CONTROL SYSTEM DESIGN AND OPTIMIZATION FOR
THE FUEL CELL POWERED BUCKEYE BULLET 2
LAND SPEED VEHICLE

A Thesis

Presented in Partial Fulfillment of the Requirements for
the Degree Master of Science in the
Graduate School of The Ohio State University

By

Kevin R. Ponziani, B.S., F.E.

* * * * *

The Ohio State University

2008

Master’s Examination Committee:                              Approved by
Giorgio Rizzoni, Adviser
Steve Yurkovich

________________________________________
Adviser
Electrical & Computer
Engineering Graduate
Program
ABSTRACT

The Buckeye Bullet 2 is a hydrogen fuel cell electric land speed vehicle, designed to run on the Bonneville Salt Flats in Utah. To operate the vehicle, a number of control systems are utilized, and are controlled through a high-level supervisory controller. By integrating a complex network of sensors, and inputs from the driver, the controller is able to safely and effectively maximize the power at the wheel to increase the vehicle’s top speed. A power management scheme using rule-based control is used to manage the power between a motor controller and fuel cell controller. Additionally, control is developed to manage the thermal energy generated by the fuel cell reaction, and maintain a constant operating temperature. Various safety systems are integrated to the central system as well. Finally, methods are presented to process data following a speed trial, and improve diagnosis, improvement, and optimization of the vehicle systems.
This thesis is dedicated to my parents, whose lifelong encouragement and confidence in me inspires me every day.
VITA

February 27, 1985 ......................... Born - Dayton, OH, USA

March 2007 ............................... B.S., Elec. & Comp. Engineering

2007–2008 .................................. NSF Graduate Fellow,
The Ohio State University

2003–2008 .................................. Member, Buckeye Bullet Team,
Center For Automotive Research

2003,2004 ................................. Engineering Aid,
Wright-Patterson Air Force Base

2005,2006 .................................. Engineering Intern
Northrop Grumman, Cincinnati, OH

PUBLICATIONS

Research Publications

G. Rizzoni, B. Sinsheimer, K. Ponziani, J. Gorse, N. McQuin, A. Davis, A. Miotti
Design, analysis and performance of an electric land-speed record streamliner Vehicle

FIELDS OF STUDY

Major Field: Electrical & Computer Engineering
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Abstract</th>
<th>ii</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dedication</td>
<td>iii</td>
</tr>
<tr>
<td>Vita</td>
<td>iv</td>
</tr>
<tr>
<td>List of Figures</td>
<td>ix</td>
</tr>
<tr>
<td>List of Tables</td>
<td>xiv</td>
</tr>
<tr>
<td>Chapters:</td>
<td></td>
</tr>
<tr>
<td>1. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Motivation</td>
<td>2</td>
</tr>
<tr>
<td>1.2 Buckeye Bullet 2 Project Objectives</td>
<td>2</td>
</tr>
<tr>
<td>1.3 Thesis Objectives</td>
<td>3</td>
</tr>
<tr>
<td>1.4 Thesis Summary</td>
<td>3</td>
</tr>
<tr>
<td>2. BACKGROUND AND LITERATURE REVIEW</td>
<td>5</td>
</tr>
<tr>
<td>2.1 Electric Racing at Ohio State University</td>
<td>5</td>
</tr>
<tr>
<td>2.1.1 Smokin’ Buckeye</td>
<td>5</td>
</tr>
<tr>
<td>2.1.2 Land Speed Racing</td>
<td>5</td>
</tr>
<tr>
<td>2.1.3 The Buckeye Bullet</td>
<td>9</td>
</tr>
<tr>
<td>2.1.4 The Buckeye Bullet 2</td>
<td>12</td>
</tr>
<tr>
<td>2.1.5 Buckeye Bullet 2 Layout</td>
<td>13</td>
</tr>
</tbody>
</table>
### 3. CONTROL SYSTEM AND SENSOR ARCHITECTURE

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Sensors in the Buckeye Bullet 2</td>
<td>15</td>
</tr>
<tr>
<td>3.1.1</td>
<td>GPS Speed Measurement</td>
<td>16</td>
</tr>
<tr>
<td>3.1.2</td>
<td>Pressure Measurement</td>
<td>27</td>
</tr>
<tr>
<td>3.1.3</td>
<td>Voltage Measurement</td>
<td>28</td>
</tr>
<tr>
<td>3.1.4</td>
<td>Current Measurement</td>
<td>28</td>
</tr>
<tr>
<td>3.1.5</td>
<td>Position/Speed/Acceleration Measurement</td>
<td>30</td>
</tr>
<tr>
<td>3.1.6</td>
<td>Temperature Measurement</td>
<td>30</td>
</tr>
<tr>
<td>3.1.7</td>
<td>Flow Rate Measurement</td>
<td>31</td>
</tr>
<tr>
<td>3.1.8</td>
<td>Driver Control Measurement</td>
<td>34</td>
</tr>
<tr>
<td>3.2</td>
<td>Controls in the Buckeye Bullet 2</td>
<td>35</td>
</tr>
<tr>
<td>3.2.1</td>
<td>Motor Controller and DC-AC Inverter</td>
<td>35</td>
</tr>
<tr>
<td>3.2.2</td>
<td>Ballard Fuel Cell Controller</td>
<td>38</td>
</tr>
<tr>
<td>3.2.3</td>
<td>Fuel Cell Thermal Management (Cooling) Controller</td>
<td>39</td>
</tr>
<tr>
<td>3.2.4</td>
<td>Supervisory Controller</td>
<td>40</td>
</tr>
<tr>
<td>3.3</td>
<td>Communication and Data Architecture</td>
<td>40</td>
</tr>
<tr>
<td>3.3.1</td>
<td>Data Structure</td>
<td>41</td>
</tr>
<tr>
<td>3.3.2</td>
<td>TCP/IP Ethernet Communication</td>
<td>42</td>
</tr>
<tr>
<td>3.3.3</td>
<td>CAN Communication</td>
<td>43</td>
</tr>
<tr>
<td>3.3.4</td>
<td>Serial Communication</td>
<td>44</td>
</tr>
</tbody>
</table>

### 4. CONTROL SYSTEM DEVELOPMENT AND DESIGN

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Cooling System Development</td>
<td>46</td>
</tr>
<tr>
<td>4.1.1</td>
<td>Overall System</td>
<td>47</td>
</tr>
<tr>
<td>4.1.2</td>
<td>Benefits and Challenges</td>
<td>54</td>
</tr>
<tr>
<td>4.1.3</td>
<td>Feedback Input</td>
<td>56</td>
</tr>
<tr>
<td>4.1.4</td>
<td>Pump Actuators</td>
<td>56</td>
</tr>
<tr>
<td>4.1.5</td>
<td>Potential Control Schemes</td>
<td>59</td>
</tr>
<tr>
<td>4.1.6</td>
<td>Cooling Software from 2007</td>
<td>67</td>
</tr>
<tr>
<td>4.1.7</td>
<td>Cooling Software from 2008</td>
<td>67</td>
</tr>
<tr>
<td>4.2</td>
<td>Power Management Development</td>
<td>67</td>
</tr>
<tr>
<td>4.2.1</td>
<td>Torque Control</td>
<td>70</td>
</tr>
<tr>
<td>4.2.2</td>
<td>Maximization of Power</td>
<td>70</td>
</tr>
<tr>
<td>4.2.3</td>
<td>2007 Control Strategy</td>
<td>70</td>
</tr>
<tr>
<td>4.2.4</td>
<td>2008 Control Strategy</td>
<td>73</td>
</tr>
<tr>
<td>4.3</td>
<td>Safety Systems</td>
<td>76</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Shift Lockout</td>
<td>77</td>
</tr>
<tr>
<td>4.3.2</td>
<td>High Voltage</td>
<td>77</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 The Smokin’ Buckeye</td>
<td>6</td>
</tr>
<tr>
<td>2.2 Map of the United States of America showing the Bonneville Salt Flats circled</td>
<td>7</td>
</tr>
<tr>
<td>2.3 Bonneville Salt Flats Entrance Sign</td>
<td>7</td>
</tr>
<tr>
<td>2.4 SCTA I-beams used to drag and flatten the course at the Bonneville Salt Flats</td>
<td>9</td>
</tr>
<tr>
<td>2.5 Land Speed Record Diagram</td>
<td>10</td>
</tr>
<tr>
<td>2.6 Buckeye Bullet Timing Slip from October 2004</td>
<td>10</td>
</tr>
<tr>
<td>2.7 The Buckeye Bullet</td>
<td>11</td>
</tr>
<tr>
<td>2.8 The Buckeye Bullet 2</td>
<td>11</td>
</tr>
<tr>
<td>2.9 The Buckeye Bullet 2 Layout</td>
<td>14</td>
</tr>
<tr>
<td>3.1 Trimble GPS Antenna: Test Setup</td>
<td>20</td>
</tr>
<tr>
<td>3.2 Trimble GPS Antenna: $S_{21}$ Frequency Response</td>
<td>21</td>
</tr>
<tr>
<td>3.3 GPS Satellite Planning: August 18, 2008 (Speed Week 2008, Day 1)[6]</td>
<td>23</td>
</tr>
<tr>
<td>3.4 GPS Network Layout</td>
<td>25</td>
</tr>
<tr>
<td>3.5 Trimble AgGPS 332 Receiver. This receiver is capable of receiving GPS satellite information, as well as DGPS correction information.</td>
<td>26</td>
</tr>
<tr>
<td>Section</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>3.6</td>
<td>Track position overlay at Speedweek 2008 (See Table 6.1)</td>
</tr>
<tr>
<td>3.7</td>
<td>Hydrogen Tank Regulator Temperature at Speedweek 2008 (See Table 6.1)</td>
</tr>
<tr>
<td>3.8</td>
<td>Safranics VG5 Diagram [30, 6]</td>
</tr>
<tr>
<td>3.9</td>
<td>Saminco M1-500 Electric Traction Inverter</td>
</tr>
<tr>
<td>3.10</td>
<td>Motor Torque Control [30, 153]</td>
</tr>
<tr>
<td>3.11</td>
<td>Ballard Fuel Cell Controller, containing a Visteon Automotive Controller</td>
</tr>
<tr>
<td>3.12</td>
<td>Alltrax DC Motor Controller</td>
</tr>
<tr>
<td>3.13</td>
<td>Mototron Motohawk Controller</td>
</tr>
<tr>
<td>3.14</td>
<td>Buckeye Bullet 2 Data Flow</td>
</tr>
<tr>
<td>4.1</td>
<td>Buckeye Bullet 2 Mechanical Fuel Cell Cooling Diagram</td>
</tr>
<tr>
<td>4.2</td>
<td>Buckeye Bullet 2 Fuel Cell Cooling Model</td>
</tr>
<tr>
<td>4.3</td>
<td>Water Density Lookup Table ($f_{\text{density}}(T)$)</td>
</tr>
<tr>
<td>4.4</td>
<td>Modine QED Lookup ($f_{\text{QED}}(\dot{v}<em>{\text{FC,GPM}}, \dot{v}</em>{\text{IB,GPM}})$). The contour lines represent lines of constant energy, in Joules, that can be dissipated per unit of entering temperature differential in °C.</td>
</tr>
<tr>
<td>4.5</td>
<td>Axiomatic TC-20 Thermocouple Monitoring Module</td>
</tr>
<tr>
<td>4.6</td>
<td>Pump Flow Rate versus Command Voltage</td>
</tr>
<tr>
<td>4.7</td>
<td>Fuel Cell Cooling Pump</td>
</tr>
<tr>
<td>4.8</td>
<td>Ice Bath Cooling Pump</td>
</tr>
</tbody>
</table>
4.9 Cooling system without Ice Bath Control, and 60 GPM on the fuel cells. In this set of plots, it can be seen that the run can only last for the amount of time that the fuel cells are heating up, absorbing the thermal energy that they generate into their mass and the mass of the fuel cell cooling water. The ice bath plots can be ignored in this case because there is no flow in that side, and the inlet is assumed to be the pre-heated starting temperature of 40°C.

4.10 Preset Constant Cooling Control, with 30 GPM on the Ice Bath, and 60 GPM on the Fuel Cells. Here, the fuel cells heat up very slowly, due to the ice bath being run too quickly at lower fuel cell outlet temperatures. The fuel cells do not fully heat up until after the ice has run out, and the run ends shortly thereafter due to the fuel cell outlet temperature being too high. The lower right plot shows when the ice has run out by the ice bath outlet temperature starting to increase, at approximately 85 seconds here.

4.11 Feedforward Cooling Control. Using the values from Table 4.3, the cooling system begins to show the desired behavior for the cooling system. The fuel cells heat up quicker than before, and approach the desired outlet temperature, with some steady-state error. After the ice runs out, the ice bath pump increases the flow rate to maintain the temperature, until it cannot keep up, and the fuel cells must be shut down.

4.12 PID Cooling Control, with $K_p = -3$ and $K_i = -1$. Here, the fuel cell outlet temperature eventually approaches the desired temperature, before the ice runs out and the flow rate increases to maintain the temperature. Having to tune the PID without any other elements to account for the nonlinearities of the system means that the best that could be done in tuning is with the oscillations shown in the top-left plot.
4.13 PID Cooling Control, with $K_p = -3$ and $K_i = -1$, with a feedforward component. Combining the strategies shown in Figures 4.11 and 4.12 gives the result shown here. In this case, the fuel cells approach the desired temperature more slowly, but in practical implementation this is required because some flow is necessary on the ice bath lines to keep the cold water at the heat exchanger, ready to transfer heat when necessary. In this case, the fuel cells maintain the desired temperature more closely, and make the run last as long as possible before the fuel cells have to be shut down.

4.14 PID Cooling Control with a feedforward component in Simulink

4.15 2008 Cooling Control Code

4.16 Power Distribution Versus Vehicle Speed [21]

4.17 Sankey Diagram [21]

4.18 2008 Current Request Scheme

4.19 2008 Inverter Control Scheme

4.20 Ideal Current Request to Fuel Cell Controller

4.21 2008 Shifting Control Scheme

4.22 High Speed Parachute Released from the Buckeye Bullet 2 at Speed Week 2008

4.23 Fuel Cell Overtemperature with accelerator cutting

5.1 Driver display showing a no warning case on the left side. On the right is an “Out of Hydrogen” warning condition.

5.2 Steering Wheel for the Buckeye Bullet 2

5.3 Main Data Analysis GUI

5.4 Report Cover Sheet

5.5 Sample Data Analyzed in Google Earth
6.1 Buckeye Bullet Timing Slip from Run 4 in August 2008 (See Table 6.1) 94
6.2 Speed Versus Distance Run Summary (See Table 6.1) .......................... 94
6.3 Fuel Cell Cooling for Run 4 at the Salt Flats (See Table 6.1) ............. 95
6.4 Ice Bath Cooling State Diagram for all runs at the Salt Flats (See Table 6.1) ........................................ 96
6.5 Power versus RPM for Run 1 at the Salt Flats (See Table 6.1) ........... 97
6.6 Power versus RPM for Run 3 at the Salt Flats (See Table 6.1) ........... 98
6.7 Power versus RPM for Run 4 at the Salt Flats (See Table 6.1) .......... 99
6.8 Power versus RPM for Run 5 at the Salt Flats (See Table 6.1) ........... 100
6.9 Power versus RPM for Run 6 at the Salt Flats (See Table 6.1) ........... 101
6.10 Power versus RPM for Run 7 at the Salt Flats (See Table 6.1) ........... 102
6.11 Current versus RPM for Runs 4 and 7 at the Salt Flats (See Table 6.1) compared to the ideal current .................................................. 103
### LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 Trimble GPS Antenna: Frequency Response</td>
<td>20</td>
</tr>
<tr>
<td>3.2 GPS Position Accuracy Error Sources [2]</td>
<td>24</td>
</tr>
<tr>
<td>3.3 Pressure Measurements</td>
<td>27</td>
</tr>
<tr>
<td>3.4 Voltage Measurements</td>
<td>29</td>
</tr>
<tr>
<td>3.5 Current Measurements</td>
<td>29</td>
</tr>
<tr>
<td>3.6 Position Measurements</td>
<td>31</td>
</tr>
<tr>
<td>3.7 Temperature Measurements</td>
<td>33</td>
</tr>
<tr>
<td>3.8 Flow Rate Measurements</td>
<td>33</td>
</tr>
<tr>
<td>3.9 Driver Inputs</td>
<td>34</td>
</tr>
<tr>
<td>3.10 Ethernet Connected Devices</td>
<td>43</td>
</tr>
<tr>
<td>3.11 CAN Connected Devices</td>
<td>44</td>
</tr>
<tr>
<td>4.1 Modine QED Lookup Table $(f_{\text{QED}}(\dot{v}<em>{\text{FC,GPM}}, \dot{v}</em>{\text{IB,GPM}}))$</td>
<td>52</td>
</tr>
<tr>
<td>4.2 Thermal Transport Delay Parameters</td>
<td>53</td>
</tr>
<tr>
<td>4.3 Feedforward Cooling Parameters</td>
<td>62</td>
</tr>
<tr>
<td>6.1 Bonneville Speed Week 2008 Run Summary</td>
<td>93</td>
</tr>
</tbody>
</table>
CHAPTER 1

INTRODUCTION

The world runs on energy. Whether it is energy from the sun, oil, gas, hydro-electricity, or other forms, we require the consumption of energy to maintain and advance our standard of living. We use energy to communicate, to travel, transport goods and provide services, and to provide health care and improve life expectancy. It is extremely important that we examine the origins of our energy, how we extract it, and examine how we use it, and how we will use it in the future. For decades, the transportation industry has run primarily on oil-based energy sources. It has remained relatively inexpensive with a high level of energy density, and is able to be widely and easily distributed. But with world population growth, the growth of world economies, the scarcity of oil as a commodity is steadily increasing. Scarcity has caused rising prices and it is therefore more important than ever to seek and develop alternative sources of energy that can replace oil as our energy source for transportation in the future.

Converting to new forms of energy in the transportation industry is not an easy task. There are significant infrastructure, manufacturing, and cost issues that must be overcome to even make it economically viable. But to truly explore alternatives,
test vehicles that utilize new forms of energy and energy conversion must be built, and tested to their limits to see what they are capable of.

1.1 Motivation

The motivation for this research project comes from the desire to push components to their limits. One of the best places to push a vehicle to its limits is the Bonneville Salt Flats in Utah. The limits of a vehicle can be tested by testing at top speed, specifically at sanctioned racing events at the Bonneville Salt Flats. This thesis explores the practical limits of a hydrogen fuel cell vehicle that is built for maximum speed capability. More specifically, this thesis explores control systems that support the objective of a hydrogen fuel cell vehicle. To really test a hydrogen fuel cell vehicle’s extreme capabilities, it is pushed to its limits on the Salt Flats.

1.2 Buckeye Bullet 2 Project Objectives

The objective of the Buckeye Bullet program is ultimately about breaking speed records. So, when the project began, the objective of the Buckeye Bullet 2 was to break the existing electric land speed records. The current records for electric land speed vehicles are held by the original Buckeye Bullet, also known as the Buckeye Bullet 1 (BB1). The reason for designing and fabricating a new vehicle, rather than modifying the original were largely for reasons of safety. The Buckeye Bullet had been built for a maximum speed of 300 mi/h, and it had already traveled over 321 mi/h, so the decision was made to design a new vehicle that would increase the capability of exceeding the speeds achieved by the Buckeye Bullet 1.
The Buckeye Bullet 2 was not only about increasing the electric vehicle speed records, but also improving upon land speed vehicle technology in general. Safety was of utmost concern, because traveling at record speeds on the Bonneville Salt Flats, it is vital to maintaining the safety of not only the driver but of the team members and observers as well.

1.3 Thesis Objectives

The objective of this thesis is to develop and optimize the control system architecture and strategy for the Buckeye Bullet 2. This includes the method for connecting the network of sensors available on the vehicle. The strategy for reaching and maintaining the fuel cell operating temperature, and maximizing the power at the drive wheels is examined as well. Finally, the methodology and tools for effectively interpreting the data from the vehicle following a run are developed and presented.

1.4 Thesis Summary

This thesis begins with a background on electric racing at The Ohio State University. This includes the history of land speed racing, and the process by which land speed records are attained on the Bonneville Salt Flats in Utah. Chapter 3 discusses and sets forth the control system and sensor architecture for the Buckeye Bullet 2. The various sensors that are used to monitor and control the vehicle are explored, and the control units within the vehicle are explained. The method by which sensor information is shared and logged between controllers is investigated as well.

Chapter 4 covers the development and design of the control system, especially the cooling system and the power management. The on-board safety systems that are
control-related are also detailed. Chapter 5 speaks of the information processing and analysis by which the vehicle can be diagnosed, improved, and optimized. Several methods for analyzing data, and integrating into a data analysis toolkit are explored.

Chapter 6 reviews the results from Speed Week 2008, and compares the results of the speed trials on the Salt Flats to the anticipated results from Chapter 4. Finally, Chapter 7 concludes the thesis, and provides recommendations for potential future work in the area of control development for alternative energy vehicles, particularly in land speed vehicle applications.
CHAPTER 2

BACKGROUND AND LITERATURE REVIEW

2.1 Electric Racing at Ohio State University

2.1.1 Smokin’ Buckeye

The Ohio State University’s involvement in electric racing began with the Formula Lightning series in 1993, where the team competed with their vehicle, the Smokin’ Buckeye, throughout the 1990s. The Formula Lightning series was a collegiate open-wheel, full-size formula-style race that traveled to major racetracks around the country. Thirty-one lead acid batteries powered an AC induction traction system capable of race track speeds of over 120 mi/h. Racing included pit stops, and by the end of the series the team was capable of executing a full pit stop, including a battery change, in under 17 seconds. Ohio State University dominated the competition by winning more than 50% of the races entered and winning the ABB national championship in each of the three years it was awarded (1995-1997) [7]-[10]. The series ended in 2000. Figure 2.1 shows the Smokin’ Buckeye.

2.1.2 Land Speed Racing

As the Formula Lightning series was phased out, the team found themselves with a great deal of electric racing experience, but there was no competitive venue available
Figure 2.1: The Smokin’ Buckeye

to put that experience to use. After some brainstorming and discussions with existing program sponsors, the team decided to take electric racing to a whole new level and go for all out speed. The goal was to break the world record for top speed in an electric vehicle, which at the time was 248 mi/h. This record-breaking attempt would take place at the Bonneville Salt Flats, in Utah.

**Bonneville Salt Flats**

Situated north-east of the intersection of the Utah-Nevada border and Interstate-80, the Bonneville Salt Flats is one of the best locations for land speed racing in the world. The area, known as Ancient Lake Bonneville, was once a salt lake the size of Lake Michigan, and covered one-third of present-day Utah [27]. On the drive from Salt Lake City to the salt flats area, the ancient shorelines can be seen etched into the sides of the mountains.

The natural seasonal cycle is what makes the 30,000 acre Bonneville Salt Flats such a great location for racing. In the winter, a shallow layer of standing water floods the salt flats, and slowly evaporates during the spring and summer. During
Figure 2.2: Map of the United States of America showing the Bonneville Salt Flats circled

Figure 2.3: Bonneville Salt Flats Entrance Sign
this drying process, wind works to smooth the surface, and make the surface a vast, nearly perfect flat plain [27]. This means that the condition of the salt changes not only year-to-year, but on a day-to-day basis as well. The surface tends to be best towards the middle and end of summer, and sometimes into the early fall. The surface of the salt is where all of the traction is gained to propel the vehicles forward, so understanding the nature of it helps to give a better understanding of any vehicle’s performance.

**Speed on the Salt Flats**

First recognized for its potential for racing in 1896 by W.D. Rishel, it was utilized by Teddy Tezlaff to drive a Blitzen Benz 141.73 mi/h to set an unofficial record. By 1949 the Bonneville Salt Flats had become the standard course for setting world land speed records, where the 300, 400, 500, and 600 mi/h speed barriers were broken [27].

**Setting Records**

Any racing on the Bonneville Salt Flats must take place as an officially sanctioned racing event. The most popular event, Speed Week, takes place in mid-August, and was started in 1948. The Southern California Timing Association (SCTA) is the official timing organization for the event, and Bonneville Nationals Inc. (BNI) is the U.S. record sanctioning organization.

The SCTA is responsible for setting up the race course. Through a combination of surveying and intuition learned through decades of experience, experts select and lay out a course on the salt. Once the course location has been decided upon, a team of volunteers work to smooth the course, using I-beams with weights over them, shown in Figure 2.4.
Figure 2.4: SCTA I-beams used to drag and flatten the course at the Bonneville Salt Flats

In order to set an official U.S. record, a 7-mile course is typically used. Illustrated in Figure 2.5, there are two miles to accelerate the vehicle. Then, the vehicle enters 3 timed miles, where the time to pass through each mile is recorded using laser break-beam sensors, at an accuracy level within 10 nanoseconds. Finally, the vehicle has 2 miles on the course to slow down and come to a stop. If the salt conditions allow, more distance is allowed. The “entry speed,” the time through the first quarter mile of the first timed mile, and the “exit speed,” the time through the final quarter mile of the third timed mile, are also recorded and printed onto the “timing slip” that is given following a speed run. The timing slip from the Buckeye Bullet’s top speed run in October 2004 is shown in Figure 2.6.

2.1.3 The Buckeye Bullet

Over a period of two years, between 2000 to 2002, the Buckeye Bullet team, mostly composed of undergraduate engineering students, designed and built the Buckeye Bullet. The vehicle was constructed as what is known as a “streamliner.” At nearly
Figure 2.5: Land Speed Record Diagram

Figure 2.6: Buckeye Bullet Timing Slip from October 2004
32 feet in length, 24 inches wide, and 30 inches tall, it is fully enclosed by a carbon fiber body shell. It was initially powered by approximately 12,000 sub-C sized NiMH batteries, which were later upgraded to prismatic NiMH batteries.

The Buckeye Bullet 1 (BB1) debuted on the Bonneville Salt Flats in October 2002. Over the next 2 years of racing at the Bonneville Salt Flats and through much refinement and optimization at the Ohio State University, the Bullet was able to achieve a top speed of 321 mi/h, and finally set International (272.737 mi/h) and National (315.958 mi/h) land-speed records in October 2004 [11]-[19].

The Buckeye Bullet was retired after its October 2004 runs, but still holds the both the national and international land-speed records in the E/III class (Electric power, over 1000 kg) [29]. Figure 2.7 depicts the Buckeye Bullet on the Salt Flats.
2.1.4 The Buckeye Bullet 2

Following the success of the Buckeye Bullet, the team sought a new and exciting challenge, and considered the possibility of a vehicle powered by hydrogen fuel cells. This vehicle would be known as the Buckeye Bullet 2 (BB2). The motivation for considering fuel cells was two-fold: first, from a performance standpoint, the inherent nature of fuel cells are to deliver a flat power profile, meaning that the power delivered does not decrease during the run, as is the case with batteries. Second, from a technology perspective, hydrogen fuel cells provide a new challenge to the team, since no hydrogen fuel cell vehicle had ever been developed for the purpose of attempting a land-speed record. BMW holds 9 world speed records set with their H2R vehicle. These records were all set using an internal combustion engine running on Hydrogen, while the BB2 uses a Hydrogen Fuel Cell. The GM-Opel HydroGen3 Zafira holds several endurance records for distances driven under fuel cell power in a 24-hour period, but not under the “flying mile” style of timing used at Bonneville.

In addition to the obvious technology challenge, a fuel cell land-speed record vehicle presented a major obstacle. Fuel cells remain a developing industry with pockets of proprietary technology held by relatively few companies. Therefore, finding a major fuel cell sponsor was necessary to make the project a reality. In the early stages, the team focused its efforts on obtaining suitable industry sponsors. By 2006, Ford Motor Company, Roush, and Ballard Power Systems had all agreed to participate, and the program developed quickly from that point on.

Ford Motor Company worked in conjunction with students from the Ohio State University to develop their own production-based fuel cell land-speed vehicle, known
as the Hydrogen 999 [22][20]. The Hydrogen 999 was based on a Ford Fusion body style, and had components and subsystems similar to the Buckeye Bullet 2, so designs and concepts were shared throughout development. The Hydrogen 999 became the first production based fuel cell vehicle to break 200 mi/h, recording a speed of 207.297 mi/h at the Bonneville Salt Flats in August, 2007.

The fuel cell Buckeye Bullet 2 was designed between 2004 and 2006, and the vehicle was fabricated in 2007. The Buckeye Bullet 2 ran for the first time in August 2007, and has already accomplished a portion of the goals set by the team, setting FIA (Federation Internationale de l’Automobile) records at the Bonneville Salt Flats for the flying kilometer and flying mile (213.042 km/h and 132.129 mi/h respectively), and also recording the highest speed ever achieved by a hydrogen fueled and/or fuel cell powered vehicle at 223.334 mi/h (359.421 km/h). The ultimate goal of the BB2 is to surpass the electric land-speed records set by the Buckeye Bullet 1.

2.1.5 Buckeye Bullet 2 Layout

In order to learn from past lessons and improve upon the Buckeye Bullet 1, a design summit was held in April of 2005 at The Ohio State University. At this summit, partners in industry, academia, and past team members came together to layout the basic design that would be the Buckeye Bullet 2. The primary result of this conference was a consensus of the overall layout of the vehicle. The final vehicle layout is illustrated in Figure 2.9. The design layout flexibility allowed for with electric machines enabled several noteworthy outcomes. Because the electrical power can be created, then transferred through cables, it allowed the power source to be behind the driver, and the electro-mechanical conversion (drive motor) to take
place in front of the driver, making the Buckeye Bullet 2 a front-wheel-drive vehicle. Additionally, the hydrogen and heliox tanks were able to be placed at the rear of the vehicle for improved safety. Driver visibility is greatly improved over the BB1.
CHAPTER 3

CONTROL SYSTEM AND SENSOR ARCHITECTURE

From a broad perspective, the Buckeye Bullet 2 combines hydrogen ($H_2$) and oxygen ($O_2$) through a fuel cell to produce electrical power, and puts that electrical power to the ground using an electric motor, with a byproduct of this reaction being water ($H_2O$) and heat. But, within this vehicle is a control system, ensuring that each piece comes together to work toward this common goal.

The Buckeye Bullet 2 is controlled using a variety of sensors and controllers, each with a specific purpose and varied responsibilities. Since the vehicle runs on a track, with only the driver to monitor vehicle systems, the team must rely on a network of measurements and sensors to properly diagnose system health and proper function. The following sections outline each of those sensors and controllers, and their operation.

3.1 Sensors in the Buckeye Bullet 2

A vital component of any control system is the ability to measure by the use of various sensors. The controller must receive measured inputs, compute and affect control to various vehicle systems. To achieve this on the Buckeye Bullet 2, there are numerous sensors that convey data to the various vehicle controllers. They must
be integrated with the vehicle, and as a complete system, function reliably and as
designed.

3.1.1 GPS Speed Measurement

The Buckeye Bullet 1 used tire rotation to measure speed. Combined with on-
board GPS data, using nominal tire diameter to find the vehicle speed from wheel
speed, the actual tire diameter was found. The tire diameter remained constant at
speeds under 150 mph, but grew linearly with vehicle speed beyond this value. The
result of this was that the overall gear ratio changes as a function of vehicle speed,
and therefore using wheel speed measurements proved to be an inaccurate method
of determining vehicle speed. In order to precisely measure vehicle speed, as well as
to characterize the tire diameter growth, an alternate method of determining vehicle
speed was necessary.

Thus, the measurement of speed using the global positioning system (GPS) was
explored. This system, originally designed for military use, consists of at least 24
Medium Earth Orbit satellites that transmit a navigation message, coarse acquisition
code, and precise code. There are presently 31 satellites transmitting this information.
The navigation message consists of the time-of-day, ephemeris (satellite location),
and almanac (orbit and status) information. The coarse/acquisition (C/A) code is
a spread-spectrum pseudo-random code freely available for public use. The precise
code is encrypted and reserved for military use.

In 2000, the U.S. Congress began the GPS III program, which allowed for the
modernization of the GPS system. With this, a new civilian-use signal, called the
L2C signal for its different frequency use, is meant to complement and improve the accuracy of the original C/A code.

There are several radio frequencies which are of use by the GPS system. They are:

- L1 Band: 1525 to 1595 MHz - Navigation, C/A code, and OmniStar Service
- L2 Band: 1217 to 1237 MHz - L2C code for complementary precision

**Radome Selection and Placement**

The GPS satellites are intended to be spread throughout the sky and in constant movement relative to any fixed earth position to improve precision. Optimally, full coverage of the entire hemisphere without obstruction or reflection will give the greatest accuracy. The GPS system requires at least 4 satellites to perform the calculation of position and time \((x, y, z, t)\). There are special techniques available to provide position information with fewer satellites, but with significantly degraded accuracy. Conversely, more than 4 satellites only serves to improve the accuracy of the position calculation.

The ability for the GPS antenna to have an unobstructed, line-of-sight access to the entire sky, with minimum opportunity for multipath, is key to optimizing its performance. The Bonneville Salt Flats are a near-ideal environment for satellite visibility, with no physical obstructions. The mountains adjacent to the salt flats are at least five miles away, and a negligible obstruction. The only obstruction potential is the car itself. The BB2 is a relatively large mass of metal and potentially poses a sizeable electromagnetic interference potential. Additionally, the vehicle’s high voltage, 3-phase motor further complicates matters.
Ideally, the best approach for mounting the GPS antenna would be to get it above the entire car, and as far away from any sources of electromagnetic noise as possible. This would minimize the interference and optimize the GPS satellite line-of-sight to the GPS receiver. The aerodynamics of the vehicle are a priority however, so having a dedicated radome would greatly compromise the vehicle’s maximum speeds, due to increased aerodynamic drag, even though a dedicated radome would be ideal for reception. There is a section of the car that is raised above the body, and is of non-metallic construction: the stabilizing fin located at the rear of the vehicle, which primarily provides the aerodynamic stability of the vehicle by moving the vehicle’s center of pressure aft. It was decided by the team to locate the GPS antenna inside the uppermost part of the fin, minimizing any potential interference between the antenna and the GPS satellites.

Since the body fully encloses the vehicle to improve aerodynamic performance, this means that it also fully encloses the GPS antenna. So, the material needs to be RF transparent over the GPS frequencies to be able to properly “see” the sky. The body material of choice is carbon fiber, chosen for its high strength to weight ratio, and custom molding capabilities. Unfortunately, carbon fiber has the characteristic of being mostly electrically reflective, making it less than ideal for this application. Another material, fiberglass, has some of the same molding and strength characteristics as carbon fiber, with the characteristic of being more electrically transparent.

**Fin Testing**

In order to test and find which material would be best for allowing the least amount of attenuation of the GPS signals, field testing was performed to provide a
comparison of the possible materials and their performance. This testing was done with the help of the OSU Electroscience Laboratory\textsuperscript{1}.

Figure 3.1 shows the baseline test setup. The setup has a radar horn source on the ground, facing upwards towards block foam. The antenna is an AEL H1479 horn with a range of 1.0-12.4 GHz and a gain of 10 dBi. On top of the foam, 0.724 meters from the end of the horn, rests the GPS antenna, pointed downward towards the horn. The two are then hooked up to a network analyzer and the $S_{21}$ parameter is recorded over the 1 GHz to 2 GHz range, with a measurement resolution bandwidth of 300 Hz. To test the various materials, they are placed under the GPS antenna, in a configuration replicating the fin’s effect. The carbon sample is provided by a carbon fiber and epoxy resin nosecone supplied by the OSU Formula SAE team\textsuperscript{2}. The fiberglass fin used is the version that ran on the Buckeye Bullet 2 in August 2007, which is relatively smaller than the fin used in October of 2007 and all of 2008.

Figure 3.2 shows the resulting $S_{21}$ network parameter for the test setup for various materials. For all practical purposes, and especially in the frequencies of interest, the fiberglass measurement follows the baseline measurement, indicating that it has very little adverse effect on the RF properties of the antenna. Comparatively, the carbon fiber measurement is attenuated by nearly 15 dB over the entire spectrum. The peaks necessary for GPS operation over the L1 and L2 bands are almost completely flattened by the carbon fiber, meaning that the GPS receiver would have difficulty differentiating the signal from noise, resulting in either a significantly reduced position accuracy, or a total lack of information. The plot also indicates the position of the

\textsuperscript{1}Ohio State Electroscience Laboratory: http://electroscience.osu.edu/

\textsuperscript{2}Ohio State Formula Buckeyes Team: http://www.formulabuckeyes.com/
L1 and L2 bands, as described before, showing the antenna filtering of the spectrum, particularly over the frequencies of importance.

The fiberglass fin performed as expected, allowing the GPS frequencies to pass through it with minimal attenuation. The carbon fiber, by comparison, did not allow the signals through, indicating that it would be an inferior choice of material for the vehicle fin for the receiving of GPS signals. There is however an undesired consequence to selecting fiberglass, which is that there is added weight with a fiberglass fin, as
compared to carbon fiber. This added weight, however is deemed an acceptable tradeoff compared to the added value of data in the GPS position information. Since the GPS antenna would sit at a height above the rest of the car, embedded in the front of the fin, it was decided to fabricate the car’s body out of carbon fiber, and fabricate the fin out of fiberglass.

**GPS Satellite Locations**

Since the GPS satellites follow a prescribed, programmed path, it is possible to predict where in the sky the satellites will be at a particular point in time for a given location. The center of the Bonneville Salt Flats are located at approximately 40°49'38"N latitude, 113°48'14"W longitude.
Although the GPS system is designed to give an even satellite coverage throughout the globe, there are “dark spots” in which full satellite coverage is not available for certain periods of time. Almanac information about the positioning satellites is available from Trimble Navigation Ltd. to predict the performance [6]. Figure 3.3 shows the predicted satellite positions for the first day of racing at Speed Week 2008 on the Bonneville Salt Flats, August 18, 2008. This shows that the dilution of precision (DOP) will be under 3 for most of the day, and increase slightly in the evening. The DOP is a unitless measure of measurement precision. A lower DOP is better, and values under 3 indicate very good accuracy in the positioning information. Figure 3.3 also indicates that at least 10 satellites will be available throughout the day. The AgGPS 332, manufactured by Trimble Navigation Ltd., that is used on the Buckeye Bullet 2 is capable of receiving information from up to 12 satellites at a time. It can be noted that the DOP is inversely related to the number of satellites, as it is an important factor in the DOP.

**Error Reduction with OmniSTAR Correction**

The method by which the distance from each satellite is determined is by the delay of the signal of each satellite to reach the receiver. Since each receiver transmits the precise time-of-day (which is accurate to ±340 nanoseconds[4]), each received signal will have a delay in reaching the receiver. The largest natural source of position measurement error is the unknown atmospheric conditions, which affect the signal delay differently. Some of this can be corrected using the L2C code, but a better correction is needed to improve this. The major sources of signal error are [3]:

- Ephemeris data: Errors in the transmitted location of the satellite
Satellite clock: Errors in the transmitted clock, including SA

Ionosphere: Errors in the corrections of pseudorange caused by ionospheric effects

Troposphere: Errors in the corrections of pseudorange caused by tropospheric effects

Multipath: Errors caused by reflected signals entering the receiver antenna

Receiver: Errors in the receiver’s measurement of range caused by thermal noise, software accuracy, and inter-channel biases

To further improve the position and speed measurements possible with GPS, a commercially available corrective service can be added to compensate for the real-time atmospheric delays. Table 3.2 illustrates the errors that make up a large part of the unknown of the GPS position measurement. Corrective services are typically...
Table 3.2: GPS Position Accuracy Error Sources [2]

<table>
<thead>
<tr>
<th>Accuracy</th>
<th>Error Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-30 meters</td>
<td>Ionosphere</td>
</tr>
<tr>
<td>0-30 meters</td>
<td>Troposphere</td>
</tr>
<tr>
<td>0-10 meters</td>
<td>Measurement Noise</td>
</tr>
<tr>
<td>1-5 meters</td>
<td>Ephemeris Data</td>
</tr>
<tr>
<td>0-1.5 meters</td>
<td>Clock Drift</td>
</tr>
<tr>
<td>0-1 meters</td>
<td>Multipath</td>
</tr>
<tr>
<td>0-70 meters</td>
<td>Selective Availability</td>
</tr>
</tbody>
</table>

Differential GPS (DGPS) systems which generate corrections for all non-local GPS errors through a series of base stations with precisely known positions [5].

One service, known as OmniStar, provides continental coverage, sub-meter accuracy throughout the coverage area, and automatic correction of a user’s position [5]. The OmniSTAR network is made of approximately 100 worldwide permanent reference stations. These track all GPS satellites above a 5° elevation and compute corrections every second. These are put into an industry standard message format, RTCM-104 Version II, and sent to the nearest OmniSTAR Network Control Center, where the message is checked, compressed, and put into packets to be transmitted to the satellite transponders [5]. Omnistar provides three levels of service:

- VBS: Basic service with sub-meter accuracy
- XP: Enhanced service with “better than 20cm” accuracy
- HP: Enhanced service with “better than 10cm” accuracy
The OmniSTAR corporation has generously donated their HP service for use on the Buckeye Bullet 2 while on the Salt Flats. The use of this service means that the position and vehicle dynamics of the Buckeye Bullet 2 can be known with great precision at all times. In August 2007, the accuracy was indicated by the receiver to be 3 inches (±0.5 inches), consistent with the OmniSTAR HP service.

**Trimble AgGPS 332**

Figure 3.5 shows the Trimble AgGPS 332 receiver that is used by the Buckeye Bullet 2. It includes Omnistar correction capabilities, and can output data over both a serial RS-232 link, as well as a CAN (NMEA-2000) link. The advantage of having a receiver separate from the receiving antenna is that the antenna can be placed in its optimal location, and the receiver can be located in its optimal location as well. In the Buckeye Bullet 2, the receive antenna is located inside the top front edge of
Figure 3.5: Trimble AgGPS 332 Receiver. This receiver is capable of receiving GPS satellite information, as well as DGPS correction information.

the fin, and the receiver is located alongside the rear fuel cell module, where settings can easily be changed.

**GPS Information Available**

The summary of information available from the GPS receiver and accessories on either CAN or serial line with respect to the Buckeye Bullet 2 dynamics is as follows:

- Latitude and Longitude of Vehicle Position
- Compass Bearing (Direction of Travel)
- Vehicle Speed
- Age of Differential GPS correction signal
- Standard Deviation of Position error (Latitude, Longitude, and Altitude)
- Dilution of Precision (DOP) for the Horizontal, Vertical, and Time dimensions
3.1.2 Pressure Measurement

Pressure readings are especially useful on the Buckeye Bullet 2 because they indicate the operation of the gas delivery system, which is crucial to improved fuel cell performance. Table 3.3 shows the anticipated maximum and minimum pressure measurement range of various BB2 components that are available on the vehicle side of the data collection system. This includes the pressure readings that are available from the fuel cell system. These measurements also provide other metrics such as the performance of the brake system and cooling pumps.

<table>
<thead>
<tr>
<th>Measurement</th>
<th>Units</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hydrogen Tank</td>
<td>psi</td>
<td>0</td>
<td>6500</td>
</tr>
<tr>
<td>Hydrogen after Regulator</td>
<td>psi</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>Hydrogen downstream</td>
<td>psi</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>Heliox Tank</td>
<td>psi</td>
<td>0</td>
<td>4000</td>
</tr>
<tr>
<td>Heliox after Regulator</td>
<td>psi</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>Stack Air Inlet Pressure A</td>
<td>psi</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>Coolant Pressure A</td>
<td>psi</td>
<td>0</td>
<td>15</td>
</tr>
<tr>
<td>Stack Hydrogen Inlet Pressure A</td>
<td>psi</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>Stack Hydrogen Inlet Pressure B</td>
<td>psi</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>Humidification Pump Pressure A</td>
<td>psi</td>
<td>0</td>
<td>150</td>
</tr>
<tr>
<td>Humidification Pump Pressure B</td>
<td>psi</td>
<td>0</td>
<td>150</td>
</tr>
<tr>
<td>Front Brakes</td>
<td>psi</td>
<td>0</td>
<td>3000</td>
</tr>
<tr>
<td>Rear Brakes</td>
<td>psi</td>
<td>0</td>
<td>3000</td>
</tr>
<tr>
<td>Brake Boost</td>
<td>psi</td>
<td>0</td>
<td>3000</td>
</tr>
<tr>
<td>Coolant Fuel Cell A</td>
<td>psi</td>
<td>0</td>
<td>30</td>
</tr>
<tr>
<td>Coolant Fuel Cell B</td>
<td>psi</td>
<td>0</td>
<td>30</td>
</tr>
</tbody>
</table>

Table 3.3: Pressure Measurements
3.1.3 Voltage Measurement

Measuring the electrical power that is being delivered to the drive system becomes a simple calculation, accomplished by simply multiplying the voltage and current from the high voltage bus together. For the BB2 to monitor the electrical systems of the vehicle, various voltage measurements are made, noted in Table 3.4. Due to the critical nature of the voltage information, there are three separate measurements made of the high voltage bus. The first is a LEM brand measurement module, placed in an insulated box, adjacent to the fuel cell modules. The second is a voltage measurement at the inverter, at the very front of the vehicle, where the DC power is converted into AC power. Finally, there is a measurement inside of the fuel cell modules, recorded by the fuel cell controller. All three are independent measurements.

There are also measurements on the voltage of the auxiliary batteries, because they are each critical to the car performing well. If the main 24 volt auxiliary pack becomes too low in voltage (≤ 24.3V), then the controllers, especially the fuel cell controller will not work correctly. The other measurements are there to ensure the proper function of the rest of system. These measurements, and their respective anticipated maximum and minimum measurement range are shown in Table 3.4.

3.1.4 Current Measurement

The current measurements on the vehicle measure when power is being drawn. This information is used to measure if each auxiliary pack is correctly sized for the application. As with the voltage measurements, there are 3 separate current measurements on the high voltage bus. In this case, the measurements by the LEM and
Table 3.4: Voltage Measurements

The fuel cell modules are per-module current measurements, to help diagnose any separate fuel cell problems. The measurement at the inverter is the current on the entire high voltage bus. These measurements, and their respective anticipated maximum and minimum measurement range are shown in Table 3.5.

Table 3.5: Current Measurements
3.1.5 Position/Speed/Acceleration Measurement

The course that is laid out is at the Bonneville Salt Flats is the same course for the entire meet, unless the conditions of the salt call for it to be moved, which can happen on occasion. Because of this, it makes it very easy to compare different runs to each other using track position as the dependent axis. Because a limiting factor is track distance, and the ultimate goal of the vehicle is increasing its top speed, it makes sense to pay particular attention to the position and speed measurements. So, measurements from the GPS, accelerometer, wheel speed, and motor speed (when combined with the gear indicator) all work to define in detail the vehicle dynamics for each run. Combined with measurements from the suspension travel sensors, much can be inferred about the operation of each of the four corners of the vehicle. As an example, Figure 3.6 shows the exact position of the Buckeye Bullet 2 at Speed Week 2008 on the Salt Flats. The x-axis indicates track position in miles, and the y-axis indicates the lateral track position in feet. By overlaying each of these, the driver can begin to recognize similar paths or subconscious corrections that are made at various points on the track during a run. This information can be used to help refine the steering system and help the driver remember additional information regarding a run, and assist in improving subsequent runs.

3.1.6 Temperature Measurement

Temperature measurements on the vehicle can be useful in monitoring components on the vehicle. There are numerous temperature sensors in various location within the cooling system, to monitor heat exchanger performance, check design parameters, and ensure that the fuel cells do not overheat. Table 3.7 shows the anticipated range
of minimum and maximum values for the temperature measurements that are taken. In addition, monitoring the hydrogen tank regulator to ensure it does not get too cold is an important design consideration. The o-rings within the regulator maintain functionality to $-40^\circ C$, and below that their performance is at risk. Figure 3.7 shows the hydrogen tank temperatures during the runs at Speed Week 2008. The chart shows in max and min measurements that the temperature drops relatively linearly throughout the run. From this, it is clear that a higher starting temperature helps keep the regulator ending temperature higher and away from the design limit of the o-rings. The trade-off to the higher starting temperature is that this decreases the amount of hydrogen that can be kept in the tank.

### 3.1.7 Flow Rate Measurement

Typically, devices used to measure flow rates of a fluid tend to be large, bulky devices. Due to the bulk, only two such devices are used in the BB2. They are devices for regulating the heliox supply to each of the fuel cell modules that provide feedback of flow rate. They are listed, along with the minimum and maximum expected values

<table>
<thead>
<tr>
<th>Measurement</th>
<th>Units</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 Axis Vehicle Accelerometer (Future)</td>
<td>G’s</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>4 Wheel Speed Measurements (Future)</td>
<td>RPM</td>
<td>0</td>
<td>5000</td>
</tr>
<tr>
<td>4 Suspension Travel</td>
<td>cm</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>GPS Speed</td>
<td>mph</td>
<td>0</td>
<td>400</td>
</tr>
<tr>
<td>GPS Position</td>
<td>Degrees</td>
<td>-180</td>
<td>180</td>
</tr>
<tr>
<td>Motor Speed</td>
<td>RPM</td>
<td>0</td>
<td>12000</td>
</tr>
</tbody>
</table>

Table 3.6: Position Measurements
Figure 3.6: Track position overlay at Speedweek 2008 (See Table 6.1)

Figure 3.7: Hydrogen Tank Regulator Temperature at Speedweek 2008 (See Table 6.1)
### Table 3.7: Temperature Measurements

<table>
<thead>
<tr>
<th>Measurement</th>
<th>Units</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fuel Cell A Inlet</td>
<td>°C</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>Fuel Cell B Inlet</td>
<td>°C</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>Fuel Cell A Outlet</td>
<td>°C</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>Fuel Cell B Outlet</td>
<td>°C</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>Ice Bath A Inlet</td>
<td>°C</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>Ice Bath B Inlet</td>
<td>°C</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>Ice Bath A Outlet</td>
<td>°C</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>Ice Bath B Outlet</td>
<td>°C</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>Hydrogen Tank Regulator</td>
<td>°C</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>Motor Temperature</td>
<td>°C</td>
<td>0</td>
<td>150</td>
</tr>
<tr>
<td>Fuel Cell Coolant</td>
<td>°C</td>
<td>0</td>
<td>100</td>
</tr>
</tbody>
</table>

It would be beneficial to have flow rate measurements on the cooling system, but packaging and the bulk of the sensor does not allow for it.

### Table 3.8: Flow Rate Measurements

<table>
<thead>
<tr>
<th>Measurement</th>
<th>Units</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fuel Cell A Heliox Flow</td>
<td>g/s</td>
<td>0</td>
<td>500</td>
</tr>
<tr>
<td>Fuel Cell B Heliox Flow</td>
<td>g/s</td>
<td>0</td>
<td>500</td>
</tr>
</tbody>
</table>

Table 3.8: Flow Rate Measurements
3.1.8 Driver Control Measurement

The most important control inputs on the Buckeye Bullet 2 are those that come from the driver. His inputs and the corresponding vehicle responses reflect a safety-first design criteria. Thus, if his foot is not on the pedal, no current should be drawn. Or, if his inputs call for closed tank solenoids, then the tank solenoids should be closed. Table 3.9 shows the inputs that the driver has to the vehicle. Most of them are simple binary inputs to turn different components on and off, except for the pedals. Please note that the brake and clutch pedals are not listed here because they are pressure measurements and are listed in Table 3.3.

<table>
<thead>
<tr>
<th>Measurement</th>
<th>Units</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accelerator Pedal</td>
<td>%</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>Main Power Switch</td>
<td>on/off</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Upshift/Downshift Buttons</td>
<td>on/off</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>High/Low Speed Parachute Releases</td>
<td>on/off</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Radio Push-to-talk</td>
<td>on/off</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Fuel Cell Start Button</td>
<td>on/off</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Gas Tank Solenoid Switch</td>
<td>on/off</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>High Voltage Enable</td>
<td>on/off</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 3.9: Driver Inputs
3.2 Controls in the Buckeye Bullet 2

There are numerous controllers on the BB2, each with a particular task, that are described in the following sections. These include the motor controller, fuel cell controller, thermal controller, and supervisory controller.

3.2.1 Motor Controller and DC-AC Inverter

Utilizing carry-over technology and experience gained from the Buckeye Bullet 1, both the electric motor and DC/AC inverter were used on the Buckeye Bullet 2 as well. This allowed development to be focused on the power delivery systems of the vehicle, rather than the traction system which had been highly developed and refined with the Buckeye Bullet 1 program.

Motor Controller

The motor utilized by the Buckeye Bullet 2 is a 3-phase AC electric motor. It is a purpose-built custom motor, with a designed duty cycle of full power for 2 minutes, and 45 minutes off, with this cycle intended to match the land speed racing application [15]. The 4-pole machine has a speed range of 0-10500 rev/min operating range, and a sufficiently large air gap chosen to tolerate the high vibration environment in which it operates [16].

The control of this motor is achieved using a custom DC/AC inverter and motor controller combination. Based on the Safronics VG5 control board, seen in Figure 3.8 and Saminco M1-500 in Figure 3.9, the inverter is also purpose built for a similar duty cycle as the motor. With liquid cooling, it is capable of supplying the necessary power to the motor for the duration of a land speed run. This particular control
board provides a variety of control methods, but the method used by the Buckeye Bullet 2 is a torque control mode, achieved through a flux vector method. In this method, the estimated motor flux and torque is compared to the reference values, and corrected using a variable frequency drive [32].

While this particular control is not a system built by the Buckeye Bullet 2 team, the whole-vehicle control systems must interface to it, and a detailed understanding of the function of each controller leads to a better supervisory control scheme. The primary control input to the motor controller is the torque reference signal, provided as a 0 to 10 volt analog signal. The highest torque reference represents requesting the maximum torque setting from the electric drive. Figure 3.10 shows the strategy within the motor controller to take the input torque reference and generate the internal torque reference for the control system. This circuitry includes filtering the input signal, and adding a compensation bias and speed limit, using a preset speed limit and speed feedback. This sum is subsequently limited by the torque limit setting.
Figure 3.9: Saminco M1-500 Electric Traction Inverter

Figure 3.10: Motor Torque Control [30, 153]
3.2.2 Ballard Fuel Cell Controller

The fuel cells systems used in the Buckeye Bullet 2 are from a city transit bus program, and were designed and manufactured by Ballard Power Systems, from Burnaby, BC in Canada. Although the power demands from the city bus program were less than what was desired for the Buckeye Bullet 2, the modules were capable of producing power greater than the specifications indicate [25]. This meant that the controller and software that had been used for the transit bus program could be carried over as well, although modified for the specific conditions of land speed racing, and the increased power output of the fuel cells.

Visteon Automotive Controller

The primary control module for the fuel cells is a standard automotive engine control unit (ECU) manufactured by Visteon, shown in Figure 3.11. This is a CAN-capable controller that has numerous analog inputs, PWM outputs, and digital inputs and outputs. In addition to the control unit, Ballard Power Systems provided external circuitry to control the components within the modules that are used. The wiring harness was also provided for 2007, but in 2008 the BB2 team upgraded and modified the harness to be more integrated with the Buckeye Bullet 2 wiring.

The code that runs on this controller is proprietary software from Ballard Power Systems, and the Buckeye Bullet 2 team worked in conjunction with Ballard Power Systems to optimize and streamline the software for land speed racing, as well as to improve diagnostic capabilities. The code is natively written in C++, and is an adaptation of the code used in the city transit bus program [25].
3.2.3 Fuel Cell Thermal Management (Cooling) Controller

To control the cooling system and accurately meter the water flow required to cool the fuel cells, pump controllers from Alltrax are used. Figure 3.12 shows the controllers that are used. There are two of these in the Buckeye Bullet 2, one for the fuel cell pump, and one for all four of the ice bath pumps. This motor controller takes a 0-5V control signal to increase the output voltage, pulse width modulated from a 48V source. These controllers have a ramp time of 1 second to go from off to full on. The actual control for the cooling system takes place on the supervisory controller, but it is important to recognize that the supervisory controller is not controlling the pumps themselves, but a DC motor controller.
3.2.4 Supervisory Controller

All of the controllers that have been mentioned up to now take commands, and get their reference signals from the supervisory controller. This controller is tasked with monitoring the driver’s demands, managing the power on the vehicle, regulating the temperature of the fuel cells, and controlling the shifting. The controller that is utilized for these tasks is a Mototron Motohawk controller, shown in Figure 3.13. This is a controller designed for automotive applications, and for rapid prototyping control systems. It is programmed and monitored using a Controller Area Network (CAN).

3.3 Communication and Data Architecture

To create a working system comprised of all of the vehicle controllers and subsystems, such that each may have the information available that it needs, a relatively
complex set of networks connects each of the various devices to the vehicle’s information. Each of the network layers will be described in the following sections.

3.3.1 Data Structure

Figure 3.14 shows the basic layout of the data system. As seen, there are sometimes more than one path to transfer information between devices. The data logging for the vehicle is a 2 channel CAN logger that is placed on the high speed (1 Mbps) data line, and the low speed (250 kbps) control line while the vehicle is running. Fuel cell information is logged using the Ballard Fuel Cell Logger, which receives data through a serial link with the fuel cell controller, and a TCP/IP link with the cell voltage monitors.
3.3.2 TCP/IP Ethernet Communication

One method for moving information through the vehicle is standard ethernet communication. This is the method used by the cell voltage monitors in the fuel cell modules to transmit the voltage of each cell. This information would then be picked up and logged by the Ballard Fuel Cell Logger. This method also makes diagnostics on the fuel cell modules easier as well. The xLink diagnostic module from Ballard allows viewing of and changing parameters of the fuel cell system through proprietary software, but it all takes place through the TCP/IP link, so all that is necessary to make changes, including re-flashing software, is a standard ethernet link, available on a laptop computer. All of these devices are connected using a standard office ethernet
switch. Table 3.10 shows the different components on the vehicle that are Ethernet capable, and the data rate that each runs at.

<table>
<thead>
<tr>
<th>Device</th>
<th>Data Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ballard Fuel Cell Cell Voltage Monitor (CVM)</td>
<td>10 Mbit/s</td>
</tr>
<tr>
<td>Ballard Data Logger</td>
<td>10 Mbit/s</td>
</tr>
<tr>
<td>Ballard xDiag</td>
<td>100 Mbit/s</td>
</tr>
<tr>
<td>Compact RIO Data Logger</td>
<td>100 Mbit/s</td>
</tr>
</tbody>
</table>

Table 3.10: Ethernet Connected Devices

### 3.3.3 CAN Communication

The primary method of communication between most devices on the vehicle is achieved via Controller Area Network (CAN) communication. There are two CAN networks that exist on the vehicle. The first is the control network, whose primary function is to pass information necessary for control between the different devices. This network runs at 250 kbps, which is more robust to noise interference due to its lower speed. For example, this is how the supervisory controller communicates the current request from the driver to the fuel cell controller. The second CAN network is the data network, which runs at 1 Mbps. This is used primarily for controllers to broadcast any and all information that may be helpful to debugging the system at a later point. Anything that is not critical for control is assigned to this line.

Table 3.11 shows the various CAN capable devices and their corresponding networks on the vehicle. Notice that there are several devices that are on both networks,
such as the Motohawk vehicle controller. This device relays messages from one network to another that may be requested by other devices. The devices are listed in descending priority (the highest priority device is at the top). This prioritization is accomplished using the arbitration capabilities built into the CAN protocol. Thus, the messages that need to have a higher priority are given a lower identification number. If any two devices try to transmit at the same time, the message with the higher priority, or lower identification number, will be transmitted, and the other controller will try again when the CANbus is inactive. This allows communication between the supervisory controller and fuel cell controller to have priority over, such as the temperature monitor.

<table>
<thead>
<tr>
<th>Device</th>
<th>Protocol</th>
<th>CAN Network</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>250kbps</td>
</tr>
<tr>
<td>Motohawk Vehicle Controller</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Saminco Inverter</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Ballard Fuel Cell Controller</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Axiomatic Temperature Monitor</td>
<td>J1939</td>
<td>X</td>
</tr>
<tr>
<td>Trimble GPS Receiver</td>
<td>NMEA 2000, J1939</td>
<td>X</td>
</tr>
<tr>
<td>Compact RIO Data Logger</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Bosch Driver Display</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.11: CAN Connected Devices

### 3.3.4 Serial Communication

Also seen in Figure 3.14 are a number of serial devices. This protocol, RS-232, is limited in that it can only support communications between two devices, whereas
CAN can support many more devices. But an advantage of RS-232 is that each of the two devices have their own transmission line, so they can be sure that no other device is transmitting on it. On the Buckeye Bullet 2, the GPS receiver, the fuel cell controller, fuel cell logger, fuel cell diagnostics module, and compact RIO are all capable of RS-232 communications. The fuel cell logger uses a connection to its controller to log information for later diagnostics of the fuel cell system. The Compact RIO takes the serial information from the GPS receiver, and retransmits it over the high-speed data CAN bus. There is also internal serial communication between the fuel cell diagnostic module and the fuel cell controller, that make more information available over the ethernet bus using the proprietary fuel cell software.
The process of converting hydrogen and oxygen into power and putting that power to the ground could not take place without a control system. The design constraints for the top-level design of the control system for the Buckeye Bullet 2 were relatively basic: create a safe, robust system that would maximize the top speed of the vehicle. To this end, the types of systems used should be easy to use, configure, and diagnose. The control systems discussed here will be the vehicle cooling system, power management scheme, and various safety systems that protect the vehicle.

4.1 Cooling System Development

One of the most important functions of the supervisory controller is the control of the temperature of the fuel cells. The control in this system utilizes the thermal energy byproduct of the fuel cell reaction to heat the fuel cells themselves, and an “ice bath” is utilized to remove energy that causes the temperature to go above the setpoint.
4.1.1 Overall System

The method of removing thermal energy from the fuel cells involves a two-loop system, as outlined in Figure 4.1. On the right of Figure 4.1, an estimated 300 kW is rejected from one fuel cell module at full power. Utilizing a pump capable of up to 80 GPM, the coolant is pumped through the fuel cell module, and then through a heat exchanger. For this loop, the coolant must be de-ionized because it must pass through the fuel cells themselves, and must conduct very little electricity. For this application, de-ionized water was chosen because of its ease of availability. On the other side, the “ice bath” side, a pump pushes coolant through the ice bath module and through the heat exchanger. The primary reason for a two-loop system is that the ice bath side can become more contaminated without affecting function, and therefore does not need to be de-ionized. This way, ice can be quickly filled in the vehicle on the salt flats, without worry of ionizing the water that travels through the fuel cells.

Buckeye Bullet Thermal Model

From the basic strategy in Figure 4.1, a thermal model was developed in Simulink, in conjunction with several others involved with the project. Shown in Figure 4.2, this model represents a method of checking to see if the cooling system would be able to mechanically function as designed, and thus be able to reject enough thermal energy from the fuel cells. To make this a model suitable for control design, several modifications were made. First, delays were added between each of the major elements (ice bath, heat exchanger, and fuel cells). Next, constant flow rates were replaced with SISO controllers. The flow rate of the ice bath is controlled by the outlet temperature on the fuel cells, the variable that is ultimately regulated. The fuel cell
coolant controller is an open loop controller based on the current available from the fuel cells, controlling the coolant pump on the fuel cells.

**Fuel Cell Thermal Model**

To represent the thermal properties of the fuel cell, Equation 4.1 is used. This equation computes the outlet temperature of the fuel cells, which is the temperature to be regulated. This equation utilizes the water density lookup, given the water temperature, in °C, shown in Figure 4.3.

\[
\dot{T}_{FC, out} = \frac{1}{C_{p, FC}} \left( \frac{.003785 \cdot \dot{\nu}_{FC,GPM}}{60} \right) 1000 \cdot f_{density}(T) \cdot C_{p, water} \left[ T_{FC, in} - T_{FC, out} \right] + Q_{FC}
\]

(4.1)
Figure 4.2: Buckeye Bullet 2 Fuel Cell Cooling Model

Figure 4.3: Water Density Lookup Table ($f_{\text{density}}(T)$)
Heat Exchanger Model

An integral part of the cooling system is the heat exchanger. It keeps the de-ionized water from becoming contaminated from the ice bath loop, and also must be capable of transferring approximately 300kW per fuel cell module. Part of the model for this exchanger was provided by Modine Inc., who helped to develop the specific heat exchanger that was used. Table 4.1 shows the values used in the lookup, and Figure 4.4 shows the values as a contour plot.

To compute the outlet temperatures of the heat exchangers, equation set 4.2 is used. This utilizes both the water density lookup in Figure 4.3, as well as the lookup table in Figure 4.4 (Table 4.1).

\[
Q_{HX} = (T_{h,in} - T_{c,in}) f_{QED}(\dot{v}_{FC,GPM}, \dot{v}_{IB,GPM}) \tag{4.2a}
\]

\[
T_{h,out} = \frac{Q_{HX}}{\left(\frac{0.003785 \cdot \dot{v}_{FC,GPM}}{60}\right)1000 \cdot f_{density}(T) \cdot C_{p, water}} + T_{h,in} \tag{4.2b}
\]

\[
T_{c,out} = \frac{Q_{HX}}{\left(\frac{0.003785 \cdot \dot{v}_{IB,GPM}}{60}\right)1000 \cdot f_{density}(T) \cdot C_{p, water}} + T_{c,in} \tag{4.2c}
\]

Thermal Transport Model

The delay in the cooling lines constitutes perhaps the greatest challenge to the cooling system in the Buckeye Bullet 2. For example, the time it takes cold water to physically travel from the ice bath to the heat exchanger depends on the flow rate, and can greatly affect the performance of the control system. The addition of this delay to the cooling control model allowed for a more accurate representation of the system delay that would be seen by the control system when trying to regulate the
Figure 4.4: Modine QED Lookup ($f_{\text{QED}}(\dot{v}_{\text{FC,GPM}}, \dot{v}_{\text{IB,GPM}})$). The contour lines represent lines of constant energy, in Joules, that can be dissipated per unit of entering temperature differential in °C.
setpoint. This makes for one of the largest nonlinearities in the system because the quicker the cooling pumps are working, the faster the system will react to changes. This was accomplished in simulation using a variable time delay.

Equation 4.3 shows how to determine the amount of delay in the line, in seconds. In this equation $\dot{v}$ is the flow rate, in GPM, and $v$ is the volume of fluid between the two points, in gallons.

$$\text{Delay (seconds)} = \frac{60 \cdot \dot{v}}{v} \quad (4.3)$$

As an example, the water traveling from the ice bath outlet to the heat exchanger goes through 1.25 inch pipe, for 5 feet, which gives a volume of 0.0426 cubic feet, or 0.319 gallons. If the water is traveling at $\dot{v} = 50$ GPM, then the delay will be:

$$\text{Delay (seconds)} = \frac{60 \cdot 0.319}{50} = 0.383 \text{ seconds}$$
<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Volume (Gallons)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ice Bath</td>
<td>Heat Exchanger</td>
<td>0.319</td>
</tr>
<tr>
<td>Heat Exchanger</td>
<td>Ice Bath</td>
<td>1.036</td>
</tr>
<tr>
<td>Fuel Cell</td>
<td>Heat Exchanger</td>
<td>1.306</td>
</tr>
<tr>
<td>Heat Exchanger</td>
<td>Fuel Cell</td>
<td>0.326</td>
</tr>
</tbody>
</table>

Table 4.2: Thermal Transport Delay Parameters

Ice Bath Model

The last major piece of the system model is for the ice bath. The scope of this study will not entail micro-physical reactions that take place between a drop of warm water and an irregularly shaped chunk of ice. However, the model that was used was based on some rough estimations done in a lab. The tests showed that, using a shower-head method of spreading warm water over the ice cubes, the temperature from the outlet remained relatively constant, ranging from 5 to 10°C. Once the ice began to run out, that temperature slowly climbed, until finally there was no ice left, and the energy then had to be absorbed into the remaining liquid water.

To simplify the model, it was assumed that as long as there was ice, all of the energy was dissipated into the latent heat of fusion of the ice. As soon as the ice was fully converted to water, all of the energy was then absorbed into the heating of the water. This is designed to operate on the safe side, because as the ice is melting, there is water being added to the loop that could absorb some of that energy as the ice continues to melt. Thus, in a worst case scenario, the simulations would predict more ice than is necessary. A couple of extra pounds of ice, though not optimal, would
allow the fuel cells to be maintained at the correct temperature, and not prematurely abort a run from a condition of overheating.

Equation set 4.4 shows the equations used to model the ice bath. Again, these equations use Figure 4.3 to look up the density of water for a given temperature.

\[
\dot{V}_{kgs} = \left( \frac{0.003785 \cdot \dot{V}_{IB, GPM}}{60} \right) 1000 \cdot f_{\text{density}}(T) \tag{4.4a}
\]

\[
\dot{T}_{\text{rise, water}} = \begin{cases} 
\frac{\dot{V}_{kgs} \cdot C_{p, water}}{C_{p, FC} \left( m_{\text{water}} + m_{\text{ice}} \right)} (T_{IB, in} - \dot{T}_{\text{rise, water}}) & \text{if } Q_{\text{ice, kJ}} > Q_{\text{to melt, kJ}} \\
0 & \text{otherwise} 
\end{cases} \tag{4.4b}
\]

\[
\dot{Q}_{\text{ice}} = (T_{IB, in} - T_{IB, out}) C_{p, water} \cdot \dot{V}_{kgs} \tag{4.4c}
\]

\[
T_{IB, out} = \begin{cases} 
273 & \text{if } Q_{\text{ice}} < E_{\text{to melt, kJ}} \\
\dot{T}_{\text{rise, water}} & \text{otherwise} 
\end{cases} \tag{4.4d}
\]

### 4.1.2 Benefits and Challenges

**Decreased Mass of Ice**

The primary benefit of optimizing the cooling system is to reduce the amount of ice that is required to be carried on the vehicle during a run. Reduced weight translates directly to greater acceleration, and thus reducing ice weight will increase vehicle top speed.

**Proper Sizing of Pump Batteries**

By running the pumps no more than necessary, the auxiliary energy required to power them is reduced. Knowing how much the pumps are expected to be run, and with track testing, can help to obtain the proper sized batteries for the application.
Maximum Fuel Cell Performance

Another added benefit to an optimized cooling control is that the fuel cells will perform better. When operating at design temperature, the fuel cells produce more power and run more efficiently than when they are cold. At lower temperatures, the ratio of heliox to hydrogen has to be increased to assist in the reaction taking place. Arriving at and maintaining the proper operating temperature enables the fuel cells to run more efficiently.

Unknown State of Cubed Ice

An unknown in the cooling system is the state of the cubed ice. The ice could be any number of shapes, and of any given quality. This unknown is one of the reasons for having control, and being able to compensate for unknowns such as this.

Unknown Heat Capacity of Fuel Cells

Every mass has a certain specific heat capacity, which is the amount of energy required to raise a unit of the mass by a unit of temperature. Using information provided by Ballard Power Systems, the fuel cell modules were assumed to each have a heat capacity of 147.9 \(kJ \cdot K^{-1}\).

Limited Cooling Capability

Another challenge with a cooling system designed for land speed racing is that once the ice is melted, then cooling capacity is exhausted. There is a fixed amount of energy that can be dissipated, and once that level is attained, then the system can either heat up further, or be shut off. To save the components and reduce the
potential for permanent damage in the fuel cell system, the system is designed to automatically shut down the fuel cells in the event of a high temperature situation.

**System Delays**

Due to the streamliner configuration of the Buckeye Bullet 2, the cooling system components cannot be in close proximity to each other as desired. Therefore, there are numerous delays in transferring the thermal energy from device to device, and delays in measuring those temperatures.

### 4.1.3 Feedback Input

The means of measuring the performance of the cooling system is accomplished using thermocouples that are placed at key points throughout the cooling system as shown in Table 3.7. To receive the data provided by the thermocouples, and to provide ice point compensation to have an accurately calibrated temperature measurement, a CAN-capable device is used. The Axiomatic TC-20, originally intended for diesel engine calibration and control is shown in Figure 4.5. This particular module updates the temperature once per second, with a precision of 1°C.

### 4.1.4 Pump Actuators

To provide the power to move the coolant through both the fuel cell loop and ice bath loop, DC pumps and DC pump controllers (Figure 3.12) are utilized, allowing the pumps to be controlled with a 0-5V control signal. To access the level of flow the pumps can achieve, a controlled laboratory bench test was performed. In this test, a control voltage is applied to the DC motor controller (Figure 3.12), and the corresponding flow rate is recorded. The resulting flow versus control signal is shown
in Figure 4.6. This allows the control to be flow rate based, and use this open-loop conversion to command based on the desired flow rate.

**Fuel Cell Pump**

The fuel cell pump is shown in Figure 4.7. This is a standard commercially available 48V DC motor, with a lightweight plastic pump head, capable of pumping over 60 GPM at 30 psi of pressure, which was the estimated performance requirement to sufficiently cool the fuel cells and provide enough pressure on the fuel cell membranes. There is one pump used to push water through both fuel cell modules and both heat exchangers.

**Ice Bath Cooling Pumps**

To provide the necessary flow rate on the ice bath side of the cooling system, four pumps from Meziere Enterprises are used. These aluminum pumps are lightweight,
Figure 4.6: Pump Flow Rate versus Command Voltage

Figure 4.7: Fuel Cell Cooling Pump
and provide enough flow to effectively cool the fuel cells. Two are used per heat exchanger and are attached to the front of each of the ice tanks, shown in Figure 4.8. Since the pump is originally intended to run continuously on 12V, the control signal is designed to input 3V, in order to avoid overheating the motor.

4.1.5 Potential Control Schemes

Several different control schemes were examined that could potentially meet the cooling requirements on the Buckeye Bullet 2. The following sections examine the various schemes that were considered. In these simulations, the assumption is made that 80 lbs per fuel cell is available for cooling.

The output of each simulation (Figures 4.9-4.13) shows four plots:

1. The first plot (top left) shows the temperatures at the fuel cells. This includes the fuel cell outlet ($T_{FC,\text{out}}$), inlet ($T_{FC,\text{in}}$), and difference ($\Delta_{FC} = T_{FC,\text{out}} - T_{FC,\text{in}}$).
$T_{FC,in}$). Also indicated, in green, is the desired values for the fuel cell outlet ($80^\circ C$), and difference ($15^\circ C$).

2. The second plot (top right) shows the temperatures of the heat exchanger. These values are the cold ice bath side inlet and outlet ($T_{hx,c \, in}, T_{hx,c \, out}$), and the hot fuel cell side inlet and outlet ($T_{hx,h \, in}, T_{hx,h \, out}$).

3. The third plot (lower left) shows the control inputs for both the fuel cell and ice bath sides of the cooling system. $\dot{V}_{FC}$ is the flow rate (in GPM) for the fuel cell side, and $\dot{V}_{IB}$ is the flow rate (in GPM) for the ice bath side.

4. The fourth and final plot (lower right) shows the operation of the ice bath. Shown are the inlet and outlet to the ice bath ($T_{IB,in}, T_{IB,out}$).

No Ice Bath Control

The first simulation shows what the system would do if there were no ice bath control system in place. This scenario would not work well because the system is relying entirely on the heat capacity of the fuel cells and water in the fuel cell cooling loop to absorb all of the thermal energy created as a byproduct of the fuel cell reaction. Figure 4.9 shows how the system would operate if this were the case. This simulation shows the fuel cells only being able to operate for approximately 25 seconds before the fuel cells overheat, which falls short of the 90 second estimated run time of the Buckeye Bullet 2.

Preset Constant Control

Concluding that no ice bath cooling is insufficient to complete a run on the salt flats, another simulation is run to see how the cells would run if the flow rate on
Figure 4.9: Cooling system without Ice Bath Control, and 60 GPM on the fuel cells. In this set of plots, it can be seen that the run can only last for the amount of time that the fuel cells are heating up, absorbing the thermal energy that they generate into their mass and the mass of the fuel cell cooling water. The ice bath plots can be ignored in this case because there is no flow in that side, and the inlet is assumed to be the pre-heated starting temperature of 40°C.
the ice side were a constant 30 GPM. Figure 4.10 shows the resulting effect on the temperatures of the fuel cells. In this simulation, the fuel cells do not achieve their $80^\circ C$ operating temperature until after the ice supply is exhausted. However, with these flow rates, the fuel cell modules are over-cooled, and nearly 30 seconds of unused cooling capacity remains after the ice has run out after nearly 90 seconds.

**Feedforward Control**

If programmed preset constant values are able to sufficiently cool the system, then imagine an operator that was instructed to change the flow rate based only on the fuel cell outlet temperature. This would accomplish a sort of feedforward control, which would give an increasing flow rate as the temperature got higher. This way, when the temperature reaches the desired value, there is enough flow to reject the amount of heat being generated. Figure 4.11 shows the result of having only this feedforward component. It is an open-loop form of control, but it does a decent job of performing the cooling operation. But this is in simulation, where unknown disturbances to the system are not be accounted for, so this may not be the best choice for control.

<table>
<thead>
<tr>
<th>Fuel Cell Outlet Temperature</th>
<th>Flow Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>$40^\circ C$</td>
<td>0 GPM</td>
</tr>
<tr>
<td>$65^\circ C$</td>
<td>10 GPM</td>
</tr>
<tr>
<td>$75^\circ C$</td>
<td>20 GPM</td>
</tr>
<tr>
<td>$85^\circ C$</td>
<td>40 GPM</td>
</tr>
<tr>
<td>$90^\circ C$</td>
<td>60 GPM</td>
</tr>
</tbody>
</table>

Table 4.3: Feedforward Cooling Parameters
Figure 4.10: Preset Constant Cooling Control, with 30 GPM on the Ice Bath, and 60 GPM on the Fuel Cells. Here, the fuel cells heat up very slowly, due to the ice bath being run too quickly at lower fuel cell outlet temperatures. The fuel cells do not fully heat up until after the ice has run out, and the run ends shortly thereafter due to the fuel cell outlet temperature being too high. The lower right plot shows when the ice has run out by the ice bath outlet temperature starting to increase, at approximately 85 seconds here.
Figure 4.11: Feedforward Cooling Control. Using the values from Table 4.3, the cooling system begins to show the desired behavior for the cooling system. The fuel cells heat up quicker than before, and approach the desired outlet temperature, with some steady-state error. After the ice runs out, the ice bath pump increases the flow rate to maintain the temperature, until it cannot keep up, and the fuel cells must be shut down.
PID Control

The proportional, integral and derivative (PID) controller has been a staple of the control industry for decades. It originally existed as hardware with actual knobs that could be “tuned.” As the control industry transformed to microprocessors as the preferred method for control, the PID methods advanced as well. In this case, a PI controller was tested for use in the Buckeye Bullet 2 cooling system. Figure 4.12 shows the resulting outlet temperature using PI control. In this simulation, the PI controller is not “switched on” until the outlet temperature reaches 75 degrees, to prevent integrator “wind up” that might occur otherwise. This control does a satisfactory job of cooling the fuel cells, and maintaining their temperature values within an acceptable range.

PID with Feedforward Control

Since the feedforward worked relatively well, and the PI controller did as well, they are now combined to see if they can perform well together. Figure 4.13 shows the result of a PI controller with feedforward design on the ice bath cooling loop of the fuel cells. Keep in mind that the fuel cell loop is still running an open-loop control based on the current available from the fuel cells. This control method does a good job of letting the fuel cells heat up to their operating temperature quickly, then holding them at the operating temperature for as long as possible. It can be seen at the end that the ice bath loop ramps up because the ice is depleted and then must run at a higher flow rate to reject the same amount of energy into the water that is in the cooling loops. Figure 4.14 shows the code that was used in the model shown
Figure 4.12: PID Cooling Control, with $K_p = -3$ and $K_i = -1$. Here, the fuel cell outlet temperature eventually approaches the desired temperature, before the ice runs out and the flow rate increases to maintain the temperature. Having to tune the PID without any other elements to account for the nonlinearities of the system means that the best that could be done in tuning is with the oscillations shown in the top-left plot.
in Figure 4.2, and Table 4.3 shows the feedforward parameters that were used. This model uses Equation 4.5 to control the ice bath.

\[ V_{FC} = f_{\text{feedforward}}(T_{FC,\text{out}}) + K_{p,IB} \cdot \int (80 - T_{FC,\text{out}}) + K_{i,IB} \cdot (80 - T_{FC,\text{out}}) \]  

(4.5)

4.1.6 Cooling Software from 2007

As a point of reference, the code to operate the cooling system in 2007 was implemented in Labview for the Compact RIO. This code basically assigned a penalty function between 60°C and 70°C, ranging linearly from “off” to completely “on” between these two values. This served its purpose in the first year of the Buckeye Bullet 2 by allowing the system to be tested and not worry about the control code being at fault.

4.1.7 Cooling Software from 2008

Figure 4.15 shows the code that was implemented for Speed Week 2008. It closely matches the method used in Figure 4.14, with the addition of a function that turns everything completely on if the fuel cells overheat.

4.2 Power Management Development

The conversion of electrical to mechanical power is the most important process on the Buckeye Bullet 2. The goal, after all, is to turn as much electrical power into mechanical power, and put that on the ground. So, maximizing the power that is put onto the ground, without any component failure, is a difficult challenge. The
Figure 4.13: PID Cooling Control, with $K_p = -3$ and $K_i = -1$, with a feedforward component. Combining the strategies shown in Figures 4.11 and 4.12 gives the result shown here. In this case, the fuel cells approach the desired temperature more slowly, but in practical implementation this is required because some flow is necessary on the ice bath lines to keep the cold water at the heat exchanger, ready to transfer heat when necessary. In this case, the fuel cells maintain the desired temperature more closely, and make the run last as long as possible before the fuel cells have to be shut down.
Figure 4.14: PID Cooling Control with a feedforward component in Simulink

Figure 4.15: 2008 Cooling Control Code
following sections describe the main concept, and how it was applied to the Buckeye Bullet 2.

4.2.1 Torque Control

The basis for the power management in the Buckeye Bullet 2 is a torque based control. The inverter and motor controller use a torque reference signal, and the driver’s pedal position equates to a torque reference, so it makes sense to manage the power on the car as torque. Equation 4.6 shows the basic relationship between power, torque, and motor speed.

\[
\text{Power} = \tau \cdot \omega 
\]  

(4.6)

4.2.2 Maximization of Power

With added power comes added speed, so maximizing power is a major objective. Figure 4.16, from the work done by Buckeye Bullet team member Ben Sinsheimer [21], shows that as the vehicle speed increases, drag increases. Aerodynamic drag reduction serves to increase vehicle speed for any given power input.

4.2.3 2007 Control Strategy

To compare this new system, it is worth comparing it to the method that was used at Speed Week 2007. The 2007 system used a National Instruments Compact RIO for the control. This unit, while best suited for a laboratory application, or bench-top testing, was being used for embedded control and was therefore an inappropriate application for this hardware. It proved to be time-consuming to program, unreliable, and would occasionally warrant the unacceptable restart that home computer users had become accustomed to in the 90’s. Fortunately computers have improved since
Figure 4.16: Power Distribution Versus Vehicle Speed [21]

Figure 4.17: Sankey Diagram [21]
then, and it was completely unacceptable that this unit operated in such a poor manner, especially being marketed for controls applications. That said, much of 2007 was focused on just getting the controller to work, as opposed to refining and improving the control strategy itself. Since the code for 2007 was written in Labview for the Compact RIO, it made it difficult to analyze and improve, since the team typically works in a Simulink environment.

The control software from 2007 was tasked with looking at the driver’s accelerator pedal position input (torque request), and converting it to the appropriate current request to the fuel cell controller. The driver’s input is scaled to the “requested torque” signal. From Equation 4.6, the torque is converted to current by multiplying the motor speed and dividing by the nominal fuel cell voltage under load (in this case, 605 V). This is then saturated between 45 and 800 Amps, since there should always be a small quantity of current requested, and no more than 800 Amps. In this case, using a nominal voltage, rather than the actual voltage helped to simplify the control scheme, and provided an adequate estimate of the current that is necessary to satisfy the driver’s torque request.

Once the current has been requested to the fuel cell controller, it is tasked with bringing up the flow rates of the gasses, using the mass flow controllers, and returning a response of how much current it had available. This available current would never be higher than the amount requested, and it was based on the mass flow of heliox to the fuel cell stacks. The hydrogen flow is based on current drawn, so it will follow passively and does require active control. Next, the current available is converted back to a torque available, and compared with the requested torque from the driver. The
minimum is taken, so that if the driver isn’t requesting any torque, none is exerted by the motor.

### 4.2.4 2008 Control Strategy

With the switch to a Simulink programmed controller for 2008, the focus could shift from just getting the controller to work, to a focus of optimizing the control scheme specifically for the Buckeye Bullet 2. The general concept stayed the same for 2008 as it was in 2007, but there were a few noted revisions that helped improve the vehicle’s performance and reliability.

Figure 4.18 shows the current request scheme for Speed Week 2008. In this version, the motor speed comes in, and is converted to a maximum power that can be pulled from the motor, given that speed. Then, it is run through a software power limit, then multiplied by the accelerator pedal, thus giving the power requested by the driver. Using a current versus power curve (an adaptation of the polarization curve) for the fuel cells that has been derived through bench testing of the fuel cell modules. The code that follows merely performs a rate limit operation, which stops the current request from dropping too quickly. When driver goes to perform a shift, rather than the current request dropping to zero and shutting off the gas flow, which can take a bit of time to recover from, the current request drops slowly, allowing full power to be available as soon as the driver gets back on the accelerator. There is no limit to the positive rate, only the negative. Finally, this current request is limited to between 45 and 900 Amps, and sent as a CAN message to the fuel cell controller.
This current request is also represented in Equation 4.7. In this case, the power, as a function of motor speed, is multiplied by the accelerator pedal (as a percentage), and a current scaling factor $\alpha$.

$$I_{\text{command}} = P(\omega_{\text{motor}}) \cdot x_{\text{accelerator}} \cdot \alpha \quad (4.7)$$

Figure 4.19 shows what happens to the current available from the fuel cell controller. Of the current requested and current available, the minimum is taken (to give the driver the ultimate control over the torque to the motor), and is again transformed from a current request into a power request, using the inverse of the curve used in Figure 4.18. This power request is divided by the power max calculated in Figure 4.18, to give a power percentage. This is then scaled up to a 10V signal, multiplied by a reduction factor, and is sent to the inverter as a torque request. The reduction factor will be discussed later, as it is a protective measure to save components on the vehicle.

The maximum power equation is shown in Equation 4.8. This is calculated using a function that finds power capable from the inverter, as function of motor speed, and multiplies it by a maximum power scaling factor $\beta$. Equation 4.9 shows the calculation...
Figure 4.19: 2008 Inverter Control Scheme

for the inverter torque reference voltage to be sent to the inverter. Power as a function of the minimum current (request and available) is divided by the computed maximum power, to give a percentage of maximum power. This is multiplied by the reduction factor, and finally scaled up to 10 volts.

\[ P_{\text{max}} = P_{\text{inverter}}(\omega_{\text{motor}}) \cdot \beta \quad (4.8) \]

\[ V_{\text{inverter torque reference}} = 10 \cdot \left[ \frac{P(\min(I_{\text{available}}, I_{\text{Request}}))}{P_{\text{max}}} \right] \cdot C_{\text{Reduction}} \quad (4.9) \]

Torque Control Simulations

Figure 4.20 shows the desired operating relationship between motor speed, current request, and inverter command voltage. Up to 7500 RPM, the motor is in constant torque mode, then it switches to constant power mode. To allow higher power to be pulled in the constant power region, and yet not over torque the motor’s mechanical limits in the constant torque region, the inverter torque reference drops in the constant torque region, and then increases for the constant power region. This means that a
higher torque limit is entered as a preset in the inverter, and the supervisory controller must make sure that the actual torque limit is not exceeded.

4.3 Safety Systems

While the Buckeye Bullet 2 is designed as a high speed vehicle, its first priority is to be a safe vehicle. The design priority on the vehicle from the ground up was to make it a safe car first, and to secondarily make it a fast car. Virtually all considerations for all systems and components centered around increased safety or increased speed. That said, there are a number of systems within the umbrella of control that help to make the car safer and protect its systems from damage. This way, a fault is prevented from climbing up the fault tree and causing a severe failure. Examples of
systems that are protected in this manner are the shift lockout, high voltage system, gas delivery system, parachutes, and cut-accelerator technique.

4.3.1 Shift Lockout

The transmission used on the Buckeye Bullet 2 is a very lightweight racing transmission. It is enclosed by a magnesium case, with several custom-cut gears with months of lead-time to obtain. If the transmission were to become damaged, repairs or replacements would be costly and time consuming. With nearly 800 horsepower being sent through the transmission, and shifting designed to occur as quickly as possible, the risk of a damaged transmission is possible, despite being designed for racing. To help minimize this risk, the code shown in Figure 4.21 is added to the supervisory controller. The basic idea is that the controller is in charge of transmission shifts, when requested by the driver. It waits until both a shift is requested (by depressing either the upshift or downshift buttons) and there is adequate pressure on the clutch line to have disengaged the motor from the transmission. This simple measure helps to reduce stress on the transmission by only shifting when no power is going into the transmission. The transmission is electronically actuated using an air solenoid system with a compressed nitrogen tank providing the pressure to perform the shift.

4.3.2 High Voltage

With voltages of up to 1000 VDC, the high voltage bus on the Buckeye Bullet 2 is a major safety consideration. The high voltage bus begins at the ends of the fuel cell stacks, travels up the side of the vehicle, through the firewall, past the driver (outside
Figure 4.21: 2008 Shifting Control Scheme

the driver compartment), and up to the inverter at the front of the car. To ensure that this line is safe, from start to finish, there are contactors at each end. The first is inside the fuel cell module, and the last is in the inverter. The driver controls the high voltage to the inverter, and the fuel cell controller controls the contactors within the fuel cell module. Using the high voltage switch, the driver has the capability of manually disconnecting the power source from the inverter, thus stopping the accelerating of the motor and vehicle. If the driver turns the main switch off, the fuel cell controller is thus turned off, thereby opening the contactors within the fuel cell modules. If the driver is unable to perform the shutdown, emergency crews are able to shut off the high voltage system, and all other auxiliary systems, using the emergency power-off switch at the rear of the vehicle, mounted on the exterior within easy access by crew members.
4.3.3 Hydrogen and Heliox Tanks

Another point of concern with the vehicle safety is with the fuel and oxidizer that are both stored on-board. Hydrogen stored at over 5,000 psi can be unsafe due to the force that the pressure exerts on the walls of the tank. The hydrogen tank itself is wrapped with several inches of carbon fiber around an aluminum core tank to withstand the pressure. In addition, the tank solenoid is constructed to be a part of the tank. The solenoid itself is inside the tank, so in the event that something happens to the tank that it becomes disconnected, the solenoid shuts off the flow of hydrogen from inside the tank. The heliox tank is similar in that it constructed to hold the pressures, in this case up to about 3,000 psi. It is wrapped with fiberglass rather than carbon fiber to reduce costs. On these tanks, the solenoids are located just outside the tank as it exits to the piping.

All of the tank solenoids, hydrogen and heliox, are under the control of the driver. If he chooses to shut them off, they will close, overriding the controller. As an additional safety measure, the fuel cell controller serves as gatekeeper for the hydrogen tank. A shutoff will occur if a significant hydrogen leak or fire is detected, regardless of the driver’s input. Again, like the high voltage, if the emergency power-off switch is flipped by emergency crews, power will be cut to all of the tank solenoids. Except for the control logic to open the hydrogen tank from the fuel cell controller, the tank solenoid wiring is independent of electronic control.

4.3.4 Parachutes

Parachutes have long been a staple safety system in land speed racing. It is a low cost, lightweight, highly effective, system that works with a basic push-pull cable. It
can help right an unstable vehicle, doesn’t require traction, and can bring a vehicle to a slower, safer speed. For these reasons, the primary braking safety system on the Buckeye Bullet 2 is a parachute system, albeit with a few improvements. The system from 2007 which gave three parachutes that had a less than desired reliability was upgraded and completely redesigned for 2008. This new parachute release has proven to be much more reliable. The new system uses the traditional push-pull cable system, but adds to it an air solenoid system that can electronically pull the push-pull cable, thus releasing the parachutes. This helps to keep the driver’s hands on the steering wheel, and in the event that the electronic system is unsuccessful, the driver can still go to the push-pull mechanical actuator. This system is electrically isolated from the controller, except for using a common battery supply.

This is a safety improvement that was integrated into the BB2. In the event that the vehicle accelerometer detects unsafe lateral or vertical accelerations, the parachutes could be released to help stabilize the car.
4.3.5 Accelerator-cutting Protection

Circumstances can arise in a run where drawing less power can correct a situation or mitigate damage. By drawing less power, the fuel cells won’t heat as quickly, the inverter temperature will reduce, and certain systems can be spared. To stop potential problems from cascading, simply dialing back, or cutting the torque request to the inverter can help solve some of the problems. Figure 4.19 shows one addition that helps to keep the fuel cells from overheating and let the cooling system “catch up” if it has fallen behind in cooling the fuel cells. This protective measure was utilized at Speed Week 2008 when the control scheme for the ice bath failed to cool the fuel cells properly, shown in Figure 4.23. This is what happened in Run 7 (See Table 6.1). The driver noted that there was a “fade in power for a short while until it came back” at the end of the run, which is exactly as the system should have worked.
Figure 4.23: Fuel Cell Overtemperature with accelerator cutting
CHAPTER 5

INFORMATION PROCESSING AND ANALYSIS SYSTEMS

When the Buckeye Bullet 1 was designed and built, collecting data somewhat of an afterthought. At the Buckeye Bullet 2 design summit in April 2006, one of the areas for improvement that was identified was the need for better collection of data on the vehicle [18]. The intent was to design the Buckeye Bullet 2 as more of fully functioning research testbed, and a major component to doing any research is a reliable system of collecting data.

Accurate data can be used not only in post-processing, but in real-time, while the vehicle is running to make changes and provide valuable information to the driver as it is necessary. Thus, a robust data collection and analysis system fully contained within the vehicle is necessary to satisfy such a requirement.

5.1 Centralized Data System

The use of information on the vehicle in real-time requires controllers to have access to the information which they need. The simplest way to accomplish this is to have a centralized data system. This way, all devices have access to all of the information, and can use it as required.
5.2 Driver Feedback

The driver of any vehicle is the primary input to that control system, and this is the case with the Buckeye Bullet 2. The input from the driver is based on various types of feedback. Feedback to the driver can be steering response, gauges, vibrations, sight, or anything else that informs the driver of how the vehicle is operating. From all of this input the driver must keep the vehicle on the course, perform shift operations, and deploy the parachutes. That’s a lot of duties to perform in less than two minutes, all while traveling several hundred miles per hour. With so much to pay attention to, minimizing extraneous information to the driver is a priority. The goal is to tell the driver what he needs to know, and nothing more. That said, the driver needs information regarding

- When to shift: Motor RPM + Shift Lights
- How fast he’s going: Vehicle Speed
- What gear the vehicle is in: Gear Indicator

5.2.1 Bosch Racing Display

To deliver this information, a Bosch motorsports racing display (DDU4) is utilized [34]. This particular unit is CAN capable, so it can tie into the existing vehicle network, and can display up to twelve different “pages.” While this isn’t useful while racing on the track, it provides a quick, easy way to check various sensors while waiting in line, or in the pits. As an example, there is a brake page that can show the different brake pressures, making the process of bleeding the brakes much easier.
Figure 5.1: Driver display showing a no warning case on the left side. On the right is an “Out of Hydrogen” warning condition.

5.2.2 Minimalist Display

To accomplish this minimalist dash concept, the display shown on the left in Figure 5.1. It provides essential information, nothing more. On the right side of Figure 5.1 is a display that shows an example of a textual warning that can show up if necessary to alert the driver what may be going on with the vehicle, or if he needs to shut the vehicle down.

5.2.3 Steering Wheel

In addition to the display, the steering wheel also has only what is necessary to run the vehicle. It is also intended to keep the driver’s hands on the steering wheel throughout the entire run. This way, he is not reaching for switches on the dash, or parachute release levers while traveling down the track. As an added feature, if the steering wheel is disconnected, it acts as an emergency power-off and shuts down the vehicle. Figure 5.2 shows the steering wheel on the vehicle.
5.3 Data Analysis Methods

Once the data comes back from the track, a method of quickly analyzing it is important. The data comes in the form of logged CAN traffic. Some of this data is used by different controllers, some of it is broadcast merely to be received by the logger.

5.3.1 Data Analysis Toolkit

To keep the focus on analyzing the data, rather than just trying to extract it, a toolkit is created that simplifies that process. There are several parts of the toolkit that accomplish different tasks. There is the extraction process, in which data is renamed to appropriate values and put onto a single time axis. The names for each of the channels are based upon the CAN database file that is used to describe the CAN system to the various controllers. Then, any post-processing calculations are added to the data. This can be information such as calculated power, or calculated...
track distance. Then the data can be interpreted any number of ways. This may be in the form of “standard plot,” plots that the team has determined to be beneficial to understanding the vehicle quickly. Standard plots include:

- **Fuel Cell**
  - (1.1) Fuel Cells: Current Draw
  - (1.2) Fuel Cells: Power Draw
  - (1.3) Fuel Cells: Polarization Curve
  - (1.4) Fuel Cells: Stack Pressures
  - (1.5) Fuel Cells: Stack Delta Pressures
  - (1.6) Fuel Cells: Stack Inlet Pressures

- **Gas Delivery**
  - (2.1) Gas Delivery: Hydrogen High and Low Pressure
  - (2.2) Gas Delivery: Heliox High and Low Pressure
  - (2.3) Gas Delivery: Mass Flow Controllers

- **Cooling**
  - (3.1) Cooling: Fuel Cell Temperatures
  - (3.2) Cooling: Ice Bath Temperatures
  - (3.3) Cooling: Heat Exchanger
  - (3.4) Cooling: Control + Pressures

- **Drivetrain**
  - (4.1) Drivetrain: Current (Requested, Available, Drawn)
  - (4.2) Drivetrain: Voltage
  - (4.3) Drivetrain: Power (Requested, Available, Drawn)
  - (4.4) Drivetrain: Torque (Requested, Available, Drawn)
  - (4.5) Drivetrain: Motor (RPM, Temperature)
  - (4.6) Drivetrain: Ground Fault Monitor
  - (4.7) Drivetrain: Clutch Pressure, Gear, RPM

- **Suspension**
- (5.1) Suspension: Shock Travel
- (5.2) Suspension: Wheel Speeds
- (5.3) Suspension: Tire Growth

- Brakes
  - (6.1) Brakes: Brake Pressures (Front/Rear)
  - (6.2) Brakes: Boost Pressure

- Vehicle
  - (7.1) Vehicle: Battery Voltages
  - (7.2) Vehicle: GPS Data
  - (7.3) Vehicle: Speed vs. Distance and Time
  - (7.4) Vehicle: Driver Display

These standard plots may be viewed on-screen, or through the automatic reports generated, which will be discussed later.

**Graphical User Interface**

To make it easy to utilize all of the elements of the toolkit without too steep of a learning curve, a graphical user interface (GUI) is used. Figure 5.3 shows the interface. Data is processed by going to the file menu, and selecting the raw run data, which loads it into the memory space. Then, the user associates the data with a CAN database, which is interpreted and also put into the memory space. The user can then select the data rate that the information should be resampled at, then click to process the data. The data within the workspace is now the processed data, including any calculated variables. The data in the workspace can be saved to a file through the GUI so that it does not have to be reinterpreted again to view the data.
Figure 5.3: Main Data Analysis GUI
5.3.2 Automatic Report Generation

Once interpreted, the user can select to generate a report based on the data within the workspace. This assumes that certain elements of data are available, necessary to interpret the run, generate a cover sheet, and print the standard plots. Figure 5.4 shows the cover that is generated. Here, quick information can be gained about the run without having to revert to the standard plots. The intent of this report is to be printed off, or stored for quick viewing for those without the proper data analysis software. It is stored as a standard PDF document.

5.3.3 Use of Google Earth

A new method that is being tested for its effectiveness in the Buckeye Bullet 2 is to utilize Google Earth\(^3\), a free publicly available program that associates data with coordinates on the globe. With runs taking place on the same track, in similar locations, it makes sense to view the data in this manner. This method is good for viewing event-based information, with some plotted-style information. For example, it can show when the brakes were first applied, or when a shift takes place. Using work developed as part of the Google Earth Toolbox\(^4\), this was expanded upon to apply to the Buckeye Bullet toolkit. Figure 5.5 shows information generated from actual run data.

\(^3\)http://earth.google.com

\(^4\)http://code.google.com/p/googleearthtoolbox/
Figure 5.5: Sample Data Analyzed in Google Earth
CHAPTER 6

CONTROL SYSTEM VALIDATION AND IMPLEMENTATION

This chapter describes the results of putting the previously mentioned control schemes onto the Buckeye Bullet 2.

6.1 2008 Bonneville Speed Week Run Summary

From August 19-23, 2008 the Buckeye Bullet 2 was run on the Bonneville Salt Flats in Utah. Table 6.1 shows the major runs that took place. The first run was shut down after 224 mph due to a faulty emergency power-off switch, which went undetected, then found after Run 3. Run 2 was a similar problem to Run 1, but the car only got up to 39 mph. Run 3 was showing great promise, but the parachutes were deployed prematurely by the driver due to the close proximity of the shift button to the parachute release button on the steering wheel. Run 4 was the most successful run that week, with a peak speed recorded of 297 mph (from the GPS receiver). Following Run 4, the torque limit was increased on the inverter to increase the power drawn from the fuel cells. The following three runs (Runs 5-7) each shut down due to an over-current alarm in the inverter, which trips at 1000 A. Even though the current values are not too close to this, transients can go above this level and cause a trip of
the alarm. Figure 6.1 shows the timing slip from that run, with mile-average speed of 280 mph, and an exit speed of over 286 mph. To summarize the week, Figure 6.2 shows the recorded speeds of each of the seven runs listed in Table 6.1 versus track distance.

<table>
<thead>
<tr>
<th>Run No.</th>
<th>Date</th>
<th>Peak Speed</th>
<th>Peak Power</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>August 19</td>
<td>224 mph</td>
<td>553 kW</td>
<td>Shutdown from faulty power-off switch</td>
</tr>
<tr>
<td>2</td>
<td>August 20</td>
<td>39 mph</td>
<td>178 kW</td>
<td>Shutdown from faulty power-off switch</td>
</tr>
<tr>
<td>3</td>
<td>August 21</td>
<td>227 mph</td>
<td>550 kW</td>
<td>Early parachute deployment</td>
</tr>
<tr>
<td>4</td>
<td>August 22</td>
<td>297 mph</td>
<td>560 kW</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>August 22</td>
<td>233 mph</td>
<td>590 kW</td>
<td>Shutdown from Inverter over-current</td>
</tr>
<tr>
<td>6</td>
<td>August 22</td>
<td>283 mph</td>
<td>589 kW</td>
<td>Shutdown</td>
</tr>
<tr>
<td>7</td>
<td>August 23</td>
<td>286 mph</td>
<td>588 kW</td>
<td>Shutdown</td>
</tr>
</tbody>
</table>

Table 6.1: Bonneville Speed Week 2008 Run Summary

### 6.2 Cooling System

A number of limitations in a test-bench environment didn’t allow full testing on the cooling system of the Buckeye Bullet 2, so some of the first trials of the system were at Speed Week 2008.

#### 6.2.1 Fuel Cell Loop

Figure 6.3 shows the fuel cell cooling for Run 4. This plot shows the cooling pump command ramping up with current available, and holding at full speed throughout most of the run. To verify the flow, the pressure sensors for both module A and
Figure 6.1: Buckeye Bullet Timing Slip from Run 4 in August 2008 (See Table 6.1)

Figure 6.2: Speed Versus Distance Run Summary (See Table 6.1)
6.2.2 Ice Bath Loop

Figure 6.3 also shows the performance of the ice bath loop. The flow can be seen ramping up with the fuel cell outlet temperature measurement. Figure 6.4 shows the ice bath pump command versus fuel cell outlet temperature. From this, it can be seen that after Run 4, the feedforward parameters were changed to allow the fuel cell to heat up more quickly. This reduction proved to cause problems as the accelerator
Figure 6.4: Ice Bath Cooling State Diagram for all runs at the Salt Flats (See Table 6.1)

cutback protection measure had to be used to prevent overheating of the fuel cells, shown in Figure 4.23.

6.3 Power Management

The power management system was also very well tested at Speed Week 2008. Figures 6.5-6.10 show the power versus motor RPM for the various runs at Speed Week 2008, from Table 6.1. Run 2 is not shown because it did not enough speed data to provide useful feedback about the power management system.

Figure 6.5 shows a good looking power curve that peaks at just over 8000 RPM. After the shutdown due to the faulty power-off switch, the driver went through each of the six gears to ensure that the shifting system was working well. This is why all of the gears are shown, many with little to no power drawn associated with it.
Figure 6.5: Power versus RPM for Run 1 at the Salt Flats (See Table 6.1)
Figure 6.6: Power versus RPM for Run 3 at the Salt Flats (See Table 6.1)

Figure 6.6 shows the result from Run 3. This is the first view of different power amounts being drawn in different gears.

Figure 6.7 shows the result from Run 4, the most successful run of the week. In this plot, the power bands look similar, but with varying maximums in each gear. The objective is to keep the vehicle in the highest power possible, and maximize the area underneath the power curve. This objective was met relatively well in this run.

Figure 6.8 shows Run 5, where the torque limit was turned up on the inverter. This can easily be seen, as the power levels are greater than they had been on previous runs. A jump in power once it reaches the region of constant power can be seen here.
Figure 6.7: Power versus RPM for Run 4 at the Salt Flats (See Table 6.1)
Figure 6.8: Power versus RPM for Run 5 at the Salt Flats (See Table 6.1)

as well, as this was done to avoid over-torque on the motor shaft. This run was shut down due to an over-current alarm from the inverter. Figure 6.9 and 6.10 each show similar runs, where the inverter shut down due to over-current alarms.

Figure 6.11 shows the result of fuel cell current drawn and the ideal request, versus motor RPM. Run 4 shows as well below the amount of current being requested, because of the reduced torque limit on the inverter. Run 7 comes closer in the peak power range to using the total current available from the fuel cells. In this plot, the current draw in the constant-torque region of the motor indicates that both runs
Figure 6.9: Power versus RPM for Run 6 at the Salt Flats (See Table 6.1)
Figure 6.10: Power versus RPM for Run 7 at the Salt Flats (See Table 6.1)
Figure 6.11: Current versus RPM for Runs 4 and 7 at the Salt Flats (See Table 6.1) compared to the ideal current
drew approximately the same torque, with Run 7 increasing in power once entering the constant-power region of the motor.
CHAPTER 7

CONCLUSION AND FUTURE WORK

7.1 Conclusions

This thesis explored the development of a control system optimized for land speed racing a fuel cell electric vehicle, The Buckeye Bullet 2.

The Buckeye Bullet 2 is the second land speed vehicle developed at The Ohio State University. The first, from which many lessons were learned, was the Buckeye Bullet 1, a battery-powered land speed vehicle. Using knowledge gained from this vehicle, and carrying over the motor and motor controller technology, the Buckeye Bullet 2 was developed from 2004 until the end of 2006. One area of improvement made was in the area of data collection. The Buckeye Bullet 1 had less than 10 channels of data, sampled at 10 Hz. Fabrication of the Buckeye Bullet 2 began in December of 2006. Since then, the Buckeye Bullet 2 has been continually improved and modified in the attempt to achieve the goal of surpassing the records held by the Buckeye Bullet 1.

The control system of the Buckeye Bullet 2 has a supervisory architecture, in which all of the controllers on the vehicle receive their inputs from a main vehicle controller. Using this method, the driver’s inputs to the vehicle can take precedence, and power can be maximized to ultimately increase the top speed of the vehicle. This
sophisticated control cannot take place without a network of sensors that assist in real-time control, as well as feedback to make improvements from run to run. Pressure, voltage, current, position, speed, temperature, and flow rates are all important to examine what is happening on the vehicle as it is running down the track. By structuring the information in layered networks, information needed by a controller can be quickly conveyed to it.

There are two primary control systems developed to be within the supervisory controller. The first is the cooling system. As the fuel cells heat up due to waste heat generated from the chemical conversion process, they are to be held at 80°C, and heat must be rejected from the system in order to maintain this temperature. This is accomplished with an ice bath, putting the thermal energy into the latent heat of fusion of the ice. Two pumps, one for the fuel cell side and one for the ice bath side are used to cool the system. The command for the fuel cell side is based on the current available from the fuel cell controller, to maintain proper pressure on the membranes of the fuel cell modules. The ice bath command depends on the fuel cell outlet temperature. Using a switched proportional-integrative controller with feedforward control, the proper temperature is reached, and maintained for as long as the capacity of the system allows, even after exhausting the on-board supply of ice.

The second main control system within the supervisory controller is the power management between the fuel cell and motor controllers. Using the driver’s accelerator pedal as a torque request, the appropriate current that would be required to generate the requested torque is calculated and sent as a request to the fuel cell controller. The controller returns with the amount of current actually available to be
pulled by the motor controller. This is translated back into a torque available, and
sent as a torque command to the inverter, relative to the torque limit set on the
inverter. This rule-based control technique operates continuously throughout the run
to maximize the power, and ultimately speed that is achieved by the vehicle within
the limits of the course.

There are also a number of electronic safety systems on the vehicle. There is a
shift lockout to protect the transmission. A number of high voltage contactors prevent
high voltage from being available to the motor, or even the high voltage cables. The
hydrogen and heliox tanks default to a safe, closed mode if their solenoids are not
energized, defined by both the driver and the fuel cell controller. The parachute
system works with the traditional mechanical methods, with a redundant electronic
system to allow the driver to keep both hands on the steering wheel throughout
the run. Finally, if components begin to exceed boundaries that might be aided by
reducing power to the motor, the accelerator is scaled back to allow the components
to come back into range, such as the fuel cells overheating. These measures help to
make the Buckeye Bullet 2 a safer vehicle to travel down the track.

Information systems that use the data collected on the vehicle help to make the
vehicle easier to diagnose, improve, and optimize. In real-time, the driver’s display
shows the minimum information necessary to let the driver operate the vehicle. If a
condition arises that the driver should be alerted of, then it appears and is presented
to the driver as a textual display. The driver’s hands can remain on the steering wheel
throughout the run with all of his controls at his fingertips. To analyze data that
comes from the vehicle after a run, a toolkit is developed specifically for the Buckeye
Bullet 2. This toolkit, developed in Matlab, includes software to align the data to a
single time axis, perform standard calculations after a run (such a computing power),
generate a run report document, and other new data analysis methods to help get
the vehicle back to the track sooner and make improvement more quickly.

Finally, the control systems explored were evaluated as they ran on the Bonneville
Salt Flats, at Speed Week 2008. In all, the systems performed as they were designed.
Even the accelerator cutting safety measure helped prevent the fuel cells from over-
heating. The cooling control showed good first steps in being a reliable control system
to keep the fuel cells cooled, and the power management worked just as expected.
The Buckeye Bullet 2 increased the top speed that a hydrogen powered vehicle has
ever achieved to 297 mph. The fastest “flying mile” speed was increased to 280 mph,
and the exit speed to 287 mph. The highest power produced at the salt flats by the
Buckeye Bullet team was increased to 590 kW.

The goal of the Buckeye Bullet 2 from the beginning was to create a fast, safe, fuel
cell electric land speed vehicle. Thanks to the onboard control systems, prototype
development methods, and on and off-board information systems, this was accom-
plished in under a year, and brought to nearly its intended design within less than
two years.

7.2 Future Work

In land speed racing, the work is never done because a vehicle can always go faster.
With that said, there is room for continued improvement in the Buckeye Bullet 2.
From the work performed in this thesis, a few areas for advancement are outlined here.
The full output power of the fuel cells has yet to be achieved on the Buckeye Bullet 2. Through further testing and optimization, the power able to be drawn by the inverter can be maximized. To aide this, there is room for improvement within the control to maximize the average power.

There is further optimization that can occur with the cooling system as well. Using newer, more advanced fuel cell membranes that run more efficiently can reduce the waste heat generated, and thus reduce the cooling capacity that is required to be carried on-board. This can translated to lower vehicle weight and thus improved speed.

Significant advancements in battery technology have been made since 2004 when the Buckeye Bullet 1 made its last run. It may be worthwhile to revisit the possibility of a battery vehicle to improve the power density of the vehicle, largely by dropping weight.

There is also room for improvement to the analysis toolkit that has been started. The capability for observers and sensor fusion with the data has yet to be explored, and it may help reveal important information about the dynamics of the vehicle. The framework that was laid out with the toolkit has helped to analyze data quickly, but there is room to further develop the visualizations which make it easiest to diagnose and improve the vehicle.
BIBLIOGRAPHY


[22] E. Hillstrom, K. Ponziani, B. Sinsheimer, G. Rizzoni and M. Procter, *Back with a bang. Three years after becoming the fastest electric vehicle in the world, the Buckeye Bullet return to Bonneville to break the fuel cell record*, Electric and Hybrid Vehicle Technology, pp. 116-120, December 2007.


APPENDIX A

BUCKEYE BULLET 2 DATA TOOLKIT FOR MATLAB

A.1 Graphical User Interface Callbacks

```matlab
function varargout = RunmeV2(varargin)
% RunmeV2 M-file for RunmeV2.fig
% RunmeV2, by itself, creates a new RunmeV2 or raises the existing
% singleton.
% H = RunmeV2 returns the handle to a new RunmeV2 or the handle to
% the existing singleton.
% RunmeV2('CALLBACK',hObject,eventData,handles,...) calls the local
% function named CALLBACK in RunmeV2.M with the given input
% arguments.
% RunmeV2('Property','Value',...) creates a new RunmeV2 or raises
% the existing singleton. Starting from the left, property value
% pairs are applied to the GUI before RunmeV2_OpeningFcn gets
% called. An unrecognized property name or invalid value makes
% property application stop. All inputs are passed to
% RunmeV2_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 RunmeV2
% Last Modified by GUIDE v2.5 09-Aug-2008 15:21:52
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name', mfilename, ...
    'gui_Singleton', gui_Singleton, ...
```
OpeningFcn = @RunmeV2OpeningFcn, ...
'gui_OutputFcn', @RunmeV2OutputFcn, ...
'gui_LayoutFcn', [], ...
'gui_Callback', []);

if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end

% End initialization code -- DO NOT EDIT

% —— Executes just before RunmeV2 is made visible.

function RunmeV2OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.

% hObject handle to figure
% eventdata reserved to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
% varargin command line arguments to RunmeV2 (see VARARGIN)

% Choose default command line output for RunmeV2
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

if (~isdeployed)
    addpath('SubFunctions')
    addpath('MainFunctions')
    addpath('googleearthtoolbox')
end

evalin('base','clear all')

% PDF_Logo = imread('PDFlogo.png');
% GEarthicon = imread('GEarthicon.png');
% MATIcon = imread('MATIcon.png');

BB2_Photo = imread('BB2onSalt.png');
axes(handles.axes1);
image(BB2_Photo); axis equal; axis off;
axes(handles.axes4); axis off;

BB2_Logo = imread('BB2Logo_ConW.bmp');
axes(handles.axes3);
image(BB2_Logo); axis equal; axis off;
% UIWAIT makes RunmeV2 wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% —— Outputs from this function are returned to the command line.
function varargout = RunmeV2_OutputFcn(hObject, eventdata, handles)
% varargout cell array for returning output args (see VARARGOUT);
% hObject handle to figure
% eventdata reserved -- to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

% —— Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject handle to pushbutton1 (see GCBO)
% eventdata reserved -- to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
[filename] = uigetfile('*.mat','Select Buckeye Bullet 2 Run File');
evalin('base', ['load ' filename]);

% —— Executes on button press in OpenFigure Button.
function OpenFigure_Button_Callback(hObject, eventdata, handles)
% hObject handle to OpenFigure Button (see GCBO)
% eventdata reserved -- to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
if evalin('base', 'exist(''t'')')
    val = get(handles.StandardPlotsDropdown,'Value');
    standardplots = sort(BB2_StandardPlotsV2(0));
    plotnumber = standardplots(val-1);
    fig = figure(plotnumber); clf; hold on; grid on;
    set(fig,'visible','on')
    BB2_StandardPlotsV2(plotnumber)
    pageformat(gcf,'screen')
else
    msgbox('Please Process Run Data','Need to Process Run Data','warn')
end

% —— Executes on selection change in StandardPlotsDropdown.
function StandardPlotsDropdown_Callback(hObject, eventdata, handles)
% hObject handle to StandardPlotsDropdown (see GCBO)
% eventdata reserved -- to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
% Hints: contents = get(hObject,'String')
% returns StandardPlotsDropdown contents as cell array

114
% contents{get(hObject,'Value')}
% returns selected item from StandardPlots_Dropdown
% val = get(hObject,'Value');
% standardplots = sort(BB2_StandardPlotsV2(0));
% switch val
%   case 1
%     BB2_Phot = imread('BB2onSalt.png');
%     axes(handles.axes1); clf;
%     image(BB2_Phot); axis equal; axis off;
%   case 2
%     axes(handles.axes1); clf;
%     BB2_StandardPlotsV2(handles.axes1,standardplots(val-1));
%     grid on;
%   case 3
%     plot(handles.axes1,evalin('base','t'),evalin('base','I_Avail'));
%     axes(handles.axes1); grid on;
%     xlabel('Time (s)'); ylabel('Current (A)');
%   case 4
%     plot(handles.axes1,evalin('base','t'),evalin('base','RPM')/1000);
%     axes(handles.axes1); grid on;
%     xlabel('Time (s)'); ylabel('RPM');
% end

% --- Executes during object creation, after setting all properties.
function StandardPlots_Dropdown_CreateFcn(hObject, eventdata, handles)
% hObject handle to StandardPlots_Dropdown (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),...
  get(0,'defaultUicontrolBackgroundColor'))
  set(hObject,'BackgroundColor','white');
end

function edit1_Callback(hObject, eventdata, handles)
% hObject handle to edit1 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text
% str2double(get(hObject,'String')) returns contents of edit1 as a
% double
% —— Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
    % hObject handle to edit1 (see GCBO)
    % eventdata reserved — to be defined in a future version of MATLAB
    % handles empty — handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),...
    get(0,'defaultUicontrolBackgroundColor'))
   )set(hObject,'BackgroundColor','white');
end

% —— Executes on button press in Process_Data_Button.
function Process_Data_Button_Callback(hObject, eventdata, handles)
    % hObject handle to Process_Data_Button (see GCBO)
    % eventdata reserved — to be defined in a future version of MATLAB
    % handles structure with handles and user data (see GUIDATA)
    if evalin('base','exist(''CAN_IDS''))
        h = waitbar(0,'Processing...');
        TimeStart = num2str(get(handles.TimeStart,'String'))
        TimeEnd = num2str(get(handles.TimeStop,'String'));
        contents = get(handles.DataRateDropdown,'String');
        rate = ...
        num2str(1/str2num(contents{get(handles.DataRateDropdown,'Value')}));
        evalin('base',['t = ' num2str(TimeStart) ':' rate ':' ...
        num2str(TimeEnd) '];')
        evalin('base','BB2_DataProcess')
        evalin('base','BB2_Calculations')
        vars = evalin('base','who');
        %findt = evalin('base',find('t'))
        set(handles.Xaxis,'String',vars);
        set(handles.Yaxis1,'String',vars);
        set(handles.Yaxis2,'String',vars);
        %evalin('base',['t = t-min(t);'])

        waitbar(1,h); close(h);
    else
        msgbox('Please Load Run Data, then CAN Database',...
            'Load CAN Data','warn')
    end
function edit2_Callback(hObject, eventdata, handles)
  % hObject    handle to edit2 (see GCBO)
  % eventdata reserved -- to be defined in a future version of MATLAB
  % handles    structure with handles and user data (see GUIDATA)

  % Hints: get(hObject,'String') returns contents of edit2 as text
  %       str2double(get(hObject,'String')) returns contents of edit2 as a
  %       double

  % --- Executes during object creation, after setting all properties.
  function edit2_CreateFcn(hObject, eventdata, handles)
    % hObject    handle to edit2 (see GCBO)
    % eventdata reserved -- to be defined in a future version of MATLAB
    % handles    empty -- handles not created until after all CreateFcns
    % called

    % Hint: edit controls usually have a white background on Windows.
    %       See ISPC and COMPUTER.
    if ispc && isequal(get(hObject,'BackgroundColor'),
      get(0,'defaultUicontrolBackgroundColor'))
      set(hObject,'BackgroundColor','white');
    end

  % _______________________________________________________
  function Open_Run_Data_Callback(hObject, eventdata, handles)
    % hObject    handle to Open_Run_Data (see GCBO)
    % eventdata reserved -- to be defined in a future version of MATLAB
    % handles    structure with handles and user data (see GUIDATA)
    [filename pathname] = uigetfile('*.mat',...
      'Select Buckeye Bullet 2 Run File');
    if filename
      evalin('base','clear all')
      h = waitbar(0,'Opening...');
      evalin('base',['load(''' pathname filename ''')']);
      evalin('base','clear header');
      evalin('base','vars = who;');
      waitbar(1,h); close(h);
    end

  % _______________________________________________________
  function Untitled_1_Callback(hObject, eventdata, handles)
    % hObject    handle to Untitled_1 (see GCBO)
    % eventdata reserved -- to be defined in a future version of MATLAB
    % handles    structure with handles and user data (see GUIDATA)
function edit4_Callback(hObject, eventdata, handles)
    % hObject handle to edit4 (see GCBO)
    % eventdata reserved — to be defined in a future version of MATLAB
    % handles structure with handles and user data (see GUIDATA)
    % Hints: get(hObject,'String') returns contents of edit4 as text
    % str2double(get(hObject,'String')) returns contents of edit4 as a double
    % --- Executes during object creation, after setting all properties.
    function edit4_CreateFcn(hObject, eventdata, handles)
        % hObject handle to edit4 (see GCBO)
        % eventdata reserved — to be defined in a future version of MATLAB
        % handles empty — handles not created until after all CreateFcns called
        % Hint: edit controls usually have a white background on Windows.
        % See ISPC and COMPUTER.
        if ispc && isequal(get(hObject,'BackgroundColor'),...
            get(0,'defaultUicontrolBackgroundColor'))
            set(hObject,'BackgroundColor','white');
        end
    end
    _________________________________
    function InstructionsMessage_Callback(hObject, eventdata, handles)
        % hObject handle to InstructionsMessage (see GCBO)
        % eventdata reserved — to be defined in a future version of MATLAB
        % handles structure with handles and user data (see GUIDATA)
        Message = {...
            '1. Open Data: File -> Open Run Data (Or Ctrl-F)',...
            '2. Associate CAN Database: File -> Load CAN Database',...
            '3. Select Data Resample Rate in "Prepare Data" box',...
            '4. Process Data: Click "Process Data"',...
            '* Data is now ready to be analyzed. You can...',
            '* Quickly view data in the "Standard Plots Menu",...
            '* Export converted data to a *.mat file to look at later',...
            '* Generate a "Run Summary" document, suitable for printing',...
            '*',
            'Good luck, and GO FAST!
        };
        msgbox(Message,'Buckeye Bullet 2: Runme Instructions','help')
    _________________________________
    function About_Dialog_Callback(hObject, eventdata, handles)
        % hObject handle to About_Dialog (see GCBO)
        % eventdata reserved — to be defined in a future version of MATLAB
339 % handles structure with handles and user data (see GUIDATA)
OSUicon = imread('OSUicon.png');
340
341 msgbox({'Buckeye Bullet 2', 'Runme Version 2.0', 'Developed July 2008', ...
'http://www.buckeyebullet.com'}, 'Buckeye Bullet 2', 'custom', OSUicon)
342
343 % function Untitled_2_Callback(hObject, eventdata, handles)
344 function Untitled_2_Callback(hObject, eventdata, handles)
345 % hObject handle to Untitled_2 (see GCBO)
346 % eventdata reserved — to be defined in a future version of MATLAB
347 % handles structure with handles and user data (see GUIDATA)
348
349 % --- Executes on button press in Generate_PDF.
350 function Generate_PDF_Callback(hObject, eventdata, handles)
351 % hObject handle to Generate_PDF (see GCBO)
352 % eventdata reserved — to be defined in a future version of MATLAB
353 % handles structure with handles and user data (see GUIDATA)
354 h = waitbar(0, 'Printing to PDF...');
355 evalin('base', 'SensorDummy')
356 evalin('base', 'Speed = RPM*169.3/9500;')
357 waitbar(.33, h);
358 evalin('base', 'BB2_Coverpage')
359 waitbar(.66, h);
360 standardplots = BB2_StandardPlotsV2(0);
361 for plotnumber = standardplots
362    fig = figure(plotnumber); clf; hold on; grid on;
363    set(fig, 'visible', 'off')
364    format = evalin('base', ...
365        ['BB2_StandardPlotsV2(' num2str(plotnumber) ');']);
366    switch format
367        case 1
368            pageformat(fig, 'landscape')
369        case 2
370            pageformat(fig, 'portrait')
371        end
372    print(gcf, 'RunData.ps', '-dpsc2', '-append');
373    close(gcf)
374 end
375 waitbar(1, h); close(h);
376 close all
377
378 % --- Executes when figure1 is resized.
379 function figure1_ResizeFcn(hObject, eventdata, handles)
380 % hObject handle to figure1 (see GCBO)
381 % eventdata reserved — to be defined in a future version of MATLAB
382 % handles structure with handles and user data (see GUIDATA)
383 % set(gcbo, 'Units', 'pixels');
384    figpos = get(gcbo, 'Position');
385
function Menu_Exit_Callback(hObject, eventdata, handles)
    % hObject handle to Menu_Exit (see GCBO)
    % eventdata reserved - to be defined in a future version of MATLAB
    % handles structure with handles and user data (see GUIDATA)
    close all;

    % --- Executes on button press in Generate_KML.
    function Generate_KML_Callback(hObject, eventdata, handles)
        % hObject handle to Generate_KML (see GCBO)
        % eventdata reserved - to be defined in a future version of MATLAB
        % handles structure with handles and user data (see GUIDATA)
        GE_Data = struct(
            'Longitude',1:301,...
            'Latitude',1:301,...
            'Speed',evalin('base','DisplayP1_Vehicle_Speed'),...
            'RPM',evalin('base','DisplayP1_Motor_RPM'),...
            'Throttle',1:301,...
            'CurrentAvail',evalin('base','pcm5977to6_allIavail'),...
            'Gear',zeros(1,301),...
            'MotorTemp',1:301...
        );
        BB2_GEart_KML(GE_Data);

        % --- Executes on selection change in DataRate_Dropdown.
        function DataRate_Dropdown_Callback(hObject, eventdata, handles)
            % hObject handle to DataRate_Dropdown (see GCBO)
        end
end
% eventdata reserved – to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
% Hints: contents = get(hObject,'String')
% returns DataRate_Dropdown contents as cell array
% contents{get(hObject,'Value')}
% returns selected item from DataRate_Dropdown

% —— Executes during object creation, after setting all properties.
function DataRate_Dropdown_CreateFcn(hObject, eventdata, handles)
% hObject handle to DataRate_Dropdown (see GCBO)
% eventdata reserved – to be defined in a future version of MATLAB
% handles empty – handles not created until after all CreateFcns called
% Hint: popupmenu controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),...
    get(0,'defaultUicontrolBackgroundColor'))
    get(0,'defaultUicontrolBackgroundColor')
    set(hObject,'BackgroundColor','white');
end

% —— Executes on button press in ExportMAT.
function ExportMAT_Callback(hObject, eventdata, handles)
% hObject handle to ExportMAT (see GCBO)
% eventdata reserved – to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
evalin('base','uisave')

% —— Executes on button press in Load_CANdb.
function Load_CANdb_Callback(hObject, eventdata, handles)
% hObject handle to Load_CANdb (see GCBO)
% eventdata reserved – to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
if evalin('base','exist("vars")')
    [filename pathname] = uigetfile(*.dbc,...
        'Select Buckeye Bullet 2 CAN Database');
evalin('base',[filename = "" pathame filename ""']);
    evalin('base',{'filename = "" filename ""'})
evalin('base','CAN_IDs = CAN_Decoder(filename);')
else
    msgbox('Please Load Run Data','Load Run Data','warn')
end

% —— Executes on button press in Clear Workspace.
function Clear_Workspace_Callback(hObject, eventdata, handles)
 evalin('base','clear all')

function Xaxis_Callback(hObject, eventdata, handles)

function Xaxis_CreateFcn(hObject, eventdata, handles)

function Yaxis1_Callback(hObject, eventdata, handles)

function Yaxis1_CreateFcn(hObject, eventdata, handles)
% Hint: popupmenu controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on selection change in Yaxis2.
function Yaxis2_Callback(hObject, eventdata, handles)
    % hObject handle to Yaxis2 (see GCBO)
% eventdata reserved — to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

    % Hints: contents = get(hObject,'String') returns Yaxis2 contents as cell
    % array
    % contents{get(hObject,'Value')} returns selected item from Yaxis2

% --- Executes during object creation, after setting all properties.
function Yaxis2_CreateFcn(hObject, eventdata, handles)
    % hObject handle to Yaxis2 (see GCBO)
% eventdata reserved — to be defined in a future version of MATLAB
% handles empty — handles not created until after all CreateFcns
% called

    % Hint: popupmenu controls usually have a white background on Windows.
    % See ISPC and COMPUTER.
    if ispc && isequal(get(hObject,'BackgroundColor'),
        get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
    end

% --- Executes on button press in PlotButton.
function PlotButton_Callback(hObject, eventdata, handles)
    % hObject handle to PlotButton (see GCBO)
% eventdata reserved — to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

    val = get(handles.Xaxis,'Value');
    var = get(handles.Xaxis,'String');
    X = evalin('base',char(var(val)));
    val = get(handles.Yaxis1,'Value');
    var = get(handles.Yaxis1,'String');
    Y1 = evalin('base',char(var(val)));
    val = get(handles.Yaxis2,'Value');
    var = get(handles.Yaxis2,'String');
    Y2 = evalin('base',char(var(val)))
val = get(handles.PlotStyle,'Value');
var = get(handles.PlotStyle,'String');
Style = char(var(val));

% [AX,H1,H2]=plotyy(X,Y1,X,Y2,'plot');
% set(H1,'LineStyle',Style);
% set(H2,'LineStyle',Style);
axes(handles.axes1);
plot(X,Y1,[Style 'k']); grid on;
axes(handles.axes4);
plot(X,Y2,[Style 'r']); %grid on;
set(handles.axes4,'YColor','r')
set(handles.axes4,'Color','none')
set(handles.axes4,'YAxisLocation','right')

function PlotStyle_Callback(hObject, eventdata, handles)
% hObject     handle to PlotStyle (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String')
% returns PlotStyle contents as cell array
% contents{get(hObject,'Value')}
% returns selected item from PlotStyle

function PlotStyle_CreateFcn(hObject, eventdata, handles)
% hObject     handle to PlotStyle (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      empty - handles not created until after all CreateFcns
called

% Hint: popupmenu controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),...
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function TimeStart_Callback(hObject, eventdata, handles)
% hObject     handle to TimeStart (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)
A.2  CAN Database Interpretation

1  function [CAN_IDS] = CAN_Decoder(filename)
2    %
file = fopen(filename,'r');
A = textscan(file,'%s','delimiter','
','whitespace',''); A = A{1};
fclose(file);
% clear file ans filename
%
% Find Messages
%Devicelist = A{35};
% Cell Array: ID, Signal Name,
m = 0;
for k = 1:length(A)
    temp = A{k};
    if length(A{k})>3&temp(1:4) =='BO_ ' % When you find a new Message...
        % Advance Message counter
        m = m+1;
        % Extract Message Information
        Message_ID = sscanf(temp,'%*s%d %*s ');
        Message_Name = char(sscanf(temp,'%*s%d %s %d %*s' ));
        Message_Name = Message_Name(1:end-1);
        % Write Message Information to Message Cell Array
        CAN_IDs{m,1} = Message_ID';
        CAN_IDs{m,2} = Message_Name';
        % Set Signal Index at message counter
        s = 0;
        temp2 = A{k+1};
        while length(temp2)>4&temp2(1:5) ==' SG_ '
            s = s+1;
            Signal{s} = char(sscanf(temp2,' %*s %s %*d '));
            Units_temp = char(sscanf(temp2,' %*s %*s %*s %*s %*s %*s %*s '));
            Units{s} = Units_temp(2:end-1);
            temp2 = A{k+1+s};
        end
        [Signal a] = sort(Signal);
        CAN_IDs{m,3} = Signal';
        CAN_IDs{m,4} = Units(a)';
        Signal = {' '}; Units = {' '};
    end
end
%clear temp
% Find Signals
% Cleanup
%clear A file k m s temp Signal_Name

A.3 Variable Conversion
function [newvar] = BB2_varscaling(timeaxis,oldvar)

% [newvar] = BB2_varscaling(timeaxis,oldvar)
%
% This function takes in oldvar, and converts it to newvar using the new
% time scale timeaxis, and erases oldvar from the workspace
oldvar = evalin('base',oldvar);
newvar = interp1(oldvar.time, oldvar.signals.values, timeaxis);
%evalin('base',[clear oldvar])

A.4 Data Processing

%for k = 1:length(vars)
% clear
% load('trcrun2008-07-19_19-43-30.mat');
% clear header
% vars = who;
% 
% addpath('SubFunctions')
% addpath('MainFunctions')
% filename = 'Bullet2Network.dbc';
% CAN_IDS = CAN_Decoder(filename);
% t = 200:500;

%%% Generate New Variables, with a single time scale
for k = 1:length(vars)
    variable = vars{ k };
    if length(variable) ≥ 11
        varlength = 11;
    else
        varlength = length(variable);
    end
    
    for m = 1:length(CAN_IDS)
        MessageName = CAN_IDS{m,2};
        if length(MessageName(1:end-1)) ≥ 12
            MessageShort = MessageName(1:12);
        else
            MessageShort = MessageName(1:end);
        end
        
        if variable(1:length(MessageShort))==MESSAGESHORT...
        &variable(length(MessageShort)+1)==',
            Signals = CAN_IDS{m,3};
            SignalName = char(Signals(str2num(variable(end-1:end))+1));
            evalin('base',
                [SignalName '=BB2_varscaling(t,’’ variable ‘’');']
            % Erase variable after it is converted

evalin('base', ['clear ' variable])
evalin('base', [SignalName '(isnan(' SignalName '))=[0];']);
end
end
% Unconverted variables will be left in the workspace
%
%% Clean Up Old Variables
clear vars header k m varlength variable filename
clear MessageName MessageShort Signals CAN IDs unknown

A.5 Post-Processing Calculations

vars = who;
for k = 1:length(vars)
    switch vars{ k}
    case 'MH_HighVoltage_DCVoltage'
        MH_HighVoltage_Power = MH_HighVoltage_DCVoltage.*...
        (MH_HighVoltage_CurrentA + MH_HighVoltage_CurrentB)/1000; % in kW
        MH_HighVoltage_Energy = ...
        cumtrapz(t,MH_HighVoltage_Power)/3600; % in kWh
    case 'MH_HighVoltage_CurrentA'
        MH_HighVoltage_CurrentTotal = ...
        (MH_HighVoltage_CurrentA + MH_HighVoltage_CurrentB);
    case 'GPS_VehicleSpeed'
        GPS_VehicleSpeed = unwrap(GPS_VehicleSpeed,100)/9;
        d = cumtrapz(t,GPS_VehicleSpeed)/3600; % In miles
    end
end

A.6 Standard Plots

function [plotlist]=BB2.StandardPlotsV2(standardplot)
% plotlist is used for returning either a list of the available plots, or
% whether the figure should be plotted in landscape (1) or portrait (2)

plotlist = 11; % Generate default value
t = evalin('base','t','zeros(1,length(t))'); % Bring in time variable
d = evalin('base','d','zeros(1,length(t))'); % Bring in distance variable
switch standardplot
% Fuel Cell Plots
% Give list of standard plots available


**case** 11 % Fuel Cells: Current Draw
I_R = evalin('base', ...
    'fr595_1to0_all_Request','zeros(1,length(t))');
I_A = evalin('base', ...
    'pcm597_1to0_all_Iavail','zeros(1,length(t))');
I_total = evalin('base', ...
    'MH_HighVoltage_CurrentTotal','zeros(1,length(t))');
I_F = evalin('base', ...
    'MH_HighVoltage_CurrentA','zeros(1,length(t))');
I_F = evalin('base', ...
    'MH_HighVoltage_CurrentB','zeros(1,length(t))');
plot(t,I_R,'k--')
plot(t,I_A,'k','LineWidth',2)
plot(t,I_total,'m','LineWidth',2)
plot(t,I_F,'r')
plot(t,I_F,'b')
xlabel('Time'); ylabel('Current (A)');
legend('Request','Available','Actual','FCA','FCB',1)
title('(1.1) Fuel Cells: Per Module Current Draw')
axis([t(min(find(I_A>0))) t(max(find(I_A>0))) 0 1000])
plotlist = 1;

**case** 12 % Fuel Cells: Power Draw
Voltage = evalin('base', ...
    'MH_HighVoltage_DC Voltage','zeros(1,length(t))');
I_R = evalin('base', ...
    'fr595_1to0_all_Request','zeros(1,length(t))');
I_A = evalin('base', ...
    'pcm597_1to0_all_Iavail','zeros(1,length(t))');
I_total = evalin('base', ...
    'MH_HighVoltage_CurrentTotal','zeros(1,length(t))');
I_F = evalin('base', ...
    'MH_HighVoltage_CurrentA','zeros(1,length(t))');
I_F = evalin('base', ...
    'MH_HighVoltage_CurrentB','zeros(1,length(t))');
plot(t,I_R.*Voltage/1000,'k--')
plot(t,I_A.*Voltage/1000,'k','LineWidth',2)
plot(t,I_total.*Voltage/1000,'m','LineWidth',2)
plot(t,I_F.*Voltage/1000,'r')
plot(t,I_F.*Voltage/1000,'b')
xlabel('Time'); ylabel('Power (kW)');
legend('Request','Available','Actual','FCA','FCB',1)
title('(1.2) Fuel Cells: Power Draw')
axis([t(min(find(I_A>0))) t(max(find(I_A>0))) 0 700])
plotlist = 1;

**case** 13 % Fuel Cells: Polarization Curve
I_F = evalin('base', ...
    'MH_HighVoltage_CurrentA','zeros(1,length(t))');
I_F = evalin('base', ...
    'MH_HighVoltage_CurrentB','zeros(1,length(t))');
StackVoltage = evalin('base',...
'MH_HighVoltage_DCVoltage','zeros(1,length(t))');
plot(I_FCA,StackVoltage,'r.'
plot(I_FCB,StackVoltage,'b.'
axis([0 600 500 1000]));
legend('Stack A','Stack B',1
xlabel('Stack Current'); ylabel('Stack Voltage');
title('(1.3) Fuel Cells: Polarization Curves')
plotlist = 1;

case 14 % Fuel Cells: Stack Pressures
PTA07A = evalin('base','RR405_PT_A07_A','zeros(1,length(t))');
PTA07B = evalin('base','RR407_PT_A07_B','zeros(1,length(t))');
PTH03A = evalin('base','RR405_PT_H03_A','zeros(1,length(t))');
PTH03B = evalin('base','RR405_PT_H03_B','zeros(1,length(t))');
PTH04A = evalin('base','RR406_PT_H04_A','zeros(1,length(t))');
PTH04B = evalin('base','RR406_PT_H04_B','zeros(1,length(t))');
subplot(2,1,1); hold on; grid on
plot(t,PTH03A,'b')
plot(t,PTA07A,'r')
%plot(t,[0 1],['k'])
xlabel('Time'); ylabel('Pressure (psi)');
legend('PT-H03-A','PT-A07-A',2)
title('(1.2.1) Fuel Cells: Stack A Pressures')
subplot(2,1,2); hold on; grid on
plot(t,PTH03B,'b')
plot(t,PTA07B,'r')
%plot(t,[0 1],['k'])
xlabel('Time'); ylabel('Pressure (psi)');
legend('PT-H03-B','PT-A07-B',2)
title('(1.2.2) Fuel Cells: Stack B Pressures')
plotlist = 2;

case 15 % Fuel Cells: Stack Delta Pressures
title('(1.5) Fuel Cells: Stack Delta Pressures')
plotlist = 2;

case 16 % Fuel Cells: Stack Inlet Pressures
%I_Req = evalin('base','I_Req');
plot([0 1],[0 1],['b'])
plot([0 1],[0 1],['r'])
legend('Stack A','Stack B',1
xlabel('Time'); ylabel('Pressure (psi)');
title('(1.4) Fuel Cells: Stack Inlet Pressures')
plotlist = 1;

H2High = evalin('base',...
H2LD = evalin('base',...'MH_GasPressures_H2High','zeros(1,length(t))');
H2stackinletA = evalin('base',...'MH_GasPressures_H2LD','zeros(1,length(t))');
H2stackinletB = evalin('base',...'RR405_PT_HO3_A','zeros(1,length(t))');
H2stackinletB = evalin('base',...'RR405_PT_HO3_B','zeros(1,length(t))');

subplot(3,1,1); hold on; grid on
plot(t,H2High,'b')
legend('Stack A','Stack B',1)
xlabel('Time'); ylabel('Pressure (psi)');
title('(2.1.1) Gas Delivery: Hydrogen Tank')

subplot(3,1,2); hold on; grid on
plot(t,H2LD,'b')
xlabel('Time'); ylabel('Pressure (psi)');
title('(2.1.2) Gas Delivery: Hydrogen Low Pressure')

subplot(3,1,3); hold on; grid on
plot(t,H2stackinletA,'r')
plot(t,H2stackinletB,'b')
xlabel('Time'); ylabel('Pressure (psi)');
title('(2.1.3) Gas Delivery: Hydrogen Stack Inlet')
plotlist = 2;

case 22 % Gas Delivery: Heliox
O2High = evalin('base',...'MH_GasPressures_O2High','zeros(1,length(t))');
O2L = evalin('base',...'MH_GasPressures_O2L','zeros(1,length(t))');
O2stackinletA = evalin('base',...'RR405_PT_HO3_A','zeros(1,length(t))');
O2stackinletB = evalin('base',...'RR405_PT_HO3_B','zeros(1,length(t))');

subplot(3,1,1); hold on; grid on
plot(t,O2High,'b')
xlabel('Time'); ylabel('Pressure (psi)');
title('(2.2.1) Gas Delivery: Heliox Tank')

subplot(3,1,2); hold on; grid on
plot(t,O2L,'b')
xlabel('Time'); ylabel('Pressure (psi)');
title('(2.2.2) Gas Delivery: Heliox Low Pressure')

subplot(3,1,3); hold on; grid on
plot(t,O2stackinletA,'r')
plot(t,O2stackinletB,'b')
xlabel('Time'); ylabel('Pressure (psi)');
title('(2.2.3) Gas Delivery: Heliox Stack Inlet')
plotlist = 2;

```matlab
% Gas Delivery: Mass Flow Controllers

case 23  % Gas Delivery: Mass Flow Controllers
    Temp_IBA_Out = evalin('base', ...
        'RR_Term1_Temp1', 'zeros(1,length(t))');
 subplot(2,1,1); hold on; grid on
    plot([0 1], [0 1], 'b')
    plot([0 1], [0 1], 'r')
 legend('Commanded (%)', 'Flow Feedback (slpm)', 1)
 xlabel('Time');
 title('(2.3.1) Gas Delivery: Mass Flow Controller A')
 subplot(2,1,2); hold on; grid on
    plot([0 1], [0 1], 'b')
    plot([0 1], [0 1], 'r')
 legend('Commanded (%)', 'Flow Feedback (slpm)', 1)
 xlabel('Time');
 title('(2.3.2) Gas Delivery: Mass Flow Controller B')
 plotlist = 2;
```

```matlab
%% Cooling =============================================================

% Cooling: Fuel Cell Temperatures

case 31  % Cooling: Fuel Cell Temperatures
    Temp_IBA_Out = evalin('base', ...
        'RR_Term1_Temp1', 'zeros(1,length(t))');
    Temp_PCA_In = evalin('base', ...
        'RR_Term1_Temp2', 'zeros(1,length(t))');
    Temp_PCB_Out = evalin('base', ...
        'RR_Term1_Temp3', 'zeros(1,length(t))');
    Temp_IBM_In = evalin('base', ...
        'RR_Term1_Temp4', 'zeros(1,length(t))');
    Temp_PCA_Out = evalin('base', ...
        'RR_Term2_Temp5', 'zeros(1,length(t))');
    Temp_IBA_In = evalin('base', ...
        'RR_Term2_Temp6', 'zeros(1,length(t))');
    Temp_PCB_IN = evalin('base', ...
        'RR_Term2_Temp7', 'zeros(1,length(t))');
    Temp_PCB_OUT1 = evalin('base', ...
        'RR_Term2_Temp8', 'zeros(1,length(t))');
 subplot(2,1,1);
 hold on; grid on
    plot(t, Temp_PCA_In, 'b')  % PC Inlet
    plot(t, Temp_PCB_Out, 'r')  % PC Outlet
    plot(t, Temp_PCA_Out, 'r--')  % PC Outlet
    plot(t, Temp_PCB_IN, 'b--')  % PC inlet
 legend('Inlet Temp', 'Outlet Temp', 'Delta Temp', 1)
 xlabel('Time (s)'); ylabel('Temperature (\degree C)')
 title('(3.1.1) Cooling: Fuel Cell A')
 axis([min(t) max(t) 0 90])
 subplot(2,1,2); hold on; grid on
```
plot(t,Temp_IBA_Out,'b')  % Ice bath outlet
plot(t,Temp_IBB_In,'r--') % IB outlet
plot(t,Temp_IBA_In,'r')  % IB Inlet
legend('Inlet Temp','Outlet Temp','Delta Temp',1)
xlabel('Time (s)'); ylabel('Temperature (\textdegree C)')
title('(3.1.2) Cooling: Fuel Cell B')
axis([min(t) max(t) 0 90])
plotlist = 2;

case 32 % Cooling: Ice Bath Temperatures
plotlist = 2;

case 33 % Cooling: Heat Exchanger
plotlist = 2;

case 34 % Cooling: Control + Pressures
MH_CoolingControl_FCThrottle = ...
evalin('base','MH_CoolingControl_FCThrottle','zeros(1,length(t))');
MH_CoolingControl_ICETHrottle = ...
evalin('base','MH_CoolingControl_ICETHrottle','zeros(1,length(t))');
MH_CoolingControl_FCEN = ...
evalin('base','MH_CoolingControl_FCEN','zeros(1,length(t))');
MH_CoolingControl_ICEN = ...
evalin('base','MH_CoolingControl_ICEN','zeros(1,length(t))');
subplot(2,1,1); hold on; grid on
plot(t,MH_CoolingControl_FCThrottle, 'b')
plot(t,MH_CoolingControl_ICETHrottle, 'r')
%plot(t,(MH_CoolingControl_FCEN+3)*.5,'b--')
%plot(t,(MH_CoolingControl_ICEN+4)*.5,'r--')
legend('Fuel Cell','Ice Bath',1)
xlabel('Time (s)'); ylabel('Control Voltage (V)')
title('(3.4.1) Cooling: Control')
axis([min(t) max(t) 0 6])

MH_CoolingP_PLeft = evalin('base',...  
'MH_CoolingP_PLeft','zeros(1,length(t))');
MH_CoolingP_PRight = evalin('base',...  
'MH_CoolingP_PRight','zeros(1,length(t))');
subplot(2,1,2); hold on; grid on
plot(t,MH_CoolingP_PLeft, 'b')
plot(t,MH_CoolingP_PRight, 'r')
legend('Left','Right',1)
xlabel('Time (s)'); ylabel('Pressure (psi)')
title('(3.4.2) Cooling: Pressures')
axis([min(t) max(t) 0 30])
plotlist = 2;

%% Drivetrain

133
case 41  % Drivetrain: Motor Performance
    Voltage = evalin('base', ...
        'MH_HighVoltage_DCVoltage', 'zeros(1,length(t))');
    I_total = evalin('base', ...
        'MH_HighVoltage_CurrentTotal', 'zeros(1,length(t))');
    RPM = evalin('base', ...
        'MH_DriveTrain_MotorRPM', 'zeros(1,length(t))');
    Gear = evalin('base', ...
        'MH_DriveTrain_TrannyGear', 'zeros(1,length(t))');
    Torque = I_total.*Voltage*60/(2*pi)./RPM;
    Power = I_total.*Voltage/1000;
    subplot(2,1,1); hold on; grid on;
    plot(RPM(find(Gear==1)),Torque(find(Gear==1)),'r.'),
    plot(RPM(find(Gear==2)),Torque(find(Gear==2)),'b.'),
    plot(RPM(find(Gear==3)),Torque(find(Gear==3)),'k.'),
    plot(RPM(find(Gear==4)),Torque(find(Gear==4)),'g.'),
    plot(RPM(find(Gear==5)),Torque(find(Gear==5)),'m.'),
    plot(RPM(find(Gear==6)),Torque(find(Gear==6)),'c.'),
    axis([0 12000 0 800]),
    legend('1st Gear','2nd Gear','3rd Gear',...
        '4th Gear','5th Gear','6th Gear',2)
    xlabel('Motor RPM'); ylabel ('Torque (Nm)')
    title('(4.1.1) Drivetrain: Motor Performance: Torque')
    subplot(2,1,2); hold on; grid on;
    plot(RPM(find(Gear==1)),Power(find(Gear==1)),'r.'),
    plot(RPM(find(Gear==2)),Power(find(Gear==2)),'b.'),
    plot(RPM(find(Gear==3)),Power(find(Gear==3)),'k.'),
    plot(RPM(find(Gear==4)),Power(find(Gear==4)),'g.'),
    plot(RPM(find(Gear==5)),Power(find(Gear==5)),'m.'),
    plot(RPM(find(Gear==6)),Power(find(Gear==6)),'c.'),
    axis([0 12000 0 700]),
    legend('1st Gear','2nd Gear','3rd Gear',...
        '4th Gear','5th Gear','6th Gear',2)
    xlabel('Motor RPM'); ylabel ('Power (kW)')
    title('(4.1.2) Drivetrain: Motor Performance: Power')
    plotlist = 2;

case 42  % Drivetrain: Voltage
    title('(4.2) Drivetrain: Voltage')
    plotlist = 2;

case 43  % Drivetrain: Power (Request, Avail, Drawn)
    title('(4.3) Drivetrain: Power (Request, Avail, Drawn)')
    plotlist = 2;

case 44  % Drivetrain: Torque (Request, Avail, Drawn)
    I_Request = evalin('base', ...
        'fr595_lto0_all.IRequest', 'zeros(1,length(t))');
    I_Avail = evalin('base', ...
I_total = evalin('base', ...
'MH_HighVoltage_CurrentTotal', 'zeros(1, length(t))');
I_FCA = evalin('base', ...
'MH_HighVoltage_CurrentA', 'zeros(1, length(t))');
I_FCB = evalin('base', ...
'MH_HighVoltage_CurrentB', 'zeros(1, length(t))');
Voltage = evalin('base', ...
'MH_HighVoltage_DCVoltage', 'zeros(1, length(t))');
RPM = evalin('base', ...
'MH_DriveTrain_MotorRPM', 'zeros(1, length(t))');
plot(t, I_Request.*Voltage*60/(2*pi)./RPM, 'k--')
plot(t, I_Available.*Voltage*60/(2*pi)./RPM, 'k', 'LineWidth', 2)
plot(t, I_Total.*Voltage*60/(2*pi)./RPM, 'm', 'LineWidth', 2)
plot(t, I_FCA.*Voltage*60/(2*pi)./RPM, 'r')
plot(t, I_FCB.*Voltage*60/(2*pi)./RPM, 'b')
xlabel('Time'); ylabel('Torque (Nm)');
legend('Request', 'Available', 'Actual', 'FCA', 'FCB', 1)
axis([t(min(find(I_Avail > 0))) t(max(find(I_Avail > 0))) 0 1000])
title('(4.4) Drivetrain: Torque (Request, Avail, Drawn)')
plotlist = 1;

case 45 % Drivetrain: Motor (RPM, Temperature)
MotorTemp = evalin('base', ...
'MH_DriveTrain_MotorTemp', 'zeros(1, length(t))');
plot(t, MotorTemp, 'r')
xlabel('Time (s)'); ylabel('Motor Temperature (\circ C)')
title('(4.5) Drivetrain: Motor (RPM, Temperature)')
plotlist = 2;

case 46 % Drivetrain: Ground Fault Monitor
title('(4.6) Drivetrain: Ground Fault Monitor')
plotlist = 2;

case 47 % Drivetrain: Clutch Pressure, Gear, RPM
title('(4.7) Drivetrain: Clutch Pressure, Gear, RPM')
plotlist = 2;

%% Suspension ================================================================
case 51 % Suspension: Shock Travel
title('(5.1) Suspension: Shock Travel')
plotlist = 2;

case 52 % Suspension: Wheel Speeds
title('(5.2) Suspension: Wheel Speeds')
plotlist = 2;

case 53 % Suspension: Tire Growth
title('(5.3) Suspension: Tire Growth')
plotlist = 2;
%% Brakes ===============================================================

case 61  % Brakes: Brake Pressures (Front/Rear)
    Front = evalin('base',...
               'MH_CockpitP_FrontBrakeP','zeros(1,length(t))');
    Rear = evalin('base',...
                  'MH_CockpitP_RearBrakeP','zeros(1,length(t))');
    plot(d,Front,'b')
    plot(d,Rear,'r')
    xlabel('Distance (miles)'); ylabel('Speed (mph)')
    legend('Front','Rear',2)
    title('(6.1) Brakes: Brake Pressures (Front/Rear)')
    axis([min(d) max(d) 0 1000])
    plotlist = 1;

case 62  % Brakes: Boost Pressure
    Boost = evalin('base',...
                   'MH_CockpitP_ResBrakeP','zeros(1,length(t))');
    plot(d,Boost,'b')
    xlabel('Distance (miles)'); ylabel('Speed (mph)')
    title('(6.2) Brakes: Boost Pressure')
    axis([min(d) max(d) 0 3000])
    plotlist = 1;

%% Vehicle ==============================================================

case 71  % Vehicle: Battery Voltages
    title('(6.2) Vehicle: Battery Voltages')
    plotlist = 2;

case 72  % Vehicle: GPS Data
    title('(6.2) Vehicle: GPS Data')
    plotlist = 2;

case 73  % Vehicle: Speed vs. Distance and Time
    Speed = evalin('base','GPS_VehicleSpeed','zeros(1,length(t))');
    d = evalin('base','d','zeros(1,length(t))');
    subplot(2,1,1); hold on; grid on
    plot(t,Speed,'b')
    xlabel('Time (s)'); ylabel('Speed (mph)')
    title('(7.3.1) Vehicle: Speed vs. Time')
    axis([min(t) max(t) 0 350])

    subplot(2,1,2); hold on; grid on
    plot(d,Speed,'b')
    xlabel('Distance (miles)'); ylabel('Speed (mph)')
    title('(7.3.2) Vehicle: Speed vs. Distance')
    axis([min(d) max(d) 0 350])
    plotlist = 2;

case 74  % Vehicle: Driver Display
    Speed = evalin('base','GPS_VehicleSpeed','zeros(1,length(t))');
Gear = evalin('base', 'MH_DriveTrain_TrannyGear', 'zeros(1,length(t))');
RPM = evalin('base', 'MH_DriveTrain_MotorRPM', 'zeros(1,length(t))');

subplot(3,1,1); hold on; grid on
plot(d,Speed,'b')
axis([min(d) max(d) 0 350])
xlabel('Distance (miles)'); ylabel('GPS Speed (mph)')
title('(7.4.1) Vehicle: Driver Display: Speed')

subplot(3,1,2); hold on; grid on
plot(d,Gear,'r')
axis([min(d) max(d) 0 6])
xlabel('Distance (miles)'); ylabel('Transmission Gear')
title('(7.4.2) Vehicle: Driver Display: Gear')

subplot(3,1,3); hold on; grid on
plot(d,RPM/1000,'g')
axis([min(d) max(d) 0 12])
xlabel('Distance (miles)'); ylabel('RPM (/1000)')
title('(7.4.3) Vehicle: Driver Display: RPM')
plotlist = 2;
end

A.7 Google Earth KML Creation

function [] = BB2_GEarth_KML(Data)
%BB2_GEARTH_KML Summary of this function goes here
% Detailed explanation goes here

%addpath('googleearthtoolbox');

%load('Oct10_run3_cRio.mat')
%data = load('Sun, Jul 20, 2008 - 08_52_57 -.csv');
%load('Oct10_run2_cRio.mat')
%load('Oct10_run1_cRio.mat')
IconFolder = 'http://www.buckeyebullet.com/html/Files/GEmarkers/';

GPS_Longitude = evalin('base', 'GPS_Longitude', 'zeros(1,length(t))');
GPS_Latitude = evalin('base', 'GPS_Latitude', 'zeros(1,length(t))');
Speed = evalin('base', 'GPS_VehicleSpeed', 'zeros(1,length(t))');
RPM = evalin('base', 'MH_DriveTrain_MotorRPM', 'zeros(1,length(t))');
Throttle = evalin('base', 'MH_Cockpit_Throttle', 'zeros(1,length(t))');
Veh_Gear = evalin('base',...
'MH.DriveTrain.TrannyGear', 'zeros(1, length(t))');

Inv_Motor_T = evalin('base', ...
'MH.DriveTrain.MotorTemp', 'zeros(1, length(t))');

Veh_BrakePT_F = evalin('base', ...
'MH.CockpitP.FrontBrakeP', 'zeros(1, length(t))');

Veh_BrakePT_R = evalin('base', ...
'MH.CockpitP.RearBrakeP', 'zeros(1, length(t))');

CANr_I_Avail = evalin('base', ...
'MH.Ballard.CurrentAvail', 'zeros(1, length(t))');

Gas_H2H = evalin('base', 'MH.GasPressures.H2High', 'zeros(1, length(t))');

Gas_O2H = evalin('base', 'MH.GasPressures.O2High', 'zeros(1, length(t))');

%% GPS
BB2Path = geplot(GPS.Longitude(1:10:length(GPS.Longitude)), ...
GPS.Latitude(1:10:length(GPS.Latitude)), ...
'name', 'BB2 Track', ...
'visibility', 1, ...
'lineColor', '#FF000000');

[a b] = max(Speed);
MaxSpeed = gepoint(GPS.Longitude(b), GPS.Latitude(b), a*10, ...
'altitudeMode', 'relativeToGround', ...
'description', ' ', ...
'name', ['Max Speed: ' num2str(a,'%3.2f') ' mph'], ...
'iconURL', [IconFolder 'grn−stars.png'], ...
'iconScale', 1.0);

%% Mark Shift Points
b = min(find(Veh_Gear==2));
if length(b)>0
    Shift2 = gepoint(GPS.Longitude(b), GPS.Latitude(b), 0, ...
        'altitudeMode', 'relativeToGround', ...
        'description', ' ', ...
        'name', '2nd Gear', ...
        'iconURL', [IconFolder '2.png'], ...
        'iconScale', 1.0);
else
    Shift2 = '';
end

b = min(find(Veh_Gear==3));
if length(b)>0
    Shift3 = gepoint(GPS.Longitude(b), GPS.Latitude(b), 0, ...
        'altitudeMode', 'relativeToGround', ...
        'description', 'BB2 Pits<br>2008 Speed Week', ...
        'name', '3rd Gear', ...
        'iconURL', [IconFolder '3.png'], ...
        'iconScale', 1.0);
else
    Shift3 = '';
end

b = min(find(Veh_Gear==4));
if length(b)>0
    Shift4 = ge_point(GPS_Longitude(b),GPS_Latitude(b),0,...
        'altitudeMode','relativeToGround',...
        'description','BB2 Pits<br>2008 Speed Week',...
        'name','4th Gear',...
        'iconURL',[IconFolder '4.png'],...
        'iconScale',1.0);
else
    Shift4 = '';
end

b = min(find(Veh_Gear==5));
if length(b)>0
    Shift5 = ge_point(GPS_Longitude(b),GPS_Latitude(b),0,...
        'altitudeMode','relativeToGround',...
        'description','BB2 Pits<br>2008 Speed Week',...
        'name','5th Gear',...
        'iconURL',[IconFolder '5.png'],...
        'iconScale',1.0);
else
    Shift5 = '';
end

b = min(find(Veh_Gear==6));
if length(b)>0
    Shift6 = ge_point(GPS_Longitude(b),GPS_Latitude(b),0,...
        'altitudeMode','relativeToGround',...
        'description','BB2 Pits<br>2008 Speed Week',...
        'name','6th Gear',...
        'iconURL',[IconFolder '6.png'],...
        'iconScale',1.0);
else
    Shift6 = '';
end

%% Motor Info

[a b] = min(abs(Inv_Motor_T-30));
if length(b)>0
    Motor30 = ge_point(GPS_Longitude(b),GPS_Latitude(b),0,...
        'altitudeMode','relativeToGround',...
        'description',' ',...
        'name','Motor 30C',...
        'iconURL',[IconFolder 'm.png'],...
        'iconScale',1.0);
else
    Motor30 = '';
end
[a b] = min(abs(Inv_Motor_T-40));
if length(b)>0
Motor40 = ge_point(GPS_Longitude(b),GPS_Latitude(b),0,...
'altitudeMode','relativeToGround',... 
'description',' ',... 
'name','Motor 40C',...
'iconURL',[IconFolder 'm.png'],...
'iconScale',1.0);
else
Motor40 = '';
end

[a b] = min(abs(Inv_Motor_T-50));
if length(b)>0
Motor50 = ge_point(GPS_Longitude(b),GPS_Latitude(b),0,...
'altitudeMode','relativeToGround',... 
'description',' ',... 
'name','Motor 50C',...
'iconURL',[IconFolder 'm.png'],...
'iconScale',1.0);
else
Motor50 = '';
end

[a b] = min(abs(Inv_Motor_T-60));
if length(b)>0
Motor60 = ge_point(GPS_Longitude(b),GPS_Latitude(b),0,...
'altitudeMode','relativeToGround',... 
'description',' ',... 
'name','Motor 60C',...
'iconURL',[IconFolder 'm.png'],...
'iconScale',1.0);
else
Motor60 = '';
end

[a b] = min(abs(Inv_Motor_T-70));
if length(b)>0
Motor70 = ge_point(GPS_Longitude(b),GPS_Latitude(b),0,...
'altitudeMode','relativeToGround',... 
'description',' ',... 
'name','Motor 70C',...
'iconURL',[IconFolder 'm.png'],...
'iconScale',1.0);
else
Motor70 = '';
end

[a b] = min(abs(Inv_Motor_T-80));
if length(b)>0
Motor80 = ge_point(GPS_Longitude(b),GPS_Latitude(b),0,...
    'altitudeMode','relativeToGround',...
    'description','',...
    'name','Motor 80C',...
    'iconURL',[IconFolder 'm.png'],...
    'iconScale',1.0);
else
    Motor80 = '';
end

[a b] = min(abs(Inv_Motor_T−90));
if length(b)>0
    Motor90 = ge_point(GPS_Longitude(b),GPS_Latitude(b),0,...
        'altitudeMode','relativeToGround',...
        'description','',...
        'name','Motor 90C',...
        'iconURL',[IconFolder 'm.png'],...
        'iconScale',1.0);
else
    Motor90 = '';
end

[a b] = min(abs(Inv_Motor_T−100));
if length(b)>0
    Motor100 = ge_point(GPS_Longitude(b),GPS_Latitude(b),0,...
        'altitudeMode','relativeToGround',...
        'description','',...
        'name','Motor 100C',...
        'iconURL',[IconFolder 'm.png'],...
        'iconScale',1.0);
else
    Motor100 = '';
end

%% Brakes Applied
b = min(find(Veh_BrakePT_F>50));
BrakesFront = ge_point(GPS_Longitude(b),GPS_Latitude(b),0,...
    'altitudeMode','relativeToGround',...
    'description','',...
    'name','Front Brakes Applied',...
    'iconURL',[IconFolder '\red-square.png'],...
    'iconScale',1.0);

b = min(find(Veh_BrakePT_R>50));
BrakesRear = ge_point(GPS_Longitude(b),GPS_Latitude(b),0,...
    'altitudeMode','relativeToGround',...
    'description','',...
    'name','Rear Brakes Applied',...
    'iconURL',[IconFolder 'red-square.png'],...
    'iconScale',1.0);
%% CAN Lines

b = min(find(CANr.I.Avail>40));
IAvailFirst = ge_point(GPS.Longitude(b),GPS.Latitude(b),0,...
'altitudeMode','relativeToGround',...'description','Current First Available',...
'iconURL',[IconFolder 'go.png'],...
'iconScale',1.0);

%% Hydrogen Tank

[a b] = min(abs(Gas.H2H-4000));
H24000 = ge_point(GPS.Longitude(b),GPS.Latitude(b),0,...
'altitudeMode','relativeToGround',...'name','H2: 4000 PSI',...
'iconURL',[IconFolder 'grn-circle.png'],...
'iconScale',1.0);

[a b] = min(abs(Gas.H2H-3000));
H23000 = ge_point(GPS.Longitude(b),GPS.Latitude(b),0,...
'altitudeMode','relativeToGround',...'name','H2: 3000 PSI',...
'iconURL',[IconFolder 'grn-circle.png'],...
'iconScale',1.0);

[a b] = min(abs(Gas.H2H-2000));
H22000 = ge_point(GPS.Longitude(b),GPS.Latitude(b),0,...
'altitudeMode','relativeToGround',...'name','H2: 2000 PSI',...
'iconURL',[IconFolder 'grn-circle.png'],...
'iconScale',1.0);

[a b] = min(abs(Gas.H2H-1000));
H21000 = ge_point(GPS.Longitude(b),GPS.Latitude(b),0,...
'altitudeMode','relativeToGround',...'name','H2: 1000 PSI',...
'iconURL',[IconFolder 'grn-circle.png'],...
'iconScale',1.0);

%% Heliox Tank

[a b] = min(abs(Gas.O2H-2000));
O22000 = ge_point(GPS.Longitude(b),GPS.Latitude(b),0,...
'altitudeMode','relativeToGround',...'description',' ',...
'name', 'Heliox: 2000 PSI', ...
'iconURL', [IconFolder 'ylw-circle.png'], ...
'iconScale', 1.0);

[a b] = min(abs(Gas_O2H - 1500));
O21500 = ge_point(GPS_Longitude(b), GPS_Latitude(b), 0, ...
'altitudeMode', 'relativeToGround', ...'description', '', ...
'name', 'Heliox: 1500 PSI', ...
'iconURL', [IconFolder 'ylw-circle.png'], ...
'iconScale', 1.0);

[a b] = min(abs(Gas_O2H - 1000));
O21000 = ge_point(GPS_Longitude(b), GPS_Latitude(b), 0, ...
'altitudeMode', 'relativeToGround', ...'description', '', ...
'name', 'Heliox: 1000 PSI', ...
'iconURL', [IconFolder 'ylw-circle.png'], ...
'iconScale', 1.0);

%% 3D Plots

BB2Speed = ge_plot3(GPS_Longitude(1:10:length(GPS_Longitude)), ...
GPS_Latitude(1:10:length(GPS_Latitude)), ...
Speed(1:10:length(GPS_Latitude))*10, ...
'name', 'Speed', ...
'lineColor', '#FFFF0000', ...
'altitudeMode', 'relativeToGround');

BB2RPM = ge_plot3(GPS_Longitude(1:10:length(GPS_Longitude)), ...
GPS_Latitude(1:10:length(GPS_Latitude)), ...
RPM(1:10:length(GPS_Latitude))/10, ...
'name', 'RPM', ...
'visibility', 0, ...
'lineColor', '#FF00FF00', ...
'altitudeMode', 'relativeToGround');

BB2Throttle = ge_plot3(GPS_Longitude(1:10:length(GPS_Longitude)), ...
GPS_Latitude(1:10:length(GPS_Latitude)), ...
Throttle(1:10:length(GPS_Latitude))*10, ...
'name', 'Throttle', ...
'visibility', 0, ...
'lineColor', '#FF0000FF', ...
'altitudeMode', 'relativeToGround');

BB2IAvail = ge_plot3(GPS_Longitude(1:10:length(GPS_Longitude)), ...
GPS_Latitude(1:10:length(GPS_Latitude)), ...
CANr_I_Avail(1:10:length(GPS_Latitude)), ...
'name', 'CAN Current Avail', ...
'visibility', 0, ...
'lineColor', '#FFFF00FF', ...
'altitudeMode', 'relativeToGround');
H2Tank = ge_plot3(GPS_Longitude(1:10:length(GPS_Longitude)),...
    GPS_Latitude(1:10:length(GPS_Latitude)),...
    Gas_H2H(1:10:length(GPS_Latitude)),...
    'name','Hydrogen Tank Pressure',...
    'visibility',0,...
    'lineColor','#FF00FF00',...
    'altitudeMode','relativeToGround');

HelioxTank = ge_plot3(GPS_Longitude(1:10:length(GPS_Longitude)),...
    GPS_Latitude(1:10:length(GPS_Latitude)),...
    Gas_O2H(1:10:length(GPS_Latitude)),...
    'name','Heliox Tank Pressure',...
    'visibility',0,...
    'lineColor','#FF00FFFF',...
    'altitudeMode','relativeToGround');

%% Speed Plots

TrapEntry = '';
Trap1 = '';
Trap2 = '';
Trap3 = '';
TrapExit = '';

%% Organize the results into a folder structure:
% Sub-Folders
FolderPurge = ge_folder('Purges',[]);
FolderHeliox = ge_folder('Heliox',[O22000 O21500 O21000]);
FolderHydrogen = ge_folder('Hydrogen',[H24000 H23000 H22000 H21000]);
FolderMotorTemp = ge_folder('Temperature',[Motor30 Motor40 Motor50...
    Motor60 Motor70 Motor80 Motor90 Motor100]);

% Main Folders
f01 = ge_folder('GPS',[BB2Path MaxSpeed]);
f02 = ge_folder('Shifting',[Shift2 Shift3 Shift4 Shift5 Shift6]);
f03 = ge_folder('Motor',[FolderMotorTemp]);
f04 = ge_folder('Fuel Cells',[FolderPurge IAvailFirst]);
f05 = ge_folder('Inverter',[]);
f06 = ge_folder('Parachutes/Brakes',[BrakesFront BrakesRear]);
f07 = ge_folder('Speed Traps',[TrapEntry Trap1 Trap2 Trap3 TrapExit]);
f08 = ge_folder('Gas Delivery',[FolderHeliox FolderHydrogen]);
f09 = ge_folder('Driver',[]);
f10 = ge_folder('3D Data',...
    [BB2Speed BB2RPM BB2IAvail BB2Throttle H2Tank HelioxTank]);

%Write the 3 foldered kmlStr's to a file:
ge_output('RunData.kml',[f01 f02 f03 f04 f05 f06 f07 f08 f09 f10],...
    'name','August 15, Run 1');
A.8 Report Coverpage

1 % Set Up Plotting Area
2 fig = figure(1);
3 clf;
4 set(fig,'visible','off')
5 set(fig, 'PaperPositionMode', 'manual');
6 set(fig, 'PaperUnits', 'inches');
7 set(fig, 'PaperPosition', [.25,.25,8,10.5]);
8 set(fig, 'Units','inches')
9 set(fig, 'Position',[0,0,8,10.5])

%d Run Graph Plot
12 subplot('Position',[0,0,1/3,1])
13 axis on
14 RunPlot = subplot('Position',[.06,.06,1/4,.9]);
15 axis on
16 box on % Plot Axes Box
17 %hold on

% Make Track Lines
20 line([-50,50],[0,0],'Color',[0,0,0]) % 0 Mile Line
21 line([-50,50],[1,1],'Color',[0,0,0]) % 1 Mile Line
22 line([-50,50],[2,2],'Color',[0,0,0]) % 2 Mile Line
23 line([-50,50],[2.25,2.25],'Color',[0,0,0]) % 2.25 Mile Line
24 line([-50,50],[3,3],'Color',[0,0,0]) % 3 Mile Line
25 line([-50,50],[4,4],'Color',[0,0,0]) % 4 Mile Line
26 line([-50,50],[4.75,4.75],'Color',[0,0,0]) % 4.75 Mile Line
27 line([-50,50],[5,5],'Color',[0,0,0]) % 5 Mile Line
28 line([-50,50],[6,6],'Color',[0,0,0]) % 6 Mile Line
29 line([-50,50],[6,6],'Color',[0,0,0]) % 7 Mile Line
30 hold on

% Plot Run Graph
33 [b a] = butter(2,.008);
34 x = filter(b,a,...
G = filter(b,a,(GPS_Longitude-GPS_Longitude(1))*cos(GPS_Latitude(1)*pi/180)*69.172);
37 y = filter(b,a,(GPS_Latitude-GPS_Latitude(1))*69.159);
38 %angles = atan2(y,x)*180/pi;
%angle = median(angles(find(R4.MH_DriveTrain_MotorRPM>5000)))*pi/180;
angle = (40.9-90)*pi/180;
41 Rot = [cos(angle) sin(angle) 0; -sin(angle) cos(angle) 0; 0 0 1];
42 z = ones(1,length(x));
43 xnew = Rot*[x' y' z']';
44 %plot(xnew(1,:),5280,xnew(2,:),'b')
45 plot(xnew(1,:)*5280+25,xnew(2,:),'b')
46 axis([-50,50,0,7])

145
ylabel('Longitudinal Track Distance (Miles)')
xlabel('Lateral Track Distance (Feet)')
title('Run Graph Plot')

% Write Important Speeds
text(-46.23, [num2str(traps.t23,4), ' mph'], 'Rotation', 90)
text(-46.33, [num2str(traps.t34,4), ' mph'], 'Rotation', 90)
text(-46.43, [num2str(traps.t45,4), ' mph'], 'Rotation', 90)

% Plot Important Points
% Plot Pushtruck Release Point
hold on
plot(RunPlot, LatitudinalPos(pushtruck.ReleaseIndex),... 
LongitudinalPos(pushtruck.ReleaseIndex), 's')
text(LatitudinalPos(pushtruck.ReleaseIndex) + 5,... 
LongitudinalPos(pushtruck.ReleaseIndex), 'PT')

% Plot Shift Points
hold on
plot(RunPlot, LatitudinalPos(shift12.StartIndex),... 
LongitudinalPos(shift12.StartIndex), 'x')
text(LatitudinalPos(shift12.StartIndex) + 5,... 
LongitudinalPos(shift12.StartIndex), '1\rightarrow2')
plot(RunPlot, LatitudinalPos(shift23.StartIndex),... 
LongitudinalPos(shift23.StartIndex), 'x')
text(LatitudinalPos(shift23.StartIndex) + 5,... 
LongitudinalPos(shift23.StartIndex), '2\rightarrow3')
plot(RunPlot, LatitudinalPos(shift34.StartIndex),... 
LongitudinalPos(shift34.StartIndex), 'x')
text(LatitudinalPos(shift34.StartIndex) + 5,... 
LongitudinalPos(shift34.StartIndex), '3\rightarrow4')
plot(RunPlot, LatitudinalPos(shift45.StartIndex),... 
LongitudinalPos(shift45.StartIndex), 'x')
text(LatitudinalPos(shift45.StartIndex) + 5,... 
LongitudinalPos(shift45.StartIndex), '4\rightarrow5')

% Plot Parachute Release Points
hold on
plot(RunPlot, LatitudinalPos(parachute.HSReleaseIndex),... 
LongitudinalPos(parachute.HSReleaseIndex), 'o')
text(LatitudinalPos(parachute.HSReleaseIndex) + 13,... 
LongitudinalPos(parachute.HSReleaseIndex), 'HS')
plot(RunPlot, LatitudinalPos(parachute.LSReleaseIndex),... 
LongitudinalPos(parachute.LSReleaseIndex), 'o')
text(LatitudinalPos(parachute.LSReleaseIndex) + 13,... 
LongitudinalPos(parachute.LSReleaseIndex), 'LS')

% Plot Top Speed Point
hold on
plot(RunPlot, LatitudinalPos(important.TopSpeedIndex(1)),... 
LongitudinalPos(important.TopSpeedIndex(1)), '*')
% text(Latitudeal Pos(important.TopSpeedIndex(1)) + 5,...
% Longitudinal Pos(important.TopSpeedIndex(1)),...
% [num2str(important.TopSpeed(1),4), ' mph'])

% Team Logo
subplot('Position',[1/3, .9, 1/6, 0.1])
BB2_Logo = imread('BB2Logo_CoNw.png);
image(BB2_Logo); axis equal;
axis off;

subplot('Position', [1/3, .82, 1/6, 0.08])
axis([0, 20, 0, 16]); axis off;
text(0, 14, info.date, 'FontWeight', 'bold', 'FontSize', 12)
text(0, 10, [num2str(important.TopSpeed, '%3.0f'), ' mph'],
'FontWeight', 'bold', 'FontSize', 14)
text(0, 6, [num2str(important.PeakPower, '%3.0f'), ' kW'],
'FontWeight', 'bold', 'FontSize', 14)

% Run Specs Plot

subplot('Position', [.52, 4/5, 1/3, 1/5])
axis off
axis([0, 20, 0, 16])
text(0, 15, 'Buckeye Bullet 2 Run Data', 'FontWeight', 'bold', 'FontSize', 14)
%text(0, 15, 'Buckeye Bullet 2 Run Data', 'FontSize', 16)
text(0, 13, ['Run Date: ', info.date, '', info.time])
text(0, 12, ['Driver: ', info.driver])
text(0, 11, ['Temperature: ', num2str(info.temp, 3),...
'\circ F / ', num2str(((5/9) * (info.temp - 32)), 3), '\circ C'])
text(0, 10, ['Barometric Pressure: ', num2str(info.barometer, 4), 'in.'])
text(0, 9, ['Humidity: ', num2str(info.humidity), '%'])
text(0, 8, ['Salt Conditions: ', info.salt])

% Draw Run Type Boxes
x=19; y=13.5;
text(x, y, 'Run Type', 'FontWeight', 'bold')
text(x+1.2, y-1.5, 'Testing')
text(x+1.2, y-2.5, 'Qualification')
text(x+1.2, y-3.5, 'Back-up')
rectangle('Position', [x, y-2, 1, 1])
rectangle('Position', [x, y-3, 1, 1])
rectangle('Position', [x, y-4, 1, 1])
switch info.RunType
    case {'Testing', 'testing'}
        rectangle('Position', [x, y-2, 1, 1], 'FaceColor', [0, 0, 0])
    case {'Qualification', 'qualification'}
        rectangle('Position', [x, y-3, 1, 1], 'FaceColor', [0, 0, 0])
    case {'Backup', 'backup'}
        rectangle('Position', [x, y-4, 1, 1], 'FaceColor', [0, 0, 0])
end
end

% Draw Record Type Boxes
x=19; y=8;
text(x,y,'Record Type','FontWeight','bold')
text(x+1.2,y-1.5,'National (U.S.)')
text(x+1.2,y-2.5,'International')
rectangle('Position',[x,y-2,1,1])
rectangle('Position',[x,y-3,1,1])
switch info.RecordType
    case {'National','national'}
        rectangle('Position',[x,y-2,1,1],'FaceColor',[0,0,0])
    case {'International','international'}
        rectangle('Position',[x,y-3,1,1],'FaceColor',[0,0,0])
end

% Important Data Plot
subplot('Position',[1/3,3/5,1/3,1/5])
axis off
axis([0,10,0,16])
text(0.15,'Important Data','FontWeight','bold','FontSize',12);
text(0.13,'Top Speed.....................');
text(6.13,[num2str(max(important.TopSpeed),'%6.3f'),' mph']);
    'FontWeight','bold');
text(0.12,'Peak Power....................');
text(6.12,[num2str(important.PeakPower,'%6.3f'),' kW']);
text(0.11,'Average Power.................');
text(6.11,[num2str(important.AveragePower,'%6.3f'),' kW']);
text(0.10,'Energy Used....................');
text(6.10,[num2str(important.EnergyUsed,'%6.3f'),' kWh']);
text(0.09,'Total Shift Time................');
text(6.09,[num2str(important.TotalShiftTime,'%3.2f'),' s']);
text(0.08,'Push Time........................');
text(6.08,[num2str(important.PushTime,'%2.1f'),' s']);
text(0.07,'Push Distance..................');
text(6.07,[num2str(important.PushDistance,'%3.2f'),' miles']);
text(0.06,'Push Release Speed............');
text(6.06,[num2str(important.PushReleaseSpeed,'%3.1f'),' mph']);
text(0.05,'Start Ride Height..............');
text(6.05,[num2str(important.StartRideHeight),' in']);
text(0.04,'Top Speed Ride Height.....');
text(6.04,[num2str(important.EndRideHeight),' in']);

% Fuel Cells Plot
subplot('Position',[.38,.36,.6,.24]); hold on; grid on;
plot(d,MH_GasPressures_H2High,'b')
plot(d,MH_GasPressures_O2High,'r')
    'DeltaP (psi)','??');
xlabel('Distance (Miles)')
\begin{verbatim}
axis([0 7 0 6000])

% Big Speed Plot
subplot('Position',[.38,.06,.6,.24]); hold on; grid on;
plot(d,GPS_VehicleSpeed*2,'r')
plot(d,MH_DriveTrain_MotorRPM/10,'k')
plot(d,MH_HighVoltage_CurrentTotal,'b')
plot(d,MH_HighVoltage_DCVoltage,'g')
legend('Speed (MPH)*3','RPM/10','Current','Voltage',4)
xlabel('Distance (Miles)')
title('Vehicle Performance')
axis([0 7 0 1000])

%% Final Outputs
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
%saveas(fig,'infosheet.pdf') % Saves as a JPG File
%print -dpdf 'infosheet.pdf' % Print to vector PDF
%print -dpdc 'infosheet.ps' % Print to vector PDF
%ps2pdf('psfile','infosheet.ps','pdffile','infosheet.pdf','gspapersize','letter')
print(fig,'RunData.ps','−dpdc2');
% close(fig) % Close Figure

A.9 Page Formatting for Standard Plots

function []=pageformat(fig,pageformat)
    switch pageformat
        case 'landscape'
            set(fig, 'PaperPositionMode', 'manual');
            set(fig, 'PaperUnits', 'inches');
            set(fig, 'PaperOrientation','landscape');
            set(fig, 'PaperPosition', [0,0,11,8.5]);
            set(fig, 'Units', 'inches');
            set(fig, 'Position',[0,0,11,8.5])
        case 'portrait'
            set(fig, 'PaperPositionMode', 'manual');
            set(fig, 'PaperUnits', 'inches');
            set(fig, 'PaperOrientation','portrait');
            set(fig, 'PaperPosition', [0,0,8.5,11]);
            set(fig, 'Units', 'inches');
            set(fig, 'Position',[0,0,8.5,11])
        case 'screen'
            set(fig, 'Units', 'pixels');
            set(fig, 'Position',[100,100,600,500])
    end
\end{verbatim}
APPENDIX B

SAMPLE RUN REPORT (SELECTED FIGURES)
Buckeye Bullet 2 Run Data

Run Date: 8/22/2008 17:30
Driver: Roger Schroer
Temperature: 29°F / −1.67°C
Humidity: 0%
Salt Conditions: Salty

8/22/2008
297 mph
560 kW

Important Data

Top Speed.................... 296.966 mph
Peak Power................... 560.141 kW
Average Power............... 483.423 kW
Energy Used.................. 12.218 kWh
Total Shift Time............. 5.20 s
Push Time................... 17.8 s
Push Distance............... 0.19 miles
Push Release Speed......... 42.0 mph
Start Ride Height........... 0 in
Top Speed Ride Height..... 0 in

Vehicle Performance
(3.4.1) Cooling: Control

- Fuel Cell
- Ice Bath

(3.4.2) Cooling: Pressures

- Left
- Right

Pressure (psi) vs. Time (s)

Pressure (psi) vs. Time (s)
(4.5) Drivetrain: Motor (RPM, Temperature)
(7.4.1) Vehicle: Driver Display: Speed

(7.4.2) Vehicle: Driver Display: Gear

(7.4.3) Vehicle: Driver Display: RPM