OPTIMIZING CONTROL OF SHELL ECO-MARATHON PROTOTYPE VEHICLE TO MINIMIZE FUEL CONSUMPTION

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Electrical Engineering

by

Chad Louis Bickel

April 2017

COMMITTEE MEMBERSHIP

TITLE: Optimizing Control of Shell Eco-marathon Prototype Vehicle to Minimize Fuel Consumption

AUTHOR: Chad Louis Bickel

DATE SUBMITTED: April 2017

COMMITTEE CHAIR: Art MacCarley, Ph.D., PE.
Professor of Electrical Engineering

COMMITTEE MEMBER: Joseph Mello, Ph.D.
Professor of Mechanical Engineering

COMMITTEE MEMBER: Markus Krug, Ph.D.
Lecturer of Electrical Engineering

ABSTRACT

Optimizing Control of Shell Eco-marathon Prototype Vehicle to Minimize Fuel Consumption

Chad Louis Bickel

Every year the automotive industry strives to increase fuel efficiency in vehicles. When most vehicles are designed, fuel efficiency cannot always come first. The Shell Eco-marathon changes that by challenging students everywhere to develop the most fuel-efficient vehicle possible. There are many different factors that affect fuel efficiency, and different teams focus on different vehicle parameters. Currently, there is no straightforward design tool that can be used to help in Shell Eco-marathon vehicle design. For this reason, it is difficult to optimize every vehicle parameter for maximum fuel efficiency.

In this study, a simulation is developed by using basic vehicle models and experimental data to accurately represent any prototype-class vehicle in the Shell Eco-marathon. This simulation is verified using different experimental data from an on-vehicle data acquisition system. An easy-to-use design tool is developed, and this tool is used to optimize driving strategy and final drive ratio to maximize fuel efficiency.

ACKNOWLEDGMENTS

There are many people that helped me in the process of developing and writing this thesis. Thank you Art MacCarley for guiding me throughout the entire process; you always motivated me to do more and challenged me to do everything better. Thank you Dr. Mello for inspiring me to develop a simulation of the Supermileage vehicle, and thank you for supporting me and the team in general. Thank you Dr. Krug for bringing a different perspective to the table and encouraging me to rethink my methodologies.

Thanks to the Electrical Engineering faculty for teaching me all the skills I utilized in this study, and thanks to the Mechanical Engineering department for funding the Supermileage team.

Thanks to all the Cal Poly Supermileage presidents that supported this endeavor: Sean Michel, Dorian Capps, and Lucas Rybarczyk. Thanks to everyone else on the team for helping me out too!

Lastly, I'd like to thank all my friends and family who motivated and supported me throughout this process. I never imagined doing all this, but here I am thanks to all of you.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

NOMENCLATURE

$\dot{m}_i$	Fuel Injector Flow Rate, $mL/fuel\ ms$
η	Final Drive Ratio
γ	Drivetrain Efficiency
ω'	Clutch Shoe Engagement Speed, rad/s or rpm
ω_e	Engine Speed, rad/s or rpm
ω_i	Engine Idle Speed, rad/s or rpm
τ_c	Torque from Clutch, $lbfft$
τ_e	Torque produced by Engine, $lbfft$
θ	Angle of Road, $^\circ$ or rad
$\underline{t}_{RC}$	Time Vector Data, s
$\underline{v}_{RC}$	Speed Vector Data, ft/s
$\underline{x}$	State Variable
$BSFC(\omega_e)$	Brake Specific Fuel Consumption Table, $(slug/s)/(lbfft/s)$
C_c	Clutch Engagement Coefficient, $(lbfft)/(rad^2/s^2)$
C_D	Experimental Coefficient of Aerodynamic Drag, $lbf/(ft^2/s^2)$
C_V	Experimental Coefficient of Wheel and Bearing Resistance, $lbf/(ft/s)$
F_{dt}	Force Exerted by Drivetrain on Vehicle, lbf
f_{PW}	Fuel Injector Pulse Width, $fuel\ ms$
F_{rl}	Road Load Force on Vehicle, lbf
F_{rr}	Experimental Constant of Rolling Resistance, lbf
f_{rr}	Coefficient of Rolling Resistance
g	Acceleration Due to Gravity, ft/s^2
J_e	Engine Mass Moment of Inertia, $slugft^2$

K_ω Starting Engine Acceleration Constant, rad/s^2

K_{m_f} Starting Fuel Consumption Rate Constant, $slug/s$

m_f Total Mass of Fuel Consumed, $slug$

m_v Total Vehicle Mass, $slug$

p Vehicle Position, ft

$poly(x, y)$ Polynomial Regression Function

r Rear Wheel Radius, ft

u Driver Input to Engine

v Vehicle Speed, ft/s

v_{max} Maximum Vehicle Speed, ft/s

v_{min} Minimum Vehicle Speed, ft/s

1 Introduction

This section introduces the problem and gives background information on previous studies in the field of high-efficiency vehicles.

1.1 Background

The Cal Poly Supermileage club designs and builds an extremely efficient vehicle every year. This vehicle is powered by a small internal combustion engine, and it has enough room for one person to drive lying down. Figure 1 shows the 2016 version of the prototype vehicle.

Figure 1: Cal Poly Supermileage 2016 Vehicle

This vehicle competes in the Shell Eco-marathon, in which vehicles must travel 6 miles within 24 minutes using the least amount of fuel possible[17]. The track is

composed of portions of closed-off city streets, which are often uneven and rough. Most drivers in the competition use the "burn and coast" method. This means that a vehicle accelerates until a certain speed has been reached, and then the engine is cut off. The vehicle then coasts until a minimum speed is reached, and the engine is started again. This ensures that fuel is only consumed when acceleration is needed. When the engine is running, the throttle is set at 100%, so the driver only controls whether the engine is running or not.

Designing a new vehicle is challenging; there are many important aspects of the vehicle to consider. These include weight, engine size and power, and aerodynamic efficiency. Historically, the team has sought to minimize weight and road load forces; a small engine has also been used for maximum efficiency. There are often tradeoffs for these parameters. For example, an aerodynamic shape often results in poor driver visibility, which could affect the driver's abilities. The importance of each design parameter has mostly been unknown in the past, resulting in a sub-optimal vehicle. Additionally, the driver does not know what speeds are most efficient to drive at.

1.2 Previous Studies

There have been many previous studies looking at optimizing fuel efficiency in consumer vehicles. Many deal with the design of fuel-efficient vehicles or the determination of optimal highway speeds. However, many of these are not useful when applied to Shell Eco-marathon prototypes, which do not follow traditional vehicle designs or travel at high speeds. One applicable study looked at optimizing vehicle speed and final drive ratio to minimize fuel consumption at low speeds, and

they found that 37% of fuel could be saved with good driving technique and proper gear selection[15]. Therefore, driving strategy should have a significant affect on fuel efficiency for low-speed vehicles.

A very thorough study of a prototype Shell Eco-marathon vehicle was done by the team of Pac-car II. It was one of the most efficient vehicles ever made for the Shell Eco-marathon[16]. Many different aspects of prototype vehicles were analyzed and optimized for fuel efficiency in the study. However, Pac-car II was powered by a hydrogen fuel cell, so many of the results may not be valid for a gasoline-powered vehicle. The most important study in the Pac-car II report was optimizing driving strategy. This study was completed by using a first-order approximation of the theoretical system and then optimizing it using linear programming. In the end, a fairly complicated driving strategy was developed that is difficult for a driver to be able to memorize. Additionally, the complex driving strategy may not be truly optimal due to the approximated system.

A similar study was conducted in which the driver is assumed to be a perfect bang-bang controller[6]. A nonlinear programming method was used to find the optimal controller speeds. The resulting optimal control law is much easier for an actual driver to follow. For this reason, the simulation developed in this study will assume the driver will only need a minimum and maximum vehicle speed, acting like a bang-bang controller. This will simplify the optimization so that no system linearization is required. Additionally, for this study, the final drive ratio will be optimized at the same time as vehicle speeds, which has not been done in previous Shell Eco-marathon studies.

Other studies have been conducted to find the optimal speeds of prototype elec-

tric vehicles[8] and prototype fuel-cell vehicles[13], however these systems are quite different than gasoline-powered vehicles.

Various studies have also made detailed simulations of Shell Eco-marathon vehicles[2][12]. Most of these simulations are either difficult to use or slow. The simulation developed in this study will be much faster and easier to use, making a more practical design tool.

The most closely related study was done by Eric Griess at Cal Poly in 2015[5]. A simulation was developed of the Cal Poly Supermileage vehicle with a focus on engine and powertrain optimization. This simulation had a few uncertainties, including unknown drag coefficient, drivetrain efficiency, and clutch characteristics. Ultimately, a tool to optimize engine torque and BSFC curves was developed, but this does not help in designing other vehicle parameters. However, the engine data gathered will be used in a later part of this study(section 4.4).

2 Thesis Overview

This section presents the scope of this study.

2.1 Objective

The objective of this project is to develop and verify a simulation that can be used to quickly determine the relative importance of major vehicle parameters and optimize the speed and final drive ratio of the vehicle.

2.2 Method

This project will be completed in the following order:

1. Develop a simulation using MATLAB/Simulink
2. Gather real vehicle data using an on-board data acquisition system
3. Analyze data and determine vehicle parameters for simulation
4. Compare simulation outputs to actual vehicle outputs
5. Develop a simple user interface for simulation to allow for quick trade studies
6. Optimize speed and final drive ratio

3 Developing the Vehicle Model

In the following sections, the equations to accurately model the prototype vehicle are developed.

3.1 Objective

The objective of this portion of the project is to develop a model that accurately represents the dynamics of the vehicle and can still be simulated within a few seconds.

3.2 Assumptions

- The track has no turns, so only one positional variable is needed
- The engine intake and coolant temperatures are constant, so thermodynamics can be neglected
- The vehicle weight is constant
- No fluctuating air conditions (wind, temperature, density)
- Perfect driver (no distractions, obstactles, or other vehicles on track)
- Road load fits a second order polynomial
- Drivetrain and wheel inertia is much smaller than the inertia of the entire vehicle
- Engine acceleration and fuel consumption rate are constant before idle speed is reached

3.3 Key Equations

The following equations[4] can be used to represent a vehicle under the above assumptions:

3.3.1 Vehicle Forces

Equation 1 shows the sum of the forces on the vehicle is equal to the force from the drivetrain minus the force from road load:

$$\sum F_v = F_{dt} - F_{rl} \tag{1}$$

The road load force is a function of vehicle speed and road slope[4], shown below:

$$F_{rl}(v, \theta) = C_D v^2 + C_V v + F_{rr} + m_v g sin(\theta) \tag{2}$$

Where:

m_v is the total mass of the vehicle ($slug$)

g is the acceleration due to gravity (ft/s^2)

θ is the angle of the uneven road ($\deg$ or rad)

F_{rl} always opposes the direction of motion of the vehicle. The v^2 term is caused by aerodynamic drag; the v term is primarily caused by bearing rolling resistance and some tire effects; the constant term F_{rr} is caused by traditional rolling resistance from the tires. The last term is caused by the slope of the road. Note that C_D is not the traditional coefficient of drag. C_D in this case includes the frontal area,

drag coefficient, and air density combined into one number in units of $lbf/(ft^2/s^2)$.

Equation 3 shows the relationship between force on the rear wheel and torque from the clutch.

$$F_{dt} = \frac{\tau_c(\omega_e, v)\eta\gamma}{r} \tag{3}$$

Where:

ω_e is the engine speed (rad/s or rpm)

τ_c is the torque from the clutch ($lbfft$)

γ is the drivetrain efficiency (0 to 1)

η is the final drive ratio

r is the rear wheel radius (ft)

Equation 4 shows the relationship between speed and total vehicle force:

$$\dot{v}(t) = \frac{F_v}{m} \tag{4}$$

Equation 5 indicates the relationship between vehicle position and speed:

$$\dot{p}(t) = v(t) \tag{5}$$

3.3.2 Fuel Consumption

Equation 6 shows how fuel consumption rate relates to engine power:

$$\dot{m}_f(t) = \omega_e \tau_e(\omega_e) BSFC(\omega_e) \tag{6}$$

Where:

m_f is the total mass of fuel consumed ($slug$)

τ_e is the torque produced by the engine ($lbfft$)

$BSFC(\omega_e)$ is the Brake Specific Fuel Consumption Table ($(slug/s)/(lbfft/s)$)

When the engine is running, there are three cases to consider:

1. The engine speed is below idle
2. The centrifugal clutch is not fully engaged
3. The centrifugal clutch is fully engaged

3.3.3 Starting Model

When the driver first wishes to start the engine, it takes a small amount of time and fuel to reach idle speed. This could be modeled using an electric starter motor model, however this adds significant complexity to the system without adding much accuracy. For this reason, a constant engine acceleration and fuel consumption rate is assumed.

When $\omega_e < \omega_i$, Equations 7 and 8 are true.

$$\dot{\omega_e} = K_\omega \tag{7}$$

$$\dot{m}_f = K_{m_f} \tag{8}$$

Where:

ω_i is the engine idle speed (rad/s)

K_ω is the starting engine acceleration constant (rad/s^2)

K_{m_f} is the starting fuel consumption rate constant ($slug/s$)

3.3.4 Clutch Not Fully Engaged

For case 2, the engine and vehicle can be considered as 2 separate rigid bodies.

$$J_e\dot{\omega}_e = \sum M_e = \tau_e(\omega_e) - \tau_c(\omega_e, v) \tag{9}$$

Where:

J_e is the mass moment of inertia of the engine ($slugft^2$)

$\sum M_e$ is the sum of the moments acting on the engine ($lbfft$)

Equation 10 relates centrifugal clutch torque to engine speed[14]:

$$\tau_c(\omega_e, v) = \left\{ \begin{array}{ll} C_c(\omega_e - \omega')^2 & \omega_e \geq \omega' \\ 0 & \omega_e < \omega' \end{array} \right\} \tag{10}$$

Where:

C_c is the clutch engagement coefficient ($(lbfft)/(rad^2/s^2)$)

ω' is the engine speed at which the clutch shoe moves forward (rad/s or rpm)

During the conditions of a typical competition run, the engine speed is always greater than the clutch housing speed while the clutch is slipping. When the clutch housing speed reaches the speed of the engine, the dynamic friction between the shoe and the housing becomes static, and the clutch becomes locked. So case 2 is only true when equation 11 is true:

$$\omega_e > \frac{v\eta}{r} \tag{11}$$

The second term in equation 11 is equal to the speed of the clutch housing. This is linked to the rear wheel through the drivetrain.

3.3.5 Clutch Fully Engaged

When equation 11 no longer holds, the clutch is fully engaged. The engine and vehicle are now coupled rigid bodies, so the engine speed and vehicle speed are coupled by the following equation:

$$\omega_e = \frac{v\eta}{r} \tag{12}$$

Since the clutch is fully engaged, the torque on the drivetrain is exactly the torque produced by the engine:

$$\tau_c(\omega_e, v) = \tau_e(\omega_e) \tag{13}$$

3.4 State-Space Model

Before developing a Simulink model, it is useful to show the previous equations in state-space representation. There are 4 state variables in the system:

$$\underline{x} = \begin{bmatrix} p \\ v \\ \omega_e \\ m_f \end{bmatrix} \tag{14}$$

The system output vector is equal to the 4 state variables. There is 1 scalar input to the system, u. Since the driver can only control if the engine is running or not (not the specific throttle %), u is binary: 0 or 1. When u is 0, the engine is off, and the vehicle is coasting. When u is 1, the engine is running, and the vehicle is accelerating at full throttle. Combining the input u and equations 5, 4, 9, and 6 produce the following state equation:

$$\underline{\dot{x}} = \begin{bmatrix} \dot{p} \\ \dot{v} \\ \dot{\omega_e} \\ \dot{m_f} \end{bmatrix} = \begin{bmatrix} v \\ (\tau_c(\omega_e, v)\eta\gamma u/r - (C_D v^2 + C_V v + F_{rr}))/m_v \\ (\tau_e(\omega_e) - \tau_c(\omega_e, v))u/J_e \\ \omega_e \tau_e(\omega_e) BSFC(\omega_e) u \end{bmatrix} \tag{15}$$

However, notice that equation 15 is only valid while the clutch is not fully engaged. Therefore, equation 15 is valid when equation 11 is true. When the clutch

is fully engaged, one of the state variables, ω_e, is lost. By eliminating ω_e, and using equation 12, the following state equation is developed:

$$\begin{bmatrix} \dot{p} \\ \dot{v} \\ \dot{m_f} \end{bmatrix} = \begin{bmatrix} v \\ (\tau_e(\frac{v\eta}{r})\eta\gamma u/r - (C_D v^2 + C_V v + F_{rr}))/m_v \\ \omega_e \tau_e(\frac{v\eta}{r}) BSFC(\frac{v\eta}{r}) u \end{bmatrix} \tag{16}$$

Notice that the system is nonlinear and time-invariant. The system could be linearized by calculating its Jacobian matrix[1], however the system does not stay within a small range of its state variables. In particular, the engine speed varies between 0 to about 7000 rpm, and the vehicle speed can vary between 0 to 40 ft/s. There are a few nonlinear functions that depend on these 2 variables, so linearizing them would reduce accuracy significantly. Therefore, classic optimal control theory cannot be applied[1], and the nonlinear system must be simulated using Simulink.

3.5 Simulation Development

Figure 2 shows the highest level of the simulation. The main plant model has 1 input and 4 outputs. The driver input is controlled by a Simulink block with hysteresis. The input to the block is the vehicle speed, and the output is either 0 or 1. When the vehicle speed is below the minimum speed (a constant), the output switches to 1 until the maximum speed (another constant) is reached. The output then switches to 0 until the minimum speed is reached again.

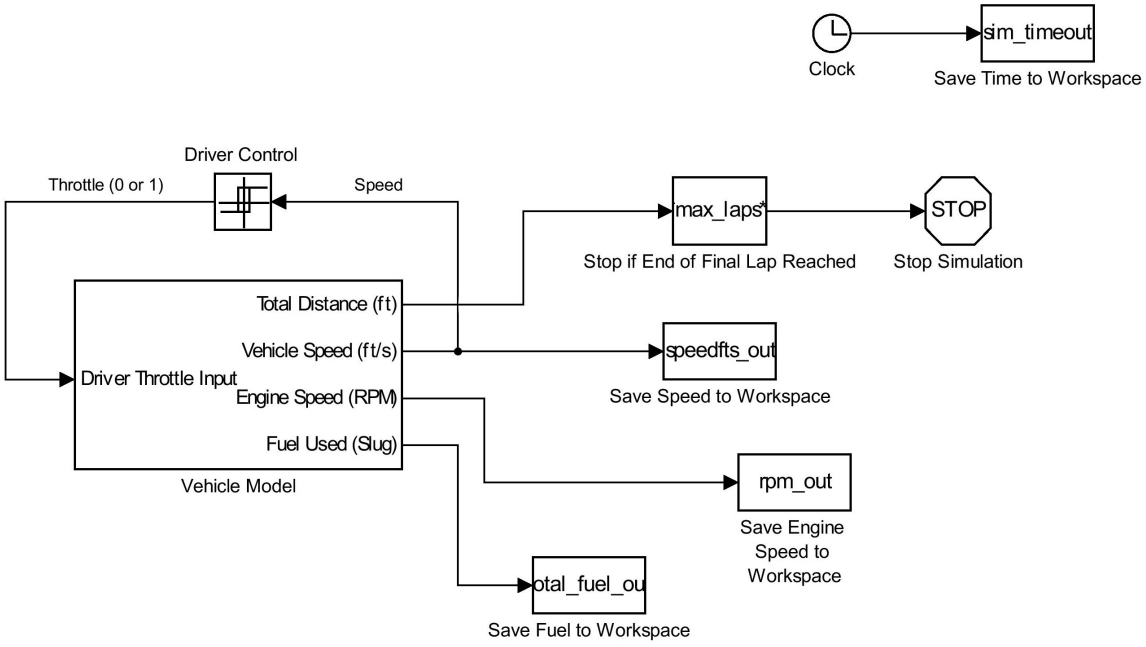

Figure 2: Top-Level View of Simulink Model

The simulation stops after the total distance for the competition has been travelled. For the current competition, this distance is 6 miles. The remaining blocks on this level simply save the outputs of the simulation and put them into another MATLAB workspace.

Figure 3 shows the inside of the plant shown in Figure 2. This level handles the integration of vehicle acceleration to get speed and position. Additionally, speed goes into the Lap Calculation subsystem to calculate the current position of the vehicle on the track. This subsystem is shown in Figure 4

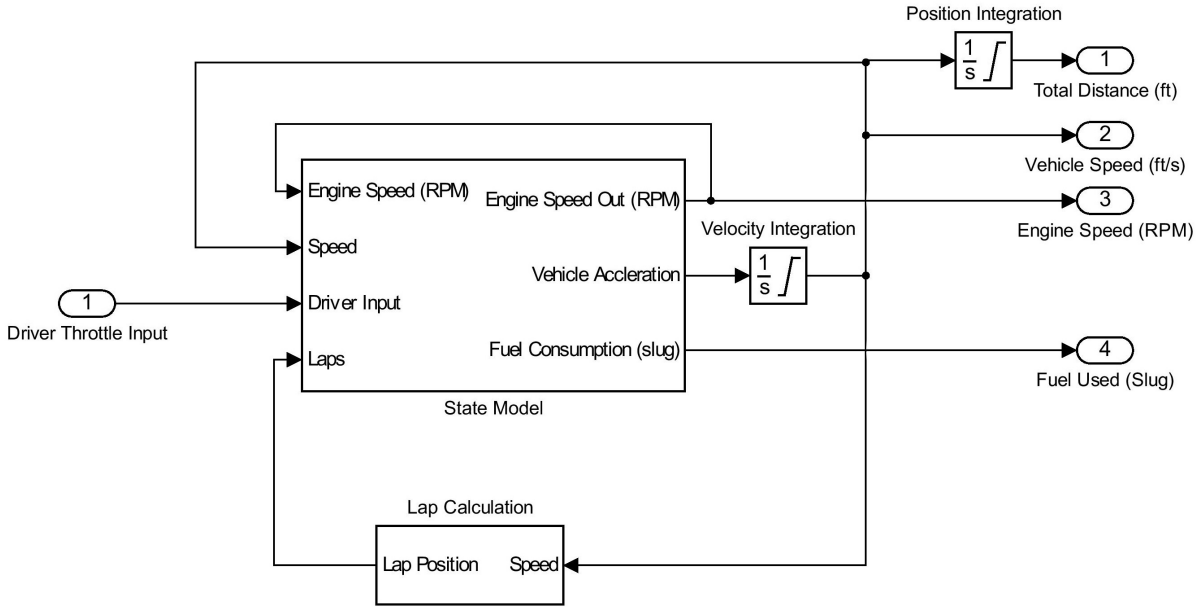

Figure 3: Plant Subsystem of Simulink Model

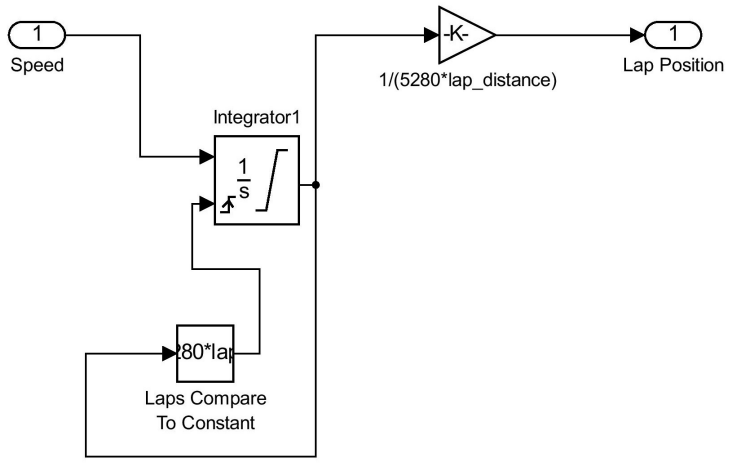

Figure 4: Lap Calculation Subsystem

Speed enters the Lap Calculation subsystem and is integrated to obtain distance travelled in feet. This output is compared to the length of a lap in feet. If the distance is greater than the total distance of a lap, the integrator is reset to 0. The

distance in feet is converted to lap position by dividing by the length of a lap in feet. The lap position output stays between 0 and 1, where 0 is at the start of a lap, and 1 is at the end of a lap.

Figure 5 shows the inside of the State Model subsystem shown in Figure 3. The upper half of this level is dedicated to calculating torque produced by the engine/-clutch and determining how much fuel is used. When the driver input is 0, there is no torque from the drivetrain on the vehicle, and no fuel is consumed. The non-engine forces acting on the vehicle come from the road load, which is calculated in the Road Load Forces subsystem, shown in Figure 6. Drivetrain torque is converted to rear-wheel force through Equation 3. Then, the force from the engine is subtracted from the road load force and then divided by the total mass of the vehicle to get vehicle acceleration.

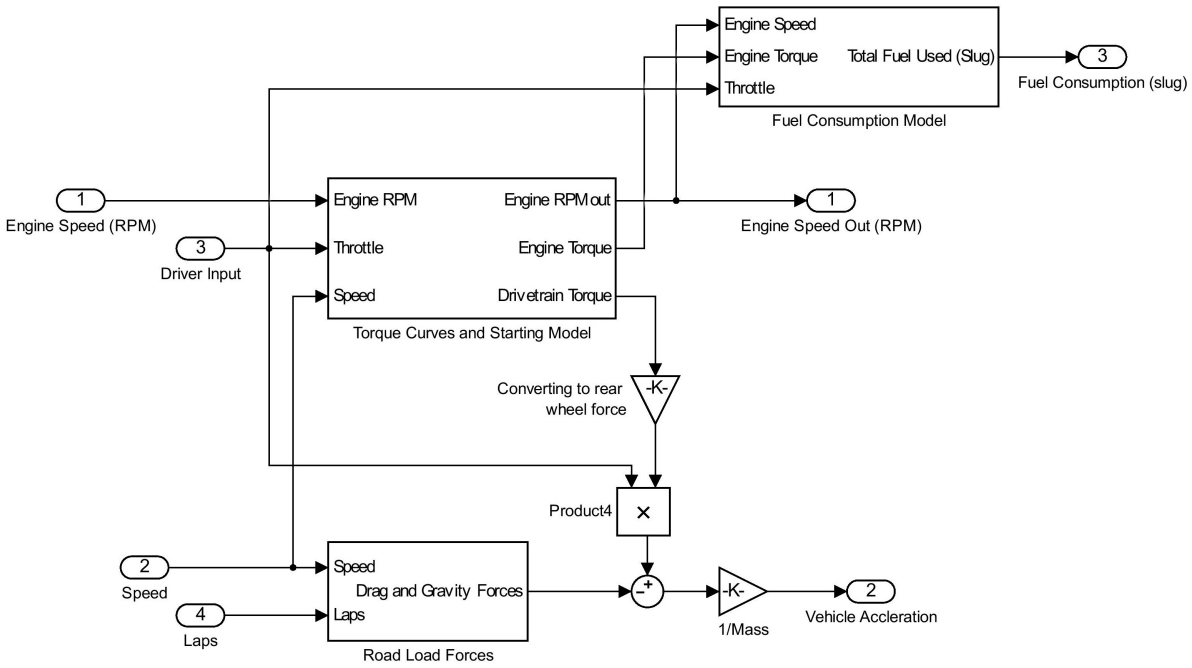

Figure 5: State Model Subsystem

Figure 6 shows how the road load forces are calculated in the Road Load Forces subsystem. Speed enters a lookup table generated from the first 3 terms of the road load equation(2), which outputs the road load force. The lap position (0 to 1) enters a lookup table that represents the last term of the road load equation(2), which outputs the road load force from the slope of the track.

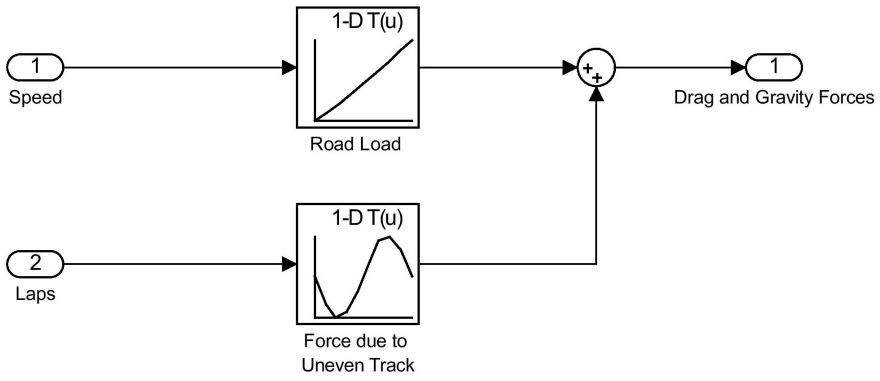

Figure 6: Road Load Forces Subsystem

Figure 7 shows the Fuel Consumption Model subsystem. If the input engine speed is less than ω_i, the fuel consumption rate is constant. When engine speed is above ω_i, engine speed is used as an input into a BSFC table. The output is in units of $lbm/(hp * hr)$. This is multiplied by engine power in units of $lbf * ft * rpm$. This is converted to $slug/s$ by dividing the output by $g * 3600 * 5252$. That output is multiplied by the input (so the output is nonzero only when the input is 1) and integrated to get total fuel consumed in slugs.

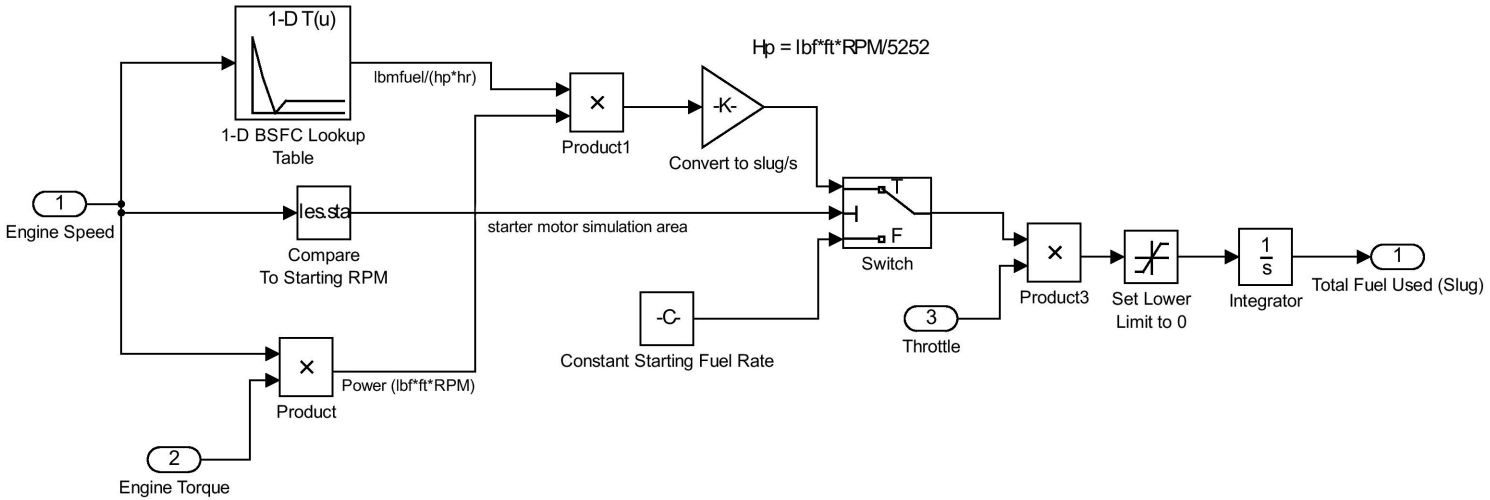

Figure 7: Fuel Consumption Model Subsystem

Figure 8 shows the inside of the Torque Curves and Starting Model subsystem. In the bottom left of the figure, engine speed is compared to ω_i. If the engine speed is less than idle, the engine acceleration is a constant. Otherwise, engine speed is used as an input to the engine torque lookup table. The clutch torque is calculated from the Clutch Curve subsystem table, which is subtracted from engine torque, as in Equation 9. That torque is then divided by J_e to get engine acceleration in rad/s^2. The engine acceleration is integrated through an integrator block that resets to 0 when the driver input changes from 0 to 1. The engine speed output from the integrator is converted to rpm, and then it is output if the clutch is not fully engaged. If the clutch is fully engaged, the engine speed is calculated from vehicle speed using Equation 12. In the upper right corner of the figure, the output torque to the drivetrain is switched depending on whether the clutch is fully engaged or not.

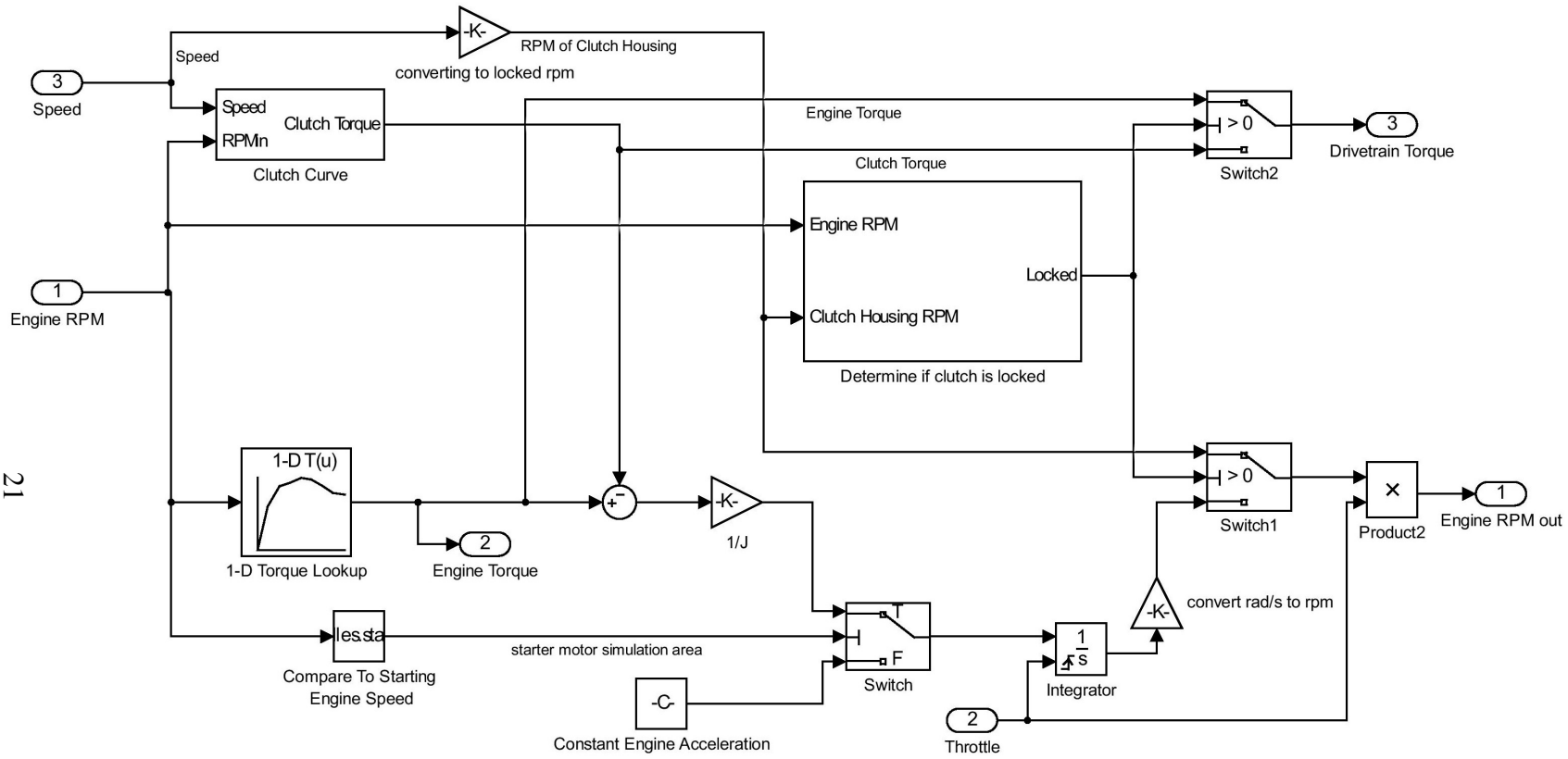

Figure 8: Torque Curves and Starting Model Subsystem

Figure 9 shows the inside of the Clutch Curve subsystem. At the bottom of the subsystem, engine speed is used to calculate clutch torque using a lookup table generated from Equation 10. The upper part of the subsystem calculates the difference between the engine speed and clutch housing speed. If the engine speed is less than the clutch housing speed, no torque should be placed on the drivetrain, so the friction term is 0. Once engine speed is greater than the clutch housing speed, the friction term is 1.

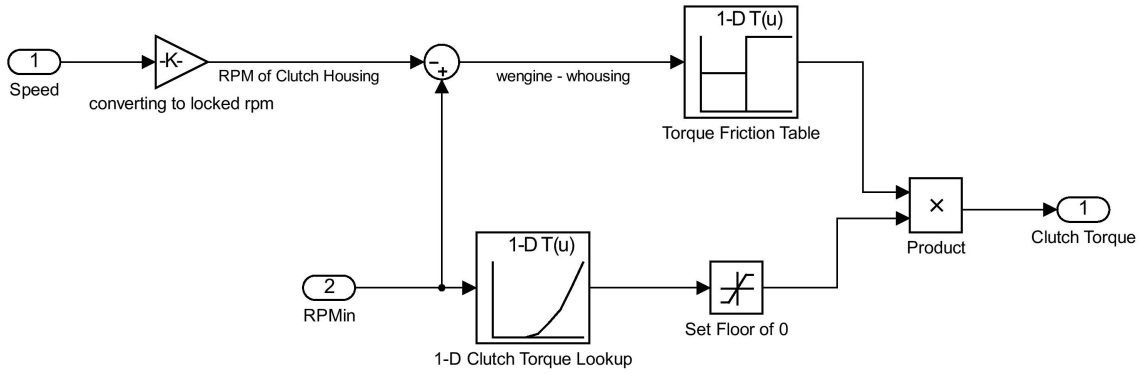

Figure 9: Clutch Curve Subsystem

Figure 10 shows the inside of the Clutch Locking subsystem. At the bottom of the subsystem, the engine speed is compared to the clutch locking speed. This clutch locking speed is calculated before a simulation begins by determining the engine speed at which clutch torque equals engine torque. If the engine speed is much less than the locking speed, the clutch will not be engaged. If the engine speed is close to the locking speed, the top part of the subsystem is used. The magnitude of the difference between engine speed and clutch housing speed is determined and then multiplied by negative 1. It must be multiplied by negative 1 since Simulink's hysteresis block needs the top limit to be higher than the bottom limit. Once the

difference between the clutch housing speed and engine speed becomes very small, the clutch is now “locked” and it will not unlock until the engine speed drops below the locking speed.

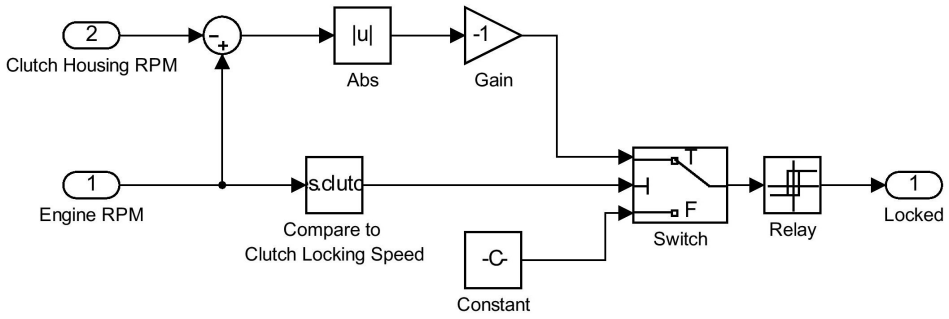

Figure 10: Clutch Locked Determination Subsystem

Once the simulation development is complete, data must be gathered to determine the various tables and coefficients.

4 Analyzing Data to Build and Validate Model

There are 2 main methods of of determining the various coefficients and constants in the Simulink model.

One method is to use detailed equations and models that can predict those coefficients from several inputs. Examples include: the aerodynamic force coefficient could be determined by using computational fluid dynamics, engine torque and BSFC curves could be determined using detailed engine models, and clutch coefficients could be determined by analyzing the geometry of the clutch. The issue with this method is the inaccuracies associated with using more models and more inputs. Each model requires different assumptions that are not completely true, and each input carries a certain amount of uncertainty, which can be amplified after a complex calculation. Additionally, if the coefficients were determined this way, and the resulting simulation did not match up with real conditions, it would be difficult to determine which model or assumption was incorrect.

A more intuitive and accurate method is to gather data using on-vehicle data acquisition systems, and utilize that data to determine the various coefficients. This method can also account for some unpredictable dynamics. Also, it is difficult to dispute the accuracy of the coefficients when they were gathered from actual data, not from simplified models and equations. For these reasons, the second method was chosen moving forward.

4.1 Data Acquisition System

The primary approach to gather data is using the RaceCapture[7] data acquisition system. Figure 11 shows RaceCapture mounted on the firewall of the Supermileage vehicle.

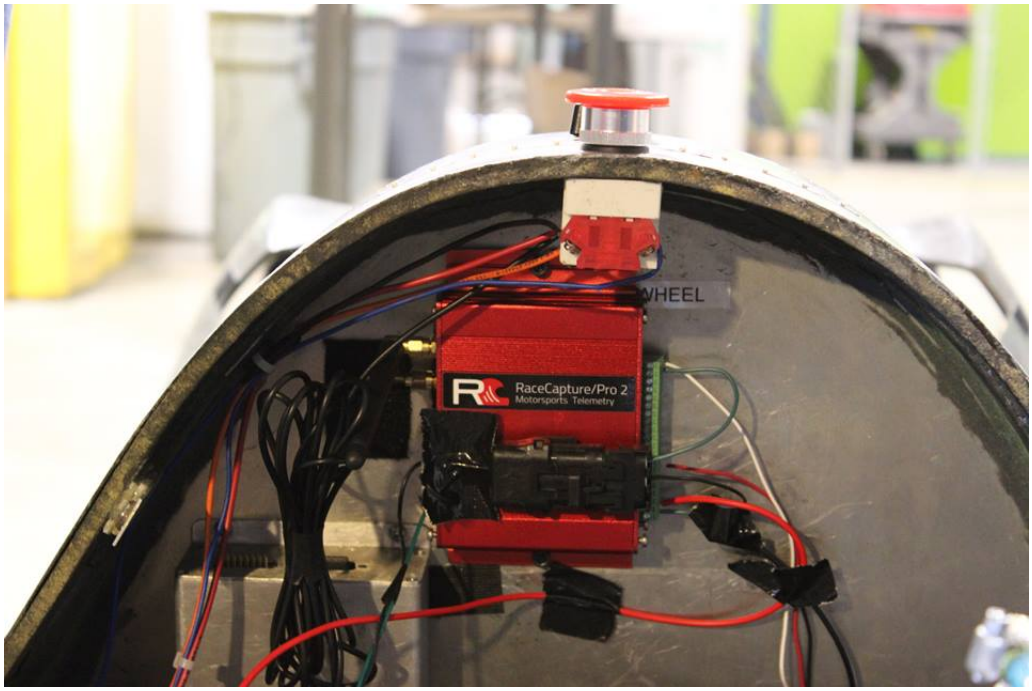

Figure 11: RaceCapture in Vehicle

RaceCapture has a few built-in sensors, and also has inputs for external sensors. Additionally, it has a CAN port, which is used to capture data from the Supermileage engine control unit. This data is logged to a micro-SD card and processed later using MATLAB. This data was gathered at a sampling rate of 50 Hz. A list of some of the important data logged is shown below:

- Acceleration (from 3 axis accelerometer)

- Roll, Pitch, Yaw (from 3 axis gyroscope)
- Engine Timing Advance
- Air to Fuel Ratio
- GPS (Latitude, Longitude, Altitude)
- Battery Voltage
- Engine Coolant Temperature
- Fuel Injector Pulse Width Duration
- Engine Speed
- Vehicle Speed (from GPS data)
- Time

Vehicle speed is one of the most important parameters that is measured, however it had to be measured by differentiating position data from the GPS sensor. RaceCapture has a built-in algorithm to determine speed, which has filtering and generally works fairly well. There is more noise in the output than there would be if a rear-wheel Hall effect sensor was used, but it was not feasible or necessary at the time to install one.

Figure 12 shows an example of speed logged using RaceCapture. Raw speed data is logged in blue, while filtered speed is shown in red. The data is filtered using an FIR moving average filter of length 100. An FIR filter was used so the time delay

associated with the filter is constant, and the large number of terms was used since any useful dynamic frequencies are at very low frequencies.

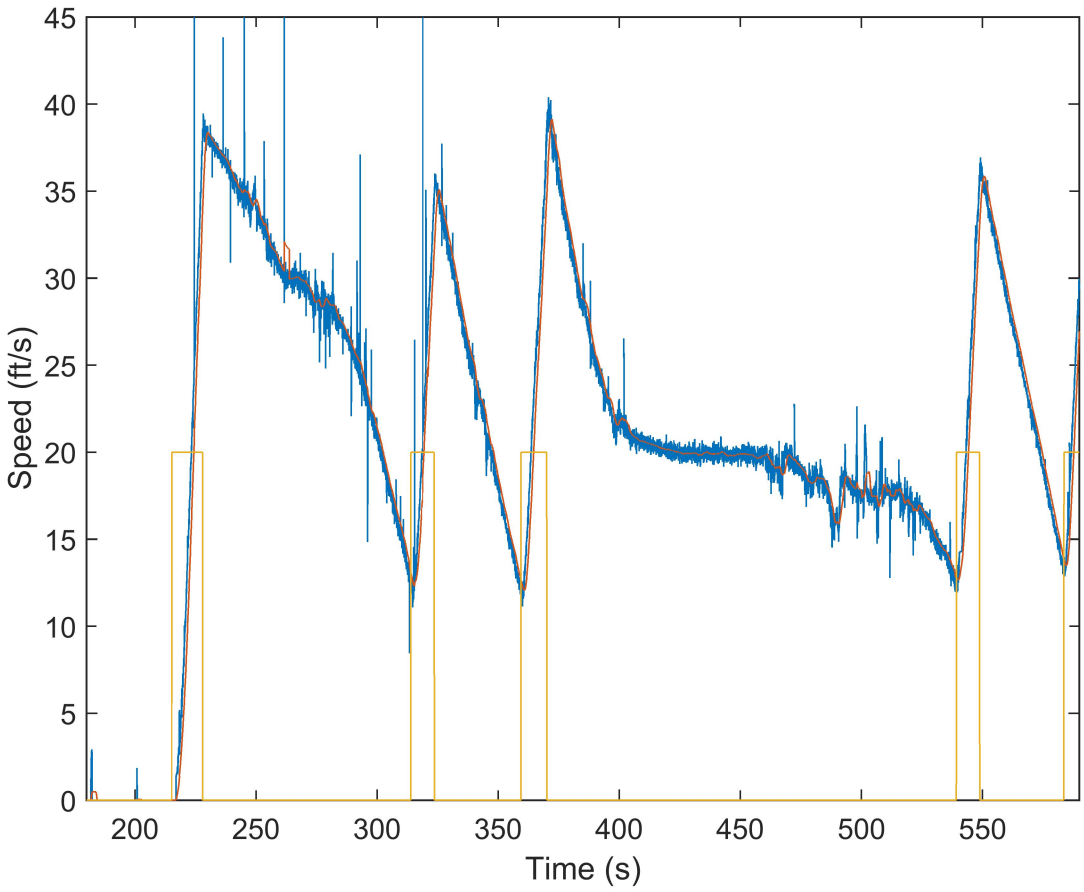

Figure 12: Speed Data Gathered from RaceCapture

In order to properly analyze the logged data, a simple algorithm was developed to automatically identify when the driver input was 0 or 1. Essentially, if the engine speed is above 0, and the engine speed is increasing, the driver input is 1 at that moment. Otherwise, the driver input is 0. The driver input is plotted in yellow in Figure 12. Note that the input of 0 or 1 has been multiplied by 20 to scale better

with the rest of the data in Figure 12.

Figure 13 shows an example of engine speed and pulse width data taken from the same run as Figure 12. This type of data will be used in the following sections to calculate various coefficients and verify the simulation.

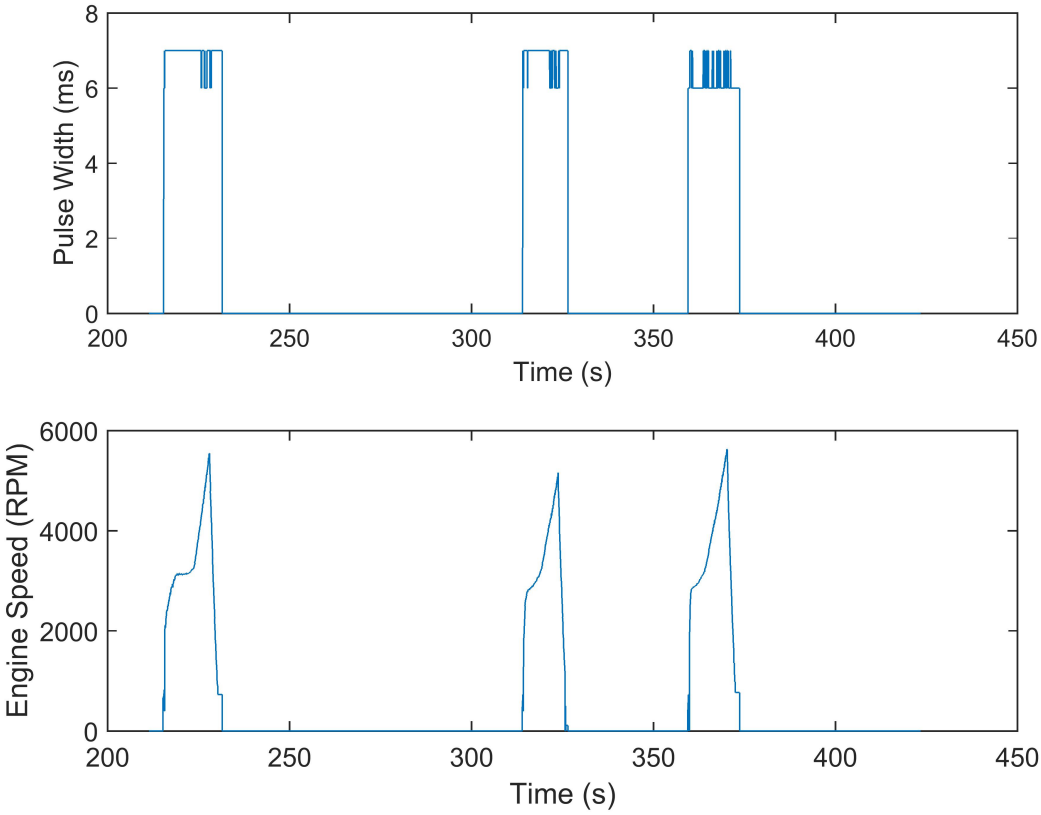

Figure 13: Engine Speed and Pulse Width Data Gathered from RaceCapture

4.2 Road Load Coefficients

The first important parameters to determine are the road load coefficients. Various studies have used coastdown testing to determine road load coefficients with a fair amount of accuracy[8]. When the vehicle is coasting, only the road load forces

are acting on it. Therefore, to determine the coefficients, only the coasting down portions of the data will be considered. The vehicle must also travel at very high and very low speeds to see significant differences in road load force. The most straightforward way to measure road load force would be to use the RaceCapture accelerometer and multiply the measured acceleration by the mass of the vehicle to get force. However, RaceCapture is mounted very close to the engine, so vibrations make the accelerometer data much too noisy to measure small changes accurately. Therefore, speed must be used directly to measure road load.

Acceleration can be attained by using polynomial regression to fit a polynomial to speed as a function of time. This polynomial can then be differentiated to get acceleration as a function of time. Road load force is a second-order polynomial function of speed if the slope of the track is 0°(Equation 2). For this reason, acceleration can be fit to a second-order polynomial of speed, and then multiplied by total vehicle mass to get road load force as a function of speed. Equations for this process are shown below:

$$\frac{d}{dt}(poly(\underline{t}_{RC}, \underline{v}_{RC})) \rightarrow \dot{v}(t) \tag{17}$$

Where:

$\underline{t}_{RC}$ is the time vector gathered from RaceCapture (s)

$\underline{v}_{RC}$ is the speed vector gathered from RaceCapture (ft/s)

$poly(x, y)$ is the polynomial regression function

$$\rightarrow m_v \cdot poly(\underline{v}_{RC}, \dot{v}(\underline{t}_{RC})) \rightarrow F'_{rl}(v) \tag{18}$$

Where:

$F'_{rl}(v)$ is the estimated road load force equation

The primary problem with this process is that a perfectly flat track is required, and Cal Poly Supermileage does not have access to a high-speed, flat track. As seen in Figure 12, coastdowns are not consistent due to variations in the track. A better way to visualize the uneven track is in Figure 14, which shows various coastdowns on different portions of the track. Notice the vehicle decelerates differently depending on where it is. Between 0 and 0.3, the vehicle is on a slight downhill. Between 0.4 and 0.9, the vehicle is on a slight uphill and decelerates faster.

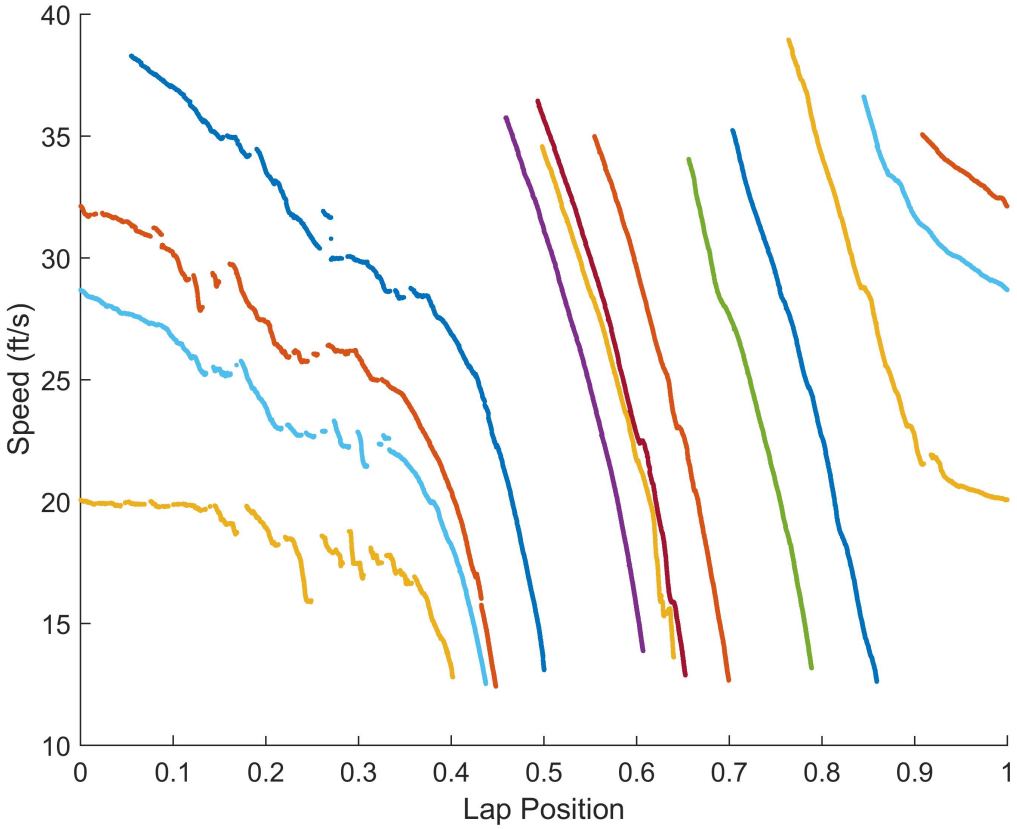

Figure 14: Logged Speed versus Lap Position

Since the track is uneven, the slope must be calculated at every point along the track to correct for it. Altitude data from GPS would be straightforward, however the altitude data gathered is noisy and inaccurate, shown in Figure 15. Even with heavy filtering, the data is too noisy to find small changes in slope.

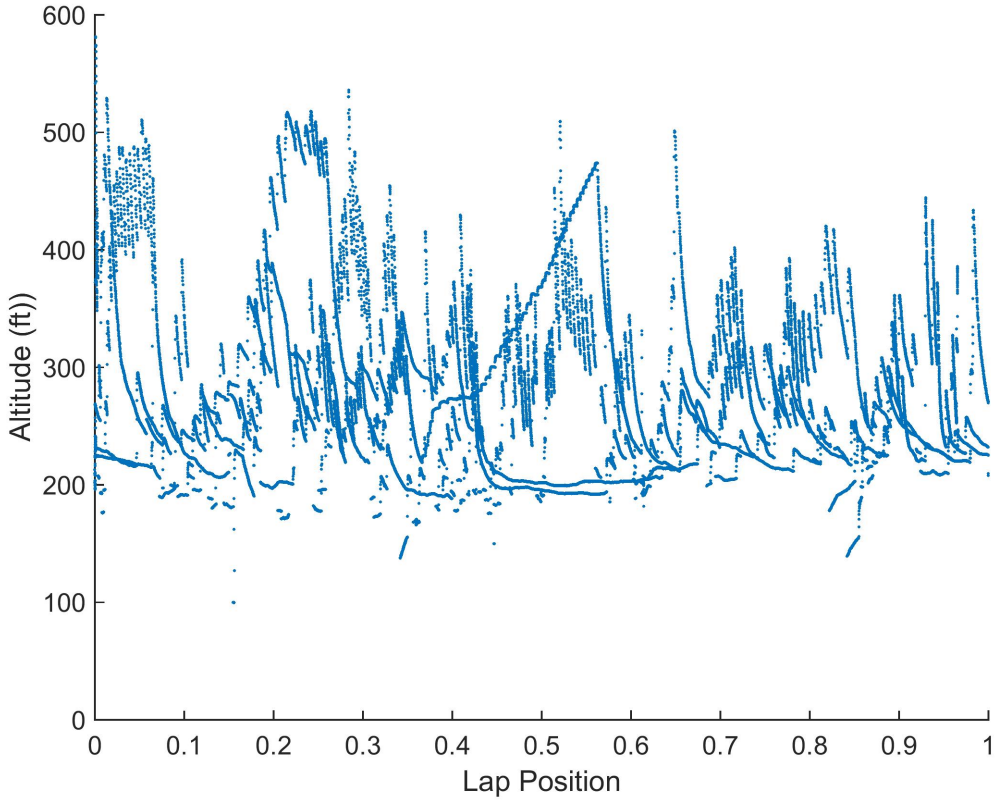

Figure 15: Altitude versus Lap Position

Therefore, an alternate source must be used to determine the slope of the track. Google Earth is an easy tool to use, and it provides an elevation profile along a track. However, the altitude accuracy from Google Earth is also limited. It uses SRTM data to determine altitude, and SRTM has a 90% linear error probable of 98.4 ft according to U.S. Military Specification [MIL]-PRF-89020B [18]. Google Earth data for the high-speed Supermileage testing track in Lompoc, CA, is shown in Figure 16.

Figure 16: Elevation Profile of Test Track

The maximum and minimum altitudes from this data are 169 ft and 194 ft respectively, which seem more reasonable than the noisy GPS data. This data also matches the previous observation of a downhill portion from 0 to 0.3 laps, and an uphill from 0.4 to 0.9 laps. However, the intricate curves on the Google Earth data might not actually be there due to the inaccuracies. Additionally, the process shown in Equations 17 and 18 does not work if the slope is changing over time or distance. Therefore, a constant uphill and constant downhill slope will be assumed for the 2 main portions of the track. This way, Equations 17 and 18 can be followed, and then corrected using Equation 19.

$$\hat{F}_{rl}(v) = F'_{rl}(v) - m_v g \cdot sin(\theta_{const}) \tag{19}$$

Where:

$\hat{F}_{rl}(v)$ is the corrected, estimated road load equation

In order to have a constant slope, the uphill and downhill portions must be analyzed separately. However, when the vehicle is on a downhill section, it takes significantly longer to coast down to a low speed. As shown in Figure 14, most downhill coastdowns end up going into part of the uphill section of the track. For this reason, it is more accurate to simply use the coastdowns from 0.5 to 0.9 lap position, where the vehicle is constantly going uphill. From 0.5 to 0.9 lap position, the vehicle increases 18 ft in altitude (from Google Earth), and it travels 1848 ft. This corresponds to a slope of about 1% or 0.56°, which is equal to about 2.4 lb of force on the vehicle. Now, Equations 17 and 18 can be applied.

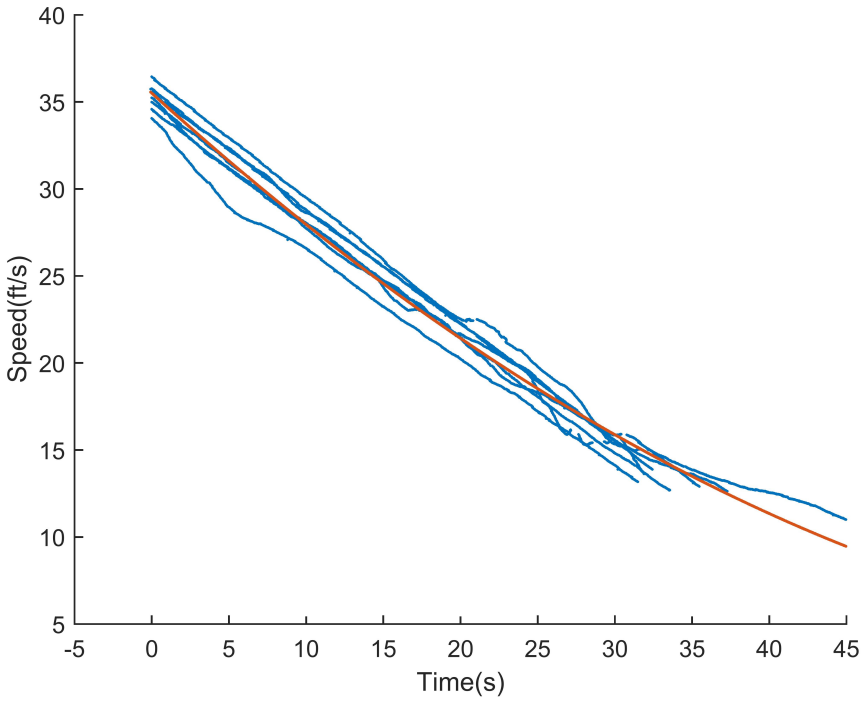

Figure 17: Coastdown Velocity Data from 0.5 to 0.9 Lap Position

Figure 17 shows all coastdown data on the uphill portions put on the same timescale. The blue lines are the various coastdowns, while the red line is the best fit curve, $poly(\underline{t}_{RC}, \underline{v}_{RC})$. After using the best fit line in Equations 18 and 19, the output is shown in Figure 18. The red line shows the second-order best fit polynomial, and the blue line is the raw acceleration versus speed data (which is beneath the red line).

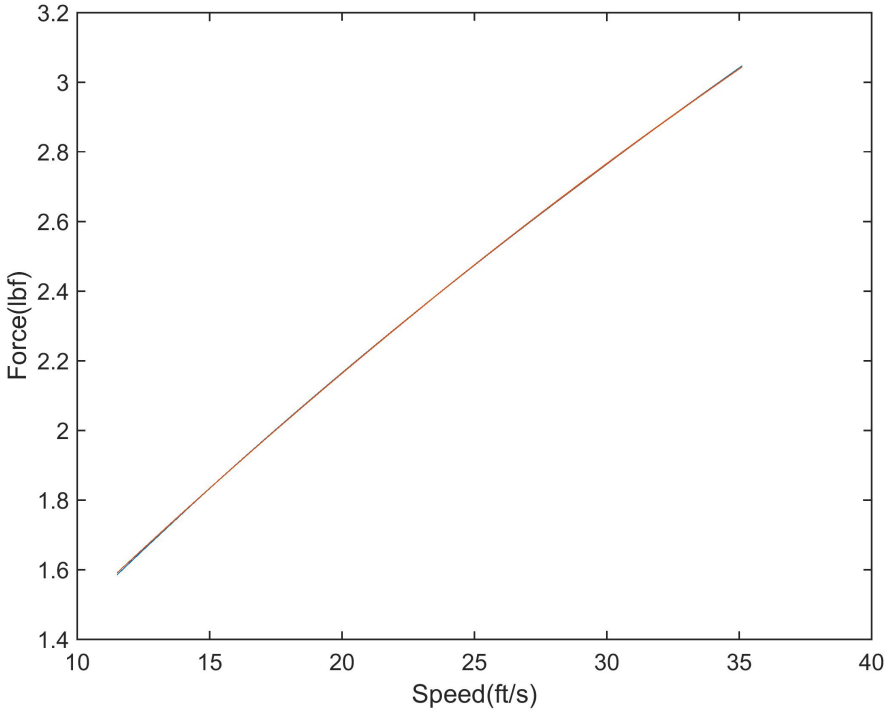

Figure 18: Road Load versus Vehicle Speed

The resulting coefficients are:

C_D = -0.000393 $lbf/(ft^2/s^2)$

C_V = 0.0799 $lbf/(ft/s)$

F_{rr} = 0.7243 lbf

Compared to the slope of the track, which applies 2.4 lbf of force to the vehicle, the calculated F_{rr} value of 0.7243 lbf is fairly small. This means that any inaccuracies in slope calculation can have a dramatic effect on the final calculated F_{rr} value. In the future, a more accurate way to measure road slope is necessary.

Also, notice that C_D is negative, meaning the vehicle has negative aerodynamic drag. This is impossible and most likely caused by a combination of errors includ-

ing changing wind speed and inaccurate speed measurements. The primary ways to get a more accurate drag number would be to install a rear-wheel Hall effect sensor and air speed sensor to more accurately capture vehicle and air speed. Additionally, Cal Poly Supermileage may have access to a wind tunnel to measure drag from a scaled-down model vehicle in the near future.

However, since a more accurate drag number is needed now, a different source will be used. Pac-car II, one of the most efficient vehicles ever made for the Shell-Eco Marathon, has an effective C_D of 0.0002437 $lbf/(ft^2/s^2)$ [16]. This sets the standard as the lowest drag number in the Shell-Eco Marathon, and the Supermileage vehicle is likely not very close. Computational fluid dynamics analysis is being performed on the current vehicle, and it indicates that the C_D of the Supermileage vehicle is about 3 times that of Pac-car II. The number may also be larger, since CFD will not capture many aerodynamic effects that come up in actual testing. For this reason, the number of 0.0007311 $lbf/(ft^2/s^2)$ will be assumed for further analysis.

4.3 The Affect of Weight on Road Load

In order to accurately determine how weight affects fuel efficiency, its affect on the road load coefficients must be found. Theoretically, increasing weight only increases the last term of the road load equation, F_{rr} [4], as shown in Equation 20.

$$F_{rr} = f_{rr} m_v g \tag{20}$$

Where:

f_{rr} is the coefficient of rolling resistance

f_{rr} does increase slightly with weight due to increased tire contact patch, but it is assumed to be very small compared to the effect of increasing weight itself. Therefore, F_{rr} should increase linearly with vehicle mass, m_v.

Under these assumptions, a test was performed with two different vehicle weights. Since only the F_{rr} constant is affected, a low-speed coastdown test is sufficient. For the test, the vehicle was placed at the bottom of a slight uphill portion of the track. It was brought up to about 20 ft/s and then allowed to coast down to 0 ft/s. The same method as shown in Equation 17 and 18 was used to determine the coefficients, except acceleration was fit to a first-order polynomial. This is because aerodynamic drag is negligible at lower speeds. This process occurred 3 times at the base vehicle weight, and 3 more times with 60.4 lb added. The uphill track did not have to be corrected for, since only the difference in rolling resistance force needed to be found.

Figure 19 shows the results of this testing.

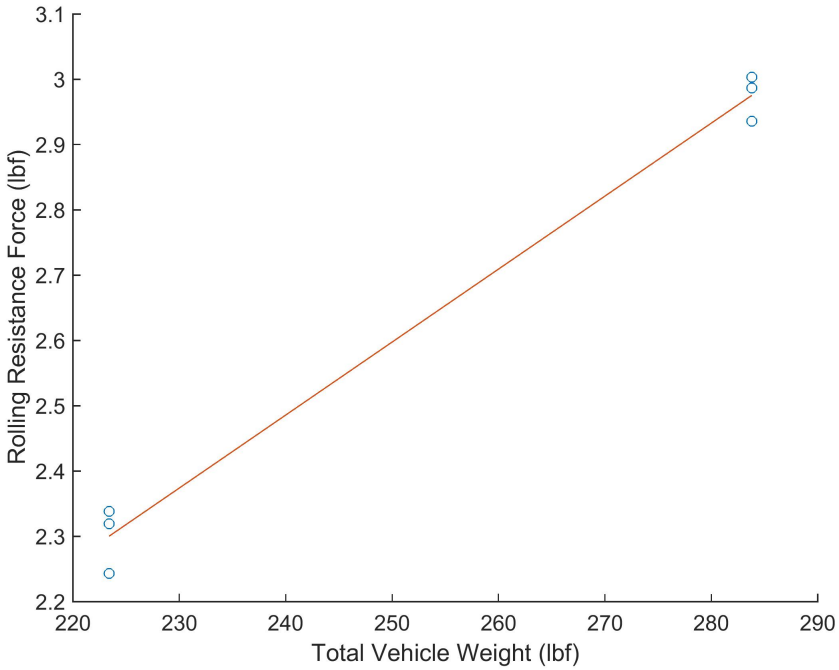

Figure 19: F_{rr} versus Total Vehicle Weight

The standard deviation of the first 3 data points is 0.0502, while the standard deviation of the last 3 are 0.0351. The red line is the best fit curve for the data. The following equation is the best fit polynomial:

$$\delta F_{rr} = 0.3596 \cdot \delta m_v \tag{21}$$

Where:

δF_{rr} is the change from initial F_{rr}

δm_v is the change from initial m_v

To calculate a new F_{rr} from an old m_v and F_{rr}, Equation 22 can be used, which is derived from Equation 21.

$$F_{rr\ new} = 0.3596 \cdot (m_{v\ new} - m_{v\ old}) + F_{rr\ old} \tag{22}$$

4.4 Engine Torque and BSFC Curves

It is possible to determine engine torque and BSFC curves from RaceCapture data, however these are more accurately determined on an engine dynamometer. A previous study determined the engine curves for the current engine and tune[5]. The BSFC data can be verified by looking at the fuel pulse duration and engine speed data.

First, the fuel consumption rate is calculated from data using Equation 23.

$$\dot{m}_f(t) = \omega_e(t)/(2\pi)/2\ (cycle/s) \cdot f_{PW}(t)(fuel\ ms) \cdot \dot{m}_i(mL/fuel\ ms) \tag{23}$$

Where:

f_{PW} is the injector pulse width in $fuel\ ms$

$\dot{m}_i$ is the fuel injector flow rate $mL/fuel\ ms$

The theoretical fuel flow rate can be calculated using Equation 6. Figure 20 shows the theoretical fuel flow rate versus engine speed in black. The blue points are experimental data, and the red line is the best fit polynomial for the experimental data.

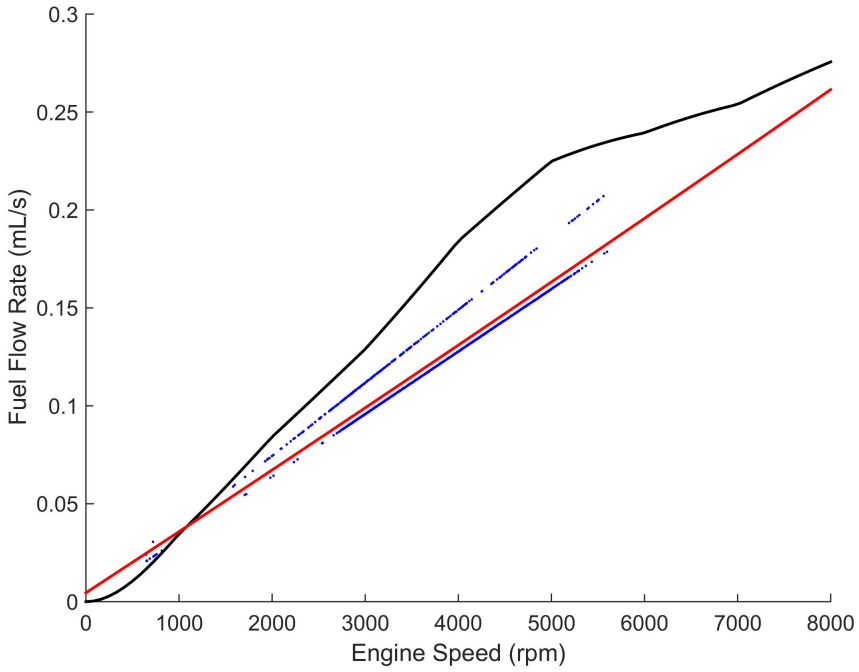

Figure 20: Fuel Flow Rate versus Engine Speed

Notice that the theoretical fuel consumption is somewhat higher than the actual fuel consumption. However, it still follows the trend of the experimental data. The injector pulse width was nearly constant during testing, so fuel flow rate has a linear relationship with engine speed. For the rest of the analysis, the best fit polynomial from the experimental data will be used for more accurate results.

The torque data cannot be verified using the available data; it is used to determine drivetrain efficiency in the next section.

4.5 Drivetrain Efficiency

Typically, to determine drivetrain efficiency, a chassis dynamometer is used. Other studies have used a chassis dynamometer to successfully measure drivetrain effi-

ciency from a Shell Eco-marathon vehicle[3]. This is the best way to acquire accurate data. However, no chassis dynamometer was available at the time of this study, so data gathered from RaceCapture will be used. To determine the drivetrain efficiency, only the data while $u = 1$ is used. Figure 21 shows how engine speed changes with time; each color is from a different burn.

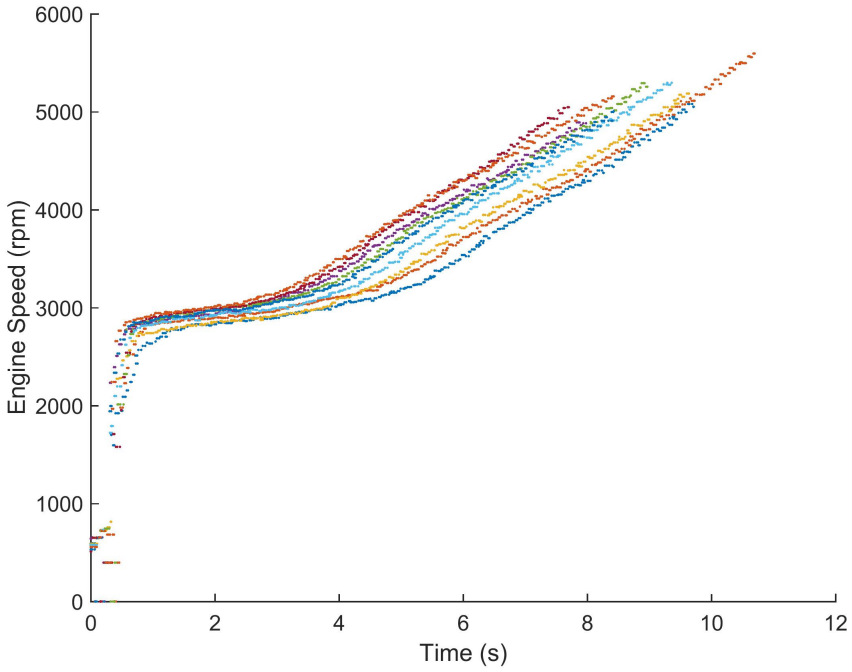

Figure 21: Engine Speed versus Time

Figure 22 shows the clutch housing speed over time. This speed was calculated using Equation 12 from the filtered GPS speed data. Notice that the engine speed and clutch housing speed nearly match around 4 seconds on. This is when the clutch is fully locked and engaged. Therefore, for the efficiency analysis, only the data after the 4 second mark will be used. For the clutch torque determination (next section), the data before the clutch is fully engaged will be used.

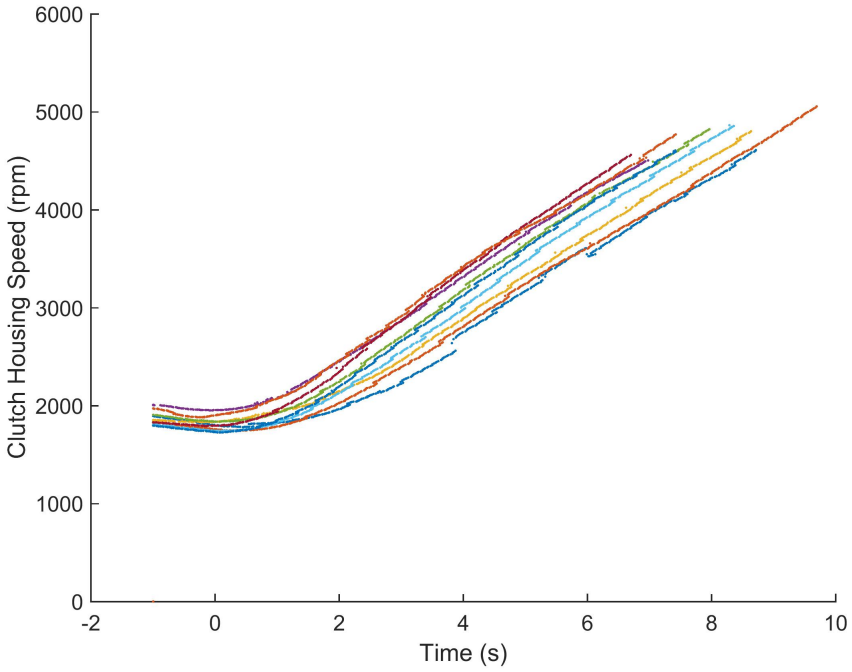

Figure 22: Clutch Housing Speed versus Time

Using the engine torque data from the previous study[5], it is possible to calculate the theoretical acceleration on the rear wheel (assuming 100% efficiency), by using the following equation:

$$\dot{v}_{th} = (\frac{\tau_e(\omega_e)\eta}{r} - F_{rl})/m_v \tag{24}$$

Where:

$\dot{v}_{th}$ is the theoretical vehicle acceleration

Notice that the equation requires an accurate road load force, so the road load found previously will be used.

To get the actual acceleration, the discrete derivative can be used on the filtered

GPS speed data. There is no need to use least squares regression as before, since the magnitude of the acceleration when burning is much larger than the sensor noise.

Figure 23 shows the theoretical vehicle acceleration in black, while the rest of the colors are experimental acceleration.

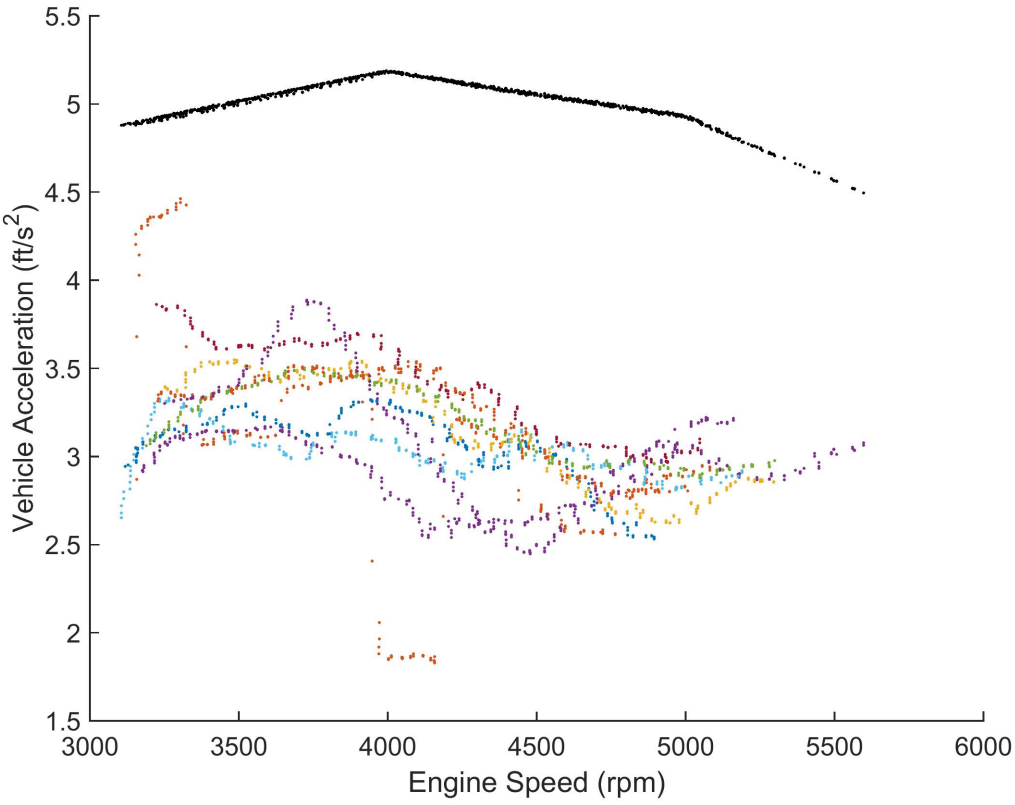

Figure 23: Vehicle Acceleration versus Engine Speed

Notice that both the theoretical and experimental accelerations follow the same trend, but the experimental acceleration seems to be a constant fraction of the theoretical. The drivetrain efficiency can be calculated at every point using the following equation:

$$\gamma = 1 - \frac{\dot{v}_{th} - \dot{v}_{ex}}{\dot{v}_{th}} \tag{25}$$

Where:

$\dot{v}_{ex}$ is the experimental vehicle acceleration

The result can be seen in Figure 24.

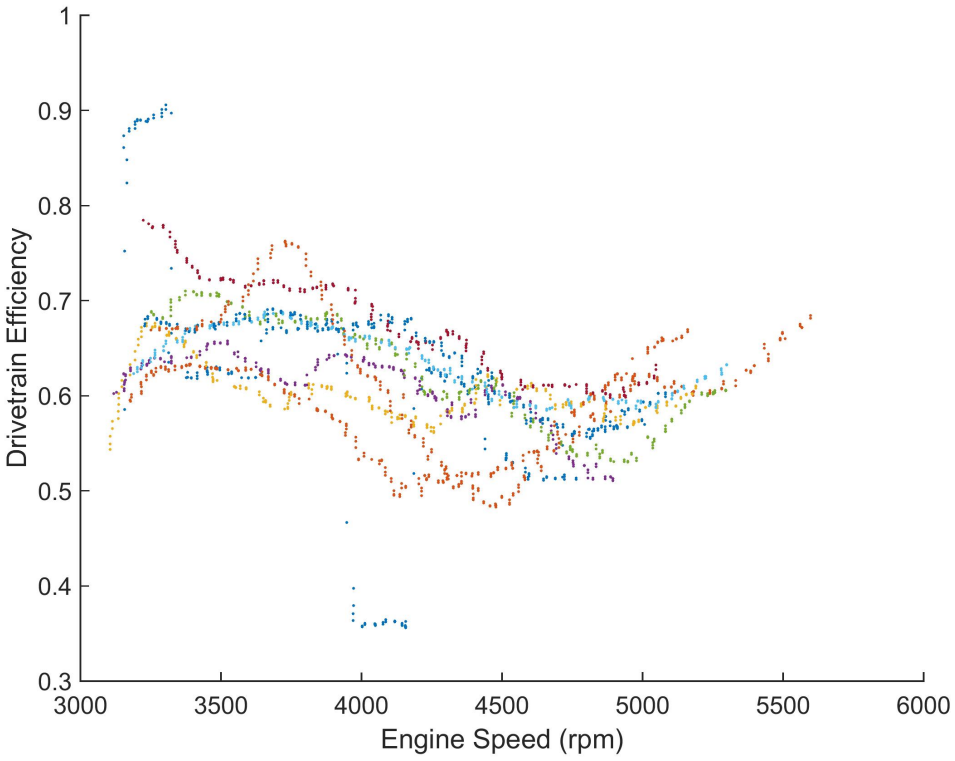

Figure 24: Drivetrain Efficiency versus Engine Speed

γ is relatively constant, so the average of all efficiencies will be used as the drivetrain efficiency. This average is 62.1%.

4.6 Clutch Torque Curve

The next step is to find the unknown coefficients of the clutch torque equation (10), C_c and ω'. Torque from the drivetrain can be calculated experimentally using Equation 26 (derived from Equation 3):

$$\tau_c(\omega_e) = \frac{\dot{v}_{ex} m_v r}{\eta \gamma} \tag{26}$$

A γ of 0.621 will be used, as calculated from the previous section. Once $\tau_c(\omega_e)$ has been determined over a range of engine speeds, polynomial regression can be used to fit the data to a second-order polynomial. Figure 25 shows the experimental clutch torque versus engine speed from various accelerations.

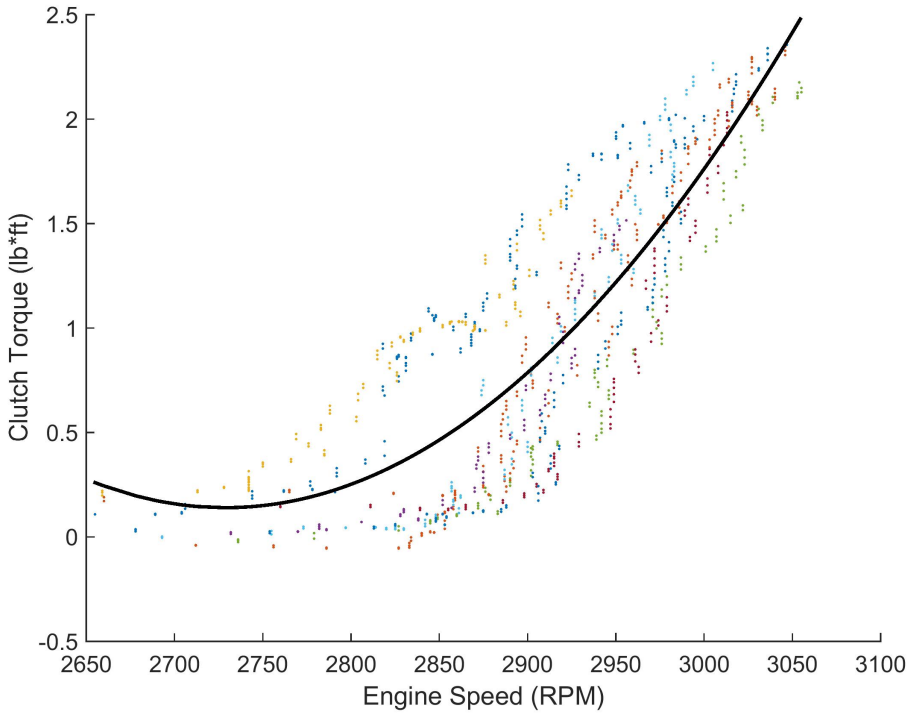

Figure 25: Clutch Torque versus Engine Speed

The black line in the center of the data is the least-squares polynomial. The coefficients of that polynomial can be used to calculate C_c and ω'. The engagement speed of clutch, ω', is 2730 rpm, and the clutch engagement coefficient, C_c, is 0.0020 $lbf/(rad/s)^2$.

4.7 Starting Penalties

The final coefficients to determine are K_ω and K_{m_f}, the average engine acceleration and fuel flow rate when the engine is below idle speed. The engine idle speed is about 1000 rpm. In order to analyze the starting motor dynamics, the motor will be considered "starting" when the engine speed is above 0 but below 1000 rpm.

Additionally, the motor will still be "starting" if it reaches 1000 rpm but does not stay above it.

Figure 26 shows the total fuel consumption of a typical testing run. The red circles are the points at which the engine is "starting." Notice that the engine does not take long to start, and little fuel is consumed.

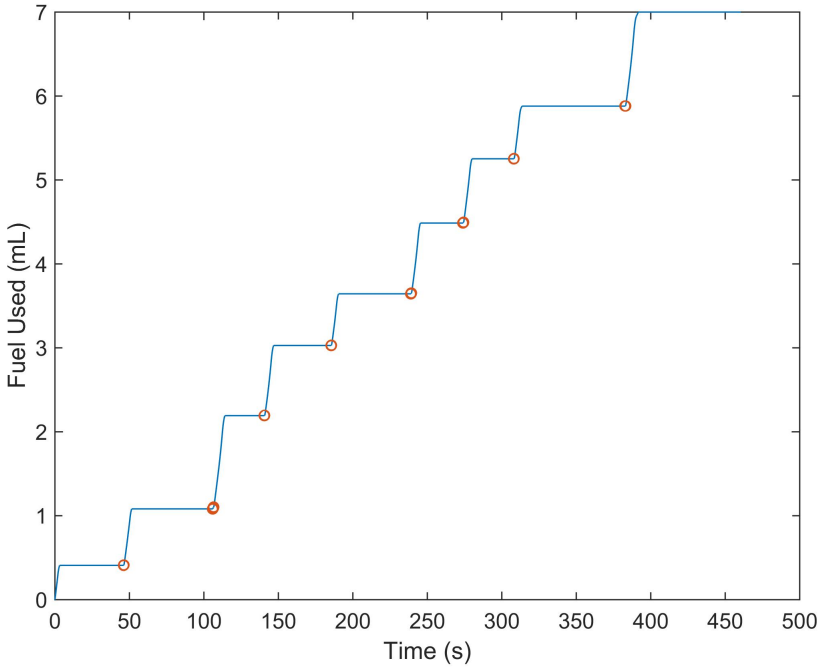

Figure 26: Total Fuel Consumption versus Time

Figure 27 shows the engine coolant temperature over time. Again, the red circles are when the engine is starting. The engine begins the testing run fairly cold but steadily warms up with every burn.

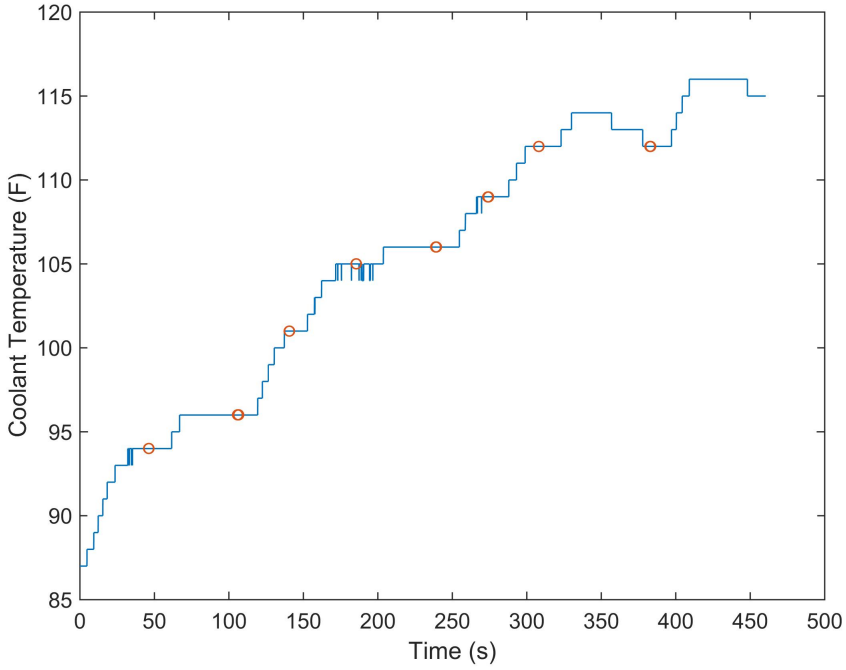

Figure 27: Engine Coolant Temperature versus Time

Figure 28 shows the time it takes for the engine to complete starting versus the start index. There were 8 starts analyzed, so there are only 8 data points in the figure.

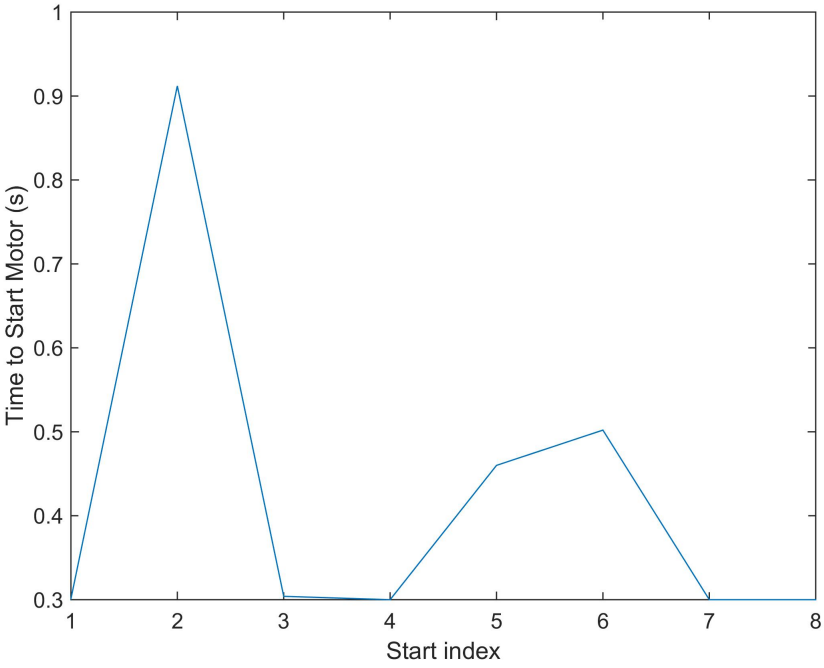

Figure 28: Time Taken to Start Motor

Figure 29 shows the total fuel consumed over each start. Notice that engine starting time and fuel consumed are relatively constant. It is expected that the engine would start faster and use less fuel as it warms up, but that is not clear from the data. Therefore, the assumption that the starting constants are not affected by temperature seems reasonable.

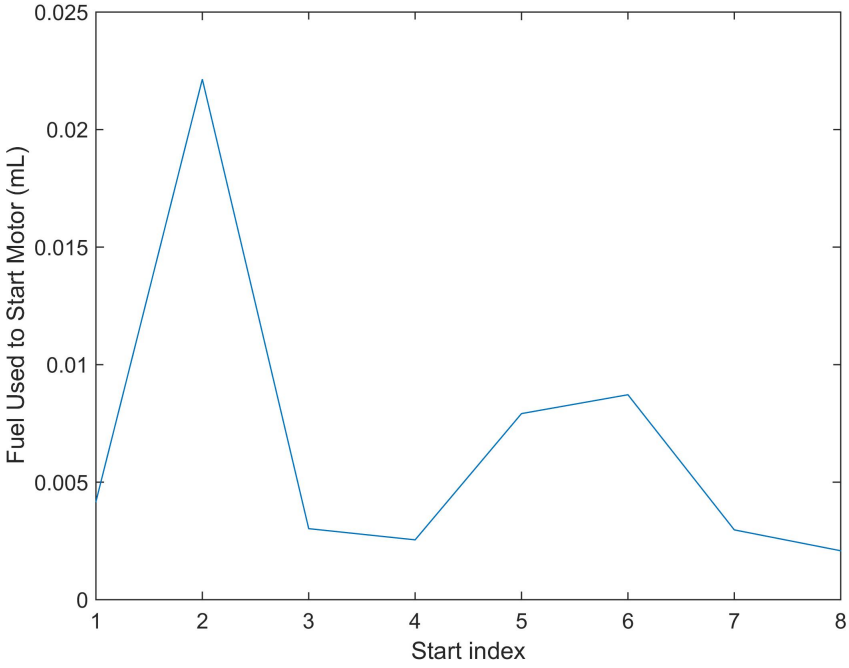

Figure 29: Fuel Consumed per Start

The average of the starting times is 0.42 seconds, while the average fuel consumed is 0.0067 mL. The average fuel consumption divided by the average time determines the average starting fuel flow rate, which is equal to 0.0159 mL/s. The starting engine acceleration can be calculated by dividing the engine idle speed, 104.72 rad/s, by the average starting time to get 247.92 rad/s^2.

4.8 Verifying the Simulation

Now that all of the required coefficients have been determined, the simulation can be executed. Since testing data is only available from uneven tracks, and the exact slopes along the track are unknown, the track will be assumed to be flat in the simulation for this section.

Figure 30 shows the measured vehicle speed in blue and the simulation output speed in red. Notice that the simulation output is perfectly consistent, while the actual speed varies significantly. Part of the difference is driver error, and part is the uneven track.

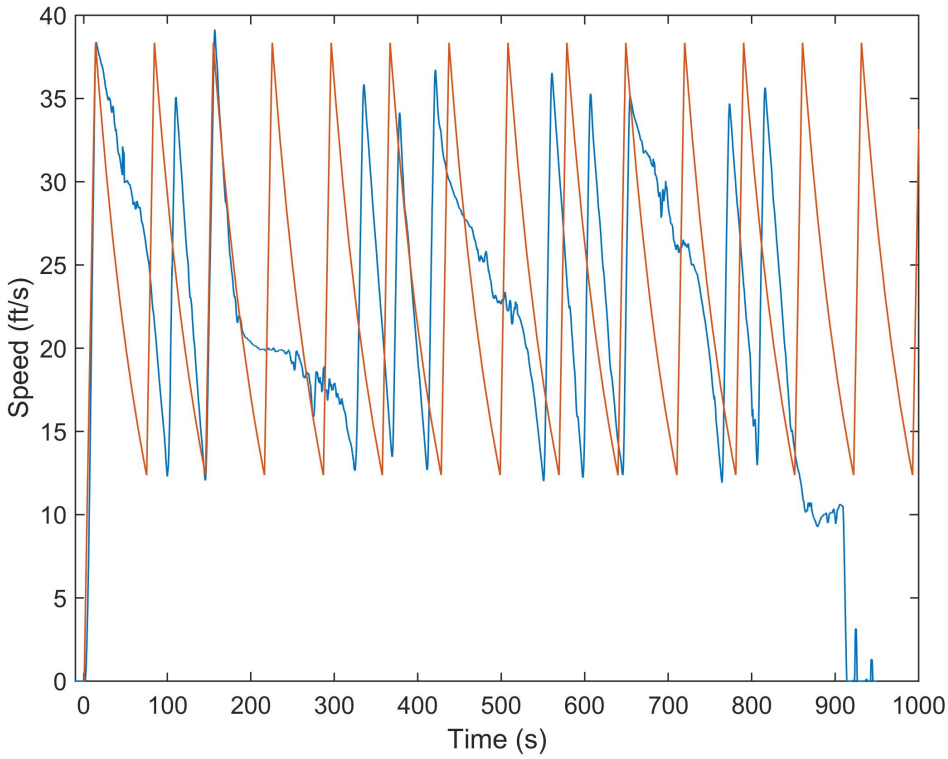

Figure 30: Vehicle Speed versus Time

Figure 31 shows a zoomed-in portion of Figure 30. The simulation output matches the actual speed fairly well, but the track unevenness makes it difficult to compare well.

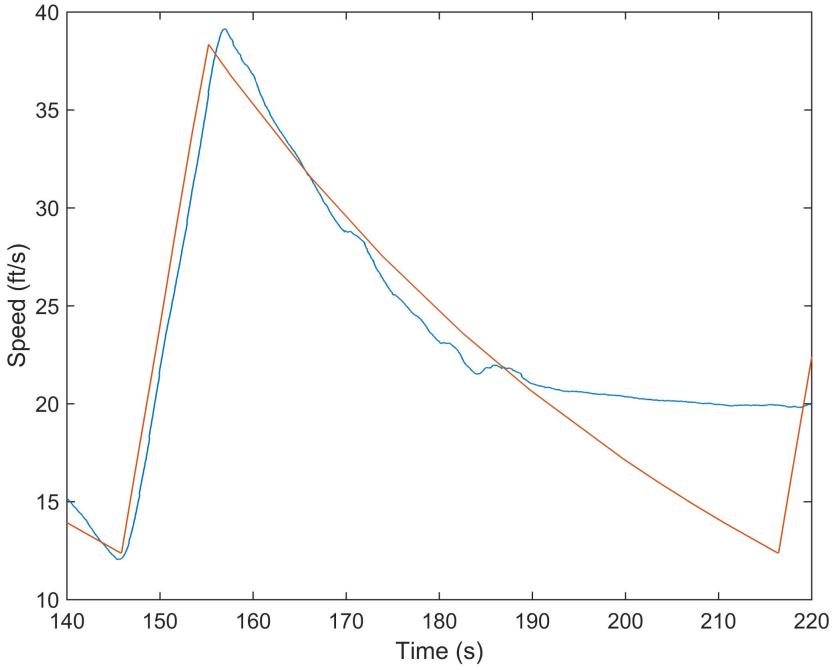

Figure 31: Zoomed In Vehicle Speed versus Time

Figure 32 shows the engine speed from the same testing data and simulation output. The actual engine speed is in blue, while the simulation output is in red.

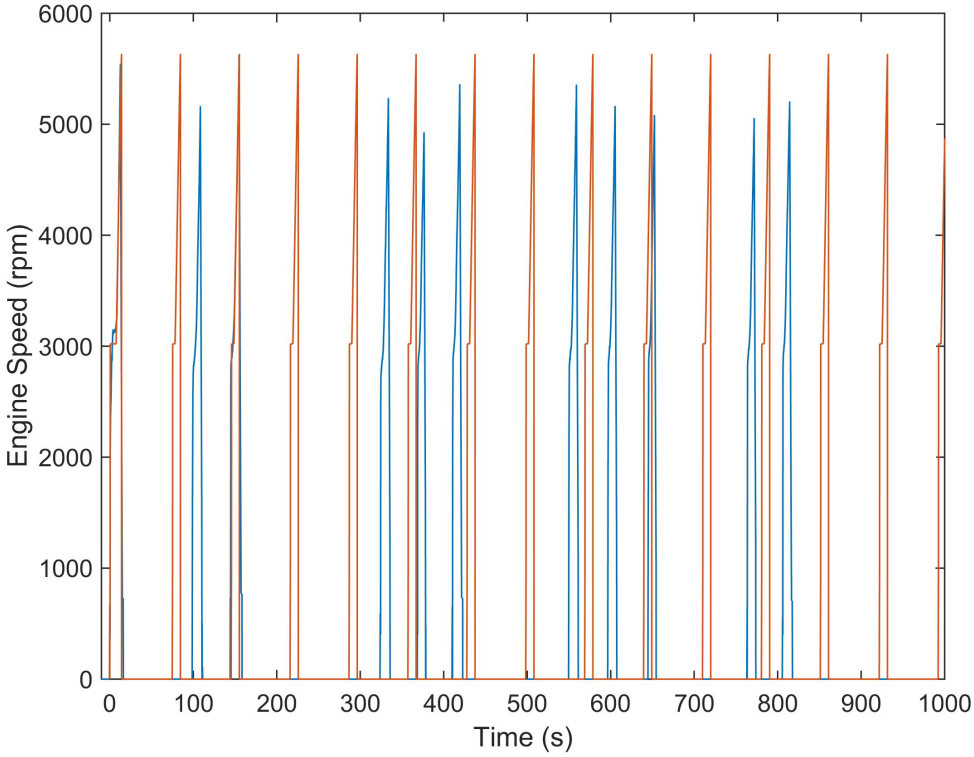

Figure 32: Engine Speed versus Time

Figure 33 shows a zoomed-in portion of Figure 32. Notice that the simulated engine speed matches fairly closely to the experimental engine speed. However, there are 2 main differences. The clutch simulation does not capture the exact dynamics of the system, but it is close. Additionally, the engine speed drops nearly instantly in the simulation, while it takes a small amount of time in the actual vehicle.

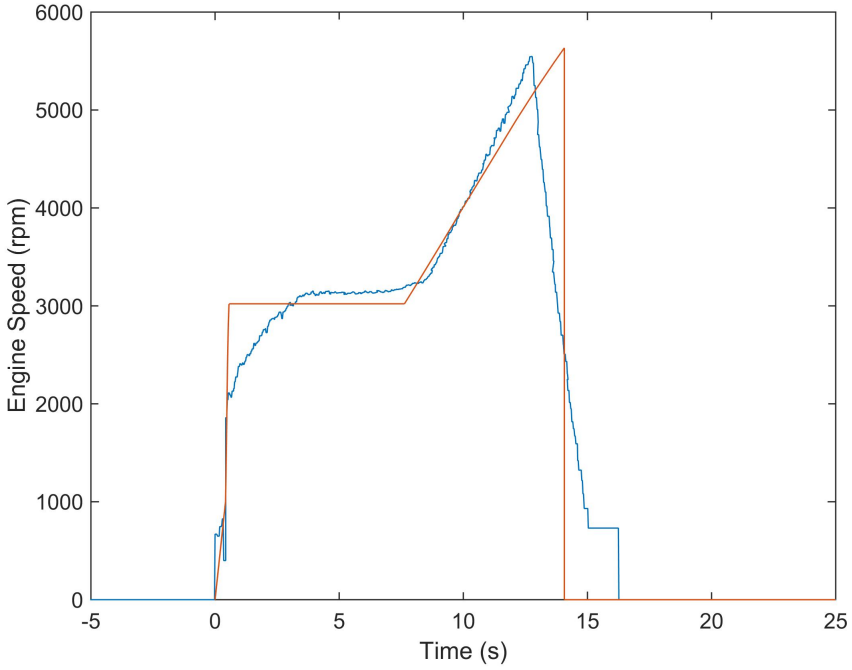

Figure 33: Zoomed In Engine Speed versus Time

Figure 34 shows the total fuel consumed over distance.

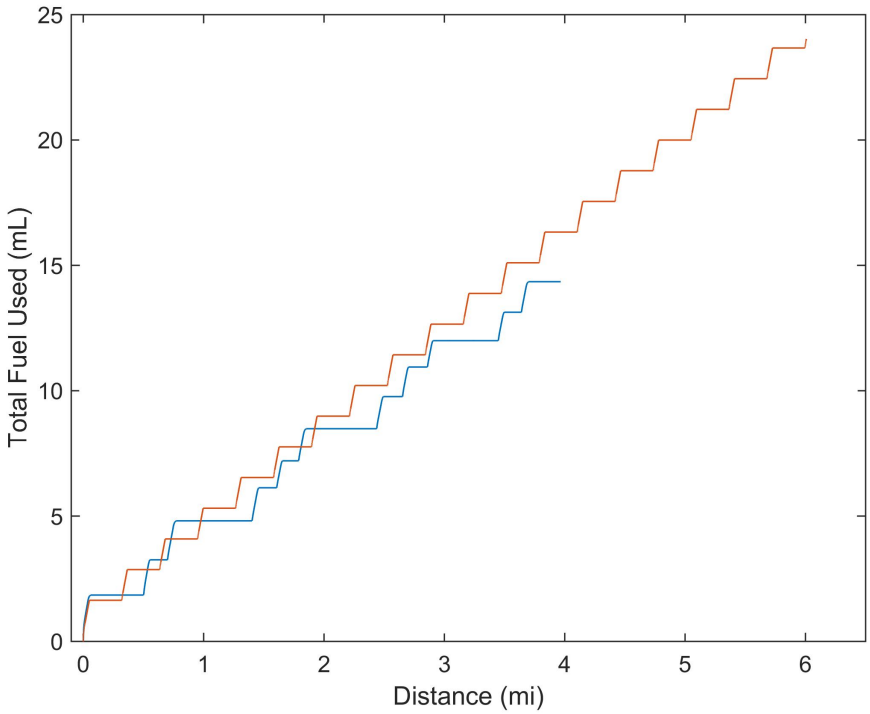

Figure 34: Total Fuel Consumption versus Distance

The fuel consumption does not exactly align, and this is likely because of 2 reasons: The engine takes a small amount of time to reach 0 rpm in reality, and the road load equation is slightly inaccurate. If the road load equation was accurate, the energy taken from the vehicle should be the same in the actual testing and simulation. This would mean that the slope of the total fuel consumption plot would be the same for experimental and simulated. Since there are 3 terms in the road load equation, it is not possible to determine which term is not accurate.

Overall, the simulation seems to match the actual vehicle well. With better data and testing methods, the simulation should be able to match the actual vehicle output even better.

5 Vehicle Parameter Analysis

In the following sections, various vehicle parameters will be analyzed using the simulation.

5.1 User Interface

In order to make trade studies easy, a simple user interface was developed using MATLAB's GUIDE[9] user interface development tool. A single window is used to alter any simulation parameters and run trade studies quickly. This window is shown in Figure 35.

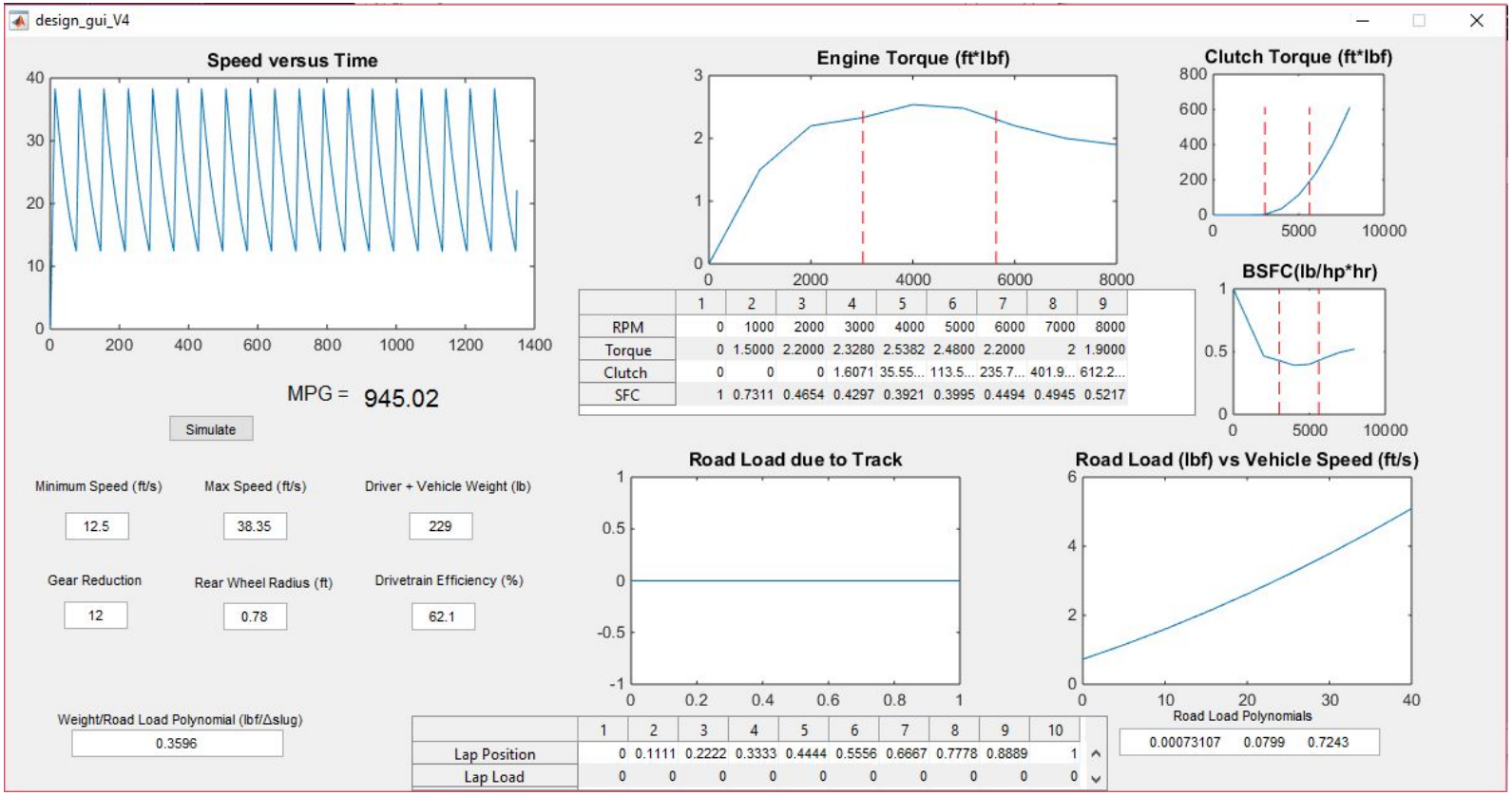

Figure 35: Simulation User Interface

Detailed instructions to use the user interface can be found in the appendices.

5.2 Vehicle Weight Analysis

Historically, one of the most important parameters for vehicle efficiency has been weight. Every Shell Eco-marathon team tries to minimize weight to maximize fuel efficiency. Using the simulation, the effect of weight on fuel efficiency can be determined. Every parameter remains constant except road load, which changes by Equation 22. The output is shown in Figure 36.

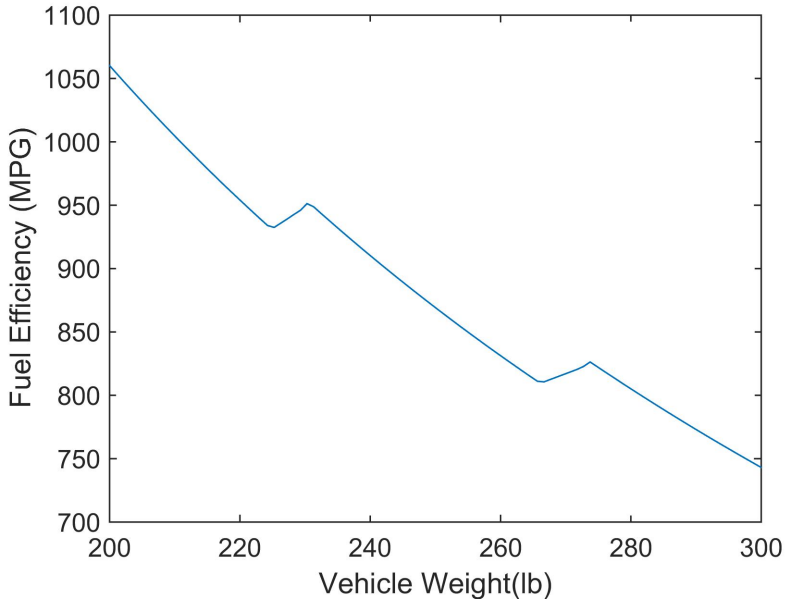

Figure 36: Fuel Efficiency versus Weight

Overall, there is a downward trend from increasing total vehicle weight, which is not surprising. There are slight increases in fuel efficiency at certain weights; these are not numerical errors. As vehicle weight increases, the total number of coasts and burns is reduced. This is due to the increased vehicle inertia, which

demands longer burns and coasts to speed up and slow down respectively. When the time spent coasting and burning is changed, the vehicle ends the competition run either burning or coasting. If the vehicle ends the competition run coasting, the fuel efficiency is better than if energy was wasted burning at the very end. That is why there are slight increases at certain weights.

5.3 Optimizing Vehicle Speed and Gear Ratio

The minimum and maximum speeds of the vehicle may have a dramatic effect on fuel economy, since they directly affect the operating range of the engine. The optimal minimum and maximum speeds could be found by keeping every other parameter constant and finding the maximum vehicle efficiency. However, the final drive ratio should also be allowed to change at the same time, since it also changes the operating range of the engine. Therefore, for the remainder of this section, the simulation output will be considered a function of 3 variables:

$$MPG = f(v_{min}, v_{max}, \eta) \tag{27}$$

Where:

v_{min} is the minimum vehicle speed (ft/s)

v_{max} is the maximum vehicle speed (ft/s)

MATLAB will be used to maximize fuel efficiency. However, its optimization functions are set to minimize, not maximize. Therefore, the optimization goal will be to minimize negative fuel efficiency. If the simulation time output is greater than 24 minutes (the time allotted for a competition run), the resulting fuel econ-

omy is not valid. Therefore, the function will return "NaN", or Not-a-Number[11]. Equation 28 shows the optimization function:

$$f_{opt}(v_{min}, v_{max}, \eta) = \left\{ \begin{array}{ll} -MPG(v_{min}, v_{max}, \eta) & t_{final} \leq 24\ min \\ NaN & t_{final} > 24\ min \end{array} \right\} \quad (28)$$

There are 2 main options available for global minimization in MATLAB: GlobalSearch and MultiStart. GlobalSearch was chosen since it runs more efficiently than MultiStart on one processor[10], and it is unknown how many processors will be available to any user running the optimization.

The GlobalSearch algorithm essentially works by evaluating the optimization function at a starting point and then finding a local minimum from that point[19]. This repeats for many different starting points to find all the local minima. The lowest minimum is then the global minimum. Ideally, each starting point should lead to a different local minimum. This can be accomplished by accurately finding the radius of the function's basin of attraction. The radius is the distance in solution space from the starting point to the local minimum. If the radius is too large, there may be local minima missed. If the radius is too small, the same local minima will be found repeatedly. The radius of the basin of attraction is estimated as part of the GlobalSearch algorithm[19], but MATLAB allows the user to specify a distance threshold factor. This factor is multiplied by the estimated basin radius to get a more conservative radius if the threshold is smaller than 1.

Additionally, a user can specify how many trial points will be tested. If a trial point is close to the previously found global minimum, the local minimum from that

point will be found. For this reason, it is expected to have a more accurate result if many trial points are used. On average, a simulation run takes about 0.37 seconds to complete, so many trial points will take a very long time to go through. Each GlobalSearch was ran with a random starting point to ensure that the true global minimum was found. The results of the optimization are summarized in Table 1.

Fuel Economy (MPG)	Speed (ft/s)	Final Drive Ratio	Simulation Runs
1146.2	17.92 – 26.86	15.17	13152
1146.0	17.36 – 27.55	14.77	7800
1137.3	17.55 – 27.41	16.23	3197
1115.2	16.96 – 28.69	16.44	3082

Table 1: Optimization Results

Notice that the maximum fuel economy was found when more runs were simulated. However, after a certain amount of simulation runs, there is not much of an advantage to looking at more trial points. The first GlobalSearch ran the simulation 13,152 times, and it found a maximum fuel economy of 1146.2 MPG. The second search found a maximum of 1146.0 MPG, which is only 0.2 MPG less. However it ran the simulation only 7,800 times.

An interesting way to visualize the GlobalSearch optimization is to plot all the various points that the algorithm tries. Since v_{min}, v_{max}, η, and MPG make up a fourth dimensional solution space, it is more practical to show the trial points on a second dimensional plot. Figure 37 shows a scatter plot of each trial point projected onto the MPG-η axes.

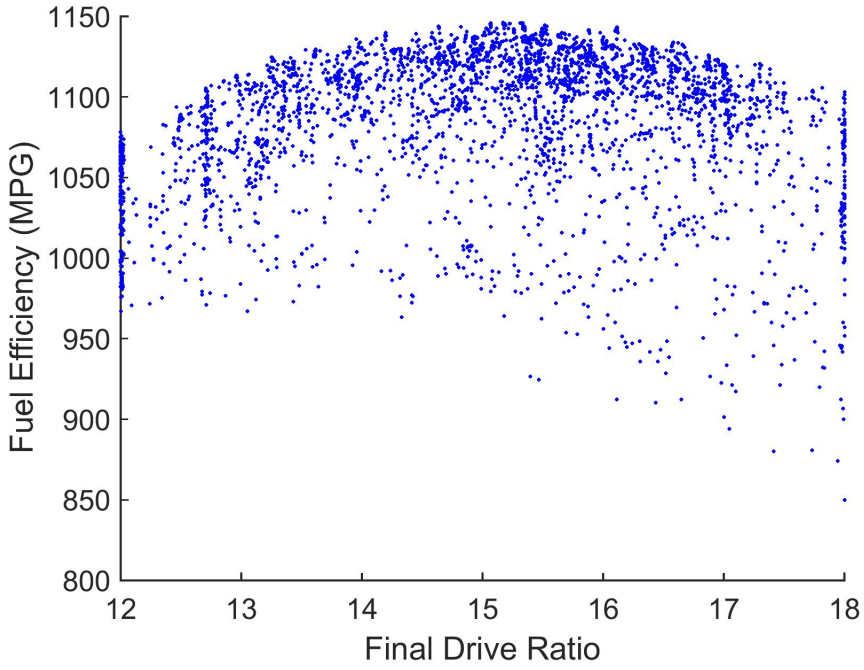

Figure 37: Fuel Efficiency versus Gear Ratio, Varying Vehicle Speed

There is a large variety in fuel economy when speed is changed, meaning that improper vehicle speeds can severely impact fuel economy. The lowest efficiency is about 850 MPG, while the highest is 1146.2 MPG. Figure 37 was generated using the search from the top of Table 1. Even though there were 13,152 simulation runs, there are not that many points on the scatter plot because many points do not complete the competition in time.

All the searches came up with reasonably close results, so the global maximum is likely the one shown at the top of Table 1. If it is not the true global maximum, it is likely very close.

The first way to verify the global maximum is to plot fuel economy versus final drive ratio while keeping v_{min} and v_{max} constant. The values of v_{min} and v_{max} used

are 17.92 and 26.86, the values from the top of Table 1. This is shown in Figure 38.

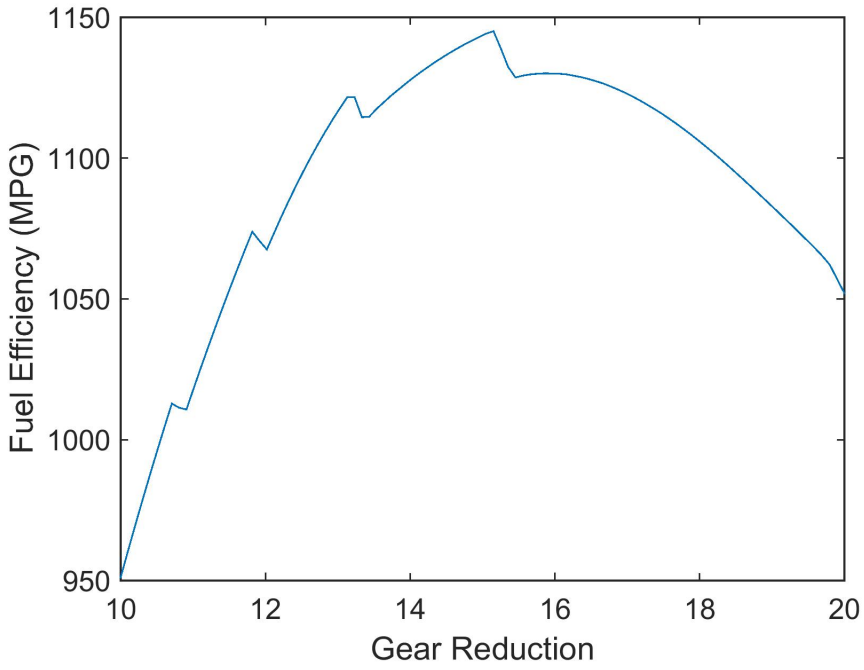

Figure 38: Fuel Efficiency versus Gear Ratio, Constant v_{min} and v_{max}

The peak fuel economy is about 1145 MPG at 15.2 gear reduction, which verifies the maximum found through GlobalSearch. The jagged sections of the plot are caused by ending the simulation on a coast or a burn. Changing the final drive ratio slightly changes when the vehicle coasts and burns, and it is always more efficient to end on a coast.

The second way to verify the global maximum is to plot MPG versus v_{min} and v_{max} in a 3-D surface plot. For the plot, final drive ratio is set to 15.17, the optimal final drive ratio found at the top of Table 1. Three views of the 3-D plot will be shown.

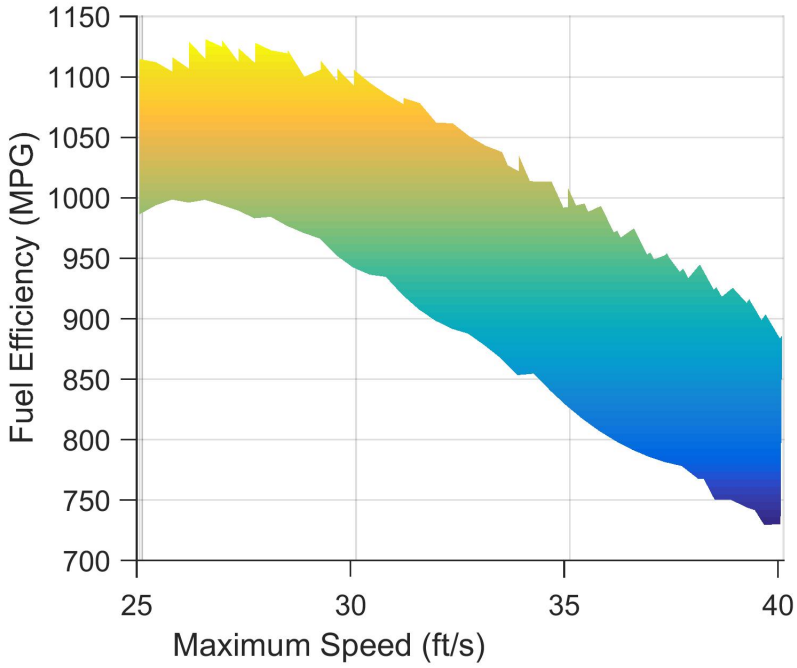

Figure 39: Fuel Efficiency versus Maximum Speed, Constant η

Figure 39 shows the relationship between maximum vehicle speed and fuel economy. The fuel efficiency decreases with top speed; this is a result of the higher engine speeds and higher road load losing more energy over time.

Figure 40 shows the relationship between minimum vehicle speed and fuel economy. There is not much of a distinct relationship as with maximum vehicle speed, but there is a clear maximum around 18 ft/s.

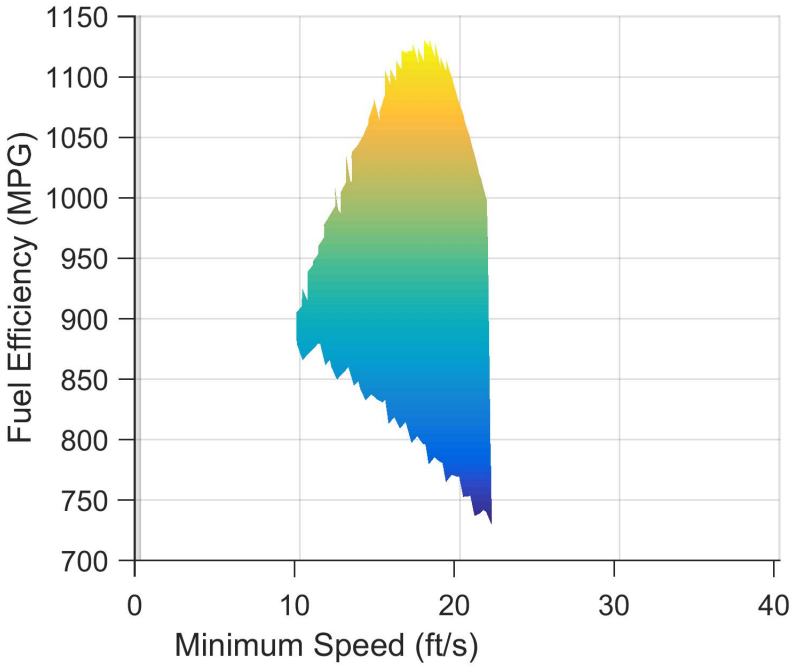

Figure 40: Fuel Efficiency versus Minimum Speed, Constant η

Figure 41 shows the relationship between maximum and minimum vehicle speed and fuel economy.

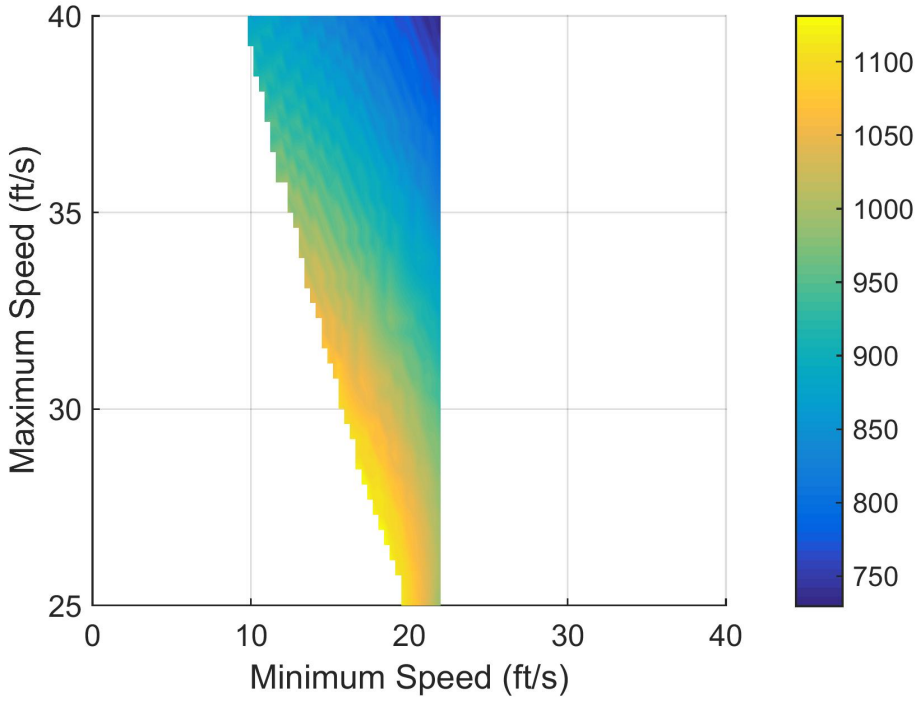

Figure 41: Fuel Efficiency versus Maximum and Minimum Speed, Constant η

The optimal minimum and maximum speeds are about 18 and 27 respectively, verifying the optimal speeds found by the GlobalSearch algorithm.

5.4 Optimization Discussion

Table 2 shows the difference between the optimized and unoptimized vehicles.

	Fuel Economy (MPG)	Vehicle Speeds (ft/s)	Final Drive Ratio
Optimized	1146.2	17.92 – 26.86	15.17
Unoptimized	1055	12.5 – 38.35	12

Table 2: Optimization Comparison

In order to further verify the optimal speeds and final drive ratio, it is useful to

look at how the vehicle operates under the optimal conditions. When the minimum vehicle speed is 17.92, the clutch housing speed is 3329 rpm. This is slightly above 3021 rpm, the speed at which centrifugal clutch torque equals the torque of the engine. This essentially means that the clutch is never slipping when the vehicle is driven at optimal speeds. Since a slipping clutch loses energy, this makes sense to maximize efficiency. Figure 42 shows actual engine speed in blue and optimal simulated engine speed in red.

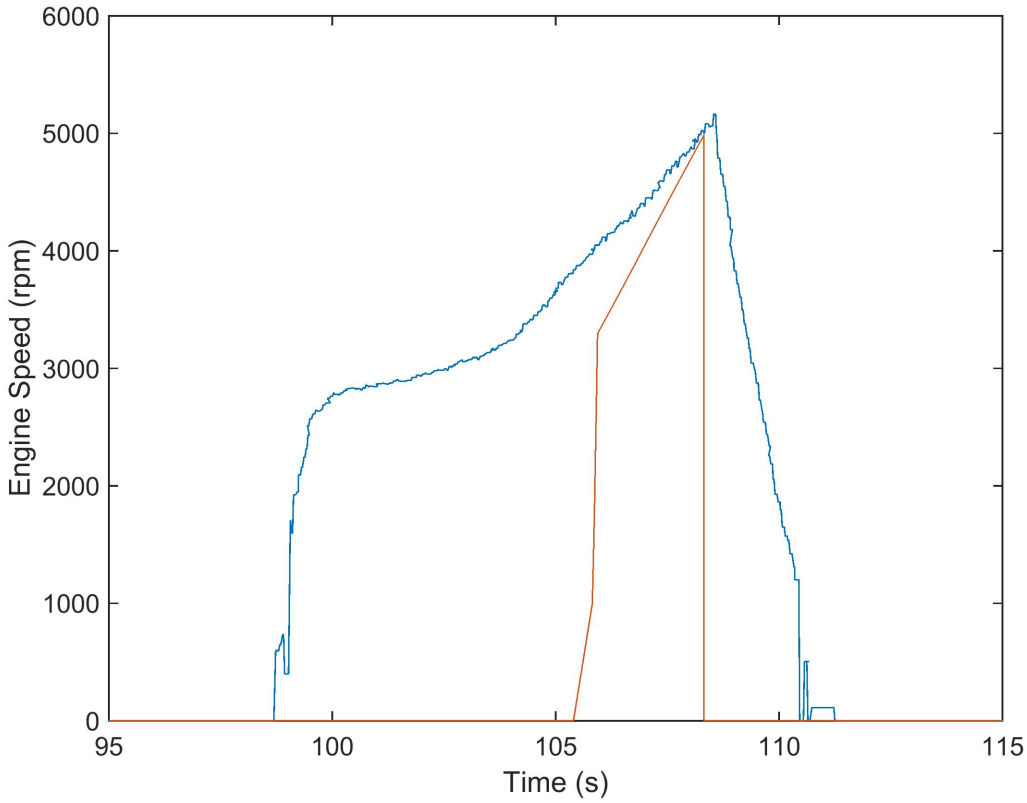

Figure 42: Engine Speed versus Time

Notice that the clutch does not slip at all in the optimal simulation, and the

optimal burn time is about 30% of the original burn time. However, at optimal speeds, the vehicle must use the engine much more often. Figure 43 shows the actual vehicle speed in blue and the optimal simulation speed in red.

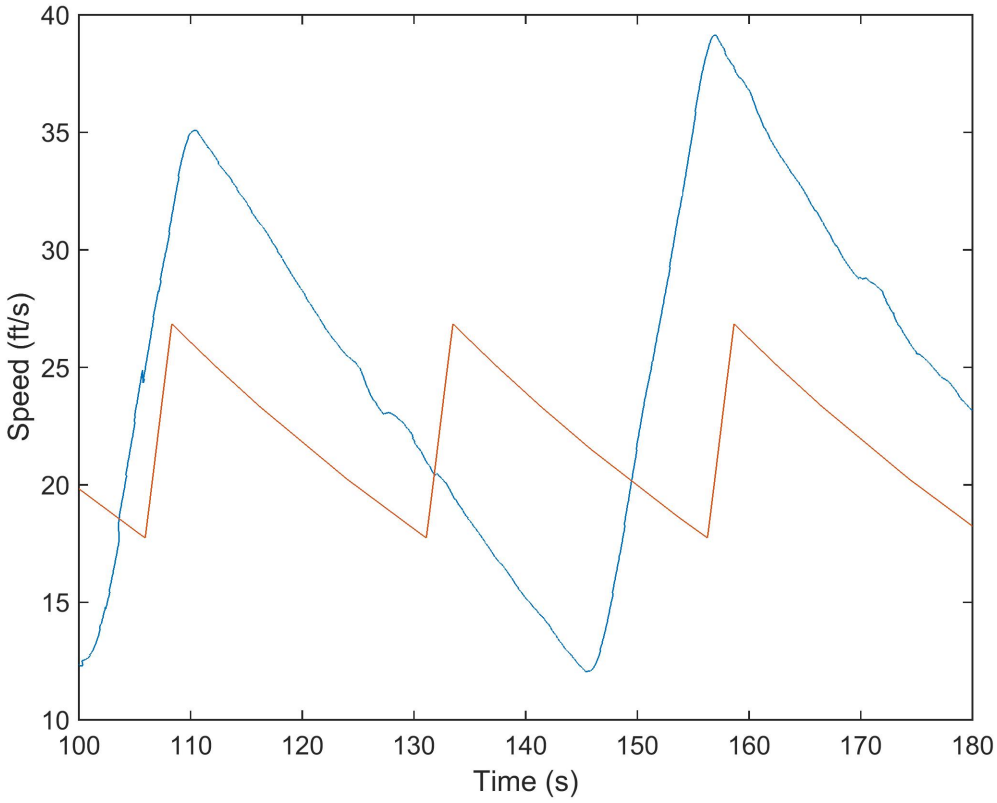

Figure 43: Vehicle Speed versus Time

The optimal top speed is the speed at which the vehicle can barely make it to the finish line in time. If the top speed is any lower, the vehicle will not finish. This makes sense, since faster speeds will increase road load and also move the peak operating engine speed higher. Both of those will increase the amount of energy wasted by the vehicle. Figure 44 shows the actual vehicle fuel consumption in blue

and the optimal fuel consumption in red.

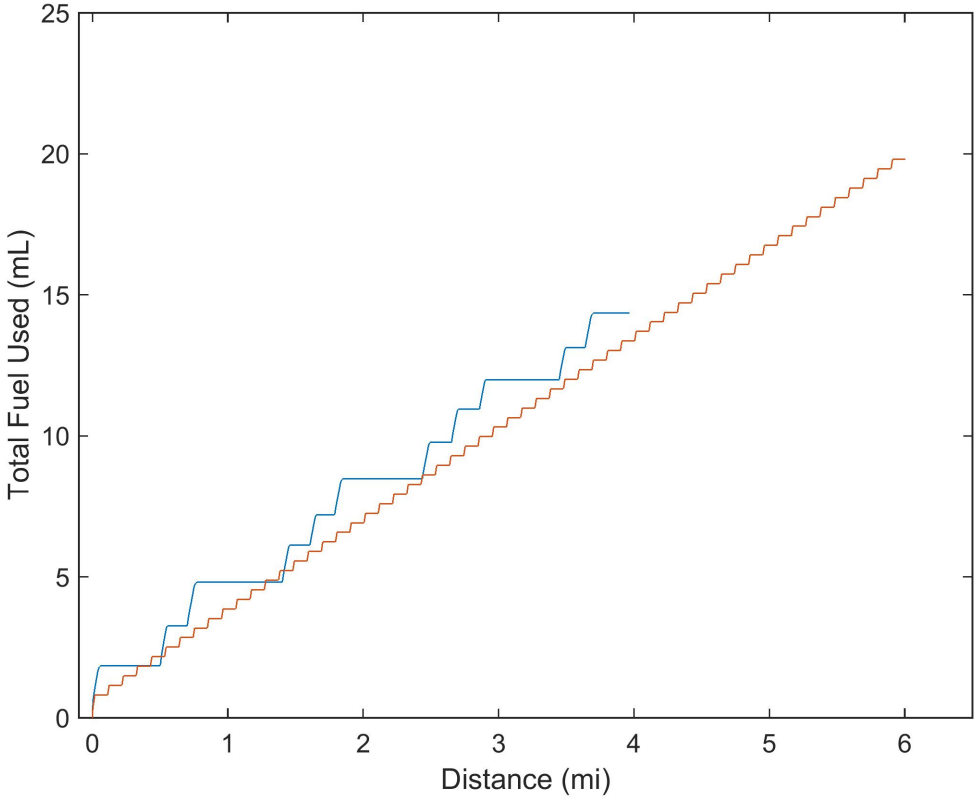

Figure 44: Fuel Consumption versus Distance

During testing, the vehicle ran at 1055.0 MPG. The optimal vehicle runs at 1146.2 MPG, which is an 8.6% increase in fuel efficiency.

6 Conclusion

Overall, the optimization study will guide future Cal Poly Supermileage teams to better fuel economy. However, there are some shortfalls that must be addressed in the future.

The calculated road load coefficients have considerable uncertainty. A rear-wheel Hall effect sensor should be installed to accurately measure vehicle speed. A pitot tube may also be mounted to the car to negate any wind effects from the aerodynamic drag calculation. Additionally, a more accurate way to measure the slope of the test track must be found.

The greatest inaccuracy from the simulation comes from the engine turning off. In the simulation, the engine dies instantly, while the actual engine takes a few seconds to actually reach 0 rpm. In the future, it may be useful to tweak the simulation equations (15 and 16) to account for the engine shutoff dynamics.

The optimization study heavily relies on accurate starting penalties. A more accurate model would take into account how much energy is consumed by the starter motor to start the engine every time. This additional penalty would likely increase the speed difference between the optimal minimum and maximum speeds, reducing the total number of burns required.

In general, a review of all the data analysis in this study, with new data would be added, would be useful to validate the results and ensure accuracy. All the MATLAB code utilized to analyze the data is easy to use, so this should be possible in the future.

From this study, a few recommendations can be asserted. Driving strategy can

be changed quickly, so the user interface should be used as a tool to ensure that the driver always drives most efficiently, no matter what vehicle is driven. In the future, a higher final drive reduction (15.2 from 12) should be used to maximize fuel economy. The user interface should also be used to determine what engine tune is most efficient. For every tune, the only information needed is new torque and BSFC curves; then a simulation can be performed. The analysis and user interface presented in this study should help guide future Supermileage teams, and hopefully it will continue to demonstrate how every vehicle can achieve the ultimate goal of higher fuel efficiency.

REFERENCES

[1] William L. Brogan. *Modern Control Theory*. Prentice Hall, Upper Saddle River, New Jersey, 1991.

[2] Pedro De Figueiredo Vieira Carvalheira. Simulation of the performance of an extra-low fuel consumption vehicle. *Proceedings of the IASTED International Conference on Modeling and Simulation*, 15(1):424–429, 2004.

[3] Karol Cichonski, Katarzyna Jezierska-Krupa, Marcin Glen, and Wojciech Skarka. The comparative study of drivetrain of high-performance electric vehicle. *Diagnostyka*, 15(2):65–70, 2014.

[4] Thomas D. Gillespie. *Fundamentals of Vehicle Dynamics*. Society of Automotive Engineers Inc, Warrendale, Pennsylvania, 1992.

[5] Eric J. Griess. Benchmarking, characterization and tuning of shell eco-marathon prototype powertrain. 2015.

[6] Sophie Jan. Minimizing the fuel consumption of a vehicle from the shell eco-marathon: A numerical study. *ESAIM - Control, Optimisation and Calculus of Variations*, 19(2):516–532, 2013.

[7] Autosport Labs. Racecapture/pro. `https://www.autosportlabs.com/racecapturepro-2/`. Accessed: 2/19/2017.

[8] T. Manrique, M. Fiacchini, T. Chambrion, and G. Millerioux. Mpc for a low consumption electric vehicle with time-varying constraints. *IFAC Symposium on System Structure and Control*, 5(1):833–838, 2013.

[9] MathWorks. Create apps with graphical user interfaces in matlab. `https://www.mathworks.com/discovery/matlab-gui.html`. Accessed: 2/27/2017.

[10] MathWorks. How globalsearch and multistart work. `https://www.mathworks.com/help/gads/how-globalsearch-and-multistart-work.html`. Accessed: 2/28/2017.

[11] MathWorks. Not-a-number. `https://www.mathworks.com/help/matlab/ref/nan.html`. Accessed: 2/28/2017.

[12] Muhammad Rizuwan Mustaffa, Wan Ahmad Najmi B. Wan Mohamed, and Rahim B Atan. Analytical approach to predict hydrogen consumption of a lightweight pem fuel cell vehicle. *IEEE International Conference on Control System, Computing and Engineering*, pages 489–494, 2013.

[13] J.-C Wasselynck Olivier, G Trichet, D Bernard, N Hmam, S Chevalier, S Josset, C Auvity, B Squadrito, and Gaetano. Multiphysics modeling and driving strategy optimization of an urban-concept vehicle. *IEEE Vehicle Power and Propulsion Conference*, 2015.

[14] S. S. Rattan. *Theory of Machines*. McGraw-Hill, 2 Pennsylvania Plaza, New York City, New York, 2014.

[15] Y. Saboohi and H. Farzaneh. Model for optimizing energy efficiency through controlling speed and gear ratio. *Energy Efficiency*, 1(1):65–76, 2008.

[16] J.J. Santin, C.H. Onder, and J. Bernard. *The World's Most Fuel Efficient Vehicle, Design and Development of Pac-car II*. ETH Zürich, Zürich/Singen, Switzerland, 2007.

[17] Shell. Shell eco-marathon 2017 rules. `http://www.shell.com/energy-and-innovation/shell-ecomarathon/for-participants/tech-tips-and-tricks/_jcr_content/par/expandablelist/expandablesection_603560196.stream/1472744744388/7b204129356de26d47c9f21e2c8f58e07f3e2e4f266ae7714c8e6b17bb7051e1/sem-2017-rules-chapter-1.pdf`. Accessed: 3/2/2017.

[18] U.S. Geological Survey. Performance specification digital terrain elevation data. `https://dds.cr.usgs.gov/srtm/version2_1/Documentation/MIL-PDF-89020B.pdf`. Accessed: 2/20/2017.

[19] Zsolt Ugray, Leon Lasdon, John C. Plummer, Fred Glover, James Kelly, and Rafael Martí. Scatter search and local nlp solvers: A multistart framework for global optimization. *INFORMS Journal on Computing*, 19(3):328–340, 2007.

APPENDICES

A. User Interface Instructions

There are two parts to user interfaces in MATLAB: a .fig and .m file. The .fig file controls the layout of the user interface (the location of buttons, etc.), while the .m file contains the MATLAB code that runs in the background when a button is pushed on the user interface. To start the user interface, make sure the .fig and .m file are in the same folder. Then run `"Design_GUI.m"`. The user interface will open and start with pre-filled in values. These default values can be changed by altering the values towards the top of the .m file.

To simulate a competition run with the starting values, click "Simulate". MATLAB figures will show up within a few seconds showing how the vehicle performed. Additionally, the vehicle's fuel efficiency in MPG is shown next to the "Simulate" button. Any parameter in the user interface can be changed by clicking on the box and typing in a new number. As soon as "Simulate" is clicked again, the new numbers will be used for another simulation.

To perform a weight sensitivity analysis, type in 2 numbers separated by a space into the "Driver + Vehicle Weight (lb)" section, and then click "Simulate". 100 simulations will be performed from the first weight to the second weight, and a figure will automatically generate showing how fuel efficiency is affected by total vehicle weight. Note that this analysis strongly depends on the accuracy of the "Weight/Road Load Polynomial" parameter, which is located in the lower left-hand corner of the user interface. This parameter is discussed in section 4.3.

To perform a final drive analysis, type in 2 numbers separated by a space into the "Gear Reduction" section, and then click "Simulate". 100 simulations will be performed from the first final drive ratio to the second, and a figure will automatically generate showing how fuel efficiency is affected by final drive reduction.

To perform an optimal speed analysis, type in 2 numbers into the Minimum and Maximum Speed sections, 4 numbers total, and then click "Simulate". 1600 simulations will be performed, so this takes significantly longer than the previous analyses. A 3D surface plot will be generated showing the relationship between minimum and maximum speeds and fuel efficiency.

To numerically optimize speed and final drive ratio using the GlobalSearch algorithm, type in 2 numbers into three sections: Minimum Speed, Maximum Speed, and Gear Reduction. Then click "Simulate" to begin the optimization. This will go through about 10,000 simulations, so expect to wait 30 minutes to an hour to get the result.

If the user interface layout needs to be changed, type in "guide" into the main MATLAB command line to bring up GUIDE, the tool used to build the interface.

experimental_supermileage_model_v3.m

```
% Chad Bickel
% 3/8/17
% Analyzes RaceCapture data and seperates data into burning and
    coasting
% sections

clear all
close all
clc

%load Apr3Test2data
load Apr3Test4data %loading track data
%load day1_run2

photopath1 = 'C:\Users\Chad\Google Drive\NEW Documents\Thesis';
photopath2 = '\Photos\MatlabOutput\';
photopath = strcat(photopath1,photopath2);

total_fuelgal = total_fuel/3785.41; %converting from mL to gallons

total_mpg = total_dist/total_fuelgal; %calculating mpg


%below, assigning variables for time,pulsewidth,rpm and speed

try
    time = (Utcms001-Utcms001(1))*1e-3; %time in seconds
    pulsewidthvec = PW1ms03050;
    rpmvec = RPMrevmin01000050;
    %speedvec = SpeedMPH00150010;
    % Do this if you're sampling GPS at 10 Hz
    ix=cellfun(@isempty,SpeedMPH00150010);
    SpeedMPH00150010(ix) = {'0'};
    keepspeed = 0;
     SpeedMPH00150010(1) = {'12'}; %fix this later
%     SpeedMPH00150010(2) = {12};
%     SpeedMPH00150010(3) = {12};
    for i = 1:length(SpeedMPH00150010)
        tempspeed = str2double(SpeedMPH00150010(i));
        if tempspeed > 0
            keepspeed = tempspeed;
        end
```

```
42          SpeedMPH00150010(i) = {keepspeed};
43      end
44      speedvec = cell2mat(SpeedMPH00150010');
45  catch
46      time = (Utcms1-Utcms1(1))*1e-3; %time in seconds
47      pulsewidthvec = PW1ms1;
48      rpmvec = RPMrevmin1;
49      speedvec = SpeedMPH1;
50
51  end
52
53
54  figure(1)
55  subplot(2,1,1)
56  plot(time(10000:20000),pulsewidthvec(10000:20000)) %plot the pulse
        width
57  xlabel('Time (s)')
58  ylabel('Pulse Width (ms)')
59  set(gca,'fontsize',12)
60  subplot(2,1,2)
61  plot(time(10000:20000),rpmvec(10000:20000))         %plot RPM
62  xlabel('Time (s)')
63  ylabel('Engine Speed (RPM)')
64  set(gca,'fontsize',13)
65  %print(strcat(photopath,'Pulse_RPM_ex'),'-djpeg','-r500')
66
67  cyclespm = rpmvec/2; %4 stroke engine, so 1 cyc per 2 revs
68  cyclesps = cyclespm/60; %engine cycles per second
69
70  %now we have cycles/s and fuel ms/cycle
71  instant_fuel_ms = pulsewidthvec.*cyclesps; %(fuel ms)/s
72
73  %now we integrate (fuel ms)/s to get total fuel ms
74  total_fuel_ms = trapz(time,instant_fuel_ms); %total ms of fuel
        burned
75
76  %now we need to convert ms of injector open to ml of fuel
77  mfdotmL = total_fuel/(total_fuel_ms);%mass flow rate of injector,
78  %in mL per fuel ms
79  %this is 0.6209 mL/s for test 2
80  %this is 0.6385 mL/s for test 4
81
82  %assume mass flow rate of injector is constant
83  instant_fuel_mL = instant_fuel_ms*mfdotmL; %instant fuel usage, in
        mL/s
84
85  n = 10;
86  instant_fuel_mL_filt = filter(ones(1,n)/n,[1],instant_fuel_mL);
```

```
87
88  % subplot(4,1,3)
89  % plot(instant_fuel_mL) %plot instant fuel usage
90  % ylabel('Instant Fuel Usage (mL/s)')
91  % hold on
92  % plot(instant_fuel_mL_filt)
93  %
94  % subplot(4,1,4)
95  % total_fuelvec = cumtrapz(time,instant_fuel_mL); %integrate
        instant fuel
96  % plot(total_fuelvec)
97  % ylabel('Total Fuel Used (mL)')
98
99  lap_dist = 1;%0.6; %1 lap =~ 1 mile
100 lap_unlimit = cumtrapz(time,speedvec/(3600*lap_dist));
101 lap_vec = mod(lap_unlimit,1); %this lap vec is between 0 and 1
102
103 figure(17)
104 scatter(lap_vec,AltitudeFeet1,2.5,'filled')
105 xlabel('Lap Position')
106 ylabel('Altitude (ft))')
107 set(gca,'fontsize',13)
108 %print(strcat(photopath,'Alt_ex'),'-djpeg','-r300')
109
110 driver_input = zeros(length(rpmvec),1); %pre-allocating
111
112 delta_t = 1/50; %s
113 speedvec_fts = speedvec*1.46667; %converting to ft/s
114
115 n = 5;
116 RPMfiltered = filter(ones(1,n)/n,[1],rpmvec); %filtering RPM data
117 RPMslopetemp = diff(RPMfiltered); %calculating discrete derivative
118 RPMslope(1) = 0;
119 RPMslope(2:(length(RPMslopetemp)+1)) = RPMslopetemp;
120 %this term shifts the left side of the input to the left
121 samples_delay1 = 0;%300; %these are used to shift the driver input
122 samples_delay2 = 150; %this should be about 1/4*n
123 burning = 0;
124
125 for i = 1:length(rpmvec)
126     if i < n + 1 || i > length(rpmvec) + 101
127         driver_input(i) = 0;
128     else
129         if rpmvec(i) > 0 && max(rpmvec((i+30):(i+200))) > 1000 &&
              ...
130                 burning == 0 && RPMslope(i) > 0
131             burning = 1;
132         elseif burning == 1 && rpmvec(i) > 2000 && RPMslope(i) <
```

```
            -10   ...
133                 && rpmvec(i+200) < 5
134             burning = 0;
135
136         end
137
138         if burning == 1
139             driver_input(i) = 1;
140         else
141             driver_input((i-n):i) = 0;
142             %the delay of ten samples is so the engine doesn't
                    stop before
143             %the driver "input" is 0
144         end
145     end
146 end
147
148 % this loop calculates when the driver is using the engine.
149 % I need to know when the driver is not using the engine so I can
150 % get correct coastdown data
151 % for i = (samples_delay2+1):length(RPMslope)
152 %
153 %     if RPMslope(i) > 0 || burning == 1
154 %         driver_input(i) = 1;
155 %         driver_input(i - samples_delay1) = 1;
156 %         burning = 1;
157 %     end
158 %     if burning == 1 && RPMslope(i) < -7 && RPMfiltered(i) > 1000
159 %         driver_input(i - samples_delay2) = 0;
160 %         burning = 0;
161 %     end
162 %     if burning == 0
163 %         driver_input(i - samples_delay2) = 0;
164 %     end
165 % end
166
167  %finding slope of course
168 %  accelmag = mean(sqrt(AccelXG1.^2+AccelYG1.^2)); %finding total
      gravity,const
169 %  track_slope = atand(AccelYG1./accelmag); %trigonometry
170 %  figure(3)
171 %  scatter(lap_vec,track_slope,1.5,'filled')
172 %  n = 2000;%30;
173 %  slopefiltered = filter(ones(1,n)/n,[1],track_slope);
174 %  hold on
175 %  scatter(lap_vec,slopefiltered,1.5,'filled')
176 %  title('Track Slope (deg) vs Distance (Lap Position)')
177 %  %to verify this makes sense, make sure the integral of slope is
```

```
      0
%  %slopetest = cumtrapz(time,slopefiltered);
%  %figure(70)
%  %plot(time,slopetest) %starts and ends at 0, so overall ok
%
%   figure(70)
%   plot(time,AltitudeFeet1)

% slope data was gathered from google earth
earth_ftvec =
    [0,531,635,1291,1687,2025,2200,0.52*5280,0.71*5280,0.77*5280,...

     0.94*5280,0.97*5280,5279] - 635; %ft
earth_mivec = earth_ftvec/5280;
earth_lapvec = mod(earth_mivec,1);
earth_height =
    [191.5,185,184,177,174,169,168,174,175,179,192,193,189];
[heighttablex,earthsort] = sort(earth_lapvec);
heighttabley = earth_height(earthsort);
% figure(3)
% plot(heighttablex,heighttabley)
% title('Track Height (ft) vs Distance (lap position)')


figure(2)
plot(time,rpmvec)
hold on
plot(time,RPMfiltered)
hold on
plot(time,driver_input*4000)
title('RPM and driver input')

%figure(7)
%plot(time,RPMslope)

n = 100; %now filtering speed from GPS data %group delay from 100
    is ~50 samples
speedfiltered = filter(ones(1,n)/n,[1],speedvec_fts);
n = 30;%100;
accltemp = filter(ones(1,n)/n,[1],diff(speedfiltered)/delta_t); %
    in ft/s^2
accl(1) = 0;
accl(2:(length(accltemp)+1)) = accltemp;

figure(4)
plot(time,speedvec_fts)
hold on
plot(time,speedfiltered)
```

```
hold on
plot(time,driver_input*20)
% hold on
% plot(time,speedvectest)
%title('Speed and driver input')
xlabel('Time (s)')
ylabel('Speed (ft/s)')
axis([180 590 0 45])
set(gca,'fontsize',13)
print(strcat(photopath,'Speed_ex'),'-djpeg','-r500')

n = 50;
testaccel = filter(ones(1,n)/n,[1],AccelYG1); %accel X is pointing
    down,Y is straight forward
%, Z is right or left
testaccel = testaccel*32.174; %converting to ft/s^2
speedinttest = cumtrapz(time,testaccel);
figure(3)
scatter(time,speedinttest,2,'filled')

figure(10)
scatter(lap_vec,testaccel,2,'filled')
%hold on
%plot(time,accl)
%  figure(10)
%  plot(time,AccelXG1)
%  hold on
%  plot(time,AccelYG1)
%  hold on
%  plot(time,AccelZG1)
%  title('Y Acceleration')

figure(2)

%coastdown_model_script_V2
%burn_model_script_V2

%%
%Trying to calculate a better speed here
figure(12)
xposvec = LongitudeDegrees1/90*6214*5280;
yposvec = LatitudeDegrees1/90*6214*5280;
plot(time,xposvec)

ninterp = 200; %the number of points to regress through

xspeed = zeros(length(xposvec),1);
yspeed = zeros(length(yposvec),1);
```

```
for j = 30000:35000%1:1:(length(xposvec)-ninterp)
    subx = xposvec(j:(j+ninterp-1));
    suby = yposvec(j:(j+ninterp-1));
    subt = time(j:(j+ninterp-1));
    [x_C,xerror] = polyfit(subt,subx,1); %1 is a linear fit
    [y_C,yerror] = polyfit(subt,suby,1); %1 is a linear fit
    xspeed_C = polyder(x_C);
    yspeed_C = polyder(y_C);
    xspeed(j) = mean(polyval(xspeed_C,mean(subt)));
    yspeed(j) = mean(polyval(yspeed_C,mean(subt)));
%     figure(13)
%     close
    h = figure(13);
    set(h,'Position',[300,300,390,300])
    plot(subt,subx,'b')
    hold on
    plot(subt,polyval(x_C,subt),'r')
%     figure(14)
%     close
    h = figure(14);
    set(h,'Position',[700,300,390,300]) % left bottom wid height
    plot(subt,suby,'b')
    hold on
    plot(subt,polyval(y_C,subt),'r')
    %pause

end
speedvectest = sqrt(xspeed.^2 + yspeed.^2);
figure(12)
plot(time,speedvectest)

% speedvectest = zeros(length(xposvec)-1,1);
% for i = 2:10:length(xposvec)
%     speedvectest(i) = (sqrt((xposvec(i)-xposvec(i-1))^2 +...
%         (yposvec(i)-yposvec(i-1))^2));%/(time(i) - time(i-1));
% end

% figure(12)
% plot(time,speedvectest)


%time to filter speed data, buildings are rough
% sigmaspeed = std(speedvec);
% for i = 2:length(speedvec)
%     if abs(speedvec(i) - speedvec(i-1)) > sigmaspeed/2
%         speedvec(i) = speedvec(i-1);
%     end
% end
```

regressclb.m

```
function [ bestfit,C] = regressclb( xs,ys,polynomialr )
% Uses least squares regression for polynomial, outputs bestfit and
% coefficients, input unknown function x points with corresponding y points
% and the degree of the polynomial

% Made by: Chad Bickel
% 06/06/2014

%turn singularity warning off
id = 'MATLAB:nearlySingularMatrix';
warning('off',id);

n = length(xs);
r = polynomialr;

B = ones(n,r);
for j = 2:r
for i = 1:n
B(i,j) = (xs(i))^(j-1);
end
end
x = (B'*B)^(-1)*B'*ys;

for i = 1:r
    X(r-i+1) = x(i);
end

C = X;

bestfit = zeros(n,1);
for i = 1:n
for j = 1:r
bestfit(i) = X(j)*(xs(i))^(r-j) + bestfit(i);
end
end

warning('on',id);
end
```

coastdown_model_script_V4.m

```
% Chad Bickel
% 3/8/17
% Analyzes coastdown portion of RaceCapture data
% Run after experiminal_supermileage_model script

%% Table-making here
% First the coastdown data
%c_indices = [1.075e4,1.16e4]; %these give an acc of -0.01
%c_indices = [7000,10200]; %these give an acc of 0 to -0.01

k = 1;
[pks,locs] = findpeaks(speedfiltered,'MinPeakDistance',1500);
input_delay = 0;

temp_start = 0;
index2 = 1;
t_cell = {};
d_cell = {};
v_cell = {};
t_temp = [];
v_temp = [];
d_temp = [];

for i = 1:length(driver_input)
    if (driver_input(i) < 1) && (speedfiltered(i) > 11) &&(accl(i)
        <0.01...
            &&accl(i) > -100)
        x_c(1,k) = speedfiltered(i+input_delay);%speedvec_fts(i+
            input_delay);
        x_c(2,k) = lap_vec(i+input_delay);
        %x_c(3,k) = slopefiltered(i+input_delay);
        y_c(1,k) = accl(i+input_delay);
        temp(1,k) = time(i+input_delay);
        if k > 2
            if (temp(k) > (temp(k-1) + 5))|| ...
                    x_c(2,k) > x_c(2,k) + 0.02%if there is a 3 sec
                        jump
                temp_start = time(i+input_delay);
                if length(t_temp) > 3
                    t_cell{index2} = t_temp;
                    v_cell{index2} = v_temp;
                    d_cell{index2} = d_temp;
                end
                t_temp = [];
                v_temp = [];
                d_temp = [];
```

```
                    index2 = index2 + 1;
                end
            end
            if x_c(1,k) ~= 0
                t_temp = [t_temp,time(i+input_delay) - temp_start];
                v_temp = [v_temp,x_c(1,k)];
                d_temp = [d_temp,x_c(2,k)];
                %slope_temp(index2,k) = x_c(3,k);
            end
            k = k+1;
        end
    end

%%
% figure(5)
% plot(x_c(1,:))
% title('Vehicle Speed')
% figure(6)
% plot(x_c(2,:))
% title('Lap Position')
% figure(7)
% plot(y_c)
% title('Vehicle Acceleration (ft/s^2) Without Slope Removed')
%
%% now to calculate slope from height data
% heights = interp1(heighttablex,heighttabley,x_c(2,:),'linear','
    extrap');
% slopetemp = diff(heights)./diff((x_c(2,:)*5280));
% slopediff(1) = 0;
% slopediff(2:(length(slopetemp)+1)) = slopetemp;
% slope = atand(slopediff);
%
% badindex = find(slope > 10); %clearly not right
% slope(badindex) = -0.7;
%
% %figure(8)
% %plot(slope)
% %pause
%
% %gravity_accel = -sind(x_c(3,:))*32.174;
% gravity_accel = -sind(slope)*32.174;
% corrected_acceleration = y_c - gravity_accel;
% hold on
% plot(corrected_acceleration)
% figure(20)
% scatter(x_c(1,:),y_c,1.5,'filled')
% hold on
% scatter(x_c(1,:),corrected_acceleration,1.5,'filled')
```

```
91  % %plot(x_c(3,:))
92  % % xinput = x_c(1,:);
93  % % [~,sortindex] = sort(xinput);
94  % % scatter(xinput(sortindex),y_c(sortindex),10,'filled')
95  % % title('Vehicle Acceleration')
96
97  % figure(8)
98  % close
99  % figure(8)
100 % for i = 1:(index2-1)
101 %     scatter(t_temp(i,:),v_temp(i,:),10,'filled')
102 %     hold on
103 %     %disp(i)
104 %     %pause
105 % end
106 % title('Speed (ft/s) vs Time (s)')
107 % %legend('1','2','3','4','5','6','7','8','9','10','11')
108
109 figure(9)
110 close
111 figure(9)
112 for i = 1:(index2-1)
113     scatter(d_cell{i},v_cell{i},5,'filled')
114     %disp(i)
115     %pause
116     hold on
117 end
118 xlabel('Lap Position')
119 ylabel('Speed (ft/s)')
120 set(gca,'fontsize',13)
121 print(strcat(photopath,'Speed_fund'),'-djpeg','-r300')
122 %% Look at downhill coefficients as a function of time
123 ninterp = 200; %the number of points to regress through
124 m_vehicle = 240/32.174; %slugs
125 k = 1;
126 road_keep = {};
127 index = 9;
128 v_temp = v_cell{index};
129 t_temp = t_cell{index};
130 d_temp = d_cell{index};
131 for j = 1:50:(length(v_temp)-ninterp)
132     subv = v_temp(j:(j+ninterp-1));
133     subt = t_temp(j:(j+ninterp-1));
134     subd = d_temp(j:(j+ninterp-1));
135     [v_C,verror] = polyfit(subt,subv,3);
136     [vbestfit,vdelta] = polyval(v_C,subt,verror);
137     accl_C = polyder(v_C);
138     t_accl = polyval(accl_C,subt);
```

```
139     [road_C,aerror] = polyfit(vbestfit,t_accl*m_vehicle,1);%subv,
            t_accl*m_vehicle,2);
140     [abestfit,adelta] = polyval(road_C,subv,aerror);
141     figure(30)
142     close
143     h=figure(30);
144     set(h,'Position',[300,300,390,300])
145     plot(subt,subv)
146     hold on
147     plot(subt,vbestfit)
148     figure(31)
149     close
150     h=figure(31);
151     set(h,'Position',[700,300,390,300]) % left bottom wid height
152     plot(subv,t_accl*m_vehicle)
153     hold on
154     plot(subv,abestfit)
155     Vr = mean(vdelta)
156     Ar = mean(adelta)
157     %pause
158     if (Vr < 0.05) || (Ar < 0.4) %make sure error is within
            tolerances
159         road_keep{k,index} = {mean(subt),road_C}; %each column is
                from the same coastdown
160         k = k+1;
161     end
162 end
163
164 [roadn,roadm] = size(road_keep);
165 timeplot = [];
166 roadloadplot = [];
167 for i = 1:roadn
168     celltemp = road_keep{i,index};
169     roadloadtemp = celltemp{2};
170     timetemp = celltemp{1};
171     timeplot = [timeplot,timetemp];
172     roadloadplot = [roadloadplot,roadloadtemp(1)];
173 end
174
175 figure(32)
176 close
177 figure(32)
178 plot(timeplot,roadloadplot)
179
180 %%
181 %title('Speed (ft/s) vs Distance (Lap Position)')
182 %now only use the data we like, t_temp is a row vector
183 good_t = t_cell{2};
```

```
184 good_v = v_cell{2};
185 good_d = d_cell{2};
186 for i = [4 5 7 8 10]%[2 4 5 7 8 10 11]
187     good_t = cat(2,good_t,t_cell{i});
188     good_v = cat(2,good_v,v_cell{i});
189     good_d = cat(2,good_d,d_cell{i});
190     %disp(i)
191     %disp(max(t_temp(i,:)))
192 end
193 k = find(good_v == 0);
194 good_v(k) = NaN;
195 figure(10)
196 close
197 figure(10)
198 scatter(good_t,good_v,1.3,'filled')
199 ylabel('Speed(ft/s)')
200 xlabel('Time(s)')
201
202 [sortedv,sortindex] = sort(good_v);
203 max_i = find (isnan(sortedv),1);
204 sortedt = good_t(sortindex);
205 real_v = sortedv(1:(max_i-1));
206 real_t = sortedt(1:(max_i-1));
207
208 [ bestfit,v_C] = regressclb(sortedt',sortedv',3);
209  hold on
210  scatter(sortedt,bestfit,1.5,'filled')
211
212  %finding acceleration coefficients here
213  accl_C = polyder(v_C);
214 % pause
215 % close
216 % figure(10)
217 % plot(good_t(sortindex),good_v(sortindex))
218 % ylabel('Speed(ft/s)')
219 % xlabel('Time(s)')
220 % [ bestfit,C] = regressclb( good_t(sortindex)',good_v(sortindex)
      ',2);
221 % hold on
222 % scatter(good_t,bestfit)
223 % p coefficients are 0.005028,-0.8052,35.51
224 % coefficients for accl are then 0.0101, -0.8052
225 accl_vec = 0.0101*sortedt -0.8052;
226 % these coefficients are 0.0001926,-0.02503,-0.1548 for
      acceleratoin v
227 % speed
228
229 figure(11)
```

```
230 close
231 figure(11)
232 m_vehicle = 240/32.174; %slugs
233 slope_on_coastdown = 0.01;%0.011; %positive if going uphill
234 gravityaccl = sind(atand(slope_on_coastdown))*32.174;
235 %should be positive if heading up a hill, slope on uphill is
        1.1-1.5%
236 true_acceleration = -polyval(accl_C,sortedt) - gravityaccl;
237 plot(polyval(v_C,sortedt),true_acceleration*m_vehicle)
238 hold on
239 [bestfit,road_C] = regressclb(polyval(v_C,sortedt)',
        true_acceleration'*m_vehicle,3);
240 plot(polyval(v_C,sortedt),bestfit)
241 ylabel('Force(lbf)')
242 xlabel('Speed(ft/s)')
243 set(gca,'fontsize',13)
244 print(strcat(photopath,'Coastdown_final'),'-djpeg','-r300')
245 road_C
246 %%
247 [bestfit,C] = regressclb( xinput',y_c',3);
248
249 figure(8)
250 close
251 figure(8)
252 plot(xinput(sortindex),y_c(sortindex))
253 hold on
254 plot(xinput(sortindex),bestfit(sortindex))
255
256 xtest = linspace(0,30,100);
257 figure(9)
258 plot(xtest,polyval(C,xtest))
259 %% Now use a neural network to fit a curve to the nonlinear data
        easily
260 close all
261 trainFcn = 'trainlm';  % Levenberg-Marquardt
262
263 disp('Training Neural Network to fit Nonlinear Coastdown data')
264
265 % Create a Fitting Network
266 hiddenLayerSize = 100; %20
267 c_net = fitnet(hiddenLayerSize,trainFcn);
268
269 % Setup Division of Data for Training, Validation, Testing
270 c_net.divideParam.trainRatio = 70/100;
271 c_net.divideParam.valRatio = 15/100;
272 c_net.divideParam.testRatio = 15/100;
273
274 % Input data is the acceleration data minus the best curve fit for
```

```
% velocity, since f_drag(v,lap) = f(v) + f(lap)
xinput = x_c(2,:);
n = 300;
yinput = filter(ones(1,n)/n,[1],(y_c - bestfit'));

% Train the Network
[c_net,tr] = train(c_net,xinput,yinput);

% Test the Network
ytest = c_net(xinput);

figure(10)
plot(yinput)
hold on
plot(ytest)
title('Actual Acceleration Data and Predicted Acceleration Data')

[~,sortindex] = sort(xinput);
figure(11)
plot(xinput(sortindex),yinput(sortindex))
hold on
plot(xinput(sortindex),ytest(sortindex))
title('Actual Acceleration Data and Predicted Acceleration Data')
% plot(speedfiltered)
% axis([c_indices(1) c_indices(2) 0 max(speedfiltered)])
%
% figure(5)
% plot(diff(speedfiltered))
% axis([c_indices(1) c_indices(2) -0.1 0.1])
%gensim(c_net)

%% Generate plots from the neural network fit
x_test(1,:) = linspace(15,30,100);
x_test(2,:) = 0.1*ones(1,100);
ytest = c_net(x_test);
x_test(2,:) = 0.6*ones(1,100);
ytest2 = c_net(x_test);
x_test(2,:) = 0.8*ones(1,100);
ytest3 = c_net(x_test);

figure(10)
close
figure(10)
plot(x_test(1,:),ytest)
hold on
plot(x_test(1,:),ytest2)
plot(x_test(1,:),ytest3)
xlabel('Vehicle Speed ft/s')
```

```
ylabel('Acceleration ft/s^2')
title('Road Load')

x_test(2,:) = linspace(0,1,100);
x_test(1,:) = 20*ones(1,100);
ytest = c_net(x_test);
figure(11)
plot(x_test(2,:),ytest)
xlabel('Position on Track (laps)')
ylabel('Acceleration ft/s^2')
title('Road Load')

%%

n = 10;
[xmesh,x2mesh] = meshgrid(linspace(15,31,n),linspace(0,1,n));

disp('Generating points for 3D plot')
ymesh = zeros(n,n);
for i = 1:n
    tic
    for j = 1:n
        ymesh(i,j) = c_net([xmesh(i,j);x2mesh(i,j)]);
    end
    disp([num2str(i/n*100) ' % Complete'])
    disp([num2str(toc*(n-i)/60) ' Minutes Remaining'])
end

figure(12)
surf(xmesh,x2mesh,ymesh)
shading interp
title('3D Surface of Vehicle Acceleration')
xlabel('Speed (ft/s)')
ylabel('Vehicle Position (lap)')
zlabel('Acceleration ft/s^2')
```

burn_model_script_V3.m

```
% Chad Bickel
% 3/8/17
% Analyzes burn portion of RaceCapture data
% Run after experimental_supermileage_model script

rwheel = 0.79; %ft
%eff = 0.95; %efficiency of final drive
gear_ratio = 12;
m_vehicle = 220/32.174; %slug
j = 0;
k = 1;
rpmaccl = RPMslope/delta_t;
%each column is a set of data
clutchlockindex = [];
for i = 2:length(driver_input)
    if driver_input(i) == 1
        if driver_input(i-1) == 0
            j = j+1;
            k = 1;
            time_start = time(i);
            clutchlockindex(j) = 0;
        end
        rpm_keep(k,j) = rpmvec(i);
        rpmaccl_keep(k,j) = rpmaccl(i);
        speed_keep(k,j) = speedfiltered(i);
        accl_keep(k,j) = accl(i);
        instant_fuel_keep(k,j) = instant_fuel_mL(i);
        time_keep(k,j) = time(i) - time_start;
        if clutchlockindex(j) == 0 && rpmvec(i) > 2700
            if abs(rpmvec(i) - ...
                    speedfiltered(i)/rwheel*gear_ratio*60/(2*pi))
                       < 600
                clutchlockindex(j) = k;
            end
        end
        k = k + 1;
    end
end

figure(7)
close
figure(7)
ikeep = [2 3 4 5 6 7 9 10 11]; %columns of data to keep
for i = ikeep
    hold on
    plot(time_keep(:,i), ...
```

```
46          rpm_keep(:,i) - speed_keep(:,i)/rwheel*gear_ratio*60/(2*pi
                ))
47      %pause
48  end
49
50  figure(8)
51  close
52  figure(8)
53  for i = ikeep
54      hold on
55      scatter(time_keep(:,i), ...
56          rpm_keep(:,i),2.5,'filled')
57      %pause
58  end
59  xlabel('Time (s)')
60  ylabel('Engine Speed (rpm)')
61  % set(gca,'fontsize',13)
62  % print(strcat(photopath,'rpmclutchtime'),'-djpeg','-r300')
63
64  %%now finding acceleration from speed
65  % for i = ikeep
66  %     [bestfit(:,i),speed_coeff] = regressclb(time_keep(:,i),
        speed_keep(:,i),8);
67  %       accl_C = polyder(speed_coeff);
68  %       accl_keep2(:,i) = polyval(accl_C,time_keep(:,i));
69  % end
70
71  figure(9)
72  close
73  figure(9)
74  for i = ikeep
75      hold on
76      scatter(time_keep(:,i) - 1,speed_keep(:,i)/rwheel*gear_ratio
            *60/(2*pi),2.5,'filled')
77  end
78  xlabel('Time (s)')
79  ylabel('Clutch Housing Speed (rpm)')
80  % set(gca,'fontsize',13)
81  % print(strcat(photopath,'houseclutchtime'),'-djpeg','-r300')
82
83  %% Determining Drivetrain Efficiency here
84  % first get rid of any zero rpm stuff
85  cancelrpm = find(rpm_keep < 500);
86  rpm_keep(cancelrpm) = NaN;
87  %
88  % figure(11)
89  % close
90  % figure(11)
```

```
% for i = ikeep
%     hold on
%     maxlength = length(rpm_keep(:,i));
%     scatter(rpm_keep(clutchlockindex(i):maxlength,i), ...
%     Tc(clutchlockindex(i):maxlength,i),2.5,'filled')
% end
% xlabel('Engine Speed (RPM)')
% ylabel('Engine Torque (lb*ft)')

coast_poly = [3*2.4369e-04 0.0799 0.7243];

engine_rpm = [0 1000 2000 3000 4000 5000 6000 7000 8000];
engine_torque = [0, 1.5, 2.2, 2.328, 2.5382, 2.48, 2.2, 2, 1.9]; %
    lb*ft

figure(12)
close
figure(12)
diffkeep = {};
j = 1;
for i = ikeep
    disp(i)
    hold on
    maxlength = length(rpm_keep(:,i));
    theory_torque = interp1(engine_rpm,engine_torque,rpm_keep(...
        clutchlockindex(i):maxlength,i));
    theory_accl = (theory_torque*gear_ratio/ ...
        rwheel - polyval(coast_poly,speed_keep(...
        clutchlockindex(i):maxlength,i)))/m_vehicle; %assuming
            perfect eff
    scatter(rpm_keep(clutchlockindex(i):maxlength,i), ...
    theory_accl,2.5,'filled','k')
    hold on
    scatter(rpm_keep(clutchlockindex(i):maxlength,i),accl_keep(...
        clutchlockindex(i):maxlength,i),2.5,'filled')

    diffkeep{j} = 1 - (theory_accl-accl_keep(...
        clutchlockindex(i):maxlength,i))./theory_accl;
    j = j + 1;
end
xlabel('Engine Speed (rpm)')
ylabel('Vehicle Acceleration (ft/s^2)')
% set(gca,'fontsize',13)
% print(strcat(photopath,'acclclutchtime'),'-djpeg','-r300')

figure(13)
close
figure(13)
```

```
137 j = 1;
138 effmat = 0;
139 for i = ikeep
140     maxlength = length(rpm_keep(:,i));
141     hold on
142     scatter(rpm_keep(clutchlockindex(i):maxlength,i),diffkeep{j
        },2.5,'filled')
143     effmat(j) = nanmean(diffkeep{j});
144     j = j+1;
145 end
146 xlabel('Engine Speed (rpm)')
147 ylabel('Drivetrain Efficiency')
148 % set(gca,'fontsize',13)
149 % print(strcat(photopath,'efftime'),'-djpeg','-r300')
150 eff = mean(effmat)
151
152 %% Now look at clutch slipping condition to determine the torque
    that the
153 % Clutch is delivering to the vehicle
154
155 % Find acceleration due to rolling resistance, which will be the
    average
156 % of the first few terms of all the vehicle accelerations
157
158 for i = ikeep
159    accl_keep(:,i) = accl_keep(:,i) - mean(accl_keep(1:10,i));
160 end
161
162 % First convert vehicle acceleration to Torque at clutch
163 for i = ikeep
164     Tc(:,i) = m_vehicle*accl_keep(:,i)*rwheel/(eff*gear_ratio);
165 end
166
167 figure(10)
168 close
169 figure(10)
170 rpm_clutch = [];
171 Tc_clutch = [];
172 for i = ikeep
173     hold on
174     rpm_temp = rpm_keep(1:(clutchlockindex(i)-50),i);
175     Tc_temp = Tc(1:(clutchlockindex(i)-50),i);
176
177     %drop Nan values and values when Tc is less than
178     ktemp = find(~isnan(rpm_temp)&(rpm_temp > 2650));
179     rpm_temp = rpm_temp(ktemp);
180     Tc_temp = Tc_temp(ktemp);
181
```

```
182     ktemp = ~isnan(Tc_temp);
183     rpm_temp = rpm_temp(ktemp);
184     Tc_temp = Tc_temp(ktemp);
185 
186     scatter(rpm_temp,Tc_temp,2.5,'filled')
187 
188     rpm_clutch = [rpm_clutch;rpm_temp];
189     Tc_clutch = [Tc_clutch;Tc_temp];
190 end
191 xlabel('Engine Speed (RPM)')
192 ylabel('Clutch Torque (lb*ft)')
193 
194 C_clutch = polyfit( rpm_clutch,Tc_clutch,2 );
195 
196 C_c = C_clutch(1)
197 wprime = sqrt(C_clutch(3)/C_c)
198 
199 %C_c*(60)^2/(2*pi)^2 C_c in lbf/(rad/s)^2
200 
201 hold on
202 plot(sort(rpm_clutch),polyval(C_clutch,sort(rpm_clutch)),'k','
    LineWidth',2)
203 
204 % set(gca,'fontsize',13)
205 % print(strcat(photopath,'clutchtorque'),'-djpeg','-r300')
206 
207 %% Verifying BSFC table here
208 
209 %engine_BSFC = [1 0.6067 0.2758 0.3909 0.3884 0.3914 0.4 0.4
    0.4];%lbm/(hp*hr);
210 engine_BSFC = [1 0.7 0.58 0.56 0.55 0.55 0.55 0.55 0.55];
211 figure(14)
212 close
213 figure(14)
214 rpm_arr = [];
215 fuel_arr = [];
216 for i = ikeep
217     hold on
218     rpm_temp = rpm_keep(:,i);
219     fuel_temp = instant_fuel_keep(:,i);
220 
221     %drop Nan values
222     ktemp = find(~isnan(rpm_temp));
223     rpm_temp = rpm_temp(ktemp);
224     fuel_temp = fuel_temp(ktemp);
225 
226     ktemp = find(~isnan(fuel_temp)&(fuel_temp>0.0001));
227     rpm_temp = rpm_temp(ktemp);
```

```
228         fuel_temp = fuel_temp(ktemp);
229         scatter(rpm_temp,fuel_temp,2.5,'filled','b') %mL/s
230         %pause
231         rpm_arr = [rpm_arr;rpm_temp];
232         fuel_arr = [fuel_arr;fuel_temp];
233     end
234
235     rpm_theory = linspace(0,8000,100);
236     bsfc_theory = interp1(engine_rpm,engine_BSFC,rpm_theory);
237     e_torque_theory = interp1(engine_rpm,engine_torque,rpm_theory);
238     fuel_use_theory = rpm_theory.*e_torque_theory/5252.*...
239         bsfc_theory/3600/6.073*3785.41;
240
241     hold on
242     plot(rpm_theory,fuel_use_theory,'k','LineWidth',1.7)
243
244     C_fuel = polyfit( rpm_arr,fuel_arr,2);
245     hold on
246     plot(rpm_theory,polyval(C_fuel,rpm_theory),'r','LineWidth',1.7)
247     xlabel('Engine Speed (rpm)')
248     ylabel('Fuel Flow Rate (mL/s)')
249     set(gca,'fontsize',13)
250     print(strcat(photopath,'bsfcverify'),'-djpeg','-r300')
251
252     % ~0.1 mL/s are used at 3000 rpm
253     fuel_use_actual = polyval(C_fuel,rpm_theory);
254     newbsfcfun = fuel_use_actual.*3600*6.073/3785.41*5252./...
255         (rpm_theory.*e_torque_theory);
256     newbsfctable = interp1(rpm_theory,newbsfcfun,engine_rpm)
```

weight_roadload_script.m

```
% Chad Bickel
% 3/8/17
% Determines effect of weight on road load coefficients

clear all
close all
clc

%% Coastdown testing script
% Changed on 12/8/16

photopath1 = 'C:\Users\Chad\Google Drive\NEW Documents\Thesis';
photopath2 = '\Photos\MatlabOutput\';
photopath = strcat(photopath1,photopath2);

start_weight = (120 + 103.4)/32.174; %slug
testing_weights = [NaN,0,0,0,60.4,60.4,60.4]/32.174;
total_weight = start_weight + testing_weights; %slugs

road_load_polyorder = 2; % 3 is x^2

for j = 1:length(testing_weights)
    clear pushindices
    clear speed
    clear speedfiltered
    clear tempspeed
    clear keepspeed
    clear M
    if isnan(testing_weights(j))
        C_coast_lbf(j,:) = NaN*ones(1,road_load_polyorder);
    else

RCstring = ['RC_',num2str(j-1),'.LOG'];
M = csvread(RCstring,1,0);

time = (M(:,1) - M(1,1))*1e-3; %time in ms
speed = M(:,32)*1.46667; %ft/s

accelx = M(:,4);
accely = M(:,5);
accelz = M(:,6);
altitude = M(:,34); %ft

n = 1000;
accelxfiltered = filter(ones(1,n)/n,[1],accelx);
figure(97)
```

```
plot(accelxfiltered)
hold on
plot(accely)
hold on
plot(accelz)

% Get rid of the weird zeros that pop on altitude
keepalt = 0;
for i = 1:length(altitude)
    tempalt = altitude(i);
    if tempalt > 0
       keepalt = tempalt;
    end
    altitude(i) = keepalt;
end
figure(98)
plot(altitude)
%this loop zero order holds the speed values taken at 10Hz
keepspeed = 0;
for i = 1:length(speed)
    tempspeed = speed(i);
    if tempspeed > 0
       keepspeed = tempspeed;
    end
    speed(i) = keepspeed;
end


n = 100;
speedfiltered = filter(ones(1,n)/n,[1],speed);

figure(1)
plot(speed)
hold on
plot(speedfiltered)
xlabel('Sample')
ylabel('Speed (ft/s)')

deltaspeed = diff(speedfiltered);
deltaspeed(length(speedfiltered)) = 0;
figure(2)
plot(deltaspeed)
xlabel('Sample')
ylabel('Delta speed')

%now to find when the push and brake occured
% pushindices = [0,0,0,0];
% for i = 1:length(deltaspeed)
```

```
95  %     if deltaspeed(i) > 0.04
96  %         if pushindices(1) == 0
97  %             pushindices(1) = i + 30;
98  %         elseif i > pushindices(1) + 500;
99  %             pushindices(3) = i + 30;
100 %         end
101 %     elseif deltaspeed(i) < -0.04
102 %         if pushindices(2) == 0
103 %             pushindices(2) = i - 120;
104 %         elseif i > pushindices(2) + 500;
105 %             pushindices(4) = i - 120;
106 %         end
107 %     end
108 %     if (pushindices(4) > 0) && (i > (pushindices(4) + 50 ))
109 %         break
110 %     end
111 % end
112
113 %speed = speedfiltered; %temporary
114
115 % finding all speeds where the delta speed is negative for a short
        period
116 % These are the valuable coastdown data points
117 % it's a coastdown test if car is rolling freely for at least 45
      seconds
118 ikeep = [];
119 istart = 0;
120 iend = 0;
121 for i = 1:length(deltaspeed)
122     if i < (length(deltaspeed) - 2300)
123         if (deltaspeed(i) < - 0.003)&&(max(deltaspeed(i:(i+2250)))
                < -0.003)&&(istart == 0)
124             istart = i;
125         end
126     end
127     if (deltaspeed(i) > - 0.003 && istart > 0)
128         iend = i;
129         ikeep = [ikeep,istart:(iend-100)];% the last number is a
                tolerance to cut off the end
130         istart = 0;
131     end
132 end
133
134 % coastdown_speeds = [speed(pushindices(1):pushindices(2)); ...
135 %     speed(pushindices(3):pushindices(4))];
136 % coastdown_times = [time(pushindices(1):pushindices(2)) - ...
137 %     time(pushindices(1)); time(pushindices(3):pushindices(4)) -
      ...
```

```
138 %      time(pushindices(3))];
139
140 coastdown_speeds = speed(ikeep);
141 coastdown_times = time(ikeep);
142
143 figure(3)
144 close
145 figure(3)
146 plot(time,speed)
147 hold on
148 scatter(coastdown_times,coastdown_speeds)
149 xlabel('Time (s)')
150 ylabel('Speed (ft/s)')
151 %pause
152
153 [~,sortedi] = sort(coastdown_times);
154 figure(4)
155 close
156 figure(4)
157 scatter(coastdown_times,coastdown_speeds)
158 xlabel('Time')
159 ylabel('Speed')
160
161 if iend > 0
162     [ bestfitspeed,C] = regressclb( coastdown_times(sortedi),
        coastdown_speeds(sortedi), 2 );
163     hold on
164     plot(coastdown_times(sortedi),bestfitspeed)
165
166     C_accl = polyder(C); %taking deriv with respect to t
167     accl = polyval(C_accl,coastdown_times(sortedi));
168     [bestfitaccl,C_coast] = regressclb(bestfitspeed,accl,
        road_load_polyorder);
169
170     figure(5)
171     scatter(coastdown_speeds,accl)
172     hold on
173     scatter(bestfitspeed,bestfitaccl)
174     %pause
175
176     C_coast_lbf(j,:) = C_coast*total_weight(j);
177 end
178
179     end
180 end
181 %%
182 Frr = C_coast_lbf(:,2);
183 Vrr = C_coast_lbf(:,1);
```

```
184 junki = find(Frr > -0.000001);
185 Frr(junki) = NaN;
186 Vrr(junki) = NaN;
187
188 figure(5)
189 close
190 figure(5)
191 scatter(total_weight*32.174,Vrr)
192 xlabel('Total Vehicle Weight (lbf)')
193 ylabel('Vehicle Speed Force Coefficient (lbf/(ft/s))')
194
195 %hold on
196 %scatter(total_weight,C_coast_lbf(:,2))
197
198 delta_Frr = Frr; %- max(Frr); %only interested in overall change
      of Frr
199 figure(6)
200 scatter(total_weight*32.174,-delta_Frr)
201
202 [Frrsort,sorti] = sort(delta_Frr);
203 temp_last = find(isnan(Frrsort),1);
204
205 sorted_weight = total_weight(sorti);
206 disp('Coefficient in lbf/deltaslug is C_final first number')
207 [bestfit_final,C_final] = regressclb(sorted_weight(1:(temp_last-1)
      ),Frrsort(1:(temp_last-1)),2)
208
209 hold on
210 plot(sorted_weight(1:(temp_last-1))*32.174,-bestfit_final)
211 %title('Change in Rolling Resistance Force (lbf) vs Total Weight (
      lbf)')
212 xlabel('Total Vehicle Weight (lbf)')
213 ylabel('Rolling Resistance Force (lbf)')
214 set(gca,'fontsize',13)
215 print(strcat(photopath,'weight_rl'),'-djpeg','-r300')
```

starting_motor_analysis.m

```
% Chad Bickel
% 1/18/17
% Script to analyze the starter motor/ get a fuel
% penalty associated with starting the engine

clear all
close all
clc

photopath1 = 'C:\Users\Chad\Google Drive\NEW Documents\Thesis';
photopath2 = '\Photos\MatlabOutput\';
photopath = strcat(photopath1,photopath2);

M = csvread('RC_1.LOG',1,0); % using data from Santa Maria Airport

time = (M(:,1) - M(1,1))*1e-3; %time in ms
speed = M(:,32)*1.46667; %ft/s

AFR = M(:,32-13);
Coolant = M(:,32-15); % F
rpmvec = M(:,32-19); %rpm
fuel_pw = M(:,32-20); %ms %usually this is between 2.5 to 3.5 for
    avg vehicles

figure(10)
plot(time,fuel_pw)

cyclespm = rpmvec/2; %4 stroke engine, so 1 cyc per 2 revs
cyclesps = cyclespm/60; %engine cycles per second

%now we have cycles/s and fuel ms/cycle
instant_fuel_ms = fuel_pw.*cyclesps; %(fuel ms)/s

%now we integrate (fuel ms)/s to get total fuel ms
total_fuel_ms = trapz(time,instant_fuel_ms); %total ms of fuel
    burned

total_fuel = 7; %mL for RC1
%now we need to convert ms of injector open to ml of fuel
mfdotmL = total_fuel/(total_fuel_ms);%mass flow rate of injector,
%in mL per fuel ms, lowest rate I could find was 95 cc/min, this
    is less
%than half

%assume mass flow rate of injector is constant (fuel ms/s)*(mL/
    fuel ms)
```

```
43 instant_fuel_mL = instant_fuel_ms*mfdotmL; %instant fuel usage, in
       mL/s
44
45 total_fuelvec = cumtrapz(time,instant_fuel_mL); %integrate instant
       fuel
46
47 figure(1)
48 plot(time,instant_fuel_mL)
49 xlabel('Time (s)')
50 ylabel('Instantaneous Fuel Use (mL/s)')
51
52 figure(2)
53 plot(time,rpmvec)
54
55 figure(3)
56 plot(time,total_fuelvec)
57
58 figure(4)
59 plot(time,Coolant)
60 ylabel('Coolant Temperature (F)')
61 xlabel('Time (s)')
62
63 % Now to analyze the data. Assume that once the engine hits 1000
       rpm, it is
64 % fully started.
65
66 ikeep = [];
67 startfuelvec = [];
68 starttimevec = [];
69 stopindexvec = [];
70 startindexvec = [];
71 startindex = 0;
72 stopindex = 0;
73 for i = 26:length(time)
74     if (rpmvec(i) > 1) && (rpmvec(i-1) < 1) && (startindex == 0)
           ...
75             && (rpmvec(i-25) < 1)
76         %if starter motor kicks on
77         startindex = i-1;
78     elseif (rpmvec(i) > 1000) && (startindex > 0) && ...
79             (min(rpmvec(i:(i+50))) > 1000)
80         % once motor reaches 1000 rpm
81         stopindex = i - 1;
82         %disp(startindex)
83         ikeep = [ikeep,startindex:stopindex]; %append to index of
               values to keep
84         % now determine how much fuel and how long it took
85         startime = time(stopindex) - time(startindex);
```

```
86          startfuel = total_fuelvec(stopindex) - total_fuelvec(
                startindex);
87          startfuelvec = [startfuelvec,startfuel];
88          starttimevec = [starttimevec,startime];
89
90          startindexvec = [startindexvec,startindex];
91          stopindexvec = [stopindexvec,stopindex];
92          startindex = 0; %now reset startindex
93      end
94
95  end
96
97  rpmkeep = rpmvec(ikeep);
98  timekeep = time(ikeep);
99  fuelkeep = total_fuelvec(ikeep);
100 temp_F_keep = Coolant(ikeep);
101 instant_fuel_mL_keep = instant_fuel_mL(ikeep);
102
103 figure(2)
104 hold on
105 scatter(timekeep,rpmkeep)
106 xlabel('Time (s)')
107 ylabel('Engine Speed (RPM)')
108
109 figure(3)
110 hold on
111 scatter(timekeep,fuelkeep)
112 ylabel('Fuel Used (mL)')
113 xlabel('Time (s)')
114 set(gca,'fontsize',13)
115 print(strcat(photopath,'startfueluse'),'-djpeg','-r300')
116
117 figure(4)
118 hold on
119 scatter(timekeep,temp_F_keep)
120 set(gca,'fontsize',13)
121 print(strcat(photopath,'starttemp'),'-djpeg','-r300')
122
123 % Now need to determine how much fuel is used and how long it
      takes when
124 % starting the motor.
125
126 figure(5)
127 plot(starttimevec)
128 ylabel('Time to Start Motor (s)')
129 xlabel('Start index')
130 set(gca,'fontsize',13)
131 print(strcat(photopath,'starttime'),'-djpeg','-r300')
```

```
figure(6)
plot(startfuelvec)
ylabel('Fuel Used to Start Motor (mL)')
xlabel('Start index')
set(gca,'fontsize',13)
print(strcat(photopath,'startfuel'),'-djpeg','-r300')

% if we assume the start time and start fuel are constants, so use the average
% of each as a penalty % 0.4224 s, 0.0067 mL total, 0.0159 mL/s
mean(starttimevec)
mean(startfuelvec)
avgfuelrate = mean(startfuelvec)/mean(starttimevec) % mL/s

%%

% for more detailed analysis, need to set all the time vectors to start at
% 0, each set of data will also be seperated for further analysis

%each column in the cells is a set of starting data
for i = 1:length(startindexvec)
    time_cell{i} = time(startindexvec(i):stopindexvec(i)) - time(startindexvec(i));
    rpm_cell{i} = rpmvec(startindexvec(i):stopindexvec(i));
    instant_fuel_mL_cell{i} = instant_fuel_mL(startindexvec(i):stopindexvec(i));
    figure(7)
    hold on
    plot(time_cell{i},rpm_cell{i})
end
```

Design_GUI.m

```
% Chad Bickel
% 3/8/17
% Design tool user interface for Shell Eco-marathon prototype vehicle

function varargout = Design_GUI(varargin)
% DESIGN_GUI MATLAB code for Design_GUI.fig
%      DESIGN_GUI, by itself, creates a new DESIGN_GUI or raises the existing
%      singleton*.
%
%      H = DESIGN_GUI returns the handle to a new DESIGN_GUI or the handle to
%      the existing singleton*.
%
%      DESIGN_GUI('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in DESIGN_GUI.M with the given input arguments.
%
%      DESIGN_GUI('Property','Value',...) creates a new DESIGN_GUI or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before Design_GUI_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to Design_GUI_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help Design_GUI

% Last Modified by GUIDE v2.5 08-Mar-2017 12:16:35

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Design_GUI_OpeningFcn, ...
```

```
36                    'gui_OutputFcn',  @Design_GUI_OutputFcn, ...
37                    'gui_LayoutFcn',  [] , ...
38                    'gui_Callback',   []);
39 if nargin && ischar(varargin{1})
40     gui_State.gui_Callback = str2func(varargin{1});
41 end
42
43 if nargout
44     [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
45 else
46     gui_mainfcn(gui_State, varargin{:});
47 end
48 % End initialization code - DO NOT EDIT
49 end
50
51 % --- Executes just before Design_GUI is made visible.
52 function Design_GUI_OpeningFcn(hObject, eventdata, handles,
       varargin)
53 % This function has no output args, see OutputFcn.
54 % hObject    handle to figure
55 % eventdata  reserved - to be defined in a future version of
       MATLAB
56 % handles    structure with handles and user data (see GUIDATA)
57 % varargin   command line arguments to Design_GUI (see VARARGIN)
58
59 % Choose default command line output for Design_GUI
60 handles.output = hObject;
61
62 % Update handles structure
63 guidata(hObject, handles);
64
65 %Here is stuff from Thesis_Design_V2
66 handles.g = 32.174; %ft/s^2, gravity, shouldn't be changed
67 handles.m_vehicle = (109+120)/handles.g; %slugs, mass of vehicle
68 handles.gear_ratio = 12; % Default gear reduction, from engine to
       rear wheel sprocket
69 handles.rwheel = 0.78; %ft, the radius of the rear wheel
70
71 %temp for presentation 0.0003728    -0.0775    -2
72 handles.coastdown_poly = [3*2.4369e-04 0.0799 0.7243];%0.0611
       0.9342];
73 %-1*[-4.5914e-04, -0.0775,      -2];
74 %first value was gained from CFD, ~0.5 lbf at max speed (22mphish)
75 % pac car is A = 2.7340332 ft^2, Cd = 0.075, rho = 0.0023769 slug/
       ft^3
76 % CD = rho*Cd*A/2 = 2.4369e-04 for pac car II
77 %[0.0003728    -0.0775    -2]; %directly from coastdown analysis
78 %[0.0001926,-0.02503,-0.1548]*handles.m_vehicle; %polynomials for
```

```
    road load
% ^^^the first value is multiplied by v^2, the last is a constant
    in lbf
handles.coastdown_v_fts = [0 5 10 15 20 25 30 35 40];
% ^^^ these are the speed points for the road load graph shown in
    the gui
%handles.coastdown_lbf = -[3 3.1 3.2 3.3 3.4 3.5 3.6 3.7 3.8]; %
    this should be negative
handles.coastdown_lbf = polyval(handles.coastdown_poly, ...
    handles.coastdown_v_fts); %evaluating the road load polynomial

handles.m_vehicleold = handles.m_vehicle; %keep track of the
    starting weight
handles.Frrold = handles.coastdown_poly(3); %keep track of
    starting Frr
% ^^these variables are important if a study on weight is
    performed

handles.coastdown_laps = linspace(0,1,10);
handles.slope_force_lbf = -sin(2*pi*handles.coastdown_laps)*0;%
    3.0; %0.5
% ^^^ These variables define the force that the track exerts on
    the vehicle
% as a function of position along the course, from 0 to 1

% starting penalty section
start_fuel_used = 0.0067; %mL of fuel used when starting
handles.startingrpm = 1000; %starting idle speed of engine in rpm
startingtime = 0.4224; % time it takes the motor to start in
    seconds

handles.start_fuel_burn = start_fuel_used/startingtime
    *0.051733933*1e-3;
%first number is mL/s of starting fuel burn
%second number is slugs/mL of gas
handles.startingrpmrate = handles.startingrpm/60*2*pi/startingtime
    ; %rad/s^2

handles.engine_rpm = [0 1000 2000 3000 4000 5000 6000 7000 8000];
handles.engine_torque = [0, 1.5, 2.2, 2.328, 2.5382, 2.48, 2.2, 2,
     1.9]; %lb*ft
% ^^^ Torque Curve data
%the first 3 and last 3 data points are guesses
%output vehicle acceleration is about 3-4 ft/s^2 when locked
woffset = 2730; %clutch start to engage at about 2850 rpm
% should use about 1.6 mL per burn
handles.engine_clutch = 2.2045e-05*(handles.engine_rpm-woffset)
    .^2;
```

```
113 zeroindex = find(handles.engine_rpm < woffset);
114 handles.engine_clutch(zeroindex) = 0;
115 %[0 0 0 1.2 3 7 12 14 17]; %Got from burn_model_scripot_newJan
116 %[0 0.01,0.3, 2, 3, 5, 8, 10, 12]; %GUESS, lb*ft
117 % ^^^ Clutch curve data
118 %handles.engine_fueluse = [0 .03 .04 .09 0.13 0.16 0.26 .36
        .46]*3600/3785.41*6.073...
119     %./(handles.engine_rpm.*handles.engine_torque*190.404e-6); %mL
            /s of fuel being used
120 handles.engine_fueluse = [1 0.7311 0.4654 0.4297 0.3921 0.3995
        0.4494 0.4945 0.5217];
121 %[1 0.6067 0.2758 0.3909 0.3884 0.3914 0.4 0.4 0.4];%lbm/(hp*hr)
122 % ^^^ amount of fuel being consumed as a function of rpm
123     %^^1st and last 3 points are guess, slope is about (0.2-0.16)
            /(400) ml/rpm
124 handles.drivetrain_eff = 0.621;%0.95; %overall drivetrain
        efficiency (0 to 1)
125 % This is the efficiency of the torque sent through the drivetrain
126
127 handles.vmax = 38.35;%35; %default max speed in (ft/s)
128 handles.vmin = 12.5;%14; %defualt min speed in (ft/s)
129
130 handles.road_load_poly_Frr = 0.3596;%-0.1746; % This number gets
        multiplied by deltam
131 % and corresponds to a change in Frr, IE Frrnew = Frrold + delta_m
        *(-0.1746)
132 % This number is in lbf/slugs
133 % Used in weight tag callback and simulation callback
134
135 %sim('Design_Table_simulink_V1')
136 %handles.uitable1
137
138 % The rest of the lines set up the tables and graphs in the GUI
139 tablefill = [handles.engine_rpm;handles.engine_torque;handles.
        engine_clutch;handles.engine_fueluse];
140 set(handles.uitable1, 'Data',tablefill);
141 tablefill = [handles.coastdown_laps;handles.slope_force_lbf];
142 set(handles.lap_load_tag, 'Data',tablefill);
143
144 set(handles.road_load_p, 'String',num2str(handles.coastdown_poly))
        ;
145
146 axes(handles.axes2)
147 plot(handles.engine_rpm,handles.engine_torque)
148 title('Engine Torque (ft*lbf)')
149 axes(handles.axes3)
150 plot(handles.engine_rpm,handles.engine_clutch)
151 title('Clutch Torque (ft*lbf)')
```

```
152 axes(handles.axes4)
153 plot(handles.engine_rpm,handles.engine_fueluse)
154 title('BSFC(lb/hp*hr)')
155 axes(handles.axes5)
156 plot(handles.coastdown_v_fts,handles.coastdown_lbf)
157 title('Road Load (lbf) vs Vehicle Speed (ft/s)')
158 guidata(hObject,handles)
159 axes(handles.axes6)
160 plot(handles.coastdown_laps,handles.slope_force_lbf)
161 title('Road Load due to Track')
162
163 %hObject.Data(1,1) = mat2cell(handles.engine_rpm(1),1);
164 set(handles.max_speed, 'String',handles.vmax);
165 set(handles.min_speed_tag, 'String',handles.vmin);
166 set(handles.weight_tag, 'String',handles.m_vehicle*handles.g);
167 set(handles.gear_tag, 'String',handles.gear_ratio);
168 set(handles.radius_tag, 'String',handles.rwheel);
169 set(handles.drivetrain_eff_tag, 'String',handles.drivetrain_eff
    *100);
170 set(handles.road_load_poly_edit, 'String', handles.
    road_load_poly_Frr)
171 guidata(hObject,handles) %this saves the handle data
172 end
173
174 % UIWAIT makes Design_GUI wait for user response (see UIRESUME)
175 % uiwait(handles.figure1);
176
177
178 % --- Outputs from this function are returned to the command line.
179 function varargout = Design_GUI_OutputFcn(hObject, eventdata,
    handles)
180 % varargout  cell array for returning output args (see VARARGOUT);
181 % hObject    handle to figure
182 % eventdata  reserved - to be defined in a future version of
    MATLAB
183 % handles    structure with handles and user data (see GUIDATA)
184
185 % Get default command line output from handles structure
186 varargout{1} = handles.output;
187 end
188
189 % --- Executes on button press in pushbutton1.
190 function pushbutton1_Callback(hObject, eventdata, handles)
191 % hObject    handle to pushbutton1 (see GCBO)
192 % eventdata  reserved - to be defined in a future version of
    MATLAB
193 % handles    structure with handles and user data (see GUIDATA)
194 %handles.current_datay(2) = 5;
```

```
195 %guidata(hObject,handles)
196 %plot(handles.current_datax,handles.current_datay
197
198 %clutchdeltapol = [0.02, -2.0817e-17]; %don't change
199 engine_inertia = 0.001; %GUESS, engine inertia in slug*ft^2
200 % ^^ the smaller the number, the longer sim takes
201 vstart = 0; %starting speed in ft/s
202
203 max_laps = 10;
204 lap_dist = 0.6; %mi per lap
205 max_time = 24*60; %seconds, max time allowed for race
206
207 % have to determine around what speed the clutch locks up
208
209 clutchfun = @(w) interp1(handles.engine_rpm,handles.engine_torque,
    w) - ...
210     interp1(handles.engine_rpm,handles.engine_clutch,w);
211
212 clutchlockrpm = fzero(clutchfun,3000);
213 handles.clutchlockrpm = clutchlockrpm - 100;
214 warning('off','all')
215 function mpgopt = speed_gear_optimization(xopt)
216     % Used for optimization of simulation with respect to gear
        ratio and
217     % Minimum and Maximum speeds
218
219     handles.vmin = xopt(1);
220     handles.vmax = xopt(2);
221     handles.gear_ratio = xopt(3);
222
223     try
224         simout = sim('Supermileage_Simulation_Final','SrcWorkspace
            ','current');
225         sim_timeout = simout.get('sim_timeout');
226         total_fuel_out = simout.get('total_fuel_out');
227         total_fuel_gal = max(total_fuel_out)*handles.g/6.073;
228         if (max(sim_timeout) > max_time)
229             mpgopt = NaN;%max(sim_timeout); %give it a pos number
230         else
231             mpgopt = -max_laps*lap_dist/total_fuel_gal; %miles per
                gallon
232             % This needs to be negative because optimization
                minimizes
233         end
234         if mpgopt < 0
235             figure(1)
236             scatter(xopt(3),-mpgopt,3,'filled','b')
237             hold on
```

```
238         end
239     catch
240         disp('Warning: Simulation Error occured at')
241         disp(xopt)
242         mpgopt = NaN;
243     end
244 
245 end
246 
247 if (length(handles.gear_ratio) > 1) && (length(handles.vmax) > 1)% new 2/12/17
248     % if gear ratio and vmin and vmax are arrays, assume global
249     % optimization needed
250     figure(1)
251     close
252     vmininput = [handles.vmin(1),handles.vmin(2)];
253     vmaxinput = [handles.vmax(1),handles.vmax(2)];
254     mingear = handles.gear_ratio(1);
255     maxgear = handles.gear_ratio(2);
256 
257     %opts = optimoptions(@fmincon,'Algorithm','interior-point');
258     %opts = optimoptions('fmincon','UseParallel',true);
259     f = 1;
260     while f > 0
261         try %need try catch here because random starting point may not
262             % return a valid mpg, and globalsearch needs actual value first
263             vminstart = rand*(vmininput(2) - vmininput(1)) + vmininput(1);
264             vmaxstart = rand*(vmaxinput(2) - vmaxinput(1)) + vmaxinput(1);
265             gearstart = rand*(maxgear-mingear) + mingear;
266             %[mean(vmininput),mean(vmaxinput),(mingear+maxgear)/2] ...
267             problem = createOptimProblem('fmincon','objective',...
268              @(xopt) speed_gear_optimization(xopt) ...
269              ,'x0',[vminstart,vmaxstart,gearstart] ...
270              ,'lb',[vmininput(1),vmaxinput(1),mingear], ...
271              'ub',[vmininput(2),vmaxinput(2),maxgear]);%,'options',opts);
272             gs = GlobalSearch('MaxTime',10000,'Display','iter','NumTrialPoints', ...
273                 1500,'NumStageOnePoints',60,'DistanceThresholdFactor',0.2);
274             [x,f] = run(gs,problem);
275         catch
276             f = 1;
```

```
277             %disp(getReport(err,'extended'));
278         end
279     end
280     figure(1)
281     xlabel('Final Drive Ratio')
282     ylabel('Fuel Efficiency (MPG)')
283     set(gca,'fontsize',13)
284     disp('Optimal speeds are: ')
285     disp([x(1),x(2)])
286     disp('Optimal Gear Ratio is: ')
287     disp(x(3))
288     disp('Maximum Fuel Efficiency is: ')
289     disp(-f)
290
291 elseif length(handles.gear_ratio) > 1 %gear ratio study
292     mingear = handles.gear_ratio(1);
293     maxgear = handles.gear_ratio(2);
294     gearvec = linspace(mingear,maxgear,100); %last number is
            points to plot
295     for i = 1:length(gearvec)
296         %disp(gearvec(i))
297         handles.gear_ratio = gearvec(i);
298         try
299             simout = sim('Supermileage_Simulation_Final','
                    SrcWorkspace','current');
300         catch
301             %disp('error')
302             handles.gear_ratio = gearvec(i) + 0.04;
303             simout = sim('Supermileage_Simulation_Final','
                    SrcWorkspace','current');
304         end
305         sim_timeout = simout.get('sim_timeout');
306         total_fuel_out = simout.get('total_fuel_out');
307         total_fuel_gal = max(total_fuel_out)*handles.g/6.073;
308         if (max(sim_timeout) > max_time)
309             mpg(i) = 0;
310         else
311             mpg(i) = max_laps*lap_dist/total_fuel_gal; %miles per
                    gallon
312         end
313         p_done = i/length(gearvec)*100;
314         set(handles.percent_tag, 'String',[num2str(p_done),' %
                Complete']);
315     end
316     axes(handles.axes1)
317     plot(gearvec,mpg)
318     title('MPG vs Gear Reduction')
319     figure(5)
```

```
320     close
321     figure(5)
322     plot(gearvec,mpg)
323     xlabel('Gear Reduction')
324     ylabel('Fuel Efficiency (MPG)')
325     set(gca,'fontsize',13)
326
327 elseif length(handles.vmax) > 1 %vmax and vmin speed study
328     disp('here')
329     n = 40;
330     minvec = linspace(handles.vmin(1),handles.vmin(2),n);
331     maxvec = linspace(handles.vmax(1),handles.vmax(2),n);
332     [vminmesh,vmaxmesh] = meshgrid(minvec,maxvec);
333     for i = 1:n
334         for j = 1:n
335             handles.vmin = vminmesh(i,j);
336             handles.vmax = vmaxmesh(i,j);
337             if handles.vmin >= handles.vmax
338                 mpg(i,j) = NaN;
339             else
340                 try
341                     simout = sim('Supermileage_Simulation_Final','
                            SrcWorkspace','current');
342                 catch
343                     handles.gear_ratio = handles.gear_ratio +
                            0.04;
344                     simout = sim('Supermileage_Simulation_Final','
                            SrcWorkspace','current');
345                 end
346                 sim_timeout = simout.get('sim_timeout');
347                 total_fuel_out = simout.get('total_fuel_out');
348                 total_fuel_gal = max(total_fuel_out)*handles.g
                        /6.073;
349                 if (max(sim_timeout) > max_time)
350                     mpg(i,j) = NaN;
351                 else
352                     mpg(i,j) = max_laps*lap_dist/total_fuel_gal; %
                            miles per gallon
353                 end
354             end
355         end
356         p_done = i/n*100;
357         set(handles.percent_tag, 'String',[num2str(p_done),' %
                Complete']);
358     end
359     figure(1)
360     close
361     figure(1)
```

```
362         surf(vminmesh,vmaxmesh,mpg)
363         shading interp
364         %title('MPG vs Min and Max Speed')
365         xlabel('Minimum Speed (ft/s)')
366         ylabel('Maximum Speed (ft/s)')
367         zlabel('Fuel Efficiency (MPG)')
368         set(gca,'fontsize',13)
369
370     elseif length(handles.m_vehicle) > 1 %weight study
371         minm_vehicle = handles.m_vehicle(1);
372         maxm_vehicle = handles.m_vehicle(2);
373         m_vehiclevec = linspace(minm_vehicle,maxm_vehicle,100);
374         for i = 1:length(m_vehiclevec)
375             handles.m_vehicle = m_vehiclevec(i);
376             delta_m = handles.m_vehicle - handles.m_vehicleold;
377             %the following number was determined from coastdown
                    testing
378             handles.coastdown_poly(3) = handles.Frrold + delta_m*
                    handles.road_load_poly_Frr;
379             handles.coastdown_lbf = polyval(handles.coastdown_poly,
                    ...
380             handles.coastdown_v_fts);
381             simout = sim('Supermileage_Simulation_Final','SrcWorkspace
                    ','current');
382
383             sim_timeout = simout.get('sim_timeout');
384             total_fuel_out = simout.get('total_fuel_out');
385             total_fuel_gal = max(total_fuel_out)*handles.g/6.073;
386             if (max(sim_timeout) > max_time)
387                 mpg(i) = 0;
388             else
389                 mpg(i) = max_laps*lap_dist/total_fuel_gal; %miles per
                        gallon
390             end
391             p_done = i/length(m_vehiclevec)*100;
392             set(handles.percent_tag, 'String',[num2str(p_done),' %
                    Complete']);
393         end
394         axes(handles.axes1)
395         plot(m_vehiclevec*handles.g,mpg)
396         title('MPG vs Vehicle Weight(lb)')
397         figure(5)
398         close
399         figure(5)
400         plot(m_vehiclevec*handles.g,mpg)
401         %title('MPG vs Vehicle Weight(lb)')
402         xlabel('Vehicle Weight(lb)')
403         ylabel('Fuel Efficiency (MPG)')
```

```
404     set(gca,'fontsize',13)
405
406 else %if everything has only one input value, do a normal
        simulation
407     try
408         simout = sim('Supermileage_Simulation_Final','SrcWorkspace
                ','current');
409     catch
410         handles.gear_ratio = handles.gear_ratio + 0.04;
411         simout = sim('Supermileage_Simulation_Final','SrcWorkspace
                ','current');
412     end
413     rpm_out = simout.get('rpm_out');
414     sim_timeout = simout.get('sim_timeout');
415     speedfts_out = simout.get('speedfts_out');
416     total_fuel_out = simout.get('total_fuel_out');
417     total_fuel_gal = max(total_fuel_out)*handles.g/6.073;
418
419     if (max(sim_timeout) > max_time)
420         mpg = 0;
421         set(handles.mpg_tag, 'String', 'DNF');
422     else
423         mpg = max_laps*lap_dist/total_fuel_gal; %miles per gallon
424         set(handles.mpg_tag, 'String', num2str(mpg));
425     end
426
427     % Put variables into workspace
428     assignin('base', 'simtime', sim_timeout)
429     assignin('base', 'simspeed', speedfts_out)
430     assignin('base', 'simrpm', rpm_out)
431     assignin('base', 'simfuel', total_fuel_out*handles.g/6.073)
432
433     axes(handles.axes1)
434     plot(sim_timeout,speedfts_out)
435     title('Speed versus Time')
436
437     figure(1)
438     close
439     figure(1)
440     %subplot(2,2,1)
441     plot(sim_timeout,speedfts_out)
442     ylabel('Speed (ft/s)')
443     xlabel('Time (s)')
444     figure(2)
445         close
446     figure(2)
447     %subplot(2,2,2)
448     plot(sim_timeout,rpm_out)
```

```
449       ylabel('Engine Speed (RPM)')
450       xlabel('Time (s)')
451       figure(3)
452           close
453       figure(3)
454       %subplot(2,2,3)
455       plot(sim_timeout,total_fuel_out*handles.g/6.073)
456       ylabel('Fuel Used (gal)')
457       xlabel('Time (s)')
458   %     subplot(2,2,4)
459   %     plot(sim_timeout,speedfts_out)
460   %     title('Speed versus Time')
461
462       %now to calculate clutch lockup
463       cfun = @(rpmq) interp1(handles.engine_rpm,handles.
              engine_torque,rpmq) - ...
464            interp1(handles.engine_rpm,handles.engine_clutch,rpmq);
465       rpmmin = fzero(cfun,2000); %this is where clutch really locks
              up
466
467       %rpmmin = handles.vmin/handles.rwheel*handles.gear_ratio
              *60/(2*pi);
468       rpmmax = handles.vmax/handles.rwheel*handles.gear_ratio*60/(2*
              pi);
469       axes(handles.axes2)
470       plot(handles.engine_rpm,handles.engine_torque)
471       title('Engine Torque (ft*lbf)')
472       hold on
473       plot([rpmmin rpmmin],[0,max(handles.engine_torque)],'r--')
474       plot([rpmmax rpmmax],[0,max(handles.engine_torque)],'r--')
475       hold off
476       axes(handles.axes3)
477       plot(handles.engine_rpm,handles.engine_clutch)
478       title('Clutch Torque (ft*lbf)')
479       hold on
480       plot([rpmmin rpmmin],[0,max(handles.engine_clutch)],'r--')
481       plot([rpmmax rpmmax],[0,max(handles.engine_clutch)],'r--')
482       hold off
483       axes(handles.axes4)
484       plot(handles.engine_rpm,handles.engine_fueluse)
485       title('BSFC(lb/hp*hr)')
486       hold on
487       plot([rpmmin rpmmin],[0,max(handles.engine_fueluse)],'r--')
488       plot([rpmmax rpmmax],[0,max(handles.engine_fueluse)],'r--')
489       hold off
490   end
491   warning('on','all')
492   end
```

```
493 %plot(handles.engine_rpm,handles.engine_torque)
494 %guidata(hObject,handles)
495
496
497
498 function edit1_Callback(hObject, eventdata, handles)
499 % hObject    handle to edit1 (see GCBO)
500 % eventdata  reserved - to be defined in a future version of
      MATLAB
501 % handles    structure with handles and user data (see GUIDATA)
502
503 % Hints: get(hObject,'String') returns contents of edit1 as text
504 %        str2double(get(hObject,'String')) returns contents of
      edit1 as a double
505 handles.engine_rpm = str2double(strsplit(get(hObject,'String')));
506 guidata(hObject,handles)
507 end
508
509 % --- Executes during object creation, after setting all
      properties.
510 function edit1_CreateFcn(hObject, eventdata, handles)
511 % hObject    handle to edit1 (see GCBO)
512 % eventdata  reserved - to be defined in a future version of
      MATLAB
513 % handles    empty - handles not created until after all
      CreateFcns called
514
515 % Hint: edit controls usually have a white background on Windows.
516 %       See ISPC and COMPUTER.
517 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
      defaultUicontrolBackgroundColor'))
518     set(hObject,'BackgroundColor','white');
519 end
520 end
521
522
523
524 function edit2_Callback(hObject, eventdata, handles)
525 % hObject    handle to edit2 (see GCBO)
526 % eventdata  reserved - to be defined in a future version of
      MATLAB
527 % handles    structure with handles and user data (see GUIDATA)
528
529 % Hints: get(hObject,'String') returns contents of edit2 as text
530 %        str2double(get(hObject,'String')) returns contents of
      edit2 as a double
531 handles.engine_torque = str2double(strsplit(get(hObject,'String'))
      );
```

```
532 guidata(hObject,handles)
533 end
534
535 % --- Executes during object creation, after setting all
      properties.
536 function edit2_CreateFcn(hObject, eventdata, handles)
537 % hObject    handle to edit2 (see GCBO)
538 % eventdata  reserved - to be defined in a future version of
      MATLAB
539 % handles    empty - handles not created until after all
      CreateFcns called
540
541 % Hint: edit controls usually have a white background on Windows.
542 %       See ISPC and COMPUTER.
543 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
      defaultUicontrolBackgroundColor'))
544     set(hObject,'BackgroundColor','white');
545 end
546 end
547
548
549 % --- Executes on button press in pushbutton2.
550 function pushbutton2_Callback(hObject, eventdata, handles)
551 % hObject    handle to pushbutton2 (see GCBO)
552 % eventdata  reserved - to be defined in a future version of
      MATLAB
553 % handles    structure with handles and user data (see GUIDATA)
554 axes(handles.axes2)
555 plot(handles.engine_rpm,handles.engine_torque)
556 end
557
558
559 % --- Executes when entered data in editable cell(s) in uitable1.
560 function uitable1_CellEditCallback(hObject, eventdata, handles)
561 % hObject    handle to uitable1 (see GCBO)
562 % eventdata  structure with the following fields (see MATLAB.UI.
      CONTROL.TABLE)
563 %       Indices: row and column indices of the cell(s) edited
564 %       PreviousData: previous data for the cell(s) edited
565 %       EditData: string(s) entered by the user
566 %       NewData: EditData or its converted form set on the Data
      property. Empty if Data was not changed
567 %       Error: error string when failed to convert EditData to
      appropriate value for Data
568 % handles    structure with handles and user data (see GUIDATA)
569 index = eventdata.Indices;
570 if index(1) == 1
571     handles.engine_rpm(index(2)) = str2num(eventdata.EditData);
```

```
572 elseif index(1) == 2
573     handles.engine_torque(index(2)) = str2num(eventdata.EditData);
574 elseif index(1) == 3
575     handles.engine_clutch(index(2)) = str2num(eventdata.EditData);
576 elseif index(1) == 4
577     handles.engine_fueluse(index(2)) = str2num(eventdata.EditData)
        ;
578 end
579 axes(handles.axes2)
580 plot(handles.engine_rpm,handles.engine_torque)
581 title('Engine Torque (ft*lbf)')
582 axes(handles.axes3)
583 plot(handles.engine_rpm,handles.engine_clutch)
584 title('Clutch Torque (ft*lbf)')
585 axes(handles.axes4)
586 plot(handles.engine_rpm,handles.engine_fueluse)
587 title('BSFC(lb/hp*hr)')
588 guidata(hObject,handles)
589 end
590
591 % --- Executes during object creation, after setting all
    properties.
592 function uitable1_CreateFcn(hObject, eventdata, handles)
593 % hObject    handle to uitable1 (see GCBO)
594 % eventdata  reserved - to be defined in a future version of
    MATLAB
595 % handles    empty - handles not created until after all
    CreateFcns called
596 %handles.engine_torque
597 %hObject.Data(1,1) = mat2cell(handles.engine_torque(1),1)
598 end
599
600 % --- Executes when selected cell(s) is changed in uitable1.
601 function uitable1_CellSelectionCallback(hObject, eventdata,
    handles)
602 % hObject    handle to uitable1 (see GCBO)
603 % eventdata  structure with the following fields (see MATLAB.UI.
    CONTROL.TABLE)
604 %       Indices: row and column indices of the cell(s) currently
    selecteds
605 % handles    structure with handles and user data (see GUIDATA)
606 end
607
608 function max_speed_Callback(hObject, eventdata, handles)
609 % hObject    handle to max_speed (see GCBO)
610 % eventdata  reserved - to be defined in a future version of
    MATLAB
611 % handles    structure with handles and user data (see GUIDATA)
```

```
612
613 % Hints: get(hObject,'String') returns contents of max_speed as
    text
614 %        str2double(get(hObject,'String')) returns contents of
    max_speed as a double
615 temp = str2double(get(hObject,'String'));
616 if isnan(temp) == 1
617     handles.vmax = str2num(get(hObject,'String'));
618 else
619     handles.vmax = temp;
620 end
621 guidata(hObject,handles);
622 end
623
624
625 function min_speed_tag_Callback(hObject, eventdata, handles)
626 % hObject    handle to min_speed_tag (see GCBO)
627 % eventdata  reserved - to be defined in a future version of
    MATLAB
628 % handles    structure with handles and user data (see GUIDATA)
629
630 % Hints: get(hObject,'String') returns contents of min_speed_tag
    as text
631 %        str2double(get(hObject,'String')) returns contents of
    min_speed_tag as a double
632 temp = str2double(get(hObject,'String'));
633 if isnan(temp) == 1
634     handles.vmin = str2num(get(hObject,'String'));
635 else
636     handles.vmin = temp;
637 end
638 guidata(hObject,handles);
639 end
640
641 % --- Executes during object creation, after setting all
    properties.
642 function min_speed_tag_CreateFcn(hObject, eventdata, handles)
643 % hObject    handle to min_speed_tag (see GCBO)
644 % eventdata  reserved - to be defined in a future version of
    MATLAB
645 % handles    empty - handles not created until after all
    CreateFcns called
646
647 % Hint: edit controls usually have a white background on Windows.
648 %       See ISPC and COMPUTER.
649 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
650     set(hObject,'BackgroundColor','white');
```

```
651 end
652 end
653
654
655 function weight_tag_Callback(hObject, eventdata, handles)
656 % hObject    handle to weight_tag (see GCBO)
657 % eventdata  reserved - to be defined in a future version of
    MATLAB
658 % handles    structure with handles and user data (see GUIDATA)
659
660 % Hints: get(hObject,'String') returns contents of weight_tag as
    text
661 %        str2double(get(hObject,'String')) returns contents of
    weight_tag as a double
662 temp = str2double(get(hObject,'String'));
663 if isnan(temp) == 1 %there have been 2 weight values input for
    design study
664     handles.m_vehicle = str2num(get(hObject,'String'))/handles.g;
665 else %there is only one weight value input
666     handles.m_vehicle = temp/handles.g;
667     delta_m = handles.m_vehicle - handles.m_vehicleold;
668     %the following number was determined from coastdown testing
669     handles.coastdown_poly(3) = handles.Frrold + delta_m*handles.
        road_load_poly_Frr;
670     set(handles.road_load_p, 'String',num2str(handles.
        coastdown_poly));
671     handles.coastdown_lbf = polyval(handles.coastdown_poly, ...
672     handles.coastdown_v_fts);
673     axes(handles.axes5)
674     plot(handles.coastdown_v_fts,handles.coastdown_lbf)
675     title('Road Load (lbf) vs Vehicle Speed (ft/s)')
676 end
677 guidata(hObject,handles);
678 end
679
680 % --- Executes during object creation, after setting all
    properties.
681 function weight_tag_CreateFcn(hObject, eventdata, handles)
682 % hObject    handle to weight_tag (see GCBO)
683 % eventdata  reserved - to be defined in a future version of
    MATLAB
684 % handles    empty - handles not created until after all
    CreateFcns called
685
686 % Hint: edit controls usually have a white background on Windows.
687 %       See ISPC and COMPUTER.
688 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
```

```
689         set(hObject,'BackgroundColor','white');
690     end
691     end
692
693
694     function gear_tag_Callback(hObject, eventdata, handles)
695     % hObject    handle to gear_tag (see GCBO)
696     % eventdata  reserved - to be defined in a future version of
        MATLAB
697     % handles    structure with handles and user data (see GUIDATA)
698
699     % Hints: get(hObject,'String') returns contents of gear_tag as
        text
700     %        str2double(get(hObject,'String')) returns contents of
        gear_tag as a double
701     temp = str2double(get(hObject,'String'));
702     if isnan(temp) == 1
703         handles.gear_ratio = str2num(get(hObject,'String'));
704     else
705         handles.gear_ratio = temp;
706     end
707     guidata(hObject,handles);
708     end
709     % --- Executes during object creation, after setting all
        properties.
710     function gear_tag_CreateFcn(hObject, eventdata, handles)
711     % hObject    handle to gear_tag (see GCBO)
712     % eventdata  reserved - to be defined in a future version of
        MATLAB
713     % handles    empty - handles not created until after all
        CreateFcns called
714
715     % Hint: edit controls usually have a white background on Windows.
716     %       See ISPC and COMPUTER.
717     if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
        defaultUicontrolBackgroundColor'))
718         set(hObject,'BackgroundColor','white');
719     end
720     end
721
722
723
724     function radius_tag_Callback(hObject, eventdata, handles)
725     % hObject    handle to radius_tag (see GCBO)
726     % eventdata  reserved - to be defined in a future version of
        MATLAB
727     % handles    structure with handles and user data (see GUIDATA)
728
```

```
729 % Hints: get(hObject,'String') returns contents of radius_tag as
      text
730 %        str2double(get(hObject,'String')) returns contents of
      radius_tag as a double
731 handles.rwheel = str2double(get(hObject,'String'));
732 guidata(hObject,handles);
733 end
734
735 % --- Executes during object creation, after setting all
      properties.
736 function radius_tag_CreateFcn(hObject, eventdata, handles)
737 % hObject    handle to radius_tag (see GCBO)
738 % eventdata  reserved - to be defined in a future version of
      MATLAB
739 % handles    empty - handles not created until after all
      CreateFcns called
740
741 % Hint: edit controls usually have a white background on Windows.
742 %       See ISPC and COMPUTER.
743 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
      defaultUicontrolBackgroundColor'))
744     set(hObject,'BackgroundColor','white');
745 end
746 end
747
748 % --- Executes during object creation, after setting all
      properties.
749 function percent_tag_CreateFcn(hObject, eventdata, handles)
750 % hObject    handle to percent_tag (see GCBO)
751 % eventdata  reserved - to be defined in a future version of
      MATLAB
752 % handles    empty - handles not created until after all
      CreateFcns called
753 end
754
755 % --- Executes when entered data in editable cell(s) in uitable2.
756 function uitable2_CellEditCallback(hObject, eventdata, handles)
757 % hObject    handle to uitable2 (see GCBO)
758 % eventdata  structure with the following fields (see MATLAB.UI.
      CONTROL.TABLE)
759 %       Indices: row and column indices of the cell(s) edited
760 %       PreviousData: previous data for the cell(s) edited
761 %       EditData: string(s) entered by the user
762 %       NewData: EditData or its converted form set on the Data
      property. Empty if Data was not changed
763 %       Error: error string when failed to convert EditData to
      appropriate value for Data
764 % handles    structure with handles and user data (see GUIDATA)
```

```
765 index = eventdata.Indices;
766 if index(1) == 1
767     handles.coastdown_v_fts(index(2)) = str2num(eventdata.EditData
        );
768 elseif index(1) == 2
769     handles.coastdown_lbf(index(2)) = str2num(eventdata.EditData);
770 end
771 axes(handles.axes5)
772 plot(handles.coastdown_v_fts,handles.coastdown_lbf)
773 title('Road Load vs Vehicle Speed (ft/s)')
774 guidata(hObject,handles)
775 end
776
777
778 function road_load_p_Callback(hObject, eventdata, handles)
779 % hObject    handle to road_load_p (see GCBO)
780 % eventdata  reserved - to be defined in a future version of
    MATLAB
781 % handles    structure with handles and user data (see GUIDATA)
782
783 % Hints: get(hObject,'String') returns contents of road_load_p as
    text
784 %        str2double(get(hObject,'String')) returns contents of
    road_load_p as a double
785 handles.coastdown_poly = str2num(get(hObject,'String'));
786 handles.coastdown_lbf = polyval(handles.coastdown_poly, ...
787     handles.coastdown_v_fts);
788 axes(handles.axes5)
789 plot(handles.coastdown_v_fts,handles.coastdown_lbf)
790 title('Road Load (lbf) vs Vehicle Speed (ft/s)')
791 guidata(hObject,handles)
792 end
793 % --- Executes during object creation, after setting all
    properties.
794 function road_load_p_CreateFcn(hObject, eventdata, handles)
795 % hObject    handle to road_load_p (see GCBO)
796 % eventdata  reserved - to be defined in a future version of
    MATLAB
797 % handles    empty - handles not created until after all
    CreateFcns called
798
799 % Hint: edit controls usually have a white background on Windows.
800 %       See ISPC and COMPUTER.
801 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
802     set(hObject,'BackgroundColor','white');
803 end
804 end
```

```
805
806
807 function drivetrain_eff_tag_Callback(hObject, eventdata, handles)
808 % hObject    handle to drivetrain_eff_tag (see GCBO)
809 % eventdata  reserved - to be defined in a future version of
    MATLAB
810 % handles    structure with handles and user data (see GUIDATA)
811
812 % Hints: get(hObject,'String') returns contents of
    drivetrain_eff_tag as text
813 %        str2double(get(hObject,'String')) returns contents of
    drivetrain_eff_tag as a double
814 handles.drivetrain_eff = str2num(get(hObject,'String'))/100;
815 guidata(hObject,handles)
816 end
817 % --- Executes during object creation, after setting all
    properties.
818 function drivetrain_eff_tag_CreateFcn(hObject, eventdata, handles)
819 % hObject    handle to drivetrain_eff_tag (see GCBO)
820 % eventdata  reserved - to be defined in a future version of
    MATLAB
821 % handles    empty - handles not created until after all
    CreateFcns called
822
823 % Hint: edit controls usually have a white background on Windows.
824 %       See ISPC and COMPUTER.
825 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
826     set(hObject,'BackgroundColor','white');
827 end
828 end
829
830 % --- Executes when entered data in editable cell(s) in
    lap_load_tag.
831 function lap_load_tag_CellEditCallback(hObject, eventdata, handles
    )
832 % hObject    handle to lap_load_tag (see GCBO)
833 % eventdata  structure with the following fields (see MATLAB.UI.
    CONTROL.TABLE)
834 %       Indices: row and column indices of the cell(s) edited
835 %       PreviousData: previous data for the cell(s) edited
836 %       EditData: string(s) entered by the user
837 %       NewData: EditData or its converted form set on the Data
    property. Empty if Data was not changed
838 %       Error: error string when failed to convert EditData to
    appropriate value for Data
839 % handles    structure with handles and user data (see GUIDATA)
840 index = eventdata.Indices;
```

```
841 if index(1) == 1
842     handles.coastdown_laps(index(2)) = str2num(eventdata.EditData)
        ;
843 elseif index(1) == 2
844     handles.slope_force_lbf(index(2)) = str2num(eventdata.EditData
        );
845 end
846 axes(handles.axes6)
847 plot(handles.coastdown_laps,handles.slope_force_lbf)
848 title('Road Load due to Track')
849 guidata(hObject,handles)
850 end
851
852
853 % This confusing callback is for the weight/road load relationship
854 function road_load_poly_edit_Callback(hObject, eventdata, handles)
855 % hObject    handle to road_load_poly_edit (see GCBO)
856 % eventdata  reserved - to be defined in a future version of
    MATLAB
857 % handles    structure with handles and user data (see GUIDATA)
858
859 % Hints: get(hObject,'String') returns contents of
    road_load_poly_edit as text
860 %        str2double(get(hObject,'String')) returns contents of
    road_load_poly_edit as a double
861 temp = str2double(get(hObject,'String'));
862 handles.road_load_poly_Frr = temp;
863 guidata(hObject,handles);
864 end
865
866 % --- Executes during object creation, after setting all
    properties.
867 % This is the callback for the polynomial relating to weight
868 function road_load_poly_edit_CreateFcn(hObject, eventdata, handles
    )
869 % hObject    handle to road_load_poly_edit (see GCBO)
870 % eventdata  reserved - to be defined in a future version of
    MATLAB
871 % handles    empty - handles not created until after all
    CreateFcns called
872
873 % Hint: edit controls usually have a white background on Windows.
874 %       See ISPC and COMPUTER.
875 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
876     set(hObject,'BackgroundColor','white');
877 end
878 end
```

sim_compare.m

```
% Chad Bickel
% 3/8/17
% Compares simulation output and actual outputs

close all
clc

load Apr3Test4data

photopath1 = 'C:\Users\Chad\Google Drive\NEW Documents\Thesis';
photopath2 = '\Photos\MatlabOutput\';
photopath = strcat(photopath1,photopath2);

%speeds are 12.5 - 38.35
%gear ratio is 12

total_fuelgal = total_fuel/3785.41; %converting from mL to gallons

total_mpg = total_dist/total_fuelgal; %calculating mpg

sim_ft_unlimit = cumtrapz(simtime,simspeed);


%below, assigning variables for time,pulsewidth,rpm and speed

try
    time = (Utcms001-Utcms001(1))*1e-3; %time in seconds
    pulsewidthvec = PW1ms03050;
    rpmvec = RPMrevmin01000050;
    %speedvec = SpeedMPH00150010;
    % Do this if you're sampling GPS at 10 Hz
    ix=cellfun(@isempty,SpeedMPH00150010);
    SpeedMPH00150010(ix) = {'0'};
    keepspeed = 0;
     SpeedMPH00150010(1) = {'12'}; %fix this later
%     SpeedMPH00150010(2) = {12};
%     SpeedMPH00150010(3) = {12};
    for i = 1:length(SpeedMPH00150010)
        tempspeed = str2double(SpeedMPH00150010(i));
        if tempspeed > 0
            keepspeed = tempspeed;
        end
        SpeedMPH00150010(i) = {keepspeed};
    end
    speedvec = cell2mat(SpeedMPH00150010');
catch
```

```
47      time = (Utcms1-Utcms1(1))*1e-3; %time in seconds
48      pulsewidthvec = PW1ms1;
49      rpmvec = RPMrevmin1;
50      speedvec = SpeedMPH1;
51
52  end
53
54  cyclespm = rpmvec/2; %4 stroke engine, so 1 cyc per 2 revs
55  cyclesps = cyclespm/60; %engine cycles per second
56
57  %now we have cycles/s and fuel ms/cycle
58  instant_fuel_ms = pulsewidthvec.*cyclesps; %(fuel ms)/s
59
60  %now we integrate (fuel ms)/s to get total fuel ms
61  total_fuel_ms = trapz(time,instant_fuel_ms); %total ms of fuel
        burned
62
63  %now we need to convert ms of injector open to ml of fuel
64  mfdotmL = total_fuel/(total_fuel_ms);%mass flow rate of injector,
65  %in mL per fuel ms
66  %this is 0.6209 mL/s for test 2
67  %this is 0.6385 mL/s for test 4
68
69  %assume mass flow rate of injector is constant
70  instant_fuel_mL = instant_fuel_ms*mfdotmL; %instant fuel usage, in
        mL/s
71
72  n = 10;
73  instant_fuel_mL_filt = filter(ones(1,n)/n,[1],instant_fuel_mL);
74
75  total_fuelvec = cumtrapz(time,instant_fuel_mL); %integrate instant
        fuel
76
77  lap_dist = 1;%0.6; %1 lap =~ 1 mile
78  lap_unlimit = cumtrapz(time,speedvec/(3600*lap_dist));
79  lap_vec = mod(lap_unlimit,1); %this lap vec is between 0 and 1
80
81  driver_input = zeros(length(rpmvec),1); %pre-allocating
82
83  delta_t = 1/50; %s
84  speedvec_fts = speedvec*1.46667; %converting to ft/s
85
86  n = 5;
87  RPMfiltered = filter(ones(1,n)/n,[1],rpmvec); %filtering RPM data
88  RPMslopetemp = diff(RPMfiltered); %calculating discrete derivative
89  RPMslope(1) = 0;
90  RPMslope(2:(length(RPMslopetemp)+1)) = RPMslopetemp;
91  %this term shifts the left side of the input to the left
```

```
92  samples_delay1 = 0;%300; %these are used to shift the driver input
93  samples_delay2 = 150; %this should be about 1/4*n
94  burning = 0;
95
96  comparetime = -1;
97
98  for i = 1:length(rpmvec)
99      if i < n + 1 || i > length(rpmvec) + 101
100         driver_input(i) = 0;
101     else
102         if rpmvec(i) > 0 && max(rpmvec((i+30):(i+200))) > 1000 &&
                ...
103                 burning == 0 && RPMslope(i) > 0
104             burning = 1;
105             if comparetime < 0
106                 comparetime = time(i);
107             end
108         elseif burning == 1 && rpmvec(i) > 2000 && RPMslope(i) <
                -10  ...
109                     && rpmvec(i+200) < 5
110             burning = 0;
111
112         end
113
114         if burning == 1
115             driver_input(i) = 1;
116         else
117             driver_input((i-n):i) = 0;
118             %the delay of ten samples is so the engine doesn't
                    stop before
119             %the driver "input" is 0
120         end
121     end
122 end
123
124 time = time - comparetime;
125
126 n = 100; %now filtering speed from GPS data
127 speedfiltered = filter(ones(1,n)/n,[1],speedvec_fts);
128 n = 30;%100;
129 accltemp = filter(ones(1,n)/n,[1],diff(speedfiltered)/delta_t); %
    in ft/s^2
130 accl(1) = 0;
131 accl(2:(length(accltemp)+1)) = accltemp;
132
133 figure(1)
134 %plot(time,speedvec_fts)
135 %hold on
```

```
plot(time,speedfiltered)
hold on
plot(simtime,simspeed)
%hold on
%plot(time,driver_input*20)
% hold on
% plot(time,speedvectest)
%title('Speed and driver input')
xlabel('Time (s)')
ylabel('Speed (ft/s)')
axis([-10 1000 0 40])
set(gca,'fontsize',13)
%print(strcat(photopath,'optspeedcompare'),'-djpeg','-r300')
axis([100 180 10 40])
print(strcat(photopath,'optspeedcompare'),'-djpeg','-r300')

figure(2)
plot(time,rpmvec)
hold on
%plot(time,RPMfiltered)
plot(simtime,simrpm)
%hold on
%plot(time,driver_input*4000)
xlabel('Time (s)')
ylabel('Engine Speed (rpm)')
axis([-10 1000 0 6000])
set(gca,'fontsize',13)
%print(strcat(photopath,'enginespeedcompare'),'-djpeg','-r300')
axis([95 115 0 6000])
print(strcat(photopath,'optenginespeedcompare'),'-djpeg','-r300')
%figure(7)
%plot(time,RPMslope)

figure(3)
plot(lap_unlimit*lap_dist,total_fuelvec)
hold on
plot(sim_ft_unlimit/5280,simfuel*3785.41)
ylabel('Total Fuel Used (mL)')
xlabel('Distance (mi)')
axis([-0.1 6.5 0 25])
set(gca,'fontsize',13)
print(strcat(photopath,'optfuelcompare'),'-djpeg','-r300')
```