UNIVERSITY OF MISSOURI – COLUMBIA
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
"A PROUD TRADITION, ENGINEERING THE FUTURE!"

Electronic Throttle Control for the University of Missouri FSAE Formula Car

ECE 4220

REAL TIME EMBEDDED SYSTEM PROJECT REPORT

May 13, 2015

— Engineering Team —

Gage Crowder, Electrical and Computer Engineering

— Course Instructor —

Mr. Luis Rivera

Department of Electrical and Computer Engineering, University of Missouri

Abstract

The project is electronic throttle control for the University of Missouri FSAE formula car. Having electronic throttle control increases responsiveness of the throttle by eliminating the delay caused by mechanical linkages. In addition it reduces the overall weight of the vehicle and decreases the number of mechanical service visits [1]. This project provides that electronic solution. The project consists of a control unit, two throttle position sensors, two pedal position sensors, a brake-system-encoder, and a throttle actuator. The control unit monitors sensors that relay back information about the engine as well as driver input. The control unit then provides control signals to the throttle actuator and fuel pump to accurately control vehicle acceleration.

Index Terms: *Drive-by-wire, electronic throttle control, FSAE ETC, vehicle driving, vehicle dynamics.*

Table of Contents

Table of Figures

Table of Tables

Chapter 1. Introduction

This chapter provides a description of the project, goals and objectives, project motivation, background information, and a report overview.

Description

This project is designed to be a new and improved method to control the throttle of the University of Missouri's FSAE team formula car. The system has been developed with vehicle integration in mind with the hope of installing the system in next year's car. By interfacing with hardware and using a micro-controller, this project is able to sense driver input as well as current vehicle conditions and perform actions based on certain design criterion.

Goals and Objectives

The short term goals for this project are to have all the hardware interfacing operational, and to meet as many of the time constraints as possible. The long term goal of this project is to have the system fully operational to meet the FSAE design standards to allow use of the project in next year's car. I expect that these goals are both realistic and attainable.

Project Motivation

The University of Missouri formula car team builds a scale race car every year and enters it into a design and racing competition [2]. However, to stay competitive the car must be as advanced as possible. This includes aerodynamics, power output, electronics, and control [2]. With that in mind an electronic throttle control system is a great addition to next year's car. This adds one more system to the car to help gain valuable points towards the teams overall score.

Background

Electronic throttle control differs from that of a mechanical control system; the throttle is monitored and controlled by an embedded system as opposed to mechanical linkages. According to [1] this has many benefits over the conventional system including: reducing the number of moving parts, reducing overall weight, increasing operational accuracy, and a decrease in the number of service visits for mechanical service. These are all beneficial design considerations when designing the car.

By implementing this project, an embedded system along with a small number of hardware devices, the formula car attains all the advantages mentioned above. There is currently no system that the team has previously used or designed so this project will be the first of its kind for the team.

Report Overview

The remainder of this report is organized as follows. The Implementation of the Electronic Throttle Control is defined in Chapter 2 including a functional block diagram and software flow diagram summarizing the operation of the system. Chapter 3 will discuss the experiments performed on the system and the results of the experimentation. Chapter 4 is an in-depth discussion of the project. This will include a discussion of the results, discussion of the time constraints imposed on the system, and how the system would benefit from a multi-threaded approach. Chapter 5 is the conclusion to the report with a discussion of the projects successes, failures, and thoughts on further improvements to the project.

Chapter 2. Implementation

This chapter provides a description of the proposed implementation, and a description of the demonstrated system. This chapter also provides a description of the demonstrated implementation, including generalized descriptions of the overall system, the units, the assemblies and subassemblies, and the software architecture.

Proposed Method

This section contains a figure that describes the proposed software flow implementation.

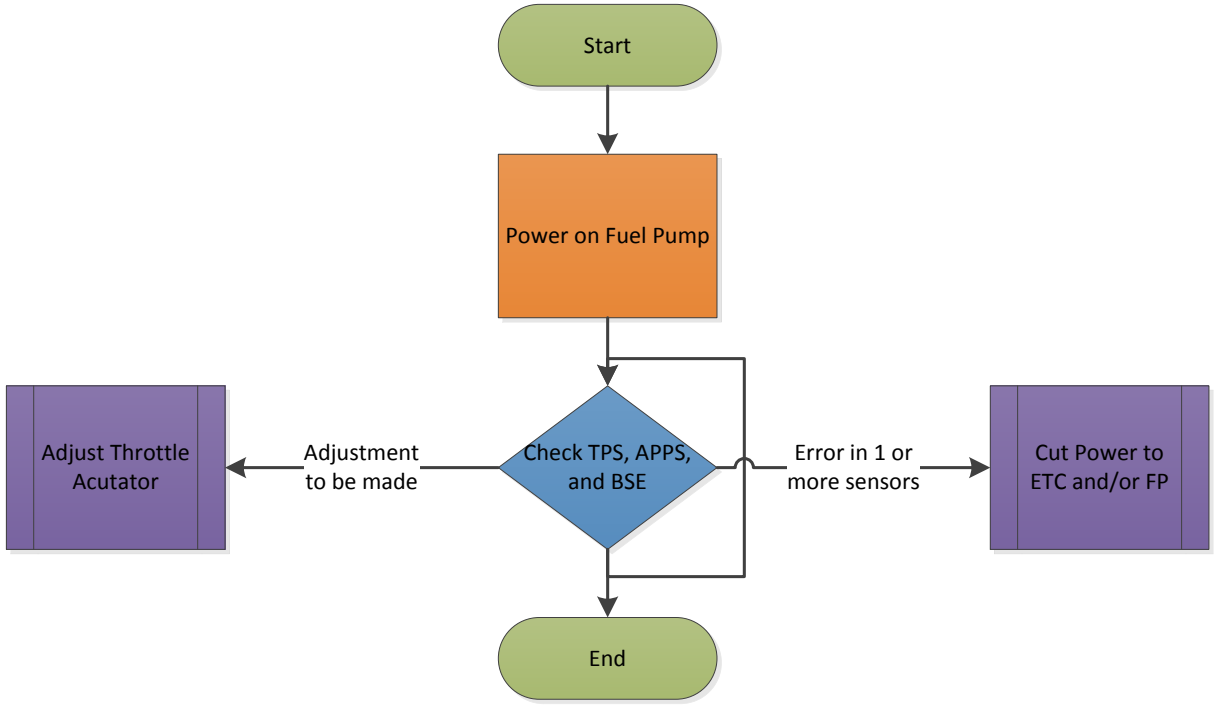

Figure 1 - Proposed software flow diagram

System Description

This section contains figures and descriptions that describe the system.

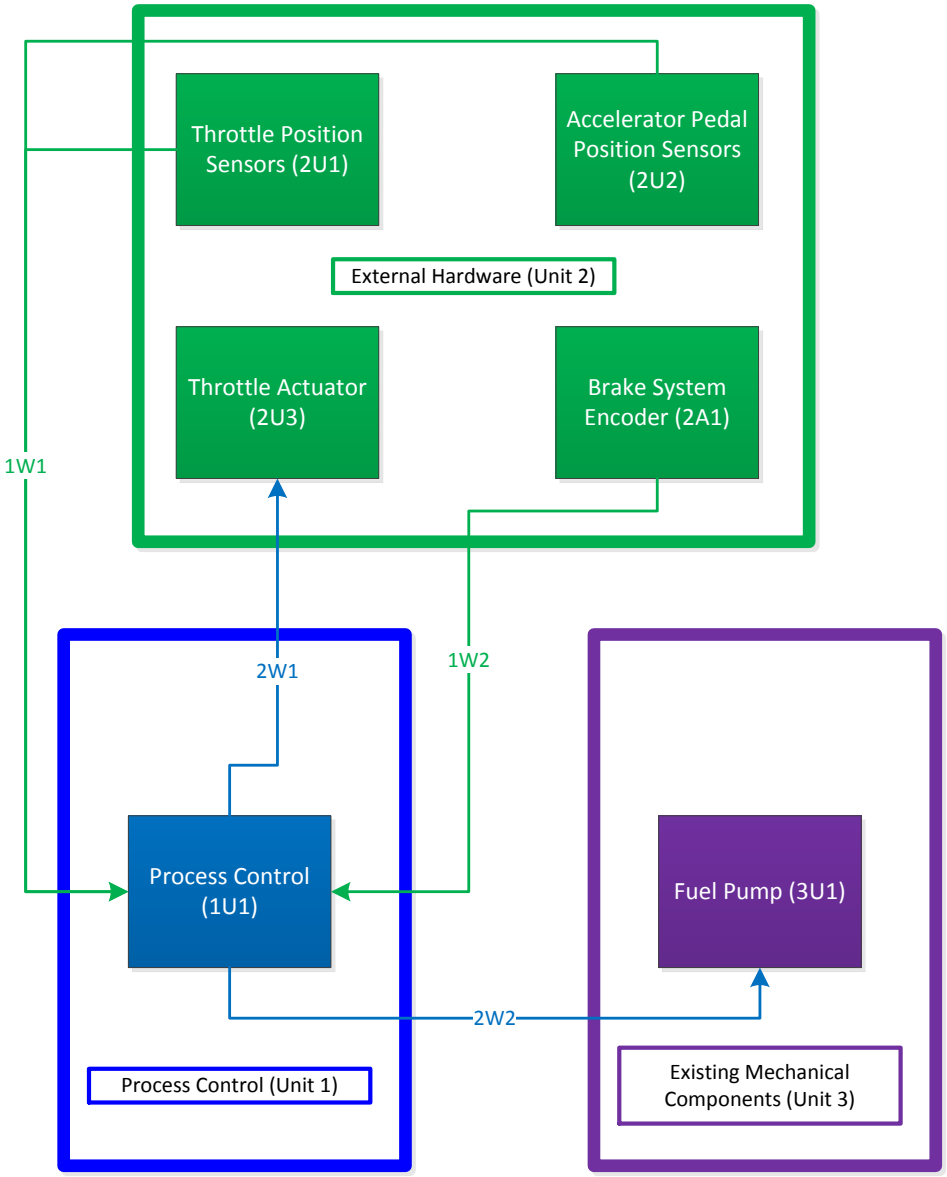

Figure 2 - Functional block diagram

Functional Blocks:

The following describes the functional blocks as shown in *Figure 2 - Functional block diagram*. Some design constraints and specifications were taken from [3] to ensure it meets minimum safe operating requirements for use by the FSAE team.

Process Control (Unit 1)

The following describes the functional blocks for *Process Control (Unit 1)*.

1U1 – Process Control

The process control unit is an Arduino Uno powered by 5 VDC from the computer USB. An Arduino was chosen because of its compact size and ease of use when interfacing with external hardware. Actual integration will require the Arduino be powered off of the existing 12V battery on the car. It is responsible for monitoring all the external sensors as well as providing a control signal to the throttle actuator and fuel pump. The process control unit will cut power to *2U3 – Throttle Actuator* and *3U1 – Fuel Pump* as defined under *2A1 – Brake System Encoder*, *2U1 – Throttle Position Sensors*, and *2U2 – Accelerator Pedal Position Sensors*.

External Hardware (Unit 2)

The following describes the functional blocks for *External Hardware (Unit 2)*.

2U1 – Throttle Position Sensors

The throttle position sensors consist of one rotary potentiometer with two leads into the Arduino. Actual integration would require two individual sensors mounted to the throttle body of the car's engine. It communicates with *1U1 – Process Control* via an analog signal.

2A1 – Brake System Encoder

The brake system encoder is push button switch to simulate a physical hardware device already on the car. It communicates with *1U1 – Process Control* via a digital signal. Power to *2U3 – Throttle Actuator* and *3U1 – Fuel Pump* will be shutdown given any of the following: the brake system encoder switch is pressed, if the error between the two pedal position values is greater than 10%, or if the error between the two throttle positions values greater than 10%.

2U2 – Accelerator Pedal Position Sensors

The accelerator pedal position sensors consist of one rotary potentiometer with two leads into the Arduino. Actual integration would require two individual sensors actuated by a foot pedal. There are two separate pedal position sensors.

2U3 – Throttle Actuator

The throttle actuator is a small servo motor. Actual integration would require a small servo mounted to the throttle body of the car's engine. It is responsible for opening and closing the throttle to allow more or less fuel and air into the engine. It communicates with *1U1 – Process Control* via a digital PWM signal.

Existing Mechanical Components (Unit 3)

The following describes the functional blocks for *Existing Mechanical Components (Unit 3)*.

3U1 – Fuel Pump

The fuel pump is represented by an LED indicator. Actual integration would require a mechanical device already installed on the car which provides fuel to the engine. Power to the fuel pump is controlled by *1U1 – Process Control*. Power to the fuel pump will be interrupted as dictated by *2A1 – Brake System Encoder*. It communicates with *1U1 – Process Control* via a digital control signal.

Signals:

The following describes the signal paths as shown in *Figure 2 - Functional block diagram.*

Process Control (Unit 1)

The following describes the signal paths for *Process Control (Unit 1)*.

1W1

This is a wired analog input signal from *2U1 – Throttle Position Sensors* and *2U2 – Accelerator Pedal Position Sensors* to *1U1 – Process Control*. This allows the process control to monitor the current throttle position and the current accelerator pedal positon.

1W2

This is a wired digital input signal from *2A1 – Brake System Encoder* to *1U1 – Process Control*. It monitors the brake system encoder for error states that require cutting power to one or more assemblies.

External Hardware (Unit 2)

The following describes the signal paths for *External Hardware (Unit 2)*.

2W1

This is a wired digital PWM output signal from *1U1 – Process Control* to *2U3 – Throttle Actuator*. This controls where the throttle should be positioned.

2W2

This is a wired digital output signal from *1U1 – Process Control* to *3U1 – Fuel Pump*. This controls the power to the fuel pump.

Implementation

This section contains figures and descriptions of the demonstrated projects software implementation.

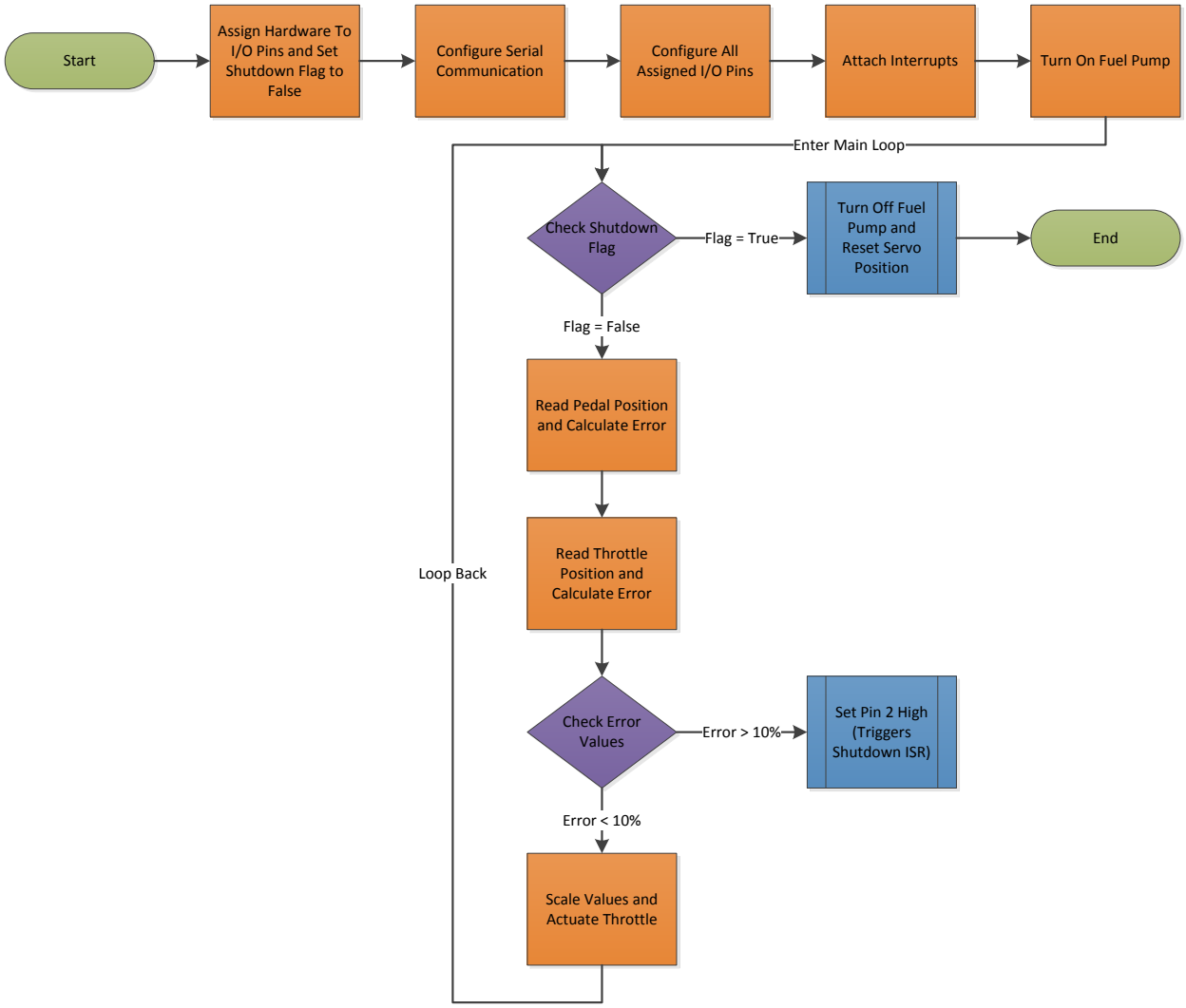

Figure 3 - Demonstrated software flow diagram

Functions

The following describes the functions used and overall software implementation as shown in *Figure 3 - Demonstrated software flow diagram*.

void setup()

This function is run once each time the Arduino is reset and serves as the entry point of the program. This function begins by starting serial communication and setting the appropriate baud rate. The function then uses pinMode() to set analog pin 2, 3, 4, and 5 for input with the internal pull-up resistors enabled for the two pedal position sensors, and the two throttle position sensors respectively. The function then uses pinMode() to set digital pin 3 for output with internal pull-up resistor enabled for the brake system encoder switch respectively. Next pinMode() is used again to set digital pin 12 and 2 for output for use with the fuel pump and shutdown interrupt respectively. The function then sets digital pin 2 to low. Next the function attaches the shutdown ISR to pins 0 and 1 triggered on the rising and falling edge respectively. The function then configures digital pin 9 for use with the servo. Lastly the function calls Control_Fuel_Pump() and turns on the fuel pump.

void loop()

This function is the main body of the program and runs continuously, until an error state occurs requiring the Arduino to be reset. The function begins by checking if the shutdown flag has been set or not. If the flag is true the function calls actuate_Throttle() to set the throttle to the neutral position, calls Control_Fuel_Pump() to turn off the fuel pump, calls Serial.println() to record that the system has shut down, and then uses an empty while loop to force the system to be reset. The rest of this function is then divided into three main sections as discussed below.

Accelerator Pedal

This part of the function is responsible for reading both pedal position sensors. This is done by calling the built in function analogRead() with the corresponding pin number for each sensor. Serial.print() is then used to record the value that was read for each sensor. The current

pedal position is then calculated by averaging the two values. The error is then found by subtracting position_1 from position_2 and taking the absolute value.

Throttle Position

This part of the function is responsible for reading both throttle position sensors. This portion of the function behaves exactly as the accelerator pedal section but uses the I/O pins that correspond to the throttle position sensors.

Check Error

This part of the function is responsible for checking the error recorded for the pedal position as well as the throttle position. If the error of either is recorded to be greater than 10% digitalWrite() is called to set pin 2 high. This triggers the ISR which in turn sets the shutdown flag to true. If the error is less than 10% this part of the function will do nothing.

Actuate Throttle

Lastly, the function scales the input values (0-1023) to valid output values (0-179) and then calls actuate_Throttle() to set the throttle to the correct position. The function then loops back to the beginning.

void Control_Fuel_Pump(bool fuel_pump_status)

This function is responsible for controlling the status of the fuel pump. The function takes in a Boolean value that corresponds to the conditional state of the fuel pump. If the parameter is true the fuel pump will be turned on. If the parameter is false the fuel pump will be turned off. To control the fuel pump the built in function digitalWrite() is used to control the digital pin that the fuel pump is connected to.

void shutDownISR()

This function is an interrupt service routine and is attached to digital pins 2 and 3.The ISR attached to digital pin 3 corresponds to interrupt 1 and is connected to the brake system

encoder switch. The switch is wired normally open so that pin 3 is normally low. This interrupt is configured to trigger on the rising edge. The ISR attached to digital pin 2 corresponds to interrupt 0 and is used for software interrupts. This pin is configured with the internal pull-up resistor enabled so this interrupt is configured to trigger on the falling edge. This function is used in both cases to set the shutdown flag to true. The main program will then see the changed flag value and shutdown the system. The table below shows the interrupt and pin numbers for the Arduino Uno.

Table 1 - Arduino interrupt pins [4]

Board	Int 0	Int 1
Uno	2	3

void actuate_Throttle(int target_position)

This function is responsible for actuating the servo motor through use of the built in servo library. The function takes in an integer that is the target throttle position. The function then calls servo.write() passing in the target throttle position. This actuates the throttle and the function returns.

Chapter 3. Experiments and Results

To test the software implementation of the demonstrated project multiple tests were performed. The first test was very straightforward; it involved turning the pedal position potentiometer left and right to observe whether or not the throttle servo moved to the corresponding position. The second test was to turn the throttle position potentiometer left and right and observe the system changes. The third test was to press the brake system encoder pushbutton switch to simulate a signal received from the actual brake system encoder. The status of the fuel pump and the position of the servo were then recorded after the button press event. The fourth test was to disconnect one wire from the pedal position potentiometer to simulate an error greater than 10% between the two pedal position sensors. The status of the fuel pump and the position of the servo were then recorded after the wire was disconnected. The fifth and final test was to disconnect one wire from the throttle positon potentiometer to simulate an error greater than 10% between the two throttle position sensors. The status of the fuel pump and the position of the servo were then recorded after the wire was disconnected. Each of these test were performed approximately five times. During each run of the various tests the same results were observed and recorded. The results of the different tests are summarized in the table below.

Table 2 - Test results

Test	Expected Result	Observed Result
Turn pedal position potentiometer left and right	Throttle servo moves left and right	Throttle servo moved left and right
Turn throttle position potentiometer left and right	Nothing visually should happen	Nothing visually happened
Press brake system encoder button	Servo returns to center position and fuel pump LED turns off	Servo returned to center position and fuel pump LED turned off
Disconnect 1 wire from pedal position potentiometer to simulate error > 10%	Servo returns to center position and fuel pump LED turns off	Servo returned to center position and fuel pump LED turned off
Disconnect 1 wire from throttle position potentiometer to simulate error > 10%	Servo returns to center position and fuel pump LED turns off	Servo returned to center position and fuel pump LED turned off

The figure below shows the output of the serial communication from the Arduino to the computer. Here it is shown that the two throttle position sensors are reading the same value within one, and the two throttle position sensors are reading the same value within one.

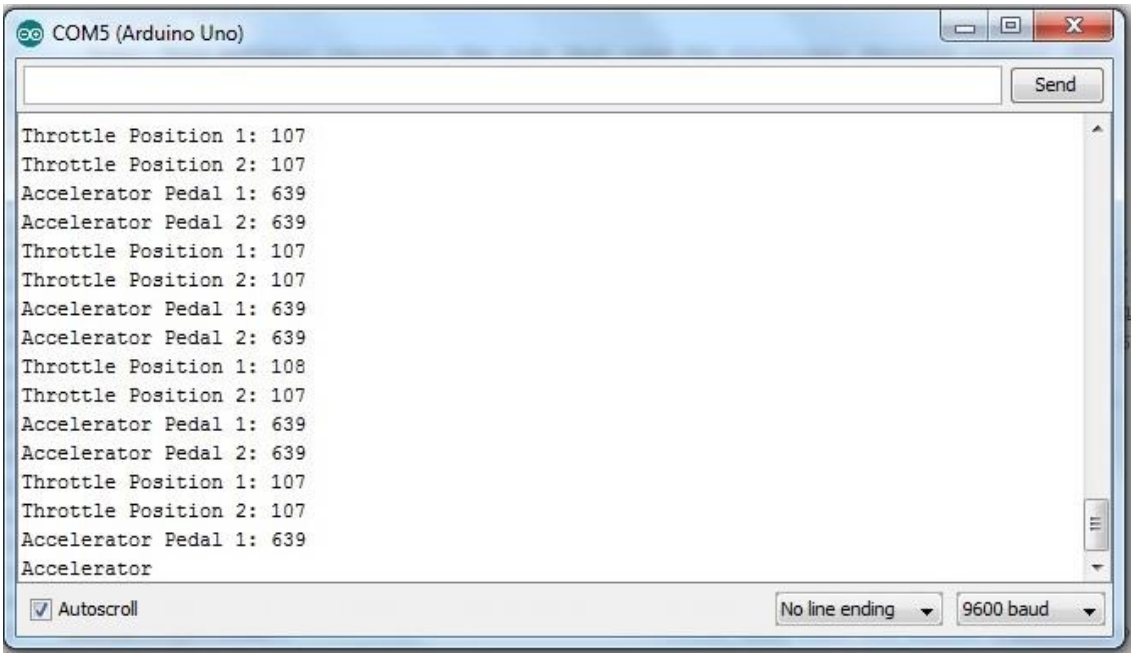

Figure 4 - Serial communication output

Chapter 4. Discussion

This chapter provides a discussion of the experimental results. It also provides a general discussion of the time constraints imposed on the system. Lastly is includes a section that discusses the benefits of implementing a multi-threaded approach.

Discussion of Results

Referring to the table in Chapter 3 it can be seen that the implementation for this project worked as expected. Turning the pedal position potentiometer resulted in the throttle servo moving to a position that directly correlated to the pedal position. Turning the throttle position potentiometer resulted in no visual changes. In the integrated setup the software would compare the target position (pedal position) and compare that to the current position (throttle position). If these two values had an error greater than 10% the shutdown flag would be set and the software would respond accordingly. To test this configuration the Arduino must be connected to an actual throttle body that has the actuator and the position sensor integrated into one unit. An alternative to this would be to use a servo motor that had an output wire for the current output shaft position. However, I only had access to a three wire servo for my tests. This would be an easy implementation change to integrate.

The remaining three tests also all performed as expected. Pressing the brake system encoder pushbutton caused the fuel pump LED to turn off and the servo to return to a neutral position. This test simulates a heaving braking situation from the brake system encoder hardware for testing. Removing one of the pedal position wires from the Arduino also caused the system to behave as expected. Because this condition was representative of a pedal position sensor error greater than 10% the fuel pump LED was turned off and the servo returned to the neutral

position. This is the expected result. Lastly removing one of the throttle position wires from the Arduino caused the system to behave as expected. Because this condition is representative of a throttle position sensor error greater than 10% the fuel pump LED was turned off and the servo returned to the neutral state. All cases that the LED was turned off and the servo reset were simulating different system shutdown conditions that all worked as expected, and as outlined in the FSAE electronic throttle control requirements document [3].

Time Constraints

In the original project proposal there were time constrains outlined for the different system shutdown cases. I was unable to implement these time constraints due to the nature of the hardware. The Arduino Uno does not support real time tasks or multi-threading. Because of this everything must run as one task. I did attempt to use the built in function mills() to get the current time in milliseconds and then compare that to later recorded times. Because this process required calling another function which then triggered a timing interrupt, the results were slow. By ignoring the time constraints altogether the overall performance of the system improved significantly. When integrating this system into the competition vehicle a different development platform should be chosen to allow for better control of timing constraints imposed on the system.

Multi-Threaded Approach

Referencing the source code in Appendix C it can be seen that the demonstrated project does not use a multi-threaded approach in carrying out its task. The main reason for this is that the Arduino Uno does not support multi-threading. A different development platform, such as the TS-7250, would alleviate this problem. By using a multi-threaded approach I believe the system could be greatly improved. First, each hardware assembly could have its own thread

dedicated to it. This would increase the responsiveness of the system and allow for a larger number of polls to be taken on the hardware. By not having to wait on the loop to return back to each hardware section they would all be allowed to run "concurrently". Before the system is integrated into the vehicle I believe I will revise the project to implement a multi-threaded design.

Problems Encountered and Lessons Learned

During the course of this project I encountered multiple problems. The first problem I encountered was correctly mapping the values of the pedal position sensors to an output position on the throttle servo. I later realized that the sensors read in values (0-1023) and the servo library only supports values (0-179). I searched the Arduino website and found a built in function that allowed value scaling which fixed this problem. The second problem I encountered during this project was implementing the interrupt handler that changes the shutdown status flag. I was able to solve this problem in multiple steps. I first had to reference the Arduino website to find what pins corresponded to the different interrupts as shown in *Table 1 - Arduino interrupt pins*. I then had to experiment with different pin configurations and triggers types before the ISR would correctly work. The correct configuration is documented in Chapter 2.

During the course of this lab I encountered multiple lessons learned. First I learned how to interface with various hardware components on the Arduino. This required learning how to correctly configure the various I/O pins. Second, I learned how to use the Servo.h library to interface with and control a three wire servo motor. Third, I learned how to implement interrupts on the Arduino Uno, which required some experimentation before working correctly. This included setting trigger types and correctly configuring the I/O pins which the interrupts were attached to.

Chapter 5. Conclusion

Overall this project serves its purpose as an electronic throttle control system. The project is able to read two pedal position sensors, two throttle positon sensors, and a brake system encoder. The project is also able to output a control signal to the fuel pump and to the throttle actuator. The project has multiple system shutdown conditions that when tested all worked appropriately. The demonstrated project utilized simulation hardware due to constraints imposed by the course and by the FSAE formula car team. However, I believe the system should operate the same given the correct hardware configuration.

While the project did behave as expected there are several limitations to the current implementation. First, the project does not have the ability to run multiple threads. This is due to limitations set forth by the embedded board that was chosen. In the future I would change the development board to something like a TS-7250 or something similar to allow for multi-threading. The second limitation is that the system is unable to track all the required timing constraints for the FSAE competition. This is again a hardware limitation and in the future could be fixed with something such as a TS-7250. However, it should be noted the Arduino platform was originally chosen due to its much smaller footprint, the micro-controller chip is all that is needed, compared to boards such as the TS-7250. With that being said I believe the tradeoff between computing power and space would make the switch worth it. Overall this project was a success, with room for improvement and an outline to implement some of those improvements.

Appendix A. Traceability

Software Used

- Arduino IDE, Arduino. 1.63
    - Servo.h, Software Library

Appendix B. Project Photos

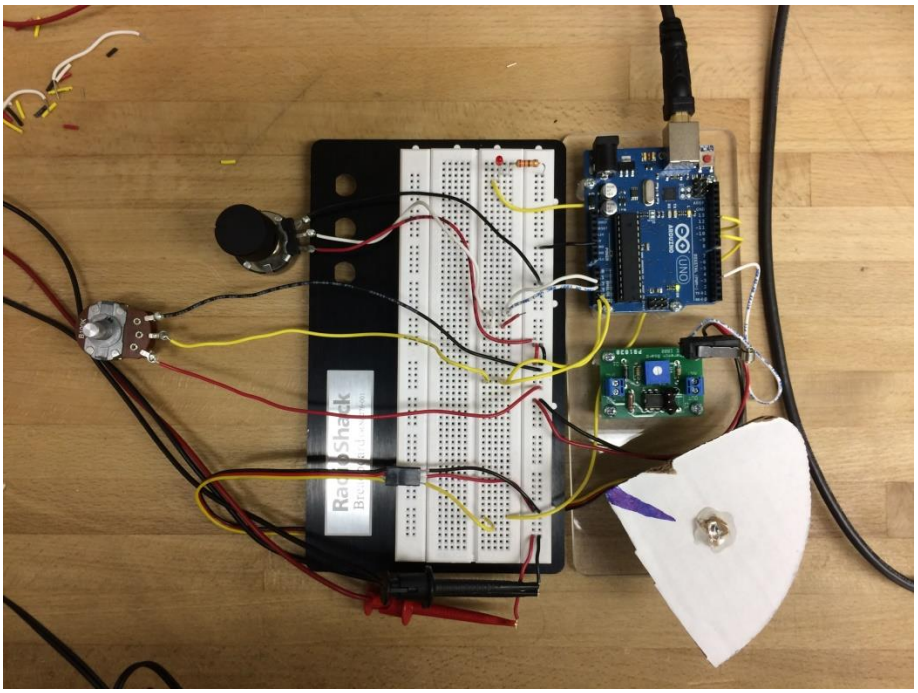

Figure 5 - Completed hardware configuration

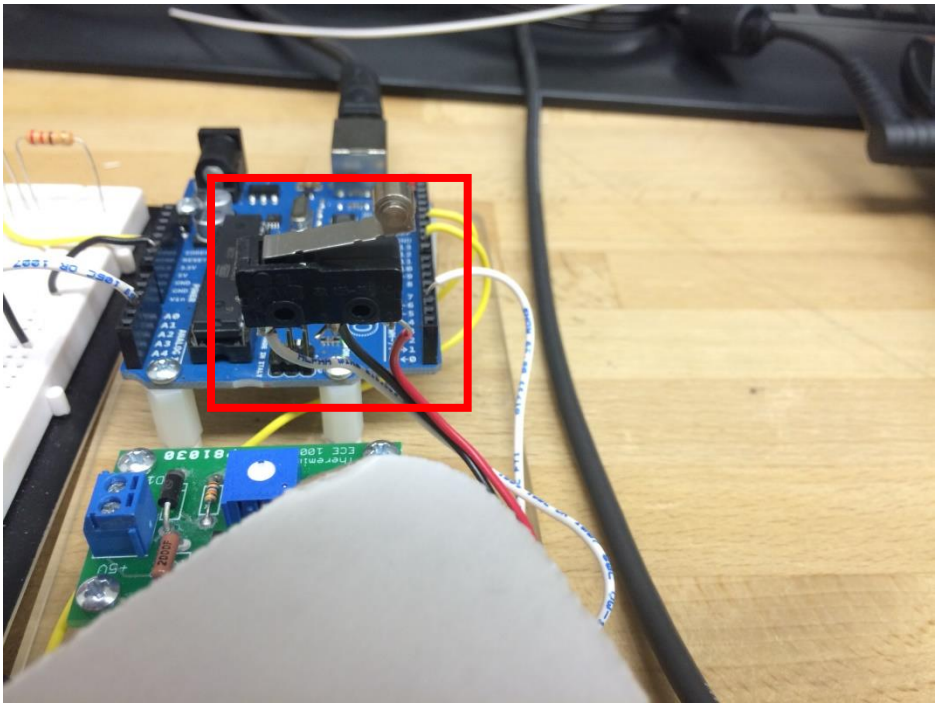

Figure 6 - Brake system encoder pushbutton

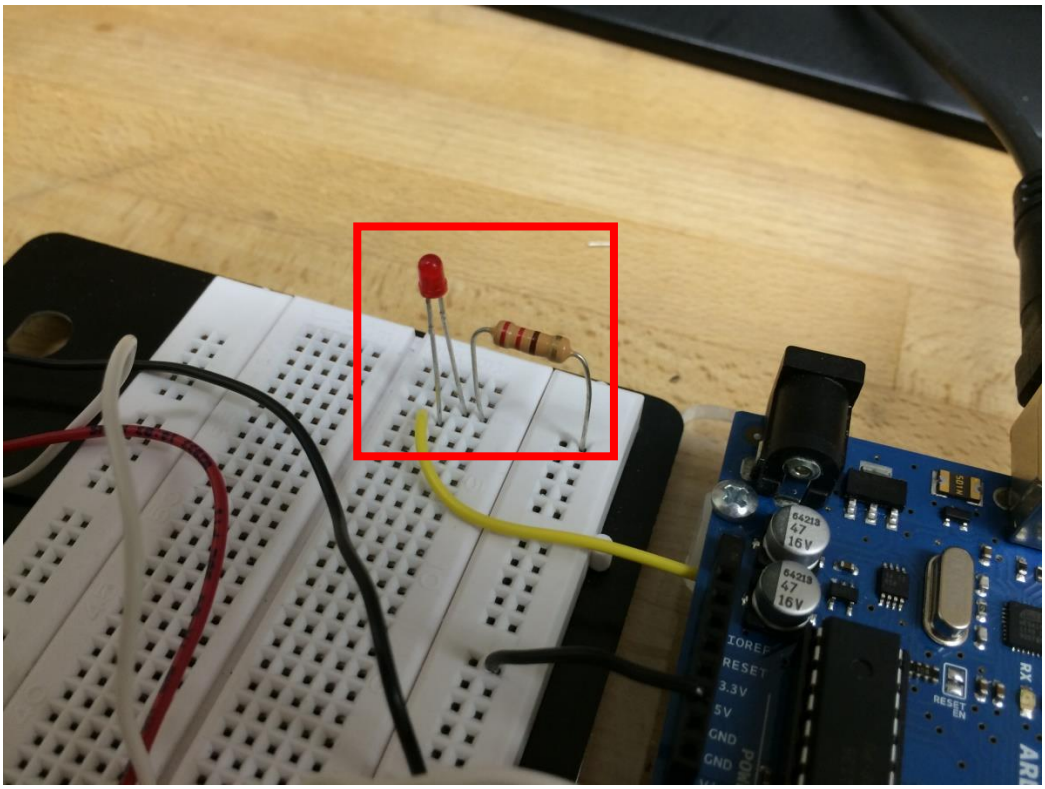

Figure 7 - Fuel pump LED indicator

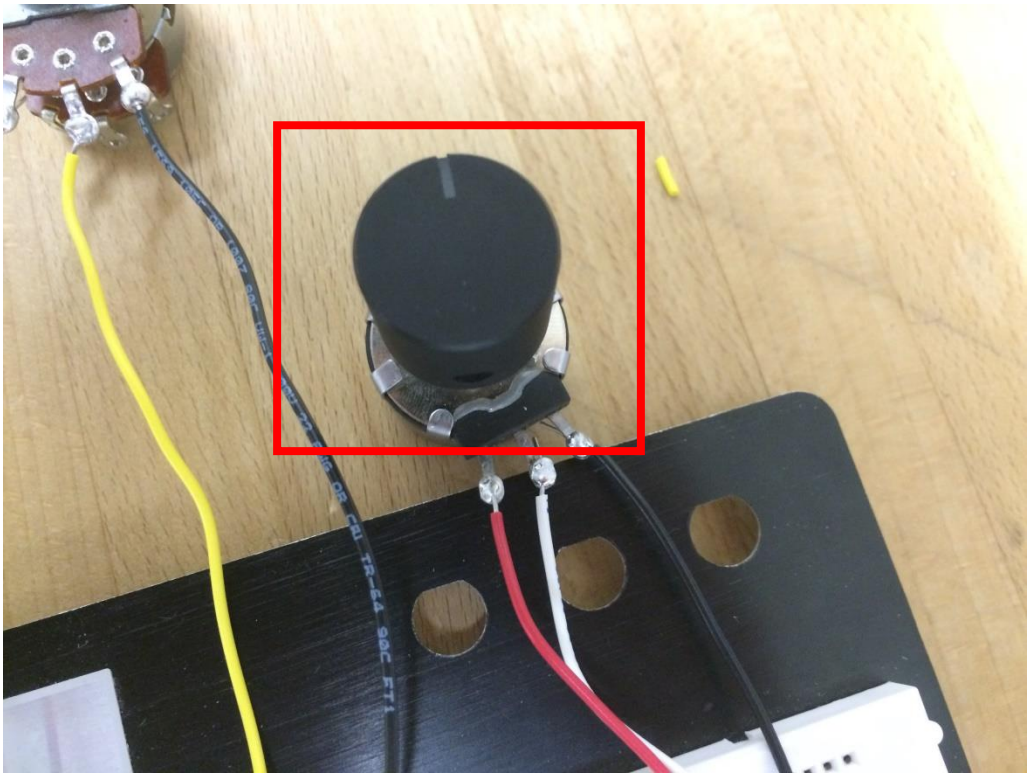

Figure 8 - Pedal position potentiometer

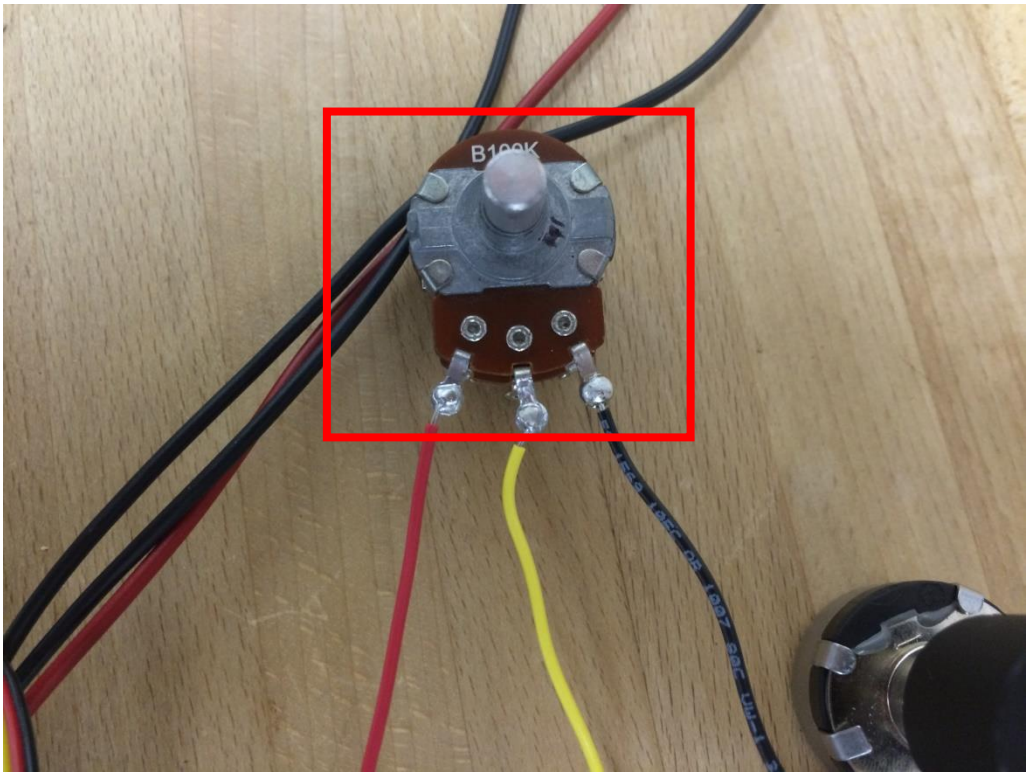

Figure 9 - Throttle position potentiometer

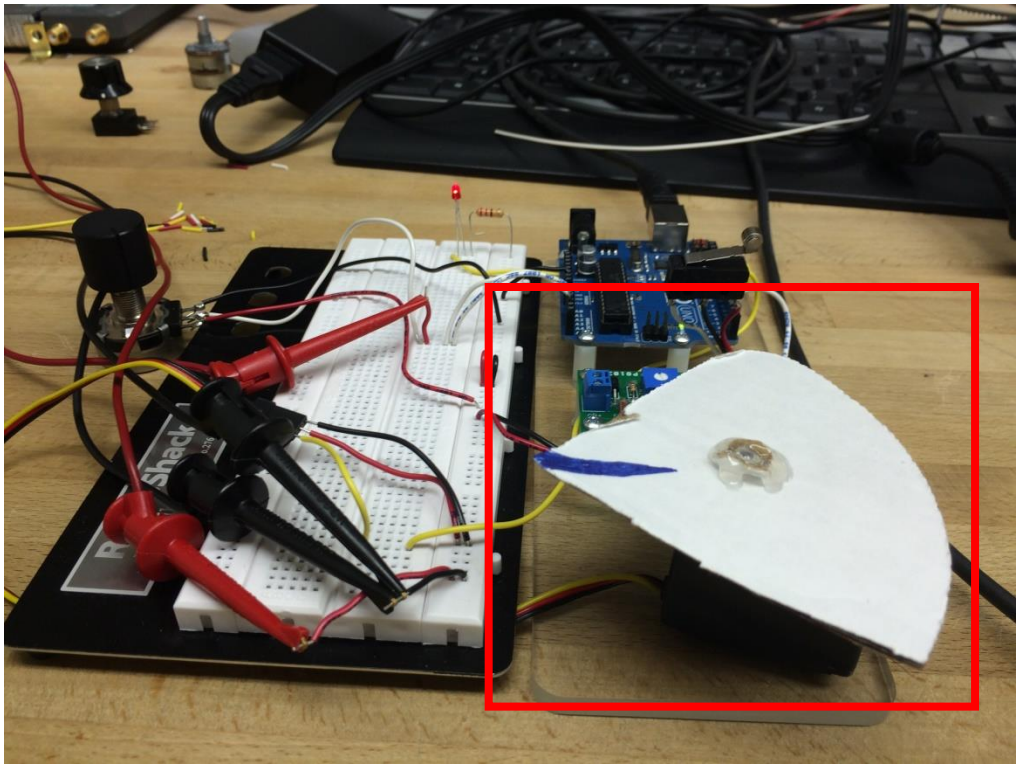

Figure 10 - Throttle servo with position indicator

Appendix C. Source Code

```
/*  Name: Gage Crowder
    Date: 04-28-15
    Course: ECE 4220
    Version: 2.2
    Desc: This software represents the code that runs the electronic throttle control (ETC) for the Mizzou
FSAE formula car
*/

/* Required libraries */
#include <Servo.h>

/* Global variables */
int APPS1 = A2;                                 //Accelerator pedal position sensor 1 (analog pin 2)
int APPS2 = A3;                                 //Accelerator pedal position sensor 2 (analog pin 3)
int TPS1  = A4;                                 //Pedal position switch 1 (analog pin 4)
int TPS2  = A5;                                 //Pedal position switch 2 (analog pin 5)
int FP1   = 12;                                 //Fuel pump 1 (digital pin 12)
int BSE   = 3;                                  //Brake system encoder (digital pin 3)
Servo Throttle_Actuator;                        //Throttle actuator (servo object)

bool shutdown_flag = false;                     //Flag for system shutdown

/* Function that only runs once to setup variables */
void setup()
{
  Serial.begin(9600);                           //Set buad rate for serial communication

  /* Accelerator pedal position sensor pins */
  pinMode(APPS1, INPUT_PULLUP);                 //Set analog pin 2 (APPS1) for input
  pinMode(APPS2, INPUT_PULLUP);                 //Set analog pin 3 (APPS2) for input

  /* Throttle position sensor pins */
  pinMode(TPS1, INPUT_PULLUP);                  //Set analog pin 4 (TPS1) for input
  pinMode(TPS2, INPUT_PULLUP);                  //Set analog pin 5 (TPS2) for input
```

```
  /* Brake system encoder pin */
  pinMode(BSE, INPUT_PULLUP);                         //Set digital pin 3 (BSE) for input with

  /* Fuel pump pin */
  pinMode(FP1, OUTPUT);                               //Set digital pin 12 (FP1) for output

  /* Interrupt pins and ISR attachment */
  pinMode(2, OUTPUT);                                 //Set digital pin 2 (interrupt 0) for input pullup
resitor on

  digitalWrite(2, LOW);                               //Set digital pin 2 (interrupt 0) to low
  attachInterrupt(0, shutDownISR, RISING);            //Attach the interrupt handler to interrupt 0
triggered on the rising edge
  attachInterrupt(1, shutDownISR, FALLING);           //Attach the interrupt handler to interrupt 1
triggered on the falling edge

  /* Throttle actuator */
  Throttle_Actuator.attach(9);                        //Attatch servo (throttle actuator) to digital pin 9

  /* Fuel pump initilize */
  Control_Fuel_Pump(true);                            //Set fuel pump to on
}

void loop()
{

  /* Check if the program has been signaled for shutdown by the ISR */
  if(shutdown_flag == true)
  {
    actuate_Throttle(90);                             //Return to center position
    Control_Fuel_Pump(false);                         //Shutdown fuel pump
    Serial.println("System Shutdown");                //System shutdown message
    while(1);                                         //Force reset of program
  }

  /****************************** Accelerator Pedal ********************************/
  /* Read the pedal position sensors */
  int pedal_position_1 = analogRead(APPS1);           //Read pedal position sensor 1
  int pedal_position_2 = analogRead(APPS2);           //Read pedal position sensor 2
```

```
  /* Pedal position debug info */
  Serial.print("Accelerator Pedal 1: "); Serial.println(pedal_position_1);
  Serial.print("Accelerator Pedal 2: "); Serial.println(pedal_position_2);

  /* Get the average of the 2 sensor for the current pedal position */
  int current_pedal_position = (pedal_position_1 + pedal_position_2) / 2;

  /* Calculate the error between the 2 pedal position sensors */
  int pedal_error = abs(pedal_position_1 - pedal_position_2);

  /****************************** Throttle Position ***********************************/
  /* Read the throttle position sensors */
  int throttle_position_1 = analogRead(TPS1);         //Read throttle position sensor 1
  int throttle_position_2 = analogRead(TPS2);         //Read throttle position sensor 2

  /* Throttle Position debug info */
  Serial.print("Throttle Position 1: "); Serial.println(throttle_position_1);
  Serial.print("Throttle Position 2: "); Serial.println(throttle_position_2);

  /* Get the average of the 2 sensors for the current throttle position */
  int current_throttle_position = (throttle_position_1 + throttle_position_2) / 2;

  /* Calculate the error between the 2 throttle position sensors */
  int throttle_error = abs(throttle_position_2 - throttle_position_1);

  /****************************** Check Error ***********************************/
  /* If the error between either of the 2 sensors in either set is greater than 100 (apprx. 10%) shutdown
the system */
  if(pedal_error > 100 || throttle_error > 100)
  {
    digitalWrite(2, HIGH);                                    //Set pin 2 to high (triggers shutdown
ISR)
  }

  /* Actuate the throttle based on the sensor inputs */
  int servo_pos = map(current_pedal_position, 0, 1023, 0, 179);    //Scale the input values 0-1023 to the
output values 0-179
  actuate_Throttle(servo_pos);                                //Actuate the throttle based on the
scaled value
}
```

```
/* Inputs: The status of the fuel pump, a bool value
   Returns: None
   Purpose: To set the fuel pump control pin to either high or low (on or off) */
void Control_Fuel_Pump(bool fuel_pump_status)
{
  if(fuel_pump_status == true)
    digitalWrite(FP1, HIGH);                          //Turn fuel pump on
  else
    digitalWrite(FP1, LOW);                           //Turn fuel pump off
}

/* Inputs: None
   Returns: None
   Purpose: ISR that signals the system to shutdown*/
void shutDownISR()
{
   shutdown_flag = true;                              //Set shutdown flag to true
}

/* Inputs: The target throttle position
   Returns: None
   Purpose: To actuate the throttle servo to the correct position */
void actuate_Throttle(int target_position)
{
  Throttle_Actuator.write(target_position);           //Writes the throttle actuator (servo) to the correct
position
      }
```

Bibliography

[1] J. Fuller, "How Drive-by-wire Technology Works," HowStuffWorks, 28 April 2009. [Online]. Available: 2015. [Accessed 12 February 2015].

[2] "About Us | Mizzou FSAE," [Online]. Available: http://mizzoufsae.com/?q=node/2. [Accessed 02 February 2015].

[3] SAE International, "SAE Collegiate Design Series - SAE International," 17 September 2014. [Online]. Available: http://students.sae.org/cds/formulaseries/rules/2015-16_fsae_rules.pdf. [Accessed 12 February 2015].

[4] "Arduino - AttachInterrupt," Arduino, [Online]. Available: http://www.arduino.cc/en/Reference/AttachInterrupt. [Accessed 28 April 2015].