = OhlinsData15_43_15_43(index15_43_15_43,1:2);
OhlinsData15_43_15_43Reb = OhlinsData15_43_15_43(~index15_43_15_43,1:2);
OhlinsData15_43_15_43Comp = sort(OhlinsData15_43_15_43Comp);
OhlinsData15_43_15_43Reb = sort(abs(OhlinsData15_43_15_43Reb));
OhlinsData15_43_15_43Reb(:,2) = -OhlinsData15_43_15_43Reb(:,2);
OhlinsData15_43_15_43Comp = OhlinsData15_43_15_43Comp(1:277,:);

index25_43_25_43 = OhlinsData25_43_25_43(:,2)>0;
OhlinsData25_43_25_43Comp = OhlinsData25_43_25_43(index25_43_25_43,1:2);
OhlinsData25_43_25_43Reb = OhlinsData25_43_25_43(~index25_43_25_43,1:2);
OhlinsData25_43_25_43Comp = sort(OhlinsData25_43_25_43Comp);
OhlinsData25_43_25_43Reb = sort(abs(OhlinsData25_43_25_43Reb));
OhlinsData25_43_25_43Reb(:,2) = -OhlinsData25_43_25_43Reb(:,2);
OhlinsData25_43_25_43Reb = OhlinsData25_43_25_43Reb(1:271,:);

%Concatenate all these files into a single master data set so that analysis
%of the different conditions can be done with a loop
OhlinsDataMaster = vertcat(OhlinsData0_0_0_0Comp, OhlinsData0_0_0_0Reb,
OhlinsData0_1_0_1Comp,...
OhlinsData0_1_0_1Reb, OhlinsData0_2_0_2Comp,OhlinsData0_2_0_2Reb,
OhlinsData0_3_0_3Comp,...
OhlinsData0_3_0_3Reb, OhlinsData0_43_0_43Comp, OhlinsData0_43_0_43Reb,
OhlinsData2_43_2_43Comp,...
OhlinsData2_43_2_43Reb, OhlinsData4_43_4_43Comp, OhlinsData4_43_4_43Reb,
OhlinsData6_43_6_43Comp,...
OhlinsData6_43_6_43Reb, OhlinsData10_43_10_43Comp, OhlinsData10_43_10_43Reb,
OhlinsData15_43_15_43Comp,...
OhlinsData15_43_15_43Reb, OhlinsData25_43_25_43Comp, OhlinsData25_43_25_43Reb);

%Define the start and end points of each sweep so that these can be indexed
%through
SweepStarts = [1 282 563 849 1135 1425 1715 1992 2269 2549 2829 3102 ...
```

```
    3375 3641 3907 4176 4445 4713 4981 5258 5535 5806];

SweepEnds = [281 562 848 1134 1424 1714 1991 2268 2548 2828 3101 3374 ...
    3640 3906 4175 4444 4712 4980 5257 5534 5805 6076];

%Define number of different data sets within the master data set
SweepsNum = numel(SweepStarts);

%Define the adjustment conditions of the dampers through each of the sweeps
OhlinsDataLSC = [0 0 0 0 0 0 0 0 0 0 2 2 4 4 6 6 10 10 15 15 25 25];
OhlinsDataHSC = [0 0 1 1 2 2 3 3 4.3 4.3 4.3 4.3 4.3 4.3 4.3 4.3 ...
    4.3 4.3 4.3 4.3 4.3 4.3];
OhlinsDataLSR = OhlinsDataLSC;
OhlinsDataHSR = OhlinsDataHSC;

disp('Valve adjustment naming convention: LSC-HSC LSR-HSR (L=low, H=high, S=speed,
C=compression, R=rebound).');
disp('LS adjustments (3mm hex) are counted in clicks from fully closed (fully
clockwise), HS adjustments (12mm hex)');
disp('are counted in turns from fully open (fully counter-clockwise)');

TheoreticalFit = zeros(length(OhlinsDataMaster),1); %Initialize exponential fit

for i=1:SweepsNum
        %Create and exponential fit for each sweep:
        TheoreticalFit =
fit(OhlinsDataMaster(SweepStarts(i):SweepEnds(i),1),OhlinsDataMaster(SweepStarts(i):Sw
eepEnds(i),2),'exp1');

        %Store the equation coefficiencts as Cd (Initial Slope) and Lambda
        %(Non-linearity coefficient):
        Cd(i) = TheoreticalFit.a;
        Lambda(i) = TheoreticalFit.b;

end

Rce = zeros(SweepsNum); %Initialize ratio of compression over extension
```

```
for i=1:SweepsNum-1
    AvgCompForce(i) = mean(OhlinsDataMaster(SweepStarts(i):SweepEnds(i),2));
    AvgRebForce(i) = mean(abs(OhlinsDataMaster(SweepStarts(i+1):SweepEnds(i+1),2)));

    % Find the average ratio of compression over extension force for each
    % sweep:
    Rce(i) = AvgCompForce(i)./AvgRebForce(i);
end

Rce = Rce(1:2:SweepsNum); %Eliminates half the entries from the previous form of the
matrix, because only every other value is used

% FZ_CLASS(i) = interp1(FZ_LIST,FZ_LIST,FZ_AVG(i),'nearest');

%Load Test Data
TestData = load(uigetfile);

%Index Test by Variables
TestDataTime = TestData(:,1);
TestDataLinDisp = TestData(:,4);
TestDataLinVel = zeros(numel(TestDataTime),1);

for i=1:numel(TestDataTime)-1

    %Find linear velocity by differentiating displacement vs. time:
    TestDataLinVel(i) = abs(TestDataLinDisp(i+1)-
TestDataLinDisp(i))./(TestDataTime(i+1)-TestDataTime(i));

end

TestDataLinVel(numel(TestDataTime)) = TestDataLinVel(numel(TestDataTime)-1);

TestDataForce = TestData(:,2);
TestDataForce = TestDataForce*-0.009806; %Unit Conversion: Grams to Newtons
% TestDataTemp = TestData(:,4);
```

```
disp('Importing Test Data...')


while(1)
AdjustmentChoice = input('What kind of adjustment is being tested? Enter 1 for Low-Speed, enter 2 for High-Speed: ');


if AdjustmentChoice == 1
TestDataLSC = input('Enter Low Speed Compression: ');
TestDataLSR = input('Enter Low Speed Rebound: ');
TestDataHSC = 4.3;
TestDataHSR = 4.3;


break


else if AdjustmentChoice == 2
TestDataLSC = 0;
TestDataLSR = 0;
TestDataHSC = input('Enter High Speed Compression: ');
TestDataHSR = input('Enter High Speed Rebound: ');
break


    end
end
end


%Create a display of all measured variables vs. time
PlotTitle = sprintf('Test Data vs. Time, LSC: %s, LSR: %s, HSC: %s, HSR: %s', ...
    num2str(TestDataLSC), num2str(TestDataLSR), ...
    num2str(TestDataHSC), num2str(TestDataHSR));
f = figure('Name',PlotTitle,'NumberTitle','off','Position',[0 0 1300 670]);
figure(f)


subplot(3,1,1)
    plot(TestDataTime, TestDataLinDisp)
    title('Damper Displacement vs . Time');
```

```
    xlabel('Time (s)');
    ylabel('Displacement (mm)');
    grid on
  %  set(gca,'color',[0 0 0])

subplot(3,1,2)
    plot(TestDataTime, TestDataLinVel)
    title('Damper Velocity vs. Time');
    xlabel('Time (s)');
    ylabel('Velocity (mm/s)');
    grid on
 %   set(gca,'color',[0 0 0])

subplot(3,1,3)
    plot(TestDataTime, TestDataForce)
    title('Force vs. Time');
    xlabel('Time (s)');
    ylabel('Force (N)');
    grid on
   % set(gca,'color',[0 0 0])

%This code below is for if a temperature sensor were to be added
% to the apparatus in the future. Shows damper temperature during test.

% subplot(3,1,4
%     plot(TestDataTime, TestDataTemp)
%     title('Damper Temperature vs. Time');
%     xlabel('Time (s)');
%     ylabel('Temperature (C)');
%     grid on
%    % set(gca,'color',[0 0 0])

%Define model parameters for the given adjustment vals:

TestRce =(Rce(2)+Rce(3))./2; %Redundant, as the two curves are being defined by their
different initial Cd's
```

```
TestCdComp = (Cd(3)+Cd(5))./2;

TestCdReb = (Cd(4)+Cd(6))./2;

TestLambdaComp = (Lambda(3)+Lambda(5))./2;

TestLambdaReb = (Lambda(4)+Lambda(6))./2;


%I just defined the model parameters manually by checking what sweeps a high

%speed adjustment of 1.5 would match up with. It would be better to make a

%curve fit of Cd and Lambda (lsqnonlin?) which can be interpolated through for
whatever

%adjustment is specified.


TheoreticalCompEqn = TestCdComp*exp(TestLambdaComp*TestDataLinVel); %Define modeled
curve for compression portion

TheoreticalRebEqn = TestCdReb*exp(TestLambdaReb*TestDataLinVel); % Define modeled
curve for rebound portion


f = figure('Name',PlotTitle, 'NumberTitle', 'off');

set(f,'units','normalized','outerposition',[0,0,1,1]);


        plot(TestDataLinVel, TestDataForce,'.k', 'DisplayName', 'Test Data',
'MarkerSize',20);

        hold on

%
plot(TestDataTime(SweepStarts(i):SweepEnds(i)),TestDataForce_Fit(SweepStarts(i):SweepE
nds(i)),'k');

%       hold on

        plot(TestDataLinVel,TheoreticalCompEqn,'.r', 'DisplayName', 'Theoretical Fit',
'MarkerSize',20);

        plot(TestDataLinVel,TheoreticalRebEqn,'.r', 'DisplayName', 'Theoretical Fit',
'MarkerSize',20);


        grid on

        title('Force vs. Velocity')

        xlabel('Velocity (mm/s)')

        ylabel('Force (N)')

        legend


%   %This code is for if a temperature sensor were to be added to the apparatus in the
future. Produces a heat plot to show how the ...

%   %damper temperature changes throughout a test.
```

```
%
% f = figure('Name',sprintf('Test Data Heat Map: LSC = %s, LSR = %s, HSC = %s, HSR =
%s', TestDataLSC, TestDataLSR, TestDataHSC, TestDataHSR),'NumberTitle','off');
% set(f,'units','normalized','outerposition',[0,0,1,1]);
% dotsz = 25;
%
scatter(TestDataTime(SweepStarts(i):SweepEnds(i)),TestDataForce(SweepStarts(i):SweepEn
ds(i)),dotsz,TestDataTemp(SweepStarts(i):SweepEnds(i)));
% title('Force vs. Velocity, Temperature')
%


%The manufacturer data will now be graphed for each type of adjustment variation.
These plots are being added
%with the intention that as more tests are done with the apparatus in the
%future, their data can be plotted against each of these curves to
%see if they can be validated/reproduced.


f = figure('Name',sprintf('Manufacturer vs. Tested Damping Plots, Ohlins TTX25 Low-
Speed Adjustment Variation'),'NumberTitle','off');
set(f,'units','normalized','outerposition',[0,0,1,1]);




%         plot(,'.g'); %These commented out lines are for adding test data
        hold on
        plot(OhlinsData0_43_0_43Comp(:,1),OhlinsData0_43_0_43Comp(:,2),'r',
'DisplayName','0-4.3 0-4.3');
        plot(OhlinsData0_43_0_43Reb(:,1),OhlinsData0_43_0_43Reb(:,2),'r',
'DisplayName','0-4.3 0-4.3');
%         plot(,'.r');
        hold on
        plot(OhlinsData2_43_2_43Comp(:,1),OhlinsData2_43_2_43Comp(:,2),'b',
'DisplayName','2-4.3 2-4.3');
        plot(OhlinsData2_43_2_43Reb(:,1),OhlinsData2_43_2_43Reb(:,2),'b',
'DisplayName','2-4.3 2-4.3');
%         plot(,'.c');
        hold on
        plot(OhlinsData4_43_4_43Comp(:,1),OhlinsData4_43_4_43Comp(:,2), 'Color',
[0.6350 0.0780 0.1840], 'DisplayName','4-4.3 4-4.3');
        plot(OhlinsData4_43_4_43Reb(:,1),OhlinsData4_43_4_43Reb(:,2), 'Color', [0.6350
0.0780 0.1840], 'DisplayName','4-4.3 4-4.3');
```

```
%         plot(,'.m');

        hold on

        plot(OhlinsData6_43_6_43Comp(:,1),OhlinsData6_43_6_43Comp(:,2), 'Color', [1
0.5 0], 'DisplayName','6-4.3 6-4.3');

        plot(OhlinsData6_43_6_43Reb(:,1),OhlinsData6_43_6_43Reb(:,2), 'Color', [1 0.5
0], 'DisplayName','6-4.3 6-4.3');

%         plot(,'.y');

        hold on

        plot(OhlinsData10_43_10_43Comp(:,1),OhlinsData10_43_10_43Comp(:,2), 'Color',
[0 0.5 0], 'DisplayName','10-4.3 10-4.3');

        plot(OhlinsData10_43_10_43Reb(:,1),OhlinsData10_43_10_43Reb(:,2), 'Color', [0
0.5 0], 'DisplayName','10-4.3 10-4.3');

%         plot(,'.b');

        hold on

        plot(OhlinsData15_43_15_43Comp(:,1),OhlinsData15_43_15_43Comp(:,2),'c',
'DisplayName','15-4.3 15-4.3');

        plot(OhlinsData15_43_15_43Reb(:,1),OhlinsData15_43_15_43Reb(:,2),'c',
'DisplayName','15-4.3 15-4.3');

%         plot(,'.k');

        hold on

        plot(OhlinsData25_43_25_43Comp(:,1),OhlinsData25_43_25_43Comp(:,2),'m',
'DisplayName','25-4.3 25-4.3');

        plot(OhlinsData25_43_25_43Reb(:,1),OhlinsData25_43_25_43Reb(:,2),'m',
'DisplayName','25-4.3 25-4.3');

        grid on

        title('Force vs. Velocity')

        xlabel('Velocity (mm/s)');

        ylabel('Force (N)');

        legend

%         legend('0-4.3 0-4.3', '2-4.3 2-4.3', '4-4.3 4-4.3', '6-4.3 6-4.3', '10-4.3
10-4.3', '15-4.3 15-4.3', '25-4.3 25-4.3')


f = figure('Name',sprintf('Manufacturer vs. Tested Damping Plots, Ohlins TTX25 High-
Speed Adjustment Variation'),'NumberTitle','off');

set(f,'units','normalized','outerposition',[0,0,1,1]);

%         plot(,'.g');

        hold on

        plot(OhlinsData0_43_0_43Comp(:,1),OhlinsData0_43_0_43Comp(:,2),'r',
'DisplayName', '0-4.3 0-4.3');

        plot(OhlinsData0_43_0_43Reb(:,1),OhlinsData0_43_0_43Reb(:,2),'r',
'DisplayName', '0-4.3 0-4.3');
```

```
%         plot(,'.r');

        hold on

        plot(OhlinsData0_3_0_3Comp(:,1),OhlinsData0_3_0_3Comp(:,2), 'Color', [0.75
0.75 0], 'DisplayName', '0-3 0-3');

        plot(OhlinsData0_3_0_3Reb(:,1),OhlinsData0_3_0_3Reb(:,2), 'Color', [0.75 0.75
0], 'DisplayName', '0-3 0-3');

%         plot(,'.c');

        hold on

        plot(OhlinsData0_2_0_2Comp(:,1),OhlinsData0_2_0_2Comp(:,2), 'Color', [0 0.75
0.75], 'DisplayName', '0-2 0-2');

        plot(OhlinsData0_2_0_2Reb(:,1),OhlinsData0_2_0_2Reb(:,2), 'Color', [0 0.75
0.75], 'DisplayName', '0-2 0-2');

%         plot(,'.m');

        hold on

        plot(OhlinsData0_1_0_1Comp(:,1),OhlinsData0_1_0_1Comp(:,2), 'Color', [0.3010
0.7450 0.9330], 'DisplayName', '0-1 0-1');

        plot(OhlinsData0_1_0_1Reb(:,1),OhlinsData0_1_0_1Reb(:,2), 'Color', [0.3010
0.7450 0.9330], 'DisplayName', '0-1 0-1');

%         plot(,'.y');

        hold on

        plot(OhlinsData0_0_0_0Comp(:,1),OhlinsData0_0_0_0Comp(:,2), 'Color', [0.2 0
0], 'DisplayName', '0-0 0-0');

        plot(OhlinsData0_0_0_0Reb(:,1),OhlinsData0_0_0_0Reb(:,2), 'Color', [0.2 0 0],
'DisplayName', '0-0 0-0');

        grid on

        title('Force vs. Velocity');

        xlabel('Velocity (mm/s)');

        ylabel('Force (N)');

        legend


toc
```