ELECTRO-PNEUMATIC SHIFTING AND SERVO CONTROL

OF A CLUTCH FOR A FSAE RACECAR

by

RAHUL JANARDHAN CHALMELA

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN MECHANICAL ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

MAY 2017

Acknowledgements

Firstly, I would like to express my special thanks to my supervising professor Dr. Robert L. Woods who has guided me throughout this project, he has continuously supported and encouraged me by providing resources, practical solutions and unmatched guidance during his supervision of this study.

Secondly, I would like to thank UTA Racing FSAE team and the teammates I have for constantly supporting and believing in my work, as this was the first time the team was moving to an entirely new system after almost 30 years. I would also like to thank all the drivers for their valuable feedback on the new system installed. The team has also been generous to allow me to convert an older car so that the setup on the new car becomes easier.

Finally, I would like to thank my family and friends who have supported me throughout this especially Abhiram Sakleshpur who has helped me start this study as a summer project and by helping me understand programming. I would also like to thank Michael Baldwin from SMC USA to provide me with the cylinders I needed to develop this system.

April 28, 2017

Abstract

ELECTRO-PNEUMATIC SHIFTING AND SERVO CONTROL

OF A CLUTCH FOR A FSAE RACECAR

Rahul Janardhan Chalmela, MS

The University of Texas at Arlington, 2017

Supervising Professor: Robert L. Woods

FSAE is an international student design competition held by SAE International where students design, build and compete in static events like cost, design and dynamic events ranging from acceleration, autocross to endurance. UTA Racing has been a part of FSAE since 1982 and have been using a mechanical design for their shifter and clutch since then. This year for the 2017 design competition an electro-pneumatic system is developed to replace the mechanical system.

This system consists of pneumatic cylinders controlling a sequential shifter and a servo clutch. The cylinders are controlled using direction control valves, these receive their signals from a microcontroller which has been programmed accordingly. The major part of this thesis defines the development of a feedback controlled servo clutch for the racecar. This sub system is unique by itself as this is the first time UTA Racing would be using an electro-pneumatic clutch. The shifter reduces the shift times from 150 milliseconds to 75 milliseconds. The report also includes selection of the pneumatic tank to last an endurance event which requires approximately 1000 gear shifts.

The Electro-pneumatic system not only reduces effort seen by the driver while shifting but also substantially improves the lap-times thereby increasing performance during the dynamic events.

Table of Contents

List of Illustrations

List of Tables

Chapter 1

Introduction

The following chapter describes about the mechanical design and a basic idea of the implementation of the electro-pneumatic design.

1.1 UTA Racing shifter design from 1985-2016

The Formula SAE racecars built by UTA have been very successful in the past due to a multitude of reasons. The previous system of manual shifting is also one of them. It consists of a mechanism termed as a Butterfly Shifter, named by the team. It includes a clutch controlled using a cable and a shifter controlled using a rod, both hinged at the same point on a lever mechanism. The way the mechanism works is that it has two levers that pivot about their axis of rotation which then actuate the clutch cable, also the entire assembly has its own pivot point which moves a set of links that then move the shifter lever. The advantage of this system over the basic systems used many teams in FSAE is that they employ a clutch pedal with a single lever for the shifter, this is considered by UTA Racing as a major flaw in racecar driving as introducing a third pedal makes driving more difficult.

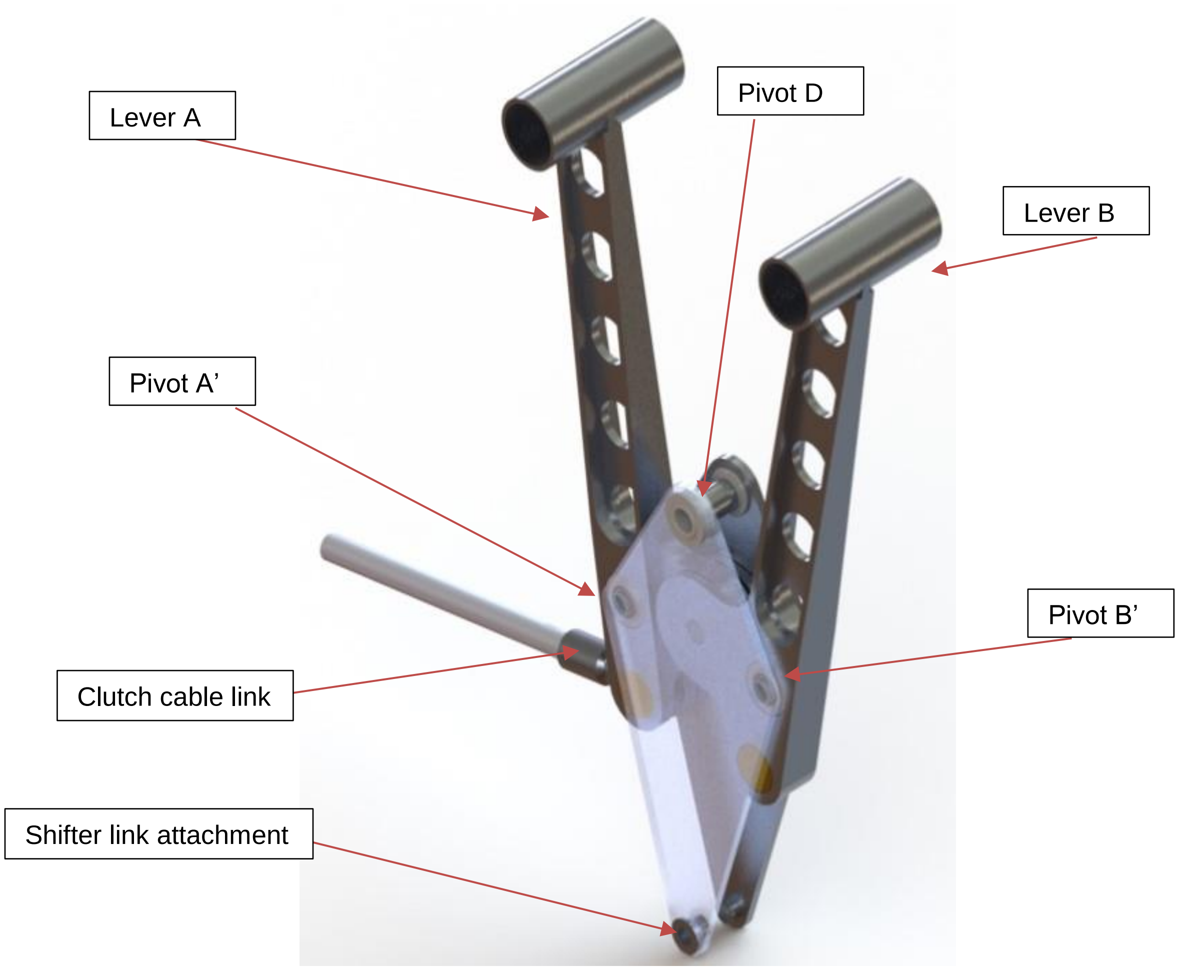

Figure 1 Butterfly Shifter Normal Position

The above image shows levers A and B, which pivot about point A’ and B’, this causes the clutch cable to be actuated as the brass components move away from each other. Point D is the point of pivot for the entire assembly and when lever a is pushed towards lever B they meet and then the entire assembly ratchets about point D, this is a downshift scenario. For upshifting lever A is pulled away from lever B and the mechanism only pivots about point D allowing an upshift.

Figure 2 Butterfly Shifter Actuated Position

The above image shows the actual actuation of the clutch system and not the shifting system. For shifting this entire assembly ratchets about point D.

1.2 Idea behind the electro-pneumatic system

The mechanical system can be improved ergonomically, the fundamental flaw being the requirement that the driver remove his/her hands from the steering wheel while shifting. This is something that adversely affects performance of the racecar, especially in very unpredictable scenarios where shifts might be required in the middle of turns. Another reason being that the endurance event lasts for 13.66 miles overall which would require inputs for the driver for shifting and clutch actuation over a long period and it becomes stressful.

Electronic shifting improves the ergonomic comfort of the driver, as well as the performance of the racecar. The system envisaged for this purpose consists of a pneumatic shifter made up of a cylinder actuating a shift lever. The cylinder is fed through a solenoid valve to which suitable, short pulses are sent. Due to the unique and competitive driving technique that UTA FSAE drivers employ, a servo clutch is a necessity. A servo clutch also definitely makes launches much smoother than otherwise. It was decided that the clutch should be actuated by another pneumatic cylinder. The idea of using an electronic servo was neglected as it increases cost and is very heavy.

The other types of clutch mechanism teams use employ a hand clutch [1] or a foot clutch [2], but that defeats the ergonomic approach. Some teams also use a pneumatic operated clutch, but it works in an on-off fashion which is not good for the clutch plates or the drivetrain system [3].

Chapter 2

Design Approach

The initial steps were to calculate the forces and the torques required for both the actuation of the clutch and the shifter. This was done by attaching the hook of a spring balance to the arm of the clutch/shifter lever on the engine and pulling it. The forces required for actuation were recorded, and on multiplication with the length of the lever arm, the torque required to actuate was determined. The maximum force value for the shifter was found to be around 30 lb with a corresponding torque value of around 4.2 lb-ft and an arm length of 1.7 inches. The maximum torque value for the clutch was measured as 7 lb-ft, slip torque was measured as 5.7 lb-ft and the start torque measured as 5 lb-ft. These values are approximate, and the true value is mentioned in the Appendix C. These same values are used to generate spreadsheets that help in determining the dimensions required for the sizing the cylinders which use standard mathematical equations and the procedure is also mentioned in Appendix C. After the cylinders were sized another spreadsheet was created to determine the size of the pneumatic that that is going to be used. This sheet also has standard mathematical equations and the combined gas law equation.

A Pulse width modulation [3][6] spreadsheet [Appendix C] was created to determine the on time for the solenoid. This was useful as it helped convert percentage duty cycle to an 8-bit value which the Arduino code understands.

Chapter 3

Initial Test Bench Setup

The following chapter describes the hardware setup that was initially available and was modified to work with the new microcontroller. It includes the basic test bench which was modified to achieve better readings, implementation of injectors, the electronic and software testing, the setup of the programmer to control the microcontroller which include the approach to program it and them test bench programing with their outputs.

3.1 Hardware test bench

The bench setup includes a basic workbench board, levers, links and springs to imitate the forces of the shifter and the clutch. It also had the cylinders, lines, valves and the microcontroller.

Figure 3 Bench Setup

The above image shows the bench setup. The primary reason to work on a test bench was to achieve maximum programming capability, and to tune a system before it is deployed. This also included reliability testing and comparing various programs. This setup helped understand programming the microcontroller and how it could be used in the most efficient way. The setup was not an exact replica of the forces observed on the actual engine, but were approximate to the behavior and loading cycle.

The hardware includes a double acting cylinder for the shifter, a single acting spring expanded cylinder for the clutch (initial setup had a double acting cylinder being used as a single acting, the spring was added to increase the static stiffness of the system as the flow of pressurized air is controlled only in one direction.), 5 port 3 position electro-pneumatic directional control valve for the shifter, 3 port 2 position directional control valve for the clutch (later replaced by a custom housed CNG injector assembly) and ¼" pneumatic lines (later replaced by ⅛" pneumatic lines).

A variable linear potentiometer was attached to the clutch cylinder. This potentiometer gives a feedback signal to the microcontroller which helps in determining the position of the actuator link/ position of the clutch.

3.1.1 CNG injectors

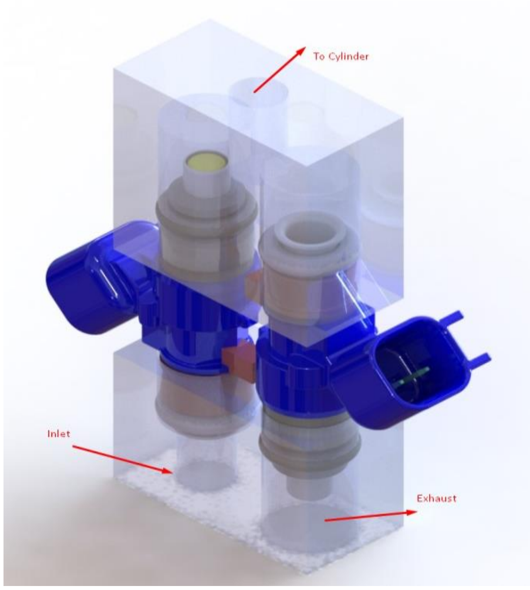

Figure 5 Injector manifold CAD render

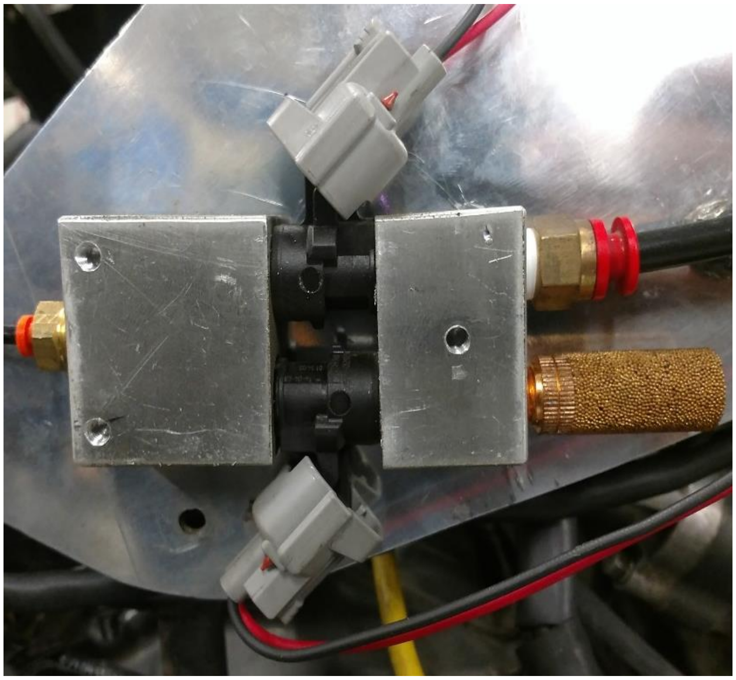

Figure 4 Injector manifold manufactured part

The use of CNG injectors is the key to the success of the servo clutch. This is because the valves conventional valves that are used to operate pneumatic cylinders cannot open and close faster than 2 milliseconds, whereas an injector can operate much faster and open and close faster than one millisecond. CNG injectors were introduced because the rate at which injectors operate is much faster. As the CNG injector is placed in a custom housing both the injectors can be operated simultaneously unlike the spool valve which was used before. This allows much smoother operation to do minor corrections while the actuator moves across the length to be travelled. Furthermore, while testing the response of the directional control valve it was observed that the range of duty cycles that the valve operates is very less as compared to that of the injectors.

3.2 Electronic and Software test bench

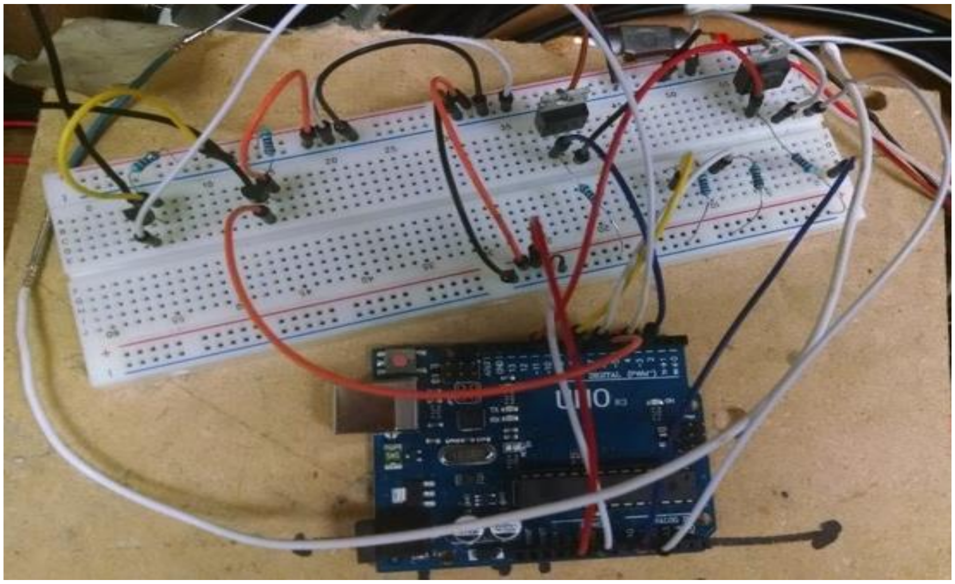

Figure 6 Microcontroller on the test bench

The above image shows how the initial test started with the Arduino Uno R3 microcontroller. Since programming was a new thing this controller was selected as it is very easy to understand and use. The circuit board was made on a breadboard to easily connect and disconnect. It consists of MOSFETs to connect the 5-volt system to the 12-volt solenoid actuators. N-channel MOSFETs were used over P-Channel due to simplicity and ease to use plus it has a wider range of options. And MOSFETs were selected over relays due to the rate at with it can switch the state. Input and feedback potentiometers are also used across the analog terminals of the microcontroller to compare the input and feedback signals. These signals are in the form of a 10-bit value. The basics of writing the code is explained in Appendix D

3.4 Test bench readings

As the programming of the controller was figured out along with the setup of the mechanical and electronic hardware, various programs were written to understand the response of the system both for the shifter and the clutch. Following are the outcomes of the programs that were used. The program code is mentioned in Appendix E. The Pulse width modulation frequency was programmed using the Arduino timers [5].

3.4.1 Shifter and Clutch Program for Test Bench Reading

From the code mentioned in Appendix E, it is visible that the carrier frequency defined is 245.10 Hz. The duty cycle defined is the “pwm” variable and the value is set to 76.5 out of 255 in terms of 8-bit code and is equal 30% duty cycle. Similarly, the duty cycle “pwm” variable value is changed in terms of 8-bit values to different percentage duty cycle from 0-100 in steps of 10. This is done for various carrier frequency setups like 30.64 Hz, 112.55 Hz and 490.20 Hz. Following are the graphs that were generated using this code.

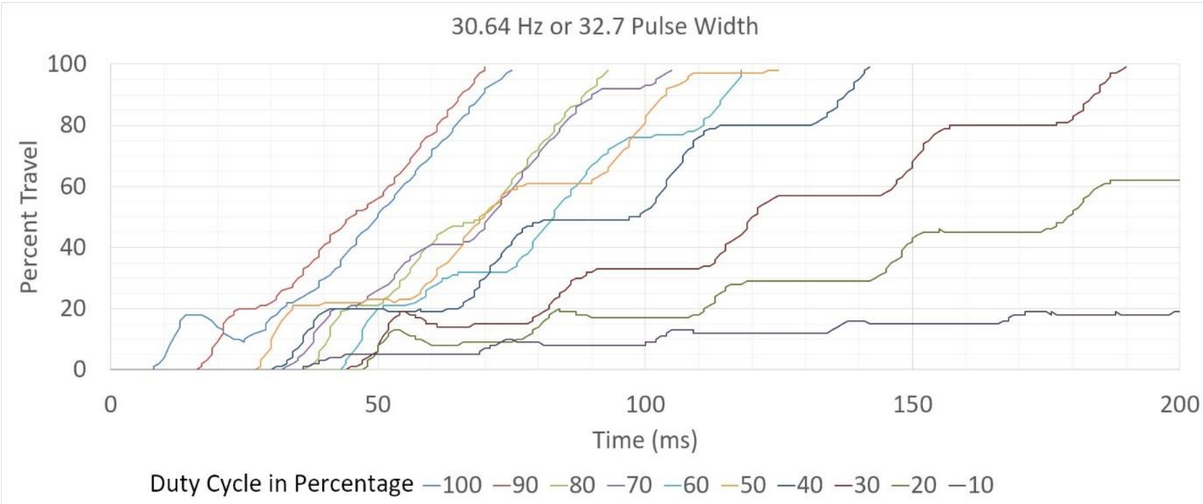

Figure 8 Graph for 30.64 Hz

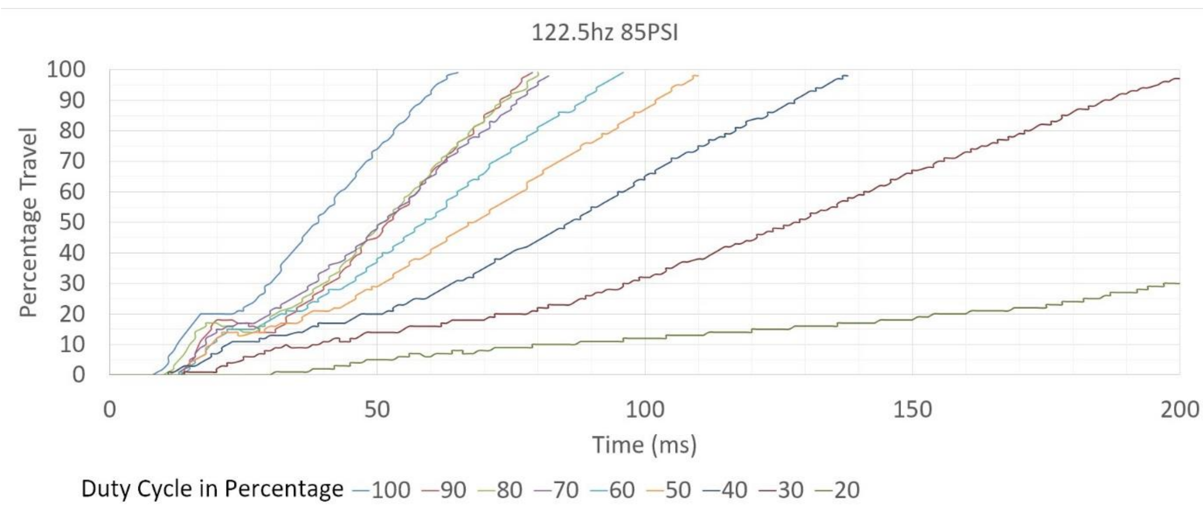

Figure 7 Graph for 122.5 Hz

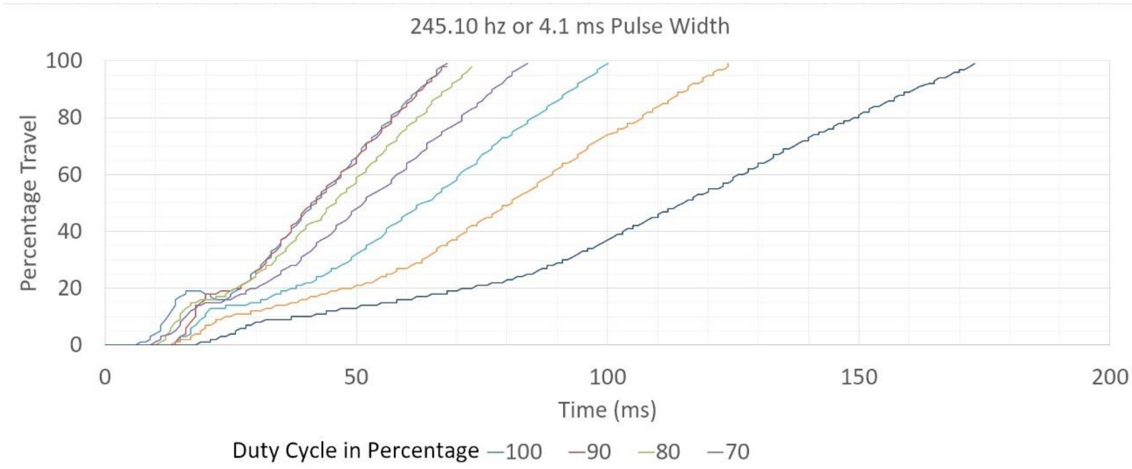

Figure 9 Graph for 245.1 Hz

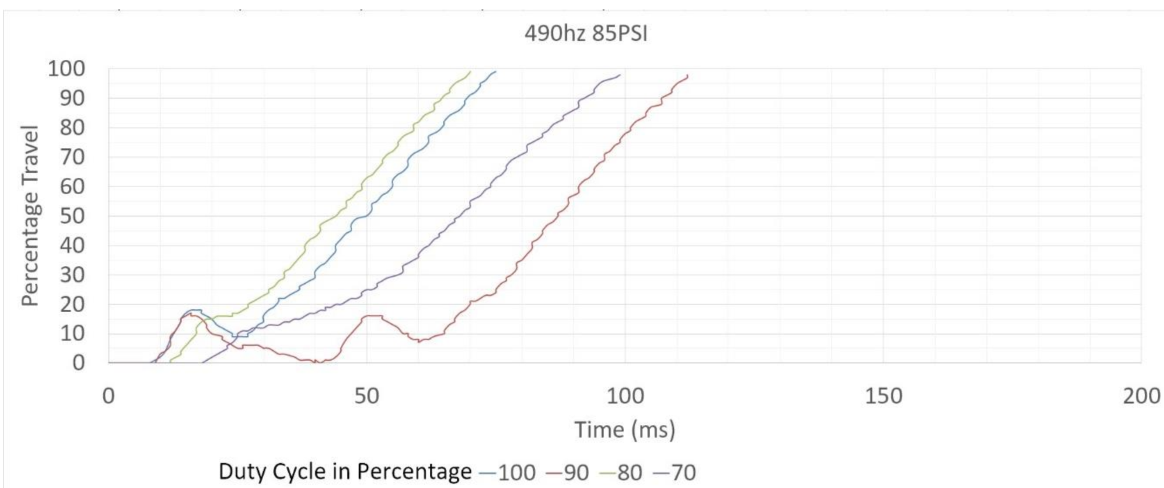

Figure 10 Graph for 490 Hz

From the above graphs, it is clearly visible that if the system is run at 30.64 Hz the actuator stops for a while before it starts moving ahead even at duty cycles as high as 70%. From the remaining three frequencies, it is observed that at 490.20 Hz the system cannot run below 70% duty cycle. Comparing 122.5 Hz and 245.1 Hz, for 122.5 Hz 70%

to 90% duty cycle the output is identical and hence does not provide much difference in operation at those duty cycles. Hence 245.1 Hz was selected as a carrier frequency.

3.4.2 Changes in the Program (operational logic)

The initial clutch program was based on the error between the input and the feedback which was tagged as error. This error value was directly proportional to the duty cycle. The response was smooth, but had issues of deceleration meaning that when the cylinder reached the end of cycle, the value of error reduced causing the duty cycle to reduce and this caused the cylinder to move slowly towards the end.

To resolve this issue, the following logic is used. The logic for negative error was the same as positive error except that the absolute value of the error is used

Figure 11 PWM duty cycle logic for clutch for positive error

The X axis in the graph shows the error value from 0-100% value and the Y axis shows the duty cycle in value from 0-100% value. As seen when the error value is less than 30% the error vs duty cycle follows a 2nd order polynomial curve to calculate duty cycle using error and it ranges from 30-40% duty cycle. This allows smooth movement of the clutch when minor changes are made to the input. The duty cycle shoots to 100% if the is more than 30%. This type of logic removed the deceleration issue. One more part of code was added to the logic which states.

```
“else if((error>=db)&&(error<=200)&&(pval>=850))
  analogWrite(mos1,240);
”
```

The above code states that if the error is more than 20% and the input is actuated more than 83% a 94% duty cycle is given to the system to disengage the clutch.

```
else if((error<=-db)&&(pval<=100)&&(error<=-200))
  analogWrite(mos2,240);
”
```

The above code states that if the input is less than 10% and the error is less than 20% a 94% duty cycle is given to the system to engage the clutch. A combination of the above codes helped remove the deceleration issue caused before.

The following graphs were generated with the final program. These graphs plot input, output and error with respect to time in milliseconds.

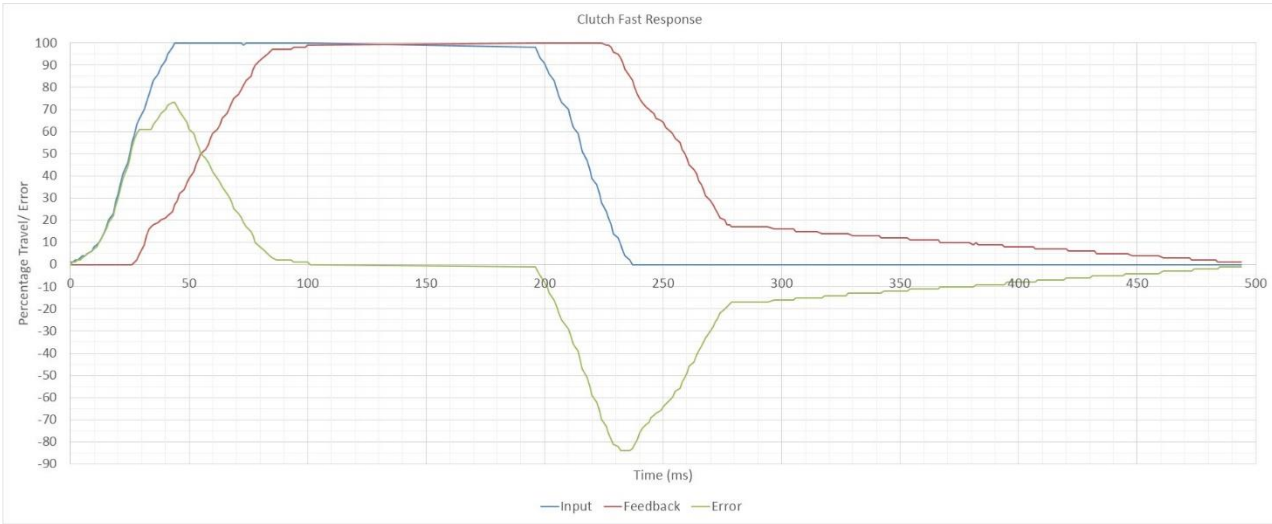

Figure 12 Clutch response fast release

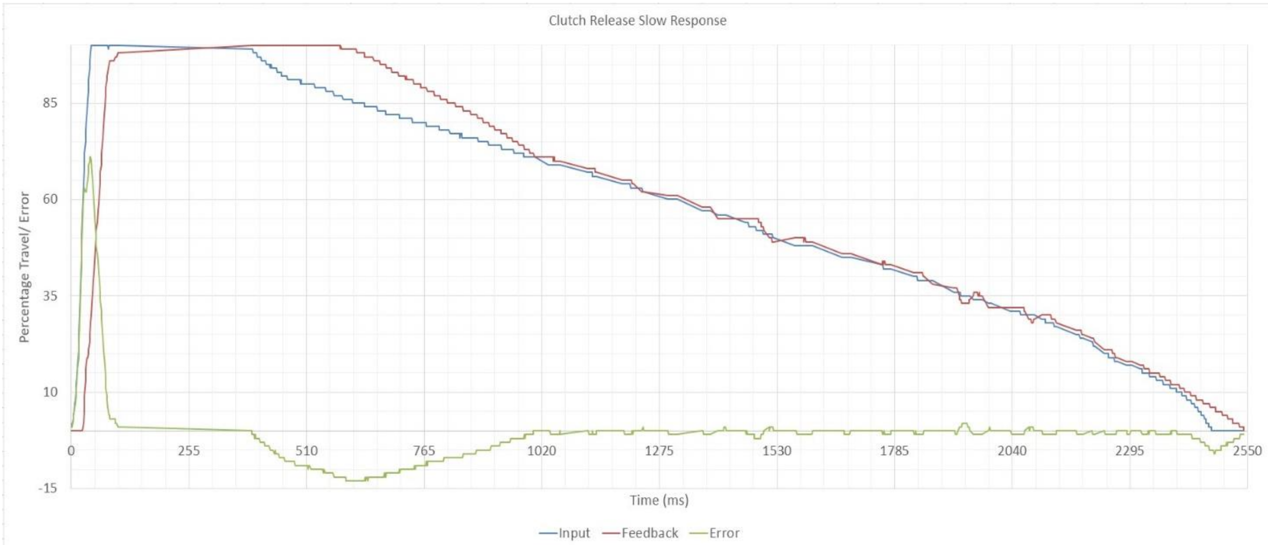

Figure 13 Clutch response slow release

Chapter 4

Chapter 5 Setup on FSAE Car with Extra Additions

The following chapter shows images of the setup on the car and how it was mounted. After seeing success in the setup on the car, the program was further developed to show a gear counter. The gear counter serves two purposes, firstly it helps understanding the gear the car is in as it is a sequential shifter, and secondly the program can be further developed to include automatic shifting.

4.1 FSAE Racecar Setup

Figure 14 Clutch cylinder setup

The above image shows the installation of the clutch cylinder on the engine with its feedback potentiometer and adjustable rod end bearing. The feedback potentiometer had a longer travel than the cylinder stroke, this was done to prevent damage to the potentiometer. The potentiometer travel is remapped on the controller.

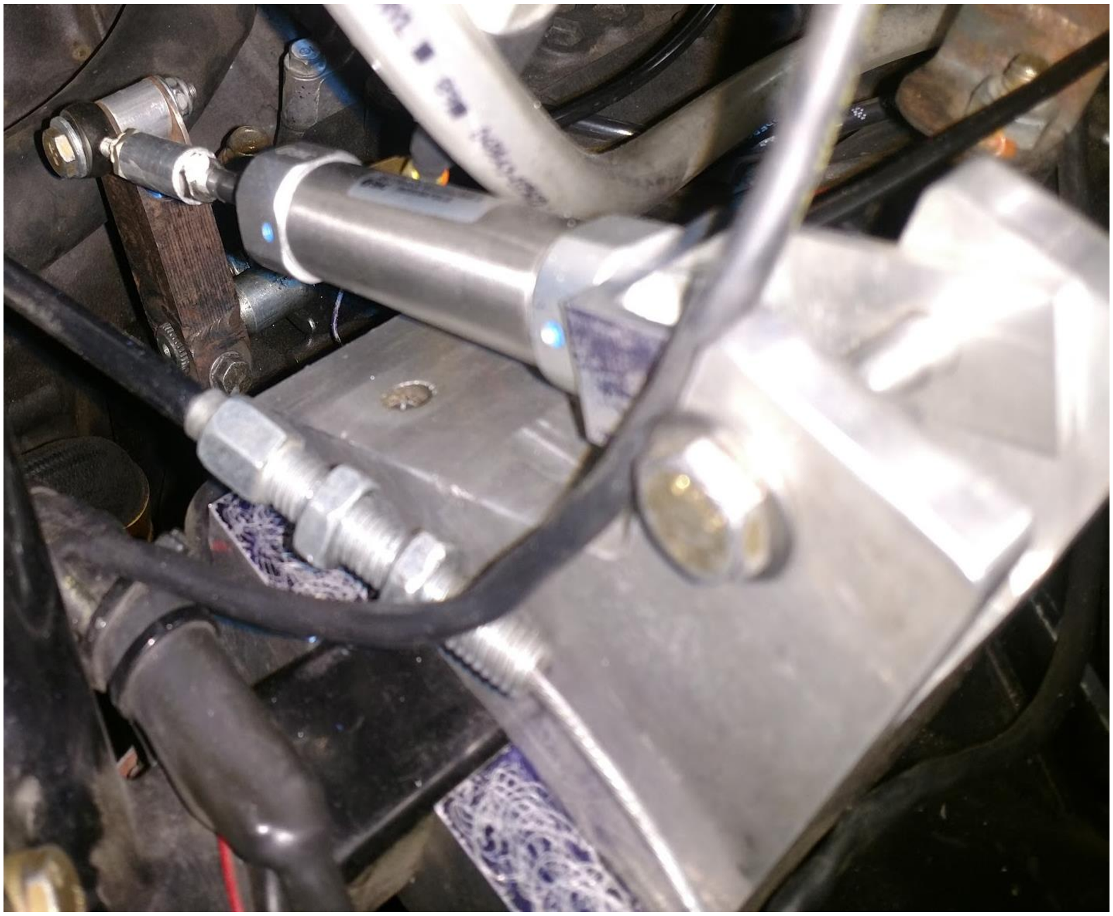

Figure 15 Shift cylinder setup

The above image shows the mounting of the shifter cylinder. A housing was made from billet aluminum to locate the cylinder. The arm length of the shift lever was made to match that of the spreadsheet, so that it can achieve the desired travel.

4.2 Change of microcontroller

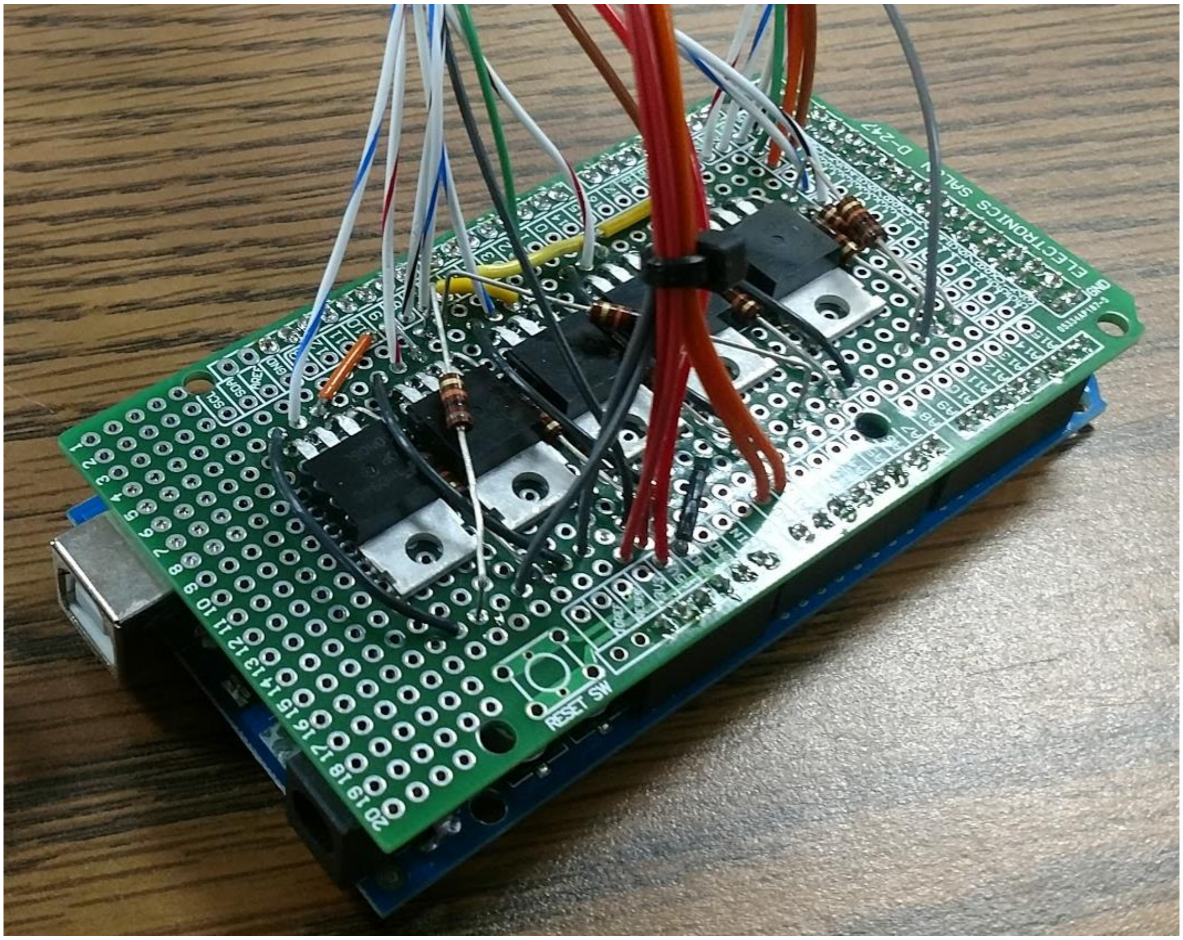

Figure 16 New board with improved wiring

A new board was implemented after seeing the success of the arduino uno r3, the new board is an arduino mega 2560. Programming the board is done using the same language and software hence it was selected. The advantages of this board over the old board is that it has more analog and digital inputs and outputs. Apart from additional input outputs, the controller also has more ram and more storage space for the program. Another reason to switch the board was that the gear counter that was to be implemented requires more inputs and outputs.

4.3 Gear Counter

The gear counter that was implemented has a 7-segment display as an output. The input to the controller comes from a magnetic reed switch which is driven by the pneumatic cylinder. The input state changes only when the reed switch receives a pick up from the magnet in the pneumatic cylinder when it reaches the end. This helps achieving a signal if and only if the cylinder reaches the end position.

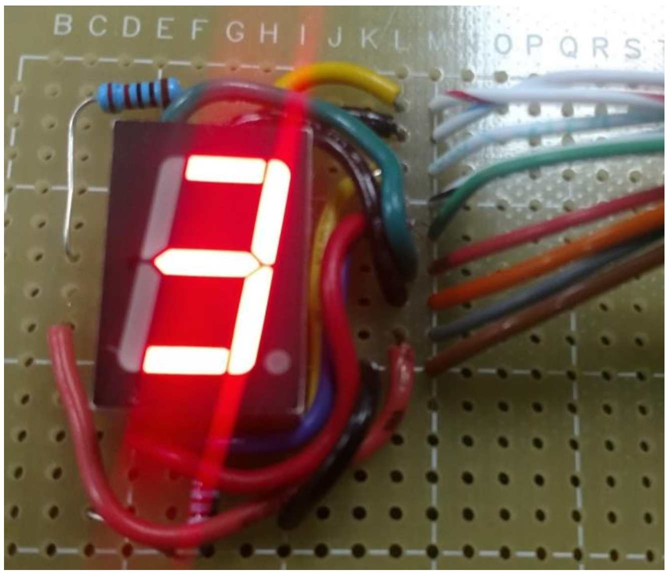

Figure 17 Gear counter showing gear count as 3

Figure 18 Shift cylinder with reed switch installed

The code for the gear counter is mentioned in the appendix. The code begins with defining a byte function to switch on and off the LED’s on the display. This reduces the length of the entire program. The gear counter resets to 0 denoting neutral each time it receives a signal from the neutral detect on the engine. The advantage of this approach is that it the system would always do correct counting whenever it crosses neutral. The code for the gear counter was modified from an existing code found online and is mentioned in appendix B. It was a basic push button counter and was modified per the working of the sequential shifter on the racecar.

Appendix A

Entire Code For the System

```
const int upsw=7;//upshift switch pin

const int downsw=8;//downshift switch pin

const int upsol=5;//the solenoid switch

const int downsol=6;//the solenoid switch

const int fbpot=A1;//feedback potentiometer pin

const int ppot=A0;//paddle potentiometer pin

const int mos1=9;//pin sending signal to MOSFET controlling cylinder input injector

const int mos2=10;//pin sending signal to MOSFET controlling cylinder exhaust injector

const int ns=2;//Neutral Detect


float pwmval=0;//pwm value for injectors

float error=0;//error between the paddle and feedback potentiometers

int fbval=0;//variable to store the feedback potentiometer value

int pval=0;//variable to store paddle potentiometer value


long unsigned int t1=0;

long unsigned int t2=0;


int st=35;//shift time

int ntrl=12;

int db=5;//deadband value (10 out of 950 units)


const int upPin = 32;                  // pushbutton attached to pin 32

const int downPin = 34;                  // pushbutton attached to pin 34

float currUpState = 1;                 // initialise currUpState as HIGH

float currDownState = 1;                  // initialise currDownState as HIGH
```

```
float prevUpState = 0;

float prevDownState = 0;

byte pins [] = {22, 23, 24, 25, 26, 27, 28, 29 };     // pin 9 allocated to DP but not used
(first element of binary array in char tenCode)

int counter = 1;                          // initialise counter as zero

byte timer = 10;                          // delay timer interval

int i[7] =  {0, 1, 2, 3, 4, 5, 6};

char tenCode[] = {B00000110, B01011111, B00111011, B00101111, B11100110,
B01101101, B01111101};

void setup() {

  // Code for setting pwm frequency

  TCCR0B = TCCR0B & B11111000 | B00000011;    // set timer 0 divisor to    64 for
PWM frequency of   976.56 Hz (The DEFAULT)

  TCCR1B = TCCR1B & B11111000 | B00000011;    // set timer 1 divisor to    64 for
PWM frequency of   490.20 Hz (The DEFAULT)

  TCCR2B = TCCR2B & B11111000 | B00000101;    // set timer 2 divisor to   128 for
PWM frequency of   245.10 Hz

//TCCR2B = TCCR2B & B11111000 | B00000110;    // set timer 2 divisor to   256 for
PWM frequency of   122.55 Hz

//TCCR2B = TCCR2B & B11111000 | B00000111;    // set timer 2 divisor to  1024 for
PWM frequency of    30.64 Hz

TCCR3B = TCCR3B & B11111000 | B00000011;    // set timer 3 divisor to    64 for
PWM frequency of   490.20 Hz  (The DEFAULT)

TCCR4B = TCCR4B & B11111000 | B00000011;    // set timer 4 divisor to    64 for
PWM frequency of   490.20 Hz  (The DEFAULT)
```

```
for(byte i = 0; i < 7; i++)                    // set digital pins as OUTPUTS
pinMode(pins[i], OUTPUT);
pinMode(upsw,INPUT);
pinMode(downsw,INPUT);
pinMode(upsol,OUTPUT);
pinMode(downsol,OUTPUT);
pinMode(fbpot,INPUT);
pinMode(ppot,INPUT);
pinMode(mos1,OUTPUT);
pinMode(mos2,OUTPUT);
pinMode(ns,INPUT_PULLUP);
Serial.begin(115200);


}


void loop()
{
pval=analogRead(ppot);
pval=map(pval,0,1023,0,1023);
fbval=analogRead(fbpot); //Reading the value of the feedback potentiometer
fbval=map(fbval,0,1023,0,1023);//"K" proportionality on feedback, for easier comparison
error=pval-fbval;
pwmval=(0.000018*pow(error,2))+75;//Formula to find the desired PWM value from
error
```

```
//The following conditions control cylinder retract motion.
if((error>=db)&&(error<300))  //polynomial proportional PWM value is determined
according to the formula as long as the error is less than 300 (out of 950 units).
  analogWrite(mos1,pwmval);
else if((error>=db)&&(error>=300)&&(pval>0))  //If error is more than 30% override the
duty cycle to 100% so as to achieve rapid movement.
  analogWrite(mos1,255);
else if((error>=db)&&(error<=200)&&(pval>=850))  //If error is less than 20% and input
is more than 85% keep duty cycle 95% so as to prevent deceleration when error reduces
  analogWrite(mos1,240);

//The following conditions control cylinder expansion i.e clutch engagement
if((error<=-db)&&(error>-300)) //polynomial proportional PWM value is determined
according to the formula as long as the error is less than 300 (out of 950 units)
  analogWrite(mos2,pwmval);
else if((error<=-db)&&(error<=-300)&&(pval=0)) //If error is more than 30% override the
duty cycle to 100% so as to achieve rapid movement.
  analogWrite(mos2,255);
else if((error<=-db)&&(pval<=100)&&(error<=-200))  //If error is less than 20% and input
is more than 85% keep duty cycle 95% so as to prevent deceleration when error reduces
  analogWrite(mos2,240);

// Following Command states that if error is within deadband not to control the solenoid
else if((error>-db)&&(error<db))
  {
    analogWrite(mos2,0);
```

```
    analogWrite(mos1,0);
  }
  if(abs(error)>=db)
  dataret();
//The next set of conditions is to control the shifter



if((digitalRead(upsw)==HIGH)&&(digitalRead(downsw)==LOW))   // case to define
upshift
{
  if(t1==0)
    t1=millis();
  else
    t1=t1;
  if((millis()-t1)<st)    //check if timer is less than shift time
  {
    digitalWrite(upsol,HIGH);   //enable upshift
   // digitalWrite(downsol,LOW);
  }
   else if(((millis()-t1)>=st)&&(digitalRead(downsw==LOW)))   //check if timer has
crossed shift time
  {
    digitalWrite(upsol,LOW);     //stop shift
   // digitalWrite(downsol,LOW);
  }
```

```
}
if((digitalRead(downsw)==HIGH)&&(digitalRead(upsw)==LOW))   //case to define
downshift
 { if(t1==0)
    t1=millis();
  else
    t1=t1;
  if((millis()-t1)<st)    //check if timer is less than shift time
 {
  digitalWrite(downsol,HIGH);   //enable downshift
 // digitalWrite(upsol,LOW);
 }
else if(((millis()-t1)>=st)&&(digitalRead((upsw)==LOW)))    //check if timer has crossed
shift time
 {
  digitalWrite(downsol,LOW);    //stop downshift
  //digitalWrite(upsol,LOW);
 }
}
if((digitalRead(downsw)==HIGH)&&(digitalRead(upsw)==HIGH)&&((millis()-t1)>=st))
//case to shift to neutral
  {
    if(t2==0)
      t2=millis();
    else
      t2=t2;
```

```
        if((millis()-t2)<ntrl)    //check if timer is less than neutral time
         digitalWrite(upsol,HIGH);   //initiate upshift
        else if(((millis()-t2)>=ntrl)||(ns==LOW))   //check if timer is greather than neutral time
or if neutral is detected from engine
         digitalWrite(upsol,LOW);    //stop upshift
       }
     else if((digitalRead(upsw)==LOW)&&(digitalRead(downsw)==LOW))   // case for no
shift
     {
       digitalWrite(downsol,LOW);    //keep both solenoids off
       digitalWrite(upsol,LOW);
       t1=0;
       t2=0;
     }
     currUpState = digitalRead(upPin);
        if (prevUpState != currUpState)           // has the state changed from
        {                                         // HIGH to LOW or vice versa
          prevUpState = currUpState;
          if (currUpState == HIGH)                // If the button was pressed
            counter++;                            // increment the counter by one

        }
        if(counter > 6)
          counter--;
        displayEleven(i[counter]);
```

```
    currDownState = digitalRead(downPin);
    if (prevDownState != currDownState)       // has the state changed from
    {                                          // HIGH to LOW or vice versa
      prevDownState = currDownState;
      if (currDownState == HIGH)              // If the button was pressed
        counter--;                             // decrement the counter by one
    }
    if(counter < 0)
      counter++;
    displayEleven(i[counter]);
  }

 void displayEleven(byte num)
 {
  byte mask = 1;
  for(byte i = 0; i < 7; i++)
  {
    if((mask & tenCode[num]) == 0)
      digitalWrite(pins[i], LOW);
    else digitalWrite(pins[i], HIGH);
    mask = mask << 1;
  }
 }
void dataret()//function to print (return) all required data
{ Serial.print(millis());
  Serial.print("\t");
```

```
    Serial.print(map(analogRead(ppot),0,1023,0,100));    //print value of input paddle from 0-100%
    Serial.print("\t");
    Serial.print(map(analogRead(fbpot),224,738,0,100));   //print value of feedback from 0-100%
    Serial.print("\t");
    Serial.print(map(error,0,1023,0,100));    //print value of error from 0-100%
    Serial.print("\t");
    Serial.print(pwmval);   //print duty cycle being calculated
    Serial.print("\t");
    Serial.print(analogRead(mos1));   //print duty cycle for mosfet 1
    Serial.print("\t");
    Serial.print(analogRead(mos2));   //print duty cycle for mosfet 2
    Serial.println();
  }
```

Appendix B

Original Gear Counter Code

The following code is used from an online source and the link is shared in the references [4].

This code is used to develop the shifter program and it is written in the following way.

```
// Thanks to Grumpy Mike  http://www.thebox.myzen.co.uk/Tutorial/Arrays.html
 // LED Segment allocation within byte = {DP ABCDEFG }

 byte upPin = 12;                              // pushbutton attached to pin 12
 byte downPin = 13;                            // pushbutton attached to pin 13
 byte currUpState = 1;                         // initialise currUpState as HIGH
 byte currDownState = 1;                       // initialise currDownState as HIGH
 byte prevUpState = 0;
 byte prevDownState = 0;
 byte pins [] = {2, 3, 4, 5, 6, 7, 8, 9 };     // pin 9 allocated to DP but not used (first element of
binary array in char tenCode)
 int counter = 0;                              // initialise counter as zero
 byte timer = 1000;                            // delay timer interval
 int i[10] =  {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
 char tenCode[] = {B01111110, B00110000, B01101101, B01111001, B00110011, B01011011,
B01011111, B01110000, B01111111, B01111011 };
 void setup()
 {
   for(byte i = 0; i < 8; i++)                 // set digital pins as OUTPUTS
     pinMode(pins[i], OUTPUT);
 }

 void loop()
 {
```

```
  currUpState = digitalRead(upPin);
  if (prevUpState != currUpState)          // has the state changed from
  {                                        // HIGH to LOW or vice versa
    prevUpState = currUpState;
    if (currUpState == HIGH)               // If the button was pressed
      counter++;                           // increment the counter by one

  }
  if(counter > 8)
    counter--;
  displayEleven(i[counter]);
  currDownState = digitalRead(downPin);
  if (prevDownState != currDownState)      // has the state changed from
  {                                        // HIGH to LOW or vice versa
    prevDownState = currDownState;
    if (currDownState == HIGH)             // If the button was pressed
      counter--;                           // decrement the counter by one
  }
  if(counter < 0)
    counter++;
  displayEleven(i[counter]);
 }

void displayEleven(byte num)
{
 byte mask = 1;
```

```
  for(byte i = 0; i < 8; i++)
  {
    if((mask & tenCode[num]) == 0)
      digitalWrite(pins[i], LOW);
    else digitalWrite(pins[i], HIGH);
    mask = mask << 1;
  }
}
```

Appendix C

Measurements and Spreadsheets

The following appendix is based on chapter 2 Design Approach. The forces and torque measured for the system are mentioned in the following tables.

Table 1 Force and Torque values for downshifts

Gear change	Force(lb)	Torque(lb-ft)
6th-5th	30	4.13
5th-4th	28	3.85
4th-3rd	30	4.13
3rd-2nd	28	3.85
2nd-1st	30	4.13

Table 2 Force and Torque values for upshifts

Gear Change	Force(lb)	Torque(lb-ft)
1st-2nd	28	3.85
2nd-3rd	28	3.85
3rd-4th	28	3.85
4th-5th	28	3.85
5th-6th	28	3.85

The torque required for clutch actuation were measured to be.

Table 3 Torque required by the clutch

Lever Position	Torque (lb ft)
Start Torque	4.83
Slip Torque	5.66
Maximum Torque	7.00

After attaining the above values a mathematical excel workbook was created. This workbook has several spreadsheets like Clutch force and cylinder dimension calculator, Shifter force and cylinder dimension calculator, compressed air tank sizing and Injector pulse calculator.

Clutch force and cylinder dimension calculator

	clutch sheet	64.00	in.Lbf
	FOS @ torque	1.31	
Measured	Clutch torque	84.00	in.Lbf
Set	Lever length	2.83	in
Measured	Clutch Force	29.63	Lbf
Set	Spring force @ bench	40.00	Lbf
Set	Factor of safety	1.80	
Set	PSI	100.00	psi
Set	Rod dia	0.25	in
Calculated	Rod end area	0.05	in^2
Calculated	Piston area	0.58	in^2
Calculated	Required area	0.53	in^2
Calculated	Piston dia	0.86	in
	Measured		
	Clutch full travel length	0.65	in
	Clutch slip travel length	0.36	in
	Clutch lever radius	3.00	in
	Calculated		
	Clutch full travel length	0.61	in
	Clutch slip travel length	0.34	in
	Clutch lever radius	2.83	in

	mm	in	
Bell crank radius	72.00	2.83	
radian to pi converter		57.30	
	(rad)	(degrees)	approximate
Clutch full travel angle	0.22	12.41	13.00
Clutch slip travel angle	0.12	6.88	7.00

Figure 19 Clutch force and cylinder dimension calculator

The above image shows a screenshot of the clutch force and cylinder dimension calculator spreadsheet. The primary inputs are the measured clutch torque, and the bell crank radius along with a defined factor of safety. After this the desired system pressure and approximate rod end diameter values are also defined.

The diameter determination goes on as following: -

1. Clutch force= (Clutch torque/lever length)

This helps determining the actuation force as the torque value is fixed.

2. Rod end area= ((pi/4)*(Rod end dia)^2)

This area is determined as a compensation value as the cylinder is a single acting spring extended one.

3. Required area= (factor of safety*Clutch force)/(Pressure)

This equation determines the minimum area required to actuate the cylinder.

4. Piston area= (Required area + Rod end area)

Since the cylinder used is a single acting spring extended, the rod end side would experience forces, but while buying a cylinder the piston diameter is considered, so it compensates for the area loss.

5. Piston dia= square root ((piston are*4)/pi)

This is the last equation which helps determine diameter required to actuate the cylinder.

Similarly, the stroke length was also calculated. This is included in the same sheet as it keeps it compact. The two set values here are the angle of rotation and the arm length.

The stroke length calculation goes on as follows: -

- Clutch full travel length= (Clutch travel angle (radians)*Clutch arm length)

Hence all the values in green and gray are either the ones that are measured or are defined, and the values in yellow are the calculated values.

Shifter force and cylinder dimension calculator

DOWNSHIFT			
Gear change	Force(lb)	Torque(lb-ft)	Torque(lb-in)
6th-5th	30	4.1	49.6
5th-4th	28	3.9	46.2
4th-3rd	30	4.1	49.6
3rd-2nd	28	3.9	46.2
2nd-1st	30	4.1	49.6
Maximum	30	lb	
UPSHIFT			
Gear Change	Force(lb)	Torque(lb-ft)	Torque(lb-in)
1st-2nd	28	3.9	46.2
2nd-3rd	28	3.9	46.2
3rd-4th	28	3.9	46.2
4th-5th	28	3.9	46.2
5th-6th	28	3.9	46.2
Maximum	28		

	Arm Length	2.04	in
	Total Max Force	30.00	lb
	Max force	28.00	Lbf
	FOS @ torque	1.07	
Measured	Shifter torque	30.00	in.Lbf
Set	Lever length	2.04	in
Measured	Shift Force	14.71	Lbf
Set	Factor of safety	1.65	
Set	PSI	90.00	psi
Set	Rod dia	0.20	in
Calculated	Rod end area	0.03	in^2
Calculated	Piston area	0.30	in^2
Calculated	Required area	0.27	in^2
Calculated	Piston dia	0.62	in

angle of rotation	16	degrees
angle of rotation	0.28	rad
pi to radian	0.02	
Length of travel	0.57	in
approx	0.60	in

Figure 20 Shifter force and cylinder dimension calculator

The above image shows a screenshot of the shifter force and cylinder dimension calculator. The tables on the left are measured values of force required to initiate shifts while upshifting and downshifting. Neutral could not be calculated as it was a small force which could not be measured properly without any trouble, it was decided to send a smaller pulse to shift to neutral. Using the table on the left the maximum of the two torques was selected along with a desired arm length. This helped calculating force required. Along with these values the system pressure and factor of safety was used as an input.

The diameter determination goes on as following: -

1. Shift force= (Shift torque/arm length)

   This helps determining the actuation force as the torque value is fixed.

2. Rod end area=(pi/4)*(Rod end dia)^2)

   This area is determined as a compensation value as the cylinder is acting in both directions.

3. Required area=(factor of safety*Shift Force)/(Pressure)

   This equation determines the minimum area required to actuate the cylinder.

4. Piston area=(Required area + Rod end area)

Since the cylinder is actuated in either direction the smaller area needs to be the bare minimum as the rod end side would experience forces, but while buying a cylinder the piston diameter is considered, so it compensates for the area loss.

5. Piston dia= square root ((piston are*4)/pi)

This is the last equation which helps determine diameter required to actuate the cylinder.

Similarly, the stroke length was also calculated. This is included in the same sheet as it keeps it compact. The two set values here are the angle of rotation and the arm length.

The stroke length calculation goes on as follows: -

- Shifter travel length= (shifter travel angle (radians)*shifter arm length) *2

The above value is multiplied by two because the shift arm rotates in either direction, depending on the type of shift.

Compressed air tank sizing

			Desired for F-17						
SHIFTER									
Bore	0.63	in		16	mm				
Stroke	0.59	in		15	mm				
Pressur	100	psi	Sides used	2					
Cycles	1000	cyc							
		1	Area Of Piston	0.31	sq.in				
		2	Consumption/Stroke	0.18	cu.in				
		3	Consumption/Cycle	0.37	cu.in				
		4	Vol. consumed	368.09	)0 psi for 1000 cycles				
CLUTCH									
Bore	0.875	in							
Stroke	1	in							
Pressur	100	psi	Sides used	1					
Cycles	1000	cyc							
		1	Area Of Piston	0.60	sq.in				
		2	Consumption/Stroke	0.60	cu.in				
		3	Consumption/Cycle	0.60	cu.in				
		4	Vol. consumed	601.32	)0 psi for 1000 cycles				
		TOTAL VOLUME CONSUMED		969.41	.in @ 100 psi				
working					tank				
P1	100	psi	7.03	kg.cm^2	P2	4000	psi	281.23	kg.cm^2
V1	969.41	cu.in	15885.77	cm^3	V2		cu.in		cm^3
T1	70	F	294.11	Kelvin	T2	70	F	294.11	Kelvin
Combined Gas Law									
V2	24.24	cu.in							
V2	397.14	cm^3							

Figure 21 Compressed air tank sizing

The above screenshot shows the spreadsheet for sizing the tank based on the number of shifts. It uses the cylinder bore and stroke for the clutch and bore and half stroke for the shifter. Number of shifts were predefined by calculating the average shifts per lap and multiplying it by the number of laps plus and extra of a few hundred shifts. The equations in the outlined box

The following equations were used.

1. Area of piston= ((pi*Bore^2)/4)

This equation calculates the area of the piston.

2. Consumption/ stroke= (area of piston*actuator stroke length)

This equation calculates the volume consumed by the cylinder each time it is actuated.

3. Consumption/ cycle= (consumption/ stroke *sides used)

This equation changes the consumption from stroke based to cycle based which changes in case of the shifter as it uses both sides.

4. Volume consumed= ((consumption/ cycle) * number of cycles)

This value is the volume consumed for the individual cylinders. They are then added to determine the final total volume consumed.

After the volume is determined Combined gas law was used to determine if the tank used enough. The equation is as following.

$$\frac{P_1 V_1}{T_1} = \frac{P_2 V_2}{T_2}$$

In the above equations; P1, P2 are pressures, T1, T2 are temperatures and V1, V2 is the Volume of the gas at state 1 and state 2 respectively. This is not the most accurate method as it is an ideal law, but it helps determine a baseline.

Injector pulse calculation

			PWM value			Percentage			Error	
Frequency	245.1	Hz	min	0		min	0		min	0
IJPU	4.079967	ms	max	255		max	100		max	1000
			set	70		calculated	27.45098			
Pulse width	1.119991	ms								

Figure 22 Injector pulse calculation

This sheet was generated to determine the frequency at which the pulse width modulation would run. It also helps determining the maximum on time for the injector. This is necessary as too high a frequency would not allow the injector to run or too low a frequency the response would be slower.

The following equations were used: -

- Pulse width maximum in ms= ((1/Frequency) *1000)

The carrier frequency is inversed and then multiplied by a thousand to get a value in milliseconds. The Pulse width mentioned in the screenshot above is the actual value of on-time for the PWM at that duty cycle.

Appendix D

Programming the Controller

The following appendix explains the wiring and the code used for the microcontroller. The circuit was connected the following way.

Table 4 Microcontroller wiring for test bench

Microcontroller pin (Type)	Component	Use
7 (Digital input)	Right Switch	Upshift Button
8 (Digital input)	Left Switch	Downshift Button
5 (Digital output)	Gate of MOSFET Up	Upshift Solenoid
6 (Digital output)	Gate of MOSFET Down	Downshift Solenoid
5 (Digital output)	Gate of MOSFET Ignition cut	ECU for Ign cut
A1 (Analog output)	Potentiometer Feedback	Clutch position sensor
A0 (Analog output)	Potentiometer Input	Clutch Input Paddle
3 (Digital output)	Gate of MOSFET Cylinder Return	Clutch Disengage
11 (Digital output)	Gate of MOSFET Cylinder Expand	Clutch Engage

All the digital input and output pins were connected to ground using a 10K-ohm resistor to set a set a logic state of LOW. The idea behind this was so that the pins do not stay floating/ may not change state because of noise. This was done for the inputs so that whenever a button is pressed it changes the pin state to HIGH changing the logic state. For the outputs this was implemented for the MOSFETS as it kept the Gate signal LOW thereby keeping the circuit open. Whenever the microcontroller sends a 5-volt signal to the MOSFET the Gate state changes from LOW to HIGH thereby closing the circuit between the DRAIN and SOURCE. The SOURCE of all the MOSFETs were connected to the common ground as they were N-Channel ones. The positive of the solenoids and the injectors were connected to +12 Volts of a battery. The MOSFETs DRAIN was connected to the ground terminal of the respective solenoid or the injectors.

Programming the microcontroller

Initial setup

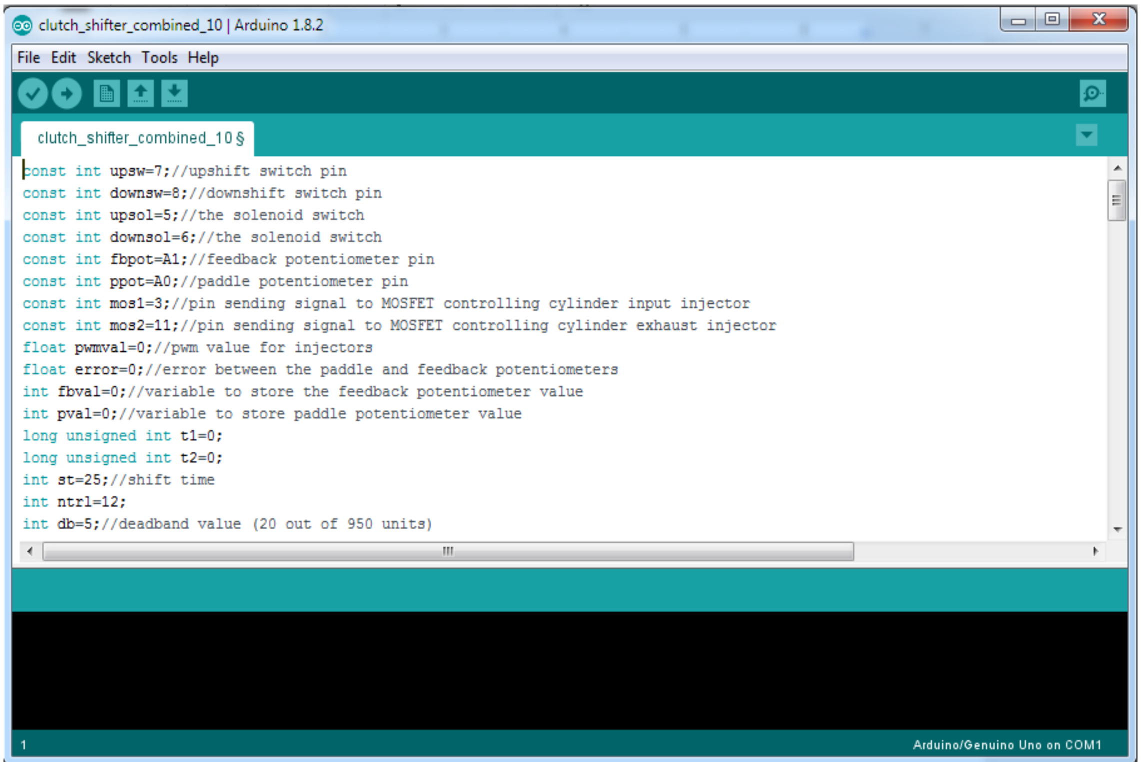

Figure 23 Arduino IDE initial setup

The Arduino UNO R3 microcontroller is programmed using a software called as Arduino IDE. The program starts with the setup of the pins. This is done using the command “const int” followed by the variable and then the pin number. “Const int” is used to define the pin number which is used for the MOSFET or switch. Then some of the variable values are defined these consist of the error value and the value for the duty cycle. These are defined using “float” command as their values keep on changing when the loop runs. The analog readings of the input and feedback are defined using an int function which are initially defined/ reset to zero in the program. Timer variables were defined using “long unsigned int” function. This was implemented in the code to calculate the time the shift solenoid is enabled. More integers were defined for example the shift time(st), neutral(ntrl) and deadband(db). These were given defined values and were used to tune the system, this helps as the entire code does not need to change, only the values in the initial setup are changed. For example, a data type is defined in the following way “const int upsw=7; “, here “const int” is the data type, “upsw “is the variable that is defined so that it can be used while programming and “7” is the pin number on the microcontroller board. The command ends with a semicolon which signifies that the command ends there.

Void Setup

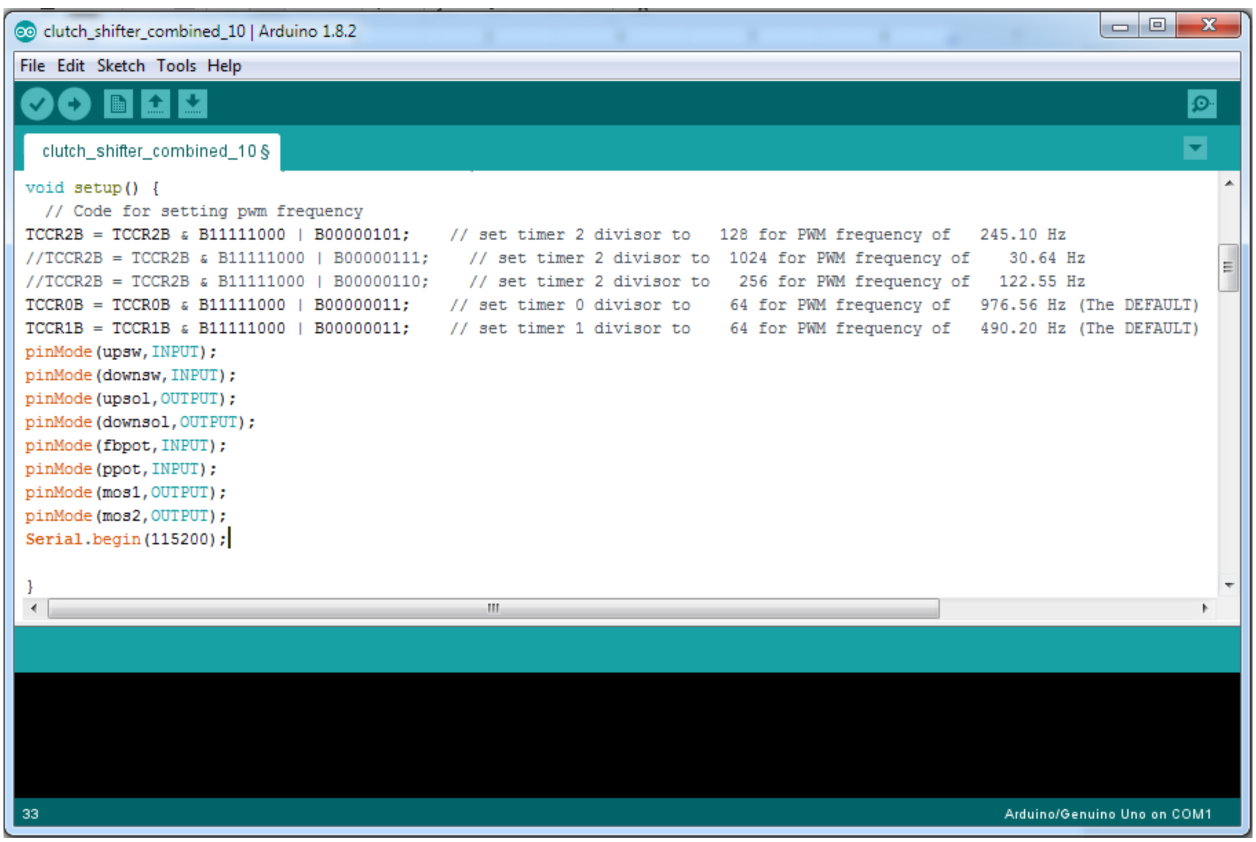

Figure 24 Arduino IDE void setup

After the initial integers and floats are defined, it is necessary to define the microcontroller pins as inputs or outputs, and to define the clock speed of the timers so that the pwm frequency can be changed.

The void setup information is written between two curly brackets. The initial part under the void setup defines the divisor of the individual timers so that the carrier frequency of the Pulse Width Modulation can be changed. The divisor of timer 0 is kept to the default value as it sets the frequency to 976.56 Hz which is closest to 1000 Hz, this is useful to run the timers for the shifter which was mentioned in the previous section. After the timers are set up the pins are set and defined if they are inputs or outputs using “pinMode” function. This function is used in the following way “pinMode(upsw,INPUT); “ here upsw is the variable defined in the initial setup, “INPUT” defines it as an input, so that the controller knows if it has to read that value or write a value there. “Serial.begin(115200) “is a communication protocol that is used to send data back to the computer to read the data, it is used as a method to log or check live data.

Void Loop

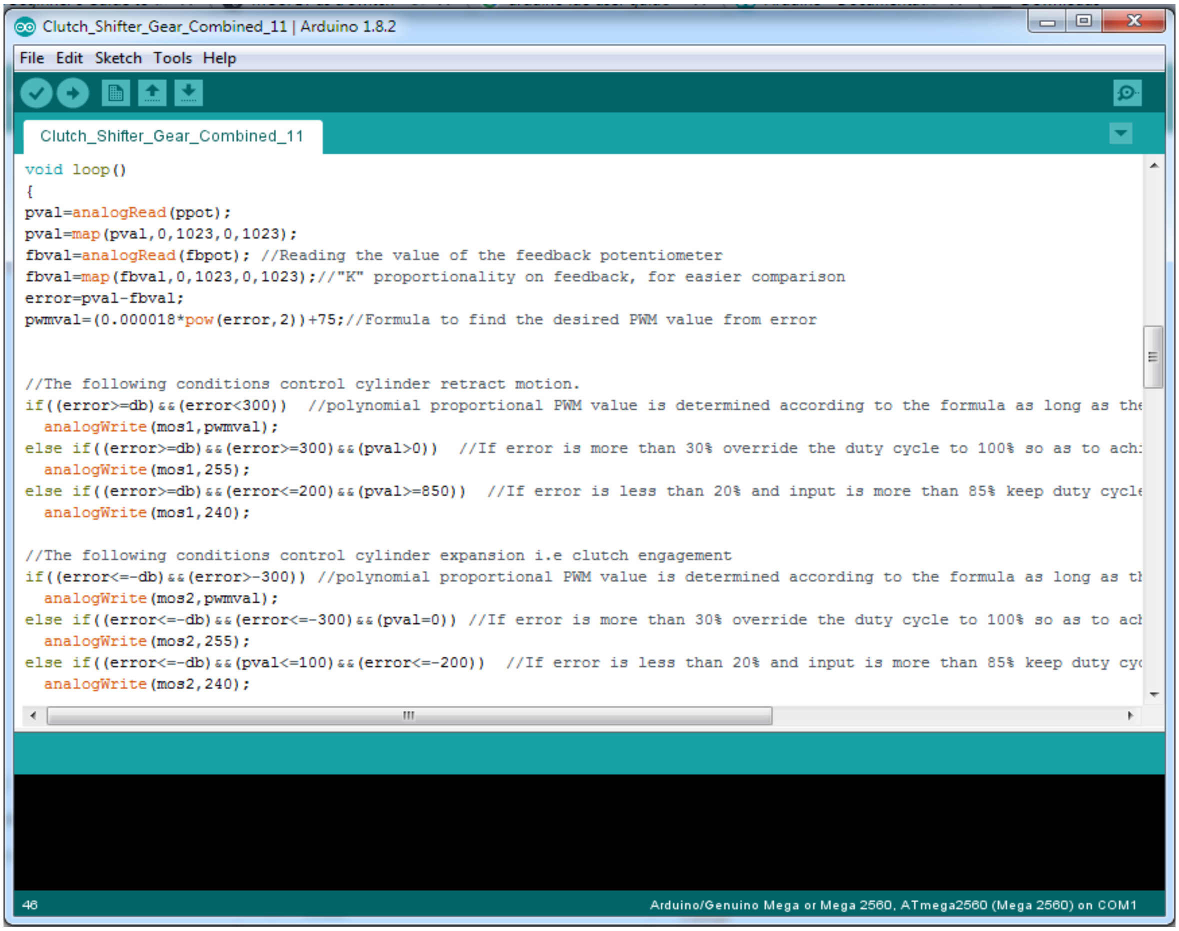

Figure 25 Arduino IDE void loop

The void loop is the section where the code for the working of the entire system is written. The previous two sections were to define the system. This part of the code includes statements like if, else if, and is basic C programming. The part of the code displayed in the above screenshot defines how the clutch should operate. It starts with re-defining variables mentioned in the first section of the programming part by reading the pins on the microcontroller. A “map “math function is used to interpolate the values read on the pins to match them while calculating error. A variable “error” is introduced which is nothing but the difference between the input paddle and the feedback from the clutch. This value is used to change the duty cycle at which the MOSFET is pulse width modulated. A predefined value of “deadband” is used in the program which helps the output settle at a faster rate, like an overdamped system, so that it does not keep oscillating while getting the error zero. This causes an intensional hysteresis in the system but helps the system respond smoother and without any jitter.

Serial Print

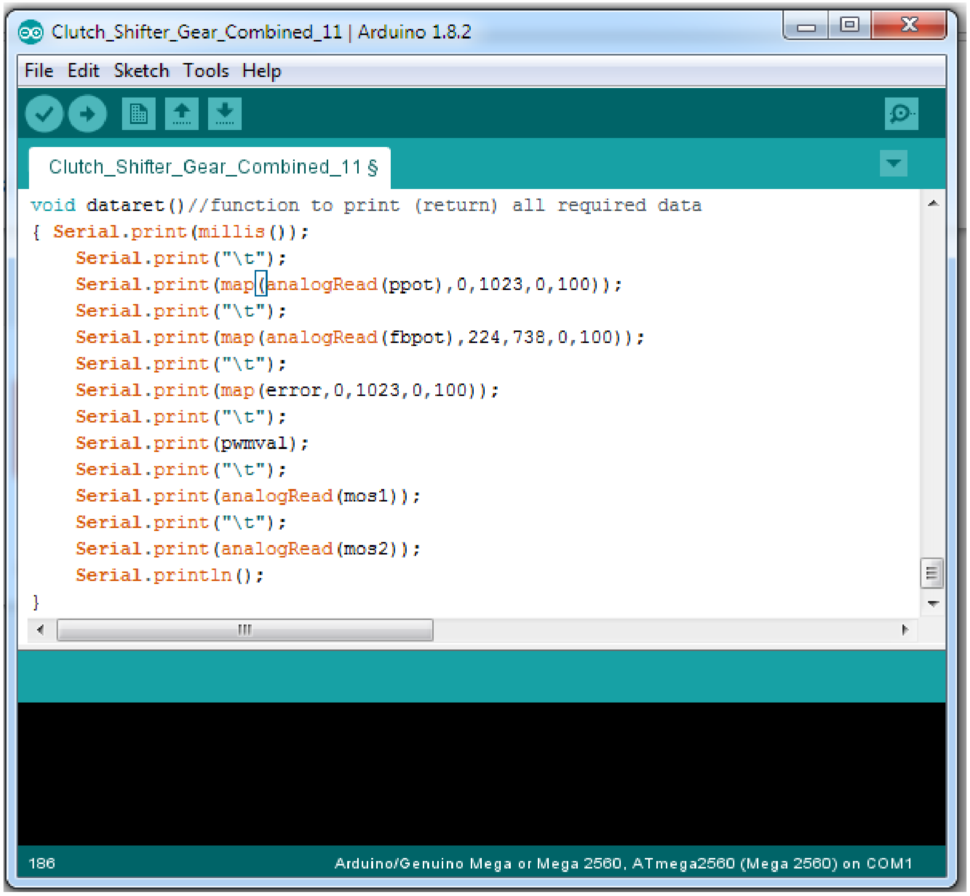

Figure 26 Arduino IDE serial print

Arduino IDE has a function called as “dataret() “ this is used when the values of the variables are required to print as output. The command works in the following way, “Serial.print(millis()); where “Serial.print” is the print function and “millis()” is the data that needs to be printed, so it prints the timestamp of the counter. Similarly map(analogRead(fbpot),224,738,0,100)) reads the 10-bit value for the feedback potentiometer and maps it to a 0-100% range so that the values are comparable on the same scale.

Appendix E

Test Code for Shifter and Clutch

Shifter Program

The following code was initially written to implement the gear shifting logic: -

```
void loop() {
  // put your main code here, to run repeatedly:
dst=digitalRead(downs);
ust=digitalRead(ups);
 if((dst==LOW)&&(ust==LOW))
  {
    digitalWrite(D,LOW);
    digitalWrite(U,LOW);
  }
else
{
if(ust==HIGH)
{ digitalWrite(U,HIGH);
  digitalWrite(D,LOW);
  delay(st);
  digitalWrite(U,LOW);
}
 if(dst==HIGH)
  {
    digitalWrite(U,LOW);
    digitalWrite(D,HIGH);
    delay(st);
    digitalWrite(D,LOW);
  }      }        }
```

This code runs on a delay function and the delay time was set by the time the shift lasted viz (st). The drawback of this system is that when the delay occurs the controller is sitting idle and does not do anything or is incapable of performing any task. This issue is corrected using a timer which is mentioned in the setup part of the programming.

The following code was implemented: -

```
if((digitalRead(upsw)==HIGH)&&(digitalRead(downsw)==LOW))
{
 if(t1==0)
  t1=millis();
 else
  t1=t1;
 if((millis()-t1)<st)
 {
  digitalWrite(upsol,HIGH);
 // digitalWrite(downsol,LOW);
 }
 else if(((millis()-t1)>=st)&&(digitalRead(downsw==LOW)))
 {
  digitalWrite(upsol,LOW);
 // digitalWrite(downsol,LOW);
```

Here the “millis “ function is used, millis is a timer in milliseconds which starts as soons as the controller is switched on. Millis is always increasing. A variable “t1” is defined as millis as soon as the button is pressed, and the logic defines that if the difference between millis and t1 is less than the shift time it starts the solenoid, or else as soon as the the button is released and the difference between millis and t1 is greater than shift time it cuts the signal to the solenoid.

The shift time is a value which was initially set to 50 ms but is reduced to 35 ms when installed on the car.

Clutch Program

The program for the clutch initially started by determining the carrier frequency of the PWM (Pulse Width Modulation). PWM was implemented because it can be operated at various speeds and the duty cycle that is used can be used as a function of the error in the system.

The initial test that helped determining the carrier frequency is by writing a program which defines a base frequency and plot the movement of the actuator with respect to time (x-axis) and position (y-axis).

An example of one of the codes used is given below: -

```
const int upsw=7;//upshift switch pin
const int downsw=8;//downshift switch pin
const int fbpot=A1;//feedback potentiometer pin
const int ppot=A0;//paddle potentiometer pin
const int mos1=3;//pin sending signal to MOSFET controlling cylinder input injector
const int mos2=11;//pin sending signal to MOSFET controlling cylinder exhaust injector
float pwmval=0;//pwm value for injectors
float error=0;//error between the paddle and feedback potentiometers
int fbval=0;//variable to store the feedback potentiometer value
int pval=0;//variable to store paddle potentiometer value
int db=10;//deadband value (10 out of 950 units)
int pwm=76.5;
void setup() {
  // Code for setting pwm frequency
```

```
TCCR2B = TCCR2B & B11111000 | B00000101;    // set timer 2 divisor to   128 for
PWM frequency of   245.10 Hz
//TCCR2B = TCCR2B & B11111000 | B00000111;    // set timer 2 divisor to  1024 for
PWM frequency of    30.64 Hz
//TCCR2B = TCCR2B & B11111000 | B00000110;    // set timer 2 divisor to   256 for
PWM frequency of   122.55 Hz
TCCR0B = TCCR0B & B11111000 | B00000011;    // set timer 0 divisor to    64 for
PWM frequency of   976.56 Hz (The DEFAULT)
TCCR1B = TCCR1B & B11111000 | B00000011;    // set timer 1 divisor to    64 for
PWM frequency of   490.20 Hz (The DEFAULT)
pinMode(upsw,INPUT);
pinMode(downsw,INPUT);
pinMode(fbpot,INPUT);
pinMode(ppot,INPUT);
pinMode(mos1,OUTPUT);
pinMode(mos2,OUTPUT);
Serial.begin(115200);
}
void loop()
{
pval=analogRead(ppot);
pval=map(pval,0,1023,0,1023);
fbval=analogRead(fbpot); //Reading the value of the feedback potentiometer
fbval=map(fbval,224,738,0,1023);//"K" proportionality on feedback, for easier
comparison
error=pval-fbval;
```

```
if (error>=db)
analogWrite(mos1,pwm);
if (error<=-db)
analogWrite(mos2,pwm);
else if((error>-db)&&(error<db))
  {
    analogWrite(mos1,0);
    analogWrite(mos2,0);
  }
  if(abs(error)>=db)
  dataret();
}
void dataret()
    {
      Serial.print(millis());
      Serial.print("\t");
      Serial.print(map(analogRead(fbpot),224,738,0,100));
      Serial.println();
    }
```

References

1. http://www.eng.fiu.edu/mme/robotics/EML4551SeniorDesignOrg/Presentation/2015Spring/Team07.pptx
2. http://www.ijetae.com/files/Volume5Issue7/IJETAE_0715_58.pdf
3. http://www.fsae.com/forums/showthread.php?8805-Pneumatic-Clutch
4. http://www.pedroduino.com/7SegLEDcircuits.php
5. https://arduino-info.wikispaces.com/Timers-Arduino
6. https://learn.sparkfun.com/tutorials/pulse-width-modulation
7. http://www.fsaeonline.com/content/2017-18%20FSAE%20Rules%209.2.16a.pdf
8. Electronic Shifting and Clutch Controls, Internal Report, UTA, MAE 4188, August 2011.
9. Pneumatic Clutch, Internal Report, UTA, MAE, August 2011.
10. Electronic Shifting and Clutch Control, Internal Report, UTA, MAE, August 2012.
11. Electronic Shifting, Internal Report, UTA, ME 5358, August 2016.
12. Control System Components, Course work, ME 5341, Robert L. Woods.

Biographical Information

Rahul Chalmela has a Master's of Science degree in Mechanical Engineering from The University of Texas at Arlington in 2017, Texas, USA. He earned his Bachelor's degree in Mechanical Engineering from University of Mumbai in 2014. He did his Diploma in Mechanical Engineering (technical course) from Maharashtra State Board of Technical Education in 2011 from Mumbai.

Rahul has been associated with FSAE since 2013 on a formula student team Orion Racing India (ORI). He joined the team and immediately was elected as the Air Intake and Exhaust Lead for the competition, Formula Student Germany 2014. After finishing his bachelors, he worked on the team as alumni and took lead of the entire powertrain system, and was also the autocross and endurance driver.

He joined UTA in August 2015 as a Graduate Master's student and got associated with UTA Racing since October 2015, where he worked as a deputy Lead for the powertrain system for the 2016 Michigan competition. For the 2017 Lincoln competition, he was elected lead for the entire powertrain system.

Rahul's research interest was the powertrain development and the summer project of the pneumatic shifter and servo clutch which he later took as his thesis so that he could develop an efficient, reliable and robust system to change the type of shifting.