

**Address:**

To: Dr. Christopher Peters and Mr. David Cinciruk

From: Philip Mak and Sanchet Hoque

Date 4/4/2019

Re: Lab 1 - Equipment Automation

**Introduction:**

The purpose of Lab 1 is to learn how to automate the digital multimeter (DMM), arbitrary waveform generator (AWG), and oscilloscope via Matlab or Python using USB commands. The group will gather a few points manually and then automate a large portion of data for each lab instrument in order to see if the automated data is similar to manually collected data. In the chosen programming language, students will apply linear regression techniques to the collected data.

**Method/Analysis**

For the first part, 1A, students setup a basic circuit involving the the DMM, AWG, and a 2k resistor. Current was measured through the resistor as students performed a voltage sweep from 0 to 10 Volts in 1 Volt increments, recording the resultant currents and calculating the resistance through Ohm's Law. The resistor itself had a resistance of 1977 ohms, while the calculated resistance based on Ohm's Law of  $V=IR$ , knowing voltage and current, was 2038 ohms.

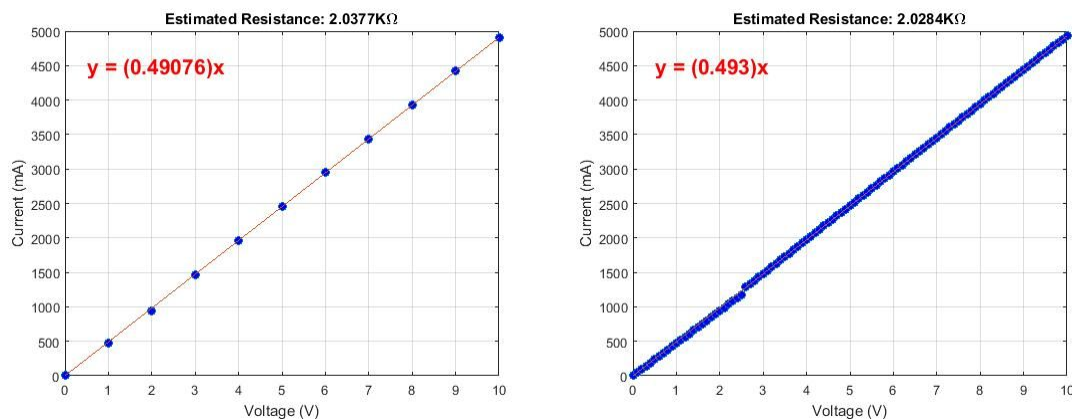


Figure I : Graph of Manual (Left) and Automated (Right) Resistance Estimation

For part 1B, the same sweep was performed via automation with a higher resolution for Voltage. Students obtained the specific address for each of the equipment and utilized the code provided in the lab package. The code used the VISA-USB object for the hardware to software connection. The calculated resistance for the automated testing was 2028 ohms, closer than the manual measurements. In the graph of the automated estimation, there is a jump at 2.5V. This happens due to the DMM switching circuits which causes an error.

For part 1C, the oscilloscope and AWG were used to measure the on-time and frequency of a periodic pulse manually with cursors on the oscilloscope. An automatic retrieval of the frequency was done via SCPI commands with 25 pulses in steps of 5% duty cycle from 20%, to 80%, and then back to 20%. The 25 on-time values were calculated by multiplying the duty cycle by the period. These 25 values vs duty cycle were used to plot the theoretical line in Figure II. The measured line (Right) was plotted in a similar fashion except the number of points that were considered high was multiplied by dt to obtain the on-time. The left plot on-times were calculated by multiple the measured frequency by the duty cycle.

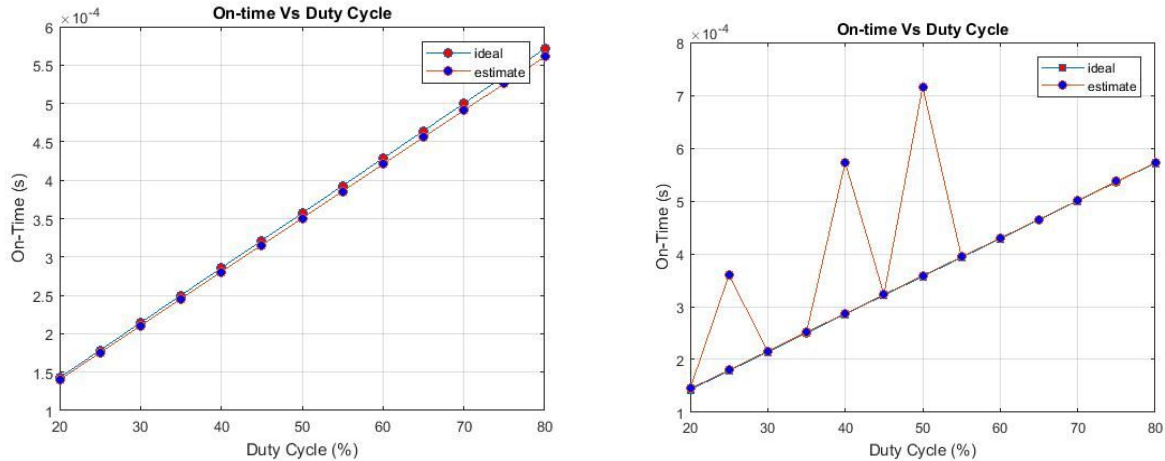


Figure II : Estimated and Ideal On-Time vs Duty Cycle using Measured Frequency (Left) and dt (Right)

### Discussion and Conclusion

Automation of lab equipment removes the human element from the tedious measuring process. This ideally makes all operations more accurate and precise without the slip of human error. Regarding potential changes to the operation, the explanation on duty cycle calculation can be revised for clarity. In Figure II, the right graph shows spikes in on-time as it goes from 20-80-20% duty cycle. The reason there are spikes lies within the data set used, as they had more than one on-time

**Address:**

To: Dr. Christopher Peters and Mr. David Cinciruk

From: Philip Mak and Sanchet Hoque

Date 4/19/2019

Re: Lab 2 - Week 1 (Prelab)

**Introduction:**

The purpose of the first week of Lab 2 was to familiarize students with Arduino programming along with software integration with MATLAB and lab equipment (DMM,AWG,Oscilloscope.) For the first part, 2A, students wrote an Arduino code that takes DC signal inputs and outputs a PWM signal, with a linearly increasing duty cycle. In the second part, 2B, students wrote an Arduino code that counts pulses from a signal. To finish the prelab in part 2C, students wrote an Arduino sketch alongside a MATLAB code that interact with each other. All of this work in the prelab would be later used in the Lab 2 - Week 2 (Lab.)

**Method/Analysis**

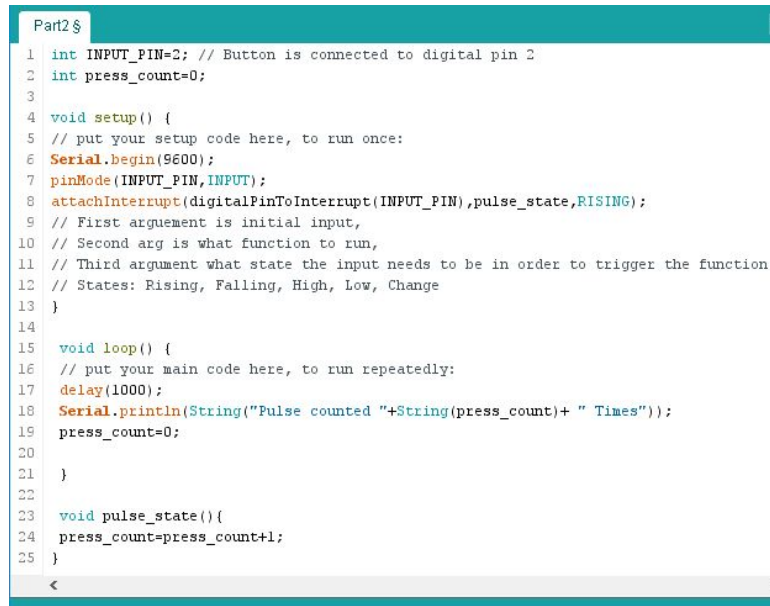
For part 2A, the students were required to have the Arduino output a PWM signal at 980.3Hz with a changing duty cycle which is linear to the DC input from 0 to 5V. In the Arduino, 5V is 1023 and 100% duty cycle is 255. In lines 4 and 6 of Figure 1, these were initialized and used in the loop. To relate the input to the duty cycle, a simple ratio was calculated: input voltage value divided by max voltage. This value was then multiple by the max duty cycle to use as the output duty cycle for the PWM. Serial Print was added for debugging.



```
Part1
1 int INPUT_PIN=A0;
2 float val = 0.0;
3 float dutycycle = 0.0;
4 float maxvol= 1023.0;
5 int OUTPUT_PIN=5;
6 float maxdc=255;
7
8
9 // pinMode(INPUT_PIN,INPUT);
10
11 void setup() {
12 // put your setup code here, to run once:
13 Serial.begin(9600);
14 //pinMode(INPUT_PIN,INPUT);
15
16 }
17
18 void loop() {
19 // put your main code here, to run repeatedly:
20
21 val=analogRead(INPUT);
22 //Serial.println(val);
23 dutycycle=(val/maxvol);
24
25 analogWrite(5, dutycycle*maxdc);
26
27 Serial.println(dutycycle);
28
29 }
```

Figure I : Arduino sketch of part 2A - DC input to PWM output

For part 2B, the students had the Arduino count the number of pulses of a signal which is displayed on the serial com each second. Since this is no output signal, only a input pin was initialized. In the setup function, the serial and interrupt were implemented. The interrupt is triggered when there is a rising pulse and the function pulse state is run. Pulse state counts the number of rising edges in the current pulse. In the main loop, a 1000ms delay is placed to account for the counting only per 1 second. The serial print is also put right after in the main loop.



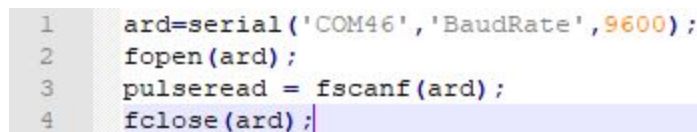
```

Part2$
1 int INPUT_PIN=2; // Button is connected to digital pin 2
2 int press_count=0;
3
4 void setup() {
5   // put your setup code here, to run once:
6   Serial.begin(9600);
7   pinMode(INPUT_PIN,INPUT);
8   attachInterrupt(digitalPinToInterrupt(INPUT_PIN),pulse_state,RISING);
9   // First argument is initial input,
10  // Second arg is what function to run,
11  // Third argument what state the input needs to be in order to trigger the function.
12  // States: Rising, Falling, High, Low, Change
13 }
14
15 void loop() {
16   // put your main code here, to run repeatedly:
17   delay(1000);
18   Serial.println(String("Pulse counted "+String(press_count)+ " Times"));
19   press_count=0;
20 }
21
22
23 void pulse_state(){
24   press_count=press_count+1;
25 }

```

Figure II : Arduino sketch of part 2B - Pulse Counting

For part 2C, the students had to read the values from their Arduino code. The built in serial commands are used to simply read the output in the Com port. Fscanf was used to convert into a readable value.



```

1 ard=serial('COM46','BaudRate',9600);
2 fopen(ard);
3 pulserread = fscanf(ard);
4 fclose(ard);

```

Figure III : Part 2C - MATLAB code interacting with the code from Part 2B

## Discussion and Conclusion

The introduction of the microcontrollers into this project is important as the Arduino generates a PWM output signal. The Arduino has an odd “translation” of volts to bits, as 5V became 1023, and the sketch was different than the usual MATLAB code. Students had some trouble with the Arduino toolbox values mismatching and opted to use serial and fscanf() instead. Flushinput was used before the fscanf so the scan doesn’t read in multiple values but rather only the most recent one.

**Address:**

To: Dr. Christopher Peters and Mr. David Cinciruk

From: Philip Mak and Sanchet Hoque

Date 4/26/2019

Re: Lab 2 - Week 2 (Lab)

**Introduction:**

The purpose of the second week of Lab 2 was to characterize the DC motor which will be used throughout the project. Students will learn how to utilize the PWM signal from the Arduino with the prelab. Following that, students use the automated lab equipment to get the RPM vs duty cycle characteristic plot through various calculation methods.

**Method/Analysis**

The circuit was set up between the Arduino, lab equipment, and new H-bridge board, powering providing the H-bridge with 12V. The PWM output from the Arduino goes into the Input pin of the H-bridge. The optical sensor on the H-bridge was attached to the oscilloscope. There was a pull-up resistor added between the 5V power and optical sensor signal in order to have well-defined voltage. The AWG was automated to cycle from 0V to 5V to 0V and back to 5V, as it will result in the varying duty cycles students wish to observe on the oscilloscope.

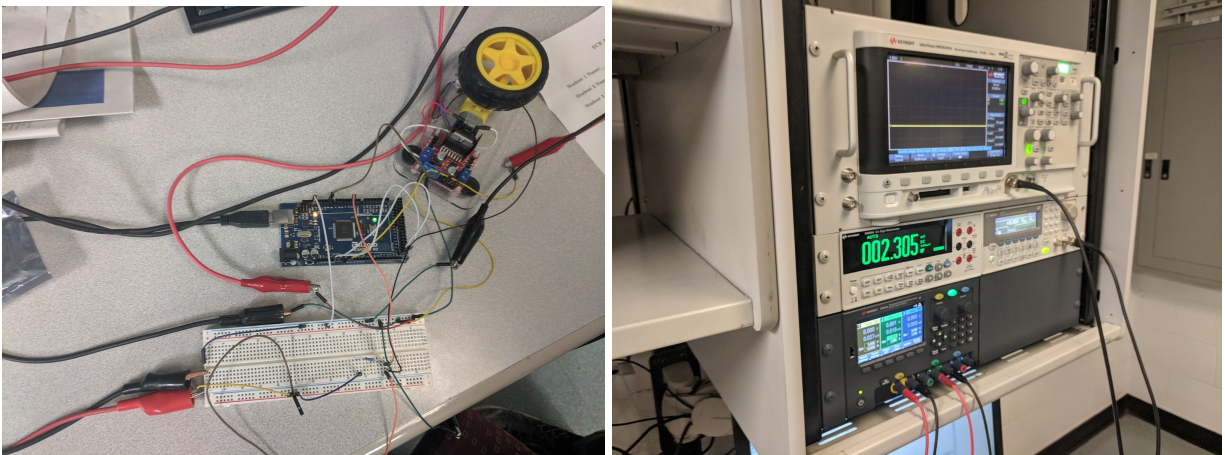


Figure I : Circuit involving breadboard, Arduino, H-Bridge(Left) and Lab Equipment(Right)

After setting up the DC motor board to the Arduino and ensuring wheel based on input from AWG, automation was done to collect data. For the Arduino side (Figure III), both part 1 and 2 of the pre-lab was combined into one script in which it outputted a PWM signal based on voltage from 0 to 5. Part 2 was used to count the pulse outputted by optical sensor. In MATLAB (Figure II), the visa commands were used to automate the voltage and extract the frequency. Serial commands were used to get the output from the Arduino. Both data sets were used to calculate the rpm of the wheel.

```

10 - V_min=0;
11 - V_max = 5;
12 - N_volts= .1;
13 - fprintf(awg,'OUTP:LOAD INF'); %Sets HIGH Z
14 - V1=0:.1:5; V2=0:.1:4.9; V3=0.1:.1:5;
15 - V=[V1,flipplr(V2),V3];
16 - h=serial('COM5','BaudRate',9600);
17 - fopen(a);
18 - countpulse=[];
19 - f=[];
20 - for K=1:length(V)
21 -     str1=['APPL:DC DEF,DEF,' num2str(V(K))];
22 -     fprintf(awg,str1); % Apply DC Voltage Output from AWG
23 -
24 -     pause(5)
25 -     %Arduino output
26 -     flushinput(a)
27 -     countpulse(:,K) = str2double(fscanf(a));
28 -     %saving frequency from oscscope
29 -     f(:,K)=str2double(query(scope,'MEAS:FREQUENCY?'));
30 -
31 - end
32 - %divide by 30 (30 holes on the pulse counter) and multiple by 60 ( 60
33 - %seconds to a minute)
34 -
35 - %rpm calculations
36 - rpmsf=2*f;
37 - rpmpc=2*countpulse;

```

Figure II. Matlab automation and serial data retrieval

```

int INPUT_PIN=A0;
float val = 0.0;
float dutycycle = 0.0;
float maxvol= 1023.0;
int OUTPUT_PIN=9;
float maxdc=255;
double press_count=0;
int Pulse_Input=2;

// pinMode(INPUT_PIN,INPUT);

void setup() {
    // put your setup code here, to run once:
    Serial.begin(9600);
    TCCR2B=(TCCR2B & (~8))|(8<3);
    pinMode(Pulse_Input,INPUT);
    attachInterrupt(digitalPinToInterrupt(Pulse_Input),pulse_state,RISING);
    //pinMode(OUTPUT_PIN,OUTPUT);
}

void loop() {
    // put your main code here, to run repeatedly:

    val=analogRead(INPUT_PIN);
    //Serial.println(val);
    dutycycle=(val/maxvol);

    analogWrite(OUTPUT_PIN, dutycycle*maxdc);

    //Serial.println(dutycycle);

    delay(1000);
    Serial.println(press_count);
    press_count=0;
}

void pulse_state(){
    press_count=press_count+1;
}

```

Figure III : Arduino PWM output and pulse counter

In each lab, RPM was plotted against the voltage points that were in 0.1V increments. The RPM was calculated by multiple 60 (the number of seconds in a minute) and divided by 30 to account for the number of holes in the circumference. The “Frequency Data” was found from the oscilloscope while the “Pulse Count” data was found from the optical sensor. The frequency plot and pulse count plot were represented separately along with their linear regression lines. The data was spliced into three sections and was used to get three linear regression lines. These three lines were then concatenated into one vector and plotted along their respective original data.

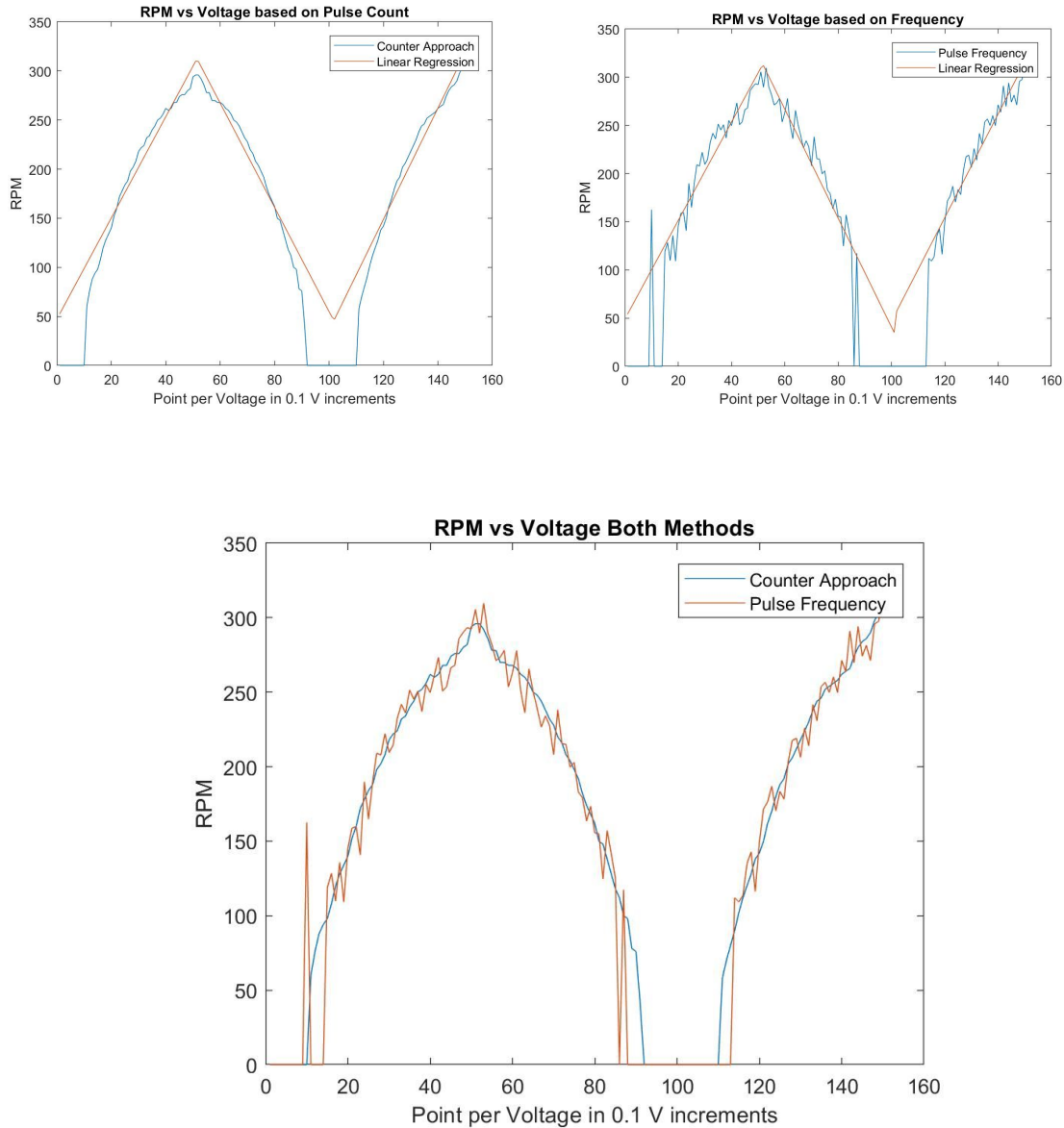


Figure IV :RPM vs Voltage: Via Pulse Count (Left), Via Frequency (Right), Both (Bottom)

## Discussion and Conclusion

The DC motor is an important part of the project to come and be able to measure its RPM is a nice variable to have at hand for later calculations, such as efficiency or battery capacity. There were some issues with the equipment. For the frequency data, any points that were above a certain threshold was set to zero, due to the oscilloscope not being able to correctly measure frequency at low voltages.



**Address:**

To: Dr. Christopher Peters and Mr. David Cinciruk

From: Philip Mak and Sanchet Hoque

Date: 5/3/2019

Re: Lab 3

**Introduction**

The purpose of lab 3 was to measure the voltage and current of a battery discharging, in which students can calculate the power and cumulative energy. The whole idea was to have a 9V battery power a DC motor, but stop after it falls underneath 4.8V. This motor will be a part of the cooling system for the electrical vehicle model. It is controlled by the arduino, which sends a signal based on the RPM of the motor whether or not to activate the fan.

**Method/Analysis**

The circuit for discharging a battery started by switching out the DC power supply for a small 9V battery. Two batteries were used: one for testing the code and the other for discharging all the way down to 4.8V. Students added the oscilloscope in parallel with the battery to measure voltage and the multimeter in series with the positive output of the battery to measure current. A small capacitor was attached across the DC motor in order to negate the noise of the DC motor for more steady measurements.

In order to retrieve the data as the battery was discharging, the oscilloscope and multimeter were automated in Matlab to measure the max voltage and current respectively each second. The time variable was measured after a 1 second pause between each measurement. However, there was uncalculated processing time from adding the data to each array. This provided us with three arrays: Voltage, Current, and Time. The power and cumulative energy over time was calculated by using  $P=I*V$  and  $E=P*T$  respectively. This was plotted against time for both battery brands to use as a comparison against each other.

```
3- scope=visa('agilent', 'USB0::0x0957::0x1799::MY58100815::0::INSTR');% - scope
4- %awg=visa('agilent', 'USB0::0x0957::0x0407::MY44043483::0::INSTR');% - AWG
5- DMM=visa('agilent', 'USB0::0x2A8D::0xB318::MY58170030::0::INSTR');% - DMM
6- fopen(DMM);
7- fopen(scope);
8- V=[];
9- C=[];
10- i=0;
11- while true
12-     i=i+1;
13-     %Curr=fprintf(DMM, 'MEAS[:PRIM]:CURR[:DC]?');
14-     pause(1);
15-     V(:,i)=str2double(query(scope, ':MEAS:VMAX?'));
16-     C(:,i)=str2double(query(DMM, 'MEAS:CURR?'));
17-     %Volt=fprintf(DMM, 'MEAS:VOLT?');
18-     if V(:,i)<4.8
19-         break;
20-     end
21- end
22- fclose(DMM);
23- fclose(scope);
24- delete(DMM);
25- delete(scope);
26-
```

Figure I: Matlab code to retrieve voltage, current, and time



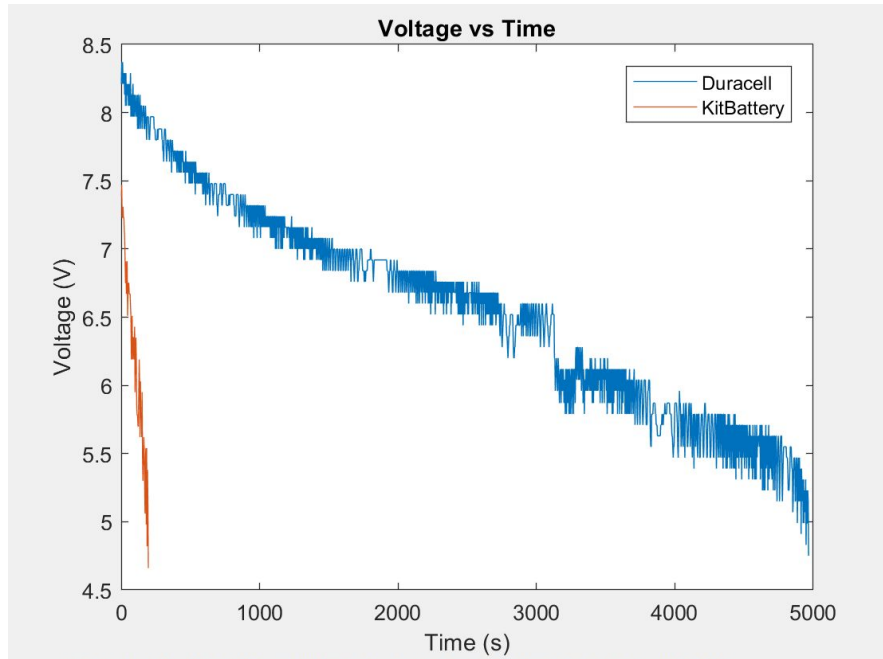


Figure II: Graph of the Two Batteries' Voltage Over Time

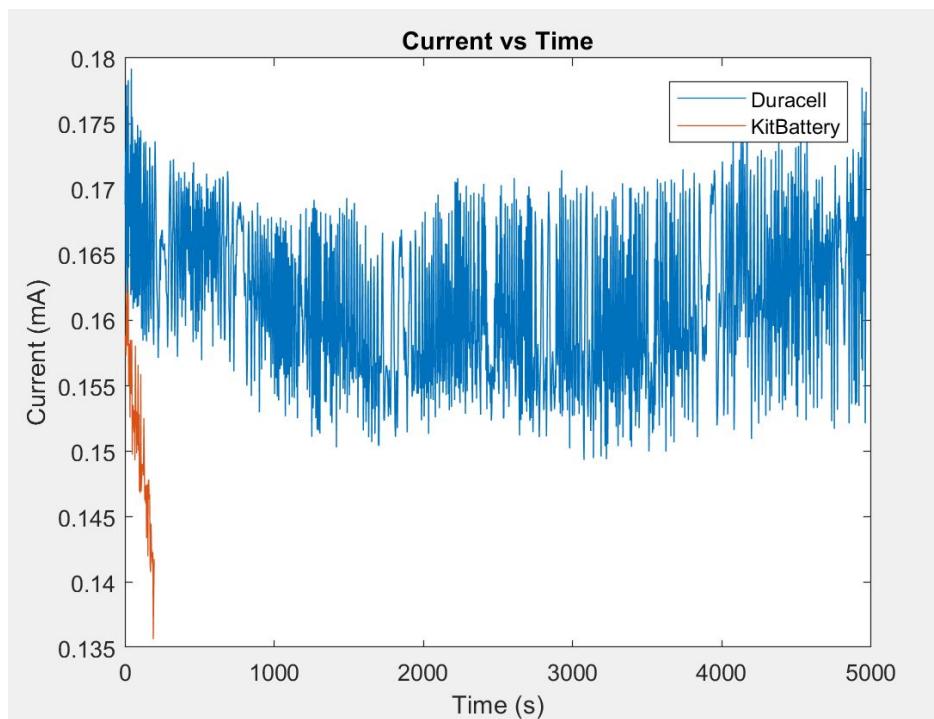


Figure III: Graph of the Two Batteries' Current Over Time

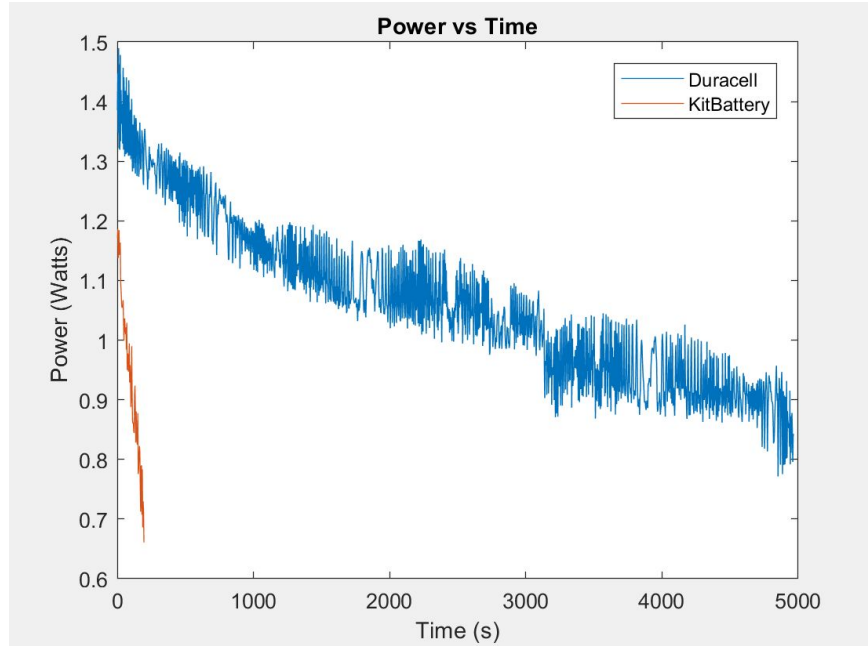


Figure IV: Graph of the Two Batteries' Power Over Time

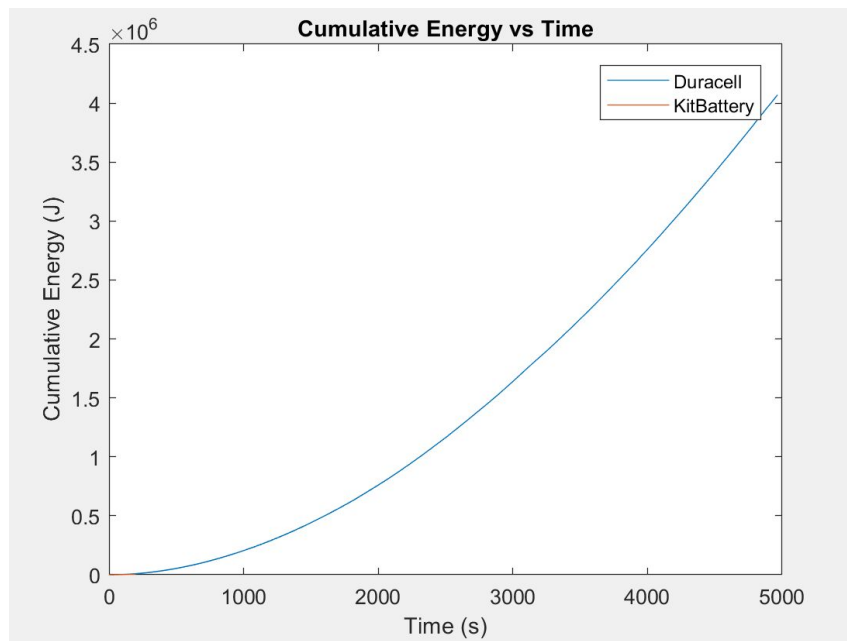


Figure V: Graph of the Two Batteries' Cumulative Energy Over Time

A fan from the given kit was used along with a packaged circuit board. With power from an outlet, there was a 3.3V output on the circuit board to provide the small DC motor with a fan blade attached. It turns on based on a digital high or low from the Arduino based on the H-Bridge being above 50% maximum RPM. There was a 220Ω resistor added between the transistor and the digital input to minimize current when the input is low. The high and low capacitance capacitors were added to filter out the high and low noise to have a steady DC voltage to the motor. The small diode was to allow the charge to dissipate if the digital input is

low. The arduino acted as the controller for the fan. The script for it stayed exactly the same as the other labs to run the motor, but in the mainloop, the RPM calculation is redone and acts as a switch to turn on the digitalWrite from low to high. The high digital output turns on the transistor, thus enabling the fan. It goes back to low once the RPM from the Hbridge is below 314 which was the peak value that was calculated from the previous lab

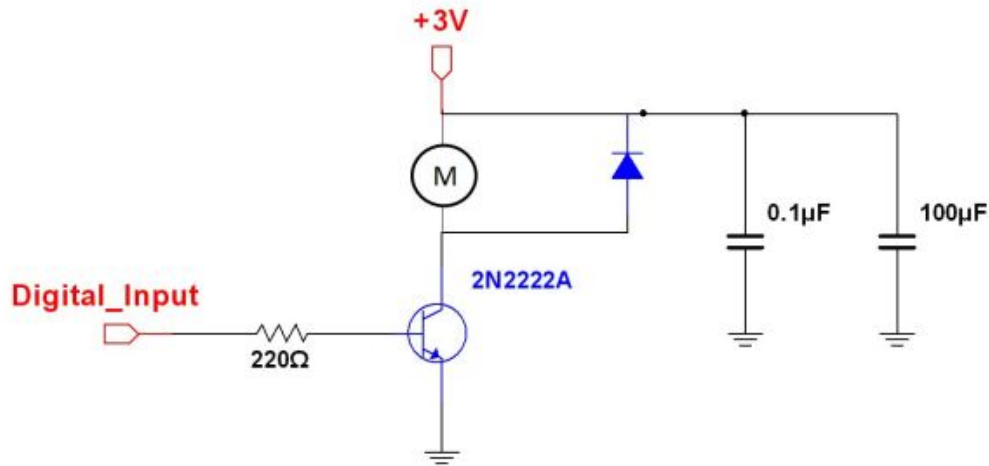


Figure VI: The Circuit for the Cooling Fan

```

delay(1000);
//Serial.println(press_count);
rpm=2*press_count;

// half rpm is calculated by dividing max value from rpmc data by 2

if rpm > halfrpm {
digitalWrite(OUTPUT_PIN, HIGH);
else
digitalWrite(OUTPUT_PIN, LOW);
}

press_count=0;
}

```

Figure VII: Arduino code to turn transistor on and off

## Discussion and Conclusion

The ability to automate the measurement of power supplies, be it a dinky Duracell battery, never-dying Amazon Battery, or an outlet, is important. The car batteries are just as important as

the fuel tanks of gasoline powered vehicles, so having a optimized motor with accurate measurements of runtime or performance is important so it doesn't run out of power in the middle of nowhere.

A cooling system is key to a power supply as running a hot motor will have an adverse effect on the performance of the motor, maybe even breaking it. It had to be run on another circuit board to avoid potential noise between the cooling and rest of the system.

In hindsight, there was a lot of optimization choices students could've made in the discharge. One example is using "tic toc" for time measurements rather than a pause(1) and then i variable adding. This had some uncalculated processing time. However, students deemed their i variable to have a 3 second interval based on the start and end of the discharge. For future usage, tic toc should be used for accurate time interval measurements to avoid the difference in processing time beyond assigned pause times.

**Address:**

To: Dr. Christopher Peters and Mr. David Cinciruk

From: Philip Mak and Sanchet Hoque

Date: 5/10/2019

Re: Lab 4

**Introduction**

The purpose of lab 4 was to set up a second arduino as a data acquisition system. It would operate the cooling motor, and the temperature and water level sensor with LEDs to denote the status of each. Additionally there'd be an audible alarm if problems arose. This DAQ would receive RPM count from the main Arduino to judge whether or not to run the cooling fan. The DAQ then communicates with the computer to tell sensor information and count. All of this is tied into a 5v relay to serve as an emergency shutdown method if an alarm comes on; it returns power if the issue settles down.

**Method/Analysis**

The DAQ used the cooling motor from the prior lab experiment. A DC motor with a fan head was used along with a packaged circuit board. The packaged circuit board took outlet power to output 3.3V to power the DC motor. It relies on a digital high or low from the Arduino based on the H-Bridge being above 50% maximum RPM. There was a  $220\Omega$  resistor added between the transistor and the digital input to minimize current when the input is low. The high and low capacitance capacitors were added to filter out the high and low noise to have a steady DC voltage to the motor. The small diode was to allow the charge to dissipate if the digital input is low. The arduino acted as the controller for the fan. The script for it stayed exactly the same as the other labs to run the motor, but in the mainloop, the RPM calculation is redone and acts as a switch to turn on the digitalWrite from low to high. The high digital output turns on the transistor, thus enabling the fan. It goes back to low once the RPM from the H-bridge goes below half the previously calculated maximum RPM. All of this is done of the DAQ rather than the old main Arduino.

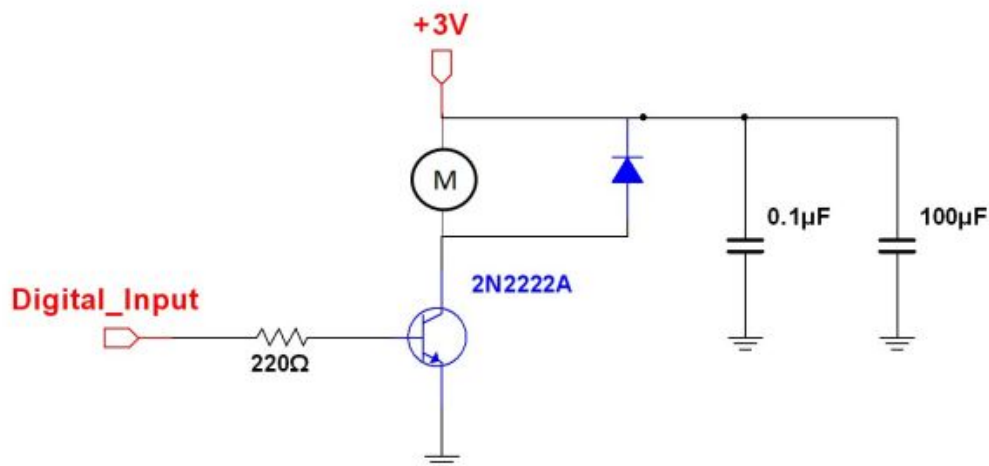


Figure I: The Circuit for the Cooling Fan

The temperature and water level sensor were added to the DAQ to take 5V to run and output a single read into the digital PWM row of the DAQ for the temperature, while the water level sensor read on analog A1 since it's based on 0-1023. For this model the 3-pin DHT11 was used to obtain the temperature. The DHT library was installed from the built in Arduino driver suite. In the setup phase for the DHT, a digital pin, DHT sensor type, dht.begin was initialized. In the main loop, the built-in dht function for reading temperature was simply used to retrieve t. This t value was a double type to improve accuracy in our if statement. The default celcius read was kept and the threshold set was 27 degrees celsius. The code to read in the water level required nothing except taking initializing an analog signal and using sensor read to get water level. The water level is read based on 0 to 1023 bit values. 200 was the threshold set for the if statement.

The 3 LEDs were set up, red, blue, and white for heat problem, liquid problem, and all clear respectively. The red LED turned on based on a given temperature threshold for testing that was a bit over room temperature. The blue LED turned on based on a given temperature threshold for testing that was pretty much any moisture at all (ie touching it with fingers.) The white LED was only on if the blue and red were off, such that it was running in a steady ok manner. The LEDs were given digital highs and lows to turn on and off based on what the sensors read in. Additionally, there was the buzzer that turns on if either the red or blue or both turn on; it gets a digital high or low if either LED is on.

---

```

1 //Fan Variables
2 int rpm = 0;
3 double halfrpm = 157;
4 int fan_control = 36;
5 int Pulse_Input = 2;
6 int press_count = 0;
7
8 //Water Level Variables
9 const int sensorPin= A1; //sensor pin connected to analog pin A1
10 int liquid_level;
11
12 // Led pins
13 int Red_Pin = 38;
14 int Blue_Pin = 40;
15 int White_Pin = 42;
16
17 //Buzzer
18 int Buzzer = 34;
19
20 //Motor Control
21 int Motor_Control = 32;
22
23 // DHT initialization
24 #include "DHT.h"
25 #define DHTPIN 7 // Digital pin connected to the DHT sensor
26 #define DHTTYPE DHT11 // DHT 11
27
28 DHT dht(DHTPIN, DHTTYPE);
29
30 void setup() {
31 // put your setup code here, to run once:
32 Serial.begin(9600);
33 pinMode(sensorPin, INPUT); //the liquid level sensor will be an input to the arduino
34 pinMode(Red_Pin, OUTPUT);
35 pinMode(Blue_Pin, OUTPUT);
36 pinMode(White_Pin, OUTPUT);
37 pinMode(fan_control, OUTPUT);
38 pinMode(Buzzer, OUTPUT);
39 pinMode(Motor_Control, OUTPUT);
40 pinMode(Pulse_Input, INPUT);
41

```

Figure II. All initializations



```

44 dht.begin();
45
46 }
47
48 void loop() {
49
50 // put your main code here, to run repeatedly:
51 //fan controller
52 //if (Serial.available() > 0) {
53 //rpm =Serial.read();
54 //Serial.println(rpm);
55 // }
56 // byte b2 =Serial.read();
57 // rpm = b2 + b1*256;
58 //Serial.println(rpm);
59 delay(1000);
60 //Serial.println(press_count);
61 rpm = 2*press_count;
62 press_count = 0;
63
64 if (rpm > halfrpm) {
65     digitalWrite(fan_control, HIGH);
66 }
67 else{
68     digitalWrite(fan_control, LOW);
69 }
70
71 liquid_level= analogRead(sensorPin); //arduino reads the value from the liquid level sensor
72
73 //float h = dht.readHumidity();
74 // Read temperature as Celsius (the default)
75 float t = dht.readTemperature();
76 Serial.println(t);
77 Serial.println(liquid_level);
78
79 if (t > 27 || liquid_level > 200) {
80     digitalWrite(White_Pin, LOW);
81     digitalWrite(Buzzer, HIGH);
82     digitalWrite(Motor_Control, LOW);
83     if (t > 27) {
84         digitalWrite(Red_Pin, HIGH);
85     }
86     if (liquid_level > 200) {
87         digitalWrite(Blue_Pin, HIGH);
88     }
89 }
90 if (t < 27 && liquid_level <200){
91     digitalWrite(Red_Pin, LOW);
92     digitalWrite(Blue_Pin, LOW);
93     digitalWrite(Buzzer, LOW);
94     digitalWrite(White_Pin, HIGH);
95     digitalWrite(Motor_Control, HIGH);
96 }
97

```

Figure III. LED, fan, motor controls

The setup to connect both the master arduino and the DAQ arduino was done through Serial. The master arduino does all the work from lab 2 (pulse counting) and does the rpm calculation in the main loop. This rpm value is sent to the DAQ to control the fan. Serial.write(rpm) was used and

to read it, `rpm= Serial.read()` was used. Tx0 and Rx0 from both arduinos were connected to each other as well. All control were handled by the DAQ as a digital high or low which shut off or turned on the motor, led, fan, and buzzer. The com ports in conjunction with `serial.println` were used to debug the code to see the values from both sensors and the rpm value.

There was a 5V relay added between the H-bridge and 12V/9V power input to act as a backup shutdown method. If any of the sensors pass the given threshold, the DAQ sends a signal to the main board to stop the PWM which is sending a 5V to the coil of the 5V. The 5V relay will switch to NO without the 5V on the coil; the H-Bridge turns on for NC; NO just goes to ground. The sensors, buzzers and LEDs will still run on the DAQ, but the H-bridge will be off until the temperature and liquid levels are resolved. The fan will also be off because the RPM will be 0.

## **Discussion and Conclusion**

All in all, having all of the sensors work in unison along with visual and audio cues with the alarm lead to a better user interface, because one cannot stare at terminals to read `println`s all the time. The ability to shutdown safely is important as fail safes in general are important. The choice to run on Normally Closed, as opposed to Normally Open signifies that the design intends to not put in voltage with the PWM when there is an issue at hand.

A big issue throughout the design of the circuitry is the common ground, since students dealt with numerous power sources and microcontrollers and breadboards. If received less than satisfactory voltage, it malfunctioned. The sensors would return -1, or the LEDs would be faded or blinking. It would also mess up the `serial.write` value from the master arduino to the DAQ. All in all, grounding is important to manage and double check via the multimeter if necessary.

Another issue that arose was in the DAQ, when writing if statements to control the LEDs, buzzer and motor. The language arduino uses allows for curly braces or no curly braces, so when writing nested ifs and using `elseifs`, a lot of the code would be ignored. To fix this, multiple standalone if statements were used. Having multiple arduinos allowed signals to distributed in a cleaner way because the DAQ mainly outputted digital signals, while the master outputted PWM signals.

**Address:**

To: Dr. Christopher Peters and Mr. David Cinciruk

From: Philip Mak and Sanchet Hoque

Date: 5/24/2019

Re: Lab 5

**Introduction**

The purpose of lab 5 was implement cruise control to the electric car model system. Cruise control is a function that keeps the wheel spinning at a given rpm no matter what load is applied to the system. The loads that were used to test the cruise control feature was change in voltage throughout the system and changing a 10k potentiometer that is connected to the 0 to 5V DC value. All of the code implementation was done in the master arduino since it controls the h-bridge.

**Method/Analysis**

The cruise control implementation main focus was controlling the duty cycle of the PWM output which originally had the linear relationship with the 0 to 5V DC value. In the code, there is a 0 or 1 value to turn on cruise control; a variable “cruising rpm” was added. When the cruise control is on, the master arduino reads the rpm of the wheel and the variable rpm. The difference is calculated and then divided by 2. This is the value to add or subtract from the duty cycle depending on whether or not the rpm is higher or lower than the set cruising rpm.

```
41  jumpval = abs(rpm-cruise)/2;
42  //Serial.println(jumpval);
43
44
45  if (ccontrol == 1) {
46
47      if (rpm < cruise) {
48          cdutycycle=cdutycycle+jumpval;
49          analogWrite(PWM_out, cdutycycle);
50      }
51      if (rpm > cruise ) {
52          cdutycycle=cdutycycle-jumpval;
53          analogWrite(PWM_out, cdutycycle);
54      }
55  }
56 }
57
```

Figure I: Cruise control code

## **Discussion and Conclusion**

At first, the cruise control implementation was done by using ratios of the current rpm of the wheel/max rpm produced by the wheel and cruise rpm/max rpm produced by the wheel. These values were subtracted by each other and multiplied by the maximum duty cycle outputted by the PWM (255). The issue with this was that during the voltage drop or increase test, the maximum rpm changed with the voltage change which messes up the ratios. This resulted in the incorrect rpm values. Thus, students adopted a binary adaptive method rather than a hard coded calculation.

**Address:**

To: Dr. Christopher Peters and Mr. David Cinciruk

From: Philip Mak and Sanchet Hoque

Date: 5/31/2019

Re: Final Lab

**Introduction**

The purpose of the final lab was to have a GUI/Interface control the entire electric car system and display values such as RPM, speed, battery life, distance traveled, temperature, and coolant level. This data can be displayed using a speedometer, gauges, and LED lights as well. Besides these requirements, the designer is free to implement any other data into their graphical display as well. Such as cruise control, time run, and other data values. This display could have been created using Python's Tkinter or Matlab App Designer. For this memo, the students used Matlab App Designer as the rest of the previous implementation has been done in Matlab thus far.

**Method/Analysis**

Code in both Arduino and Matlab was implemented to ensure success in the graphical control display. The Data Acquisition board contains all of the values needed, so only this board was sending data to Matlab. This data was sent by using serial print in a certain order and delay, so the designers could parse it better in Matlab. In Matlab, the students first implemented a base structure of all gauges, speedometers, text boxes, LEDs, and buttons using App Designer's. This was a simple drag and drop into an interface screen. In App designers there are two views: design view and code view.

The code was implemented inside of the Start/Stop button call back. This is a button with two states which made the handling a lot smoother. Upon the press of a button, all of the hardware is opened: oscilloscope, power supply, and Arduino. Tic is also started to measure time while the system is active. The code flows into a while loop that stays triggered while the button is pressed. SCPI code was used to set the power supply to 12 volts into the system at the beginning of for loop. The code also parses the Arduino data into a data structure and places them into the right area for the display.

Speed and distance traveled was calculated through RPM conversion and tic toc given a measured diameter of the wheel of about 2.625 inches.

Battery life % was calculated by using a formula based on voltage which was created by utilizing the data from lab 3. The oscilloscope was used to measure in the voltage and used in the formula. Our lab 3 data was based off a battery that started at around 8.4V, so since the power supply is outputting 12V, it will show a value over 100% until it goes below 8.4V

Time run was simply the toc value that was started at the beginning of the loop. So num2str was used heavily to convert when passing the variable into display.

Temperature and coolant level of the system was obtained through the Arduino. The LEDs attached to them lit up based on an if statement which triggered the lights depending on the values.

All the gauges attached onto the display (speed, temperature, coolant level, and battery life) took the same value as the text value counterpart except it stayed as a numerical value rather than using num2str to for the text.

Cruise control is also a constant number read in from the Arduino. If it is turned off, it will display 'Turned OFF'. If it's turned on a value will be present in there. After the while loop, the power supply is set to zero volts and all hardware is closed.

Figure I: Gui Display

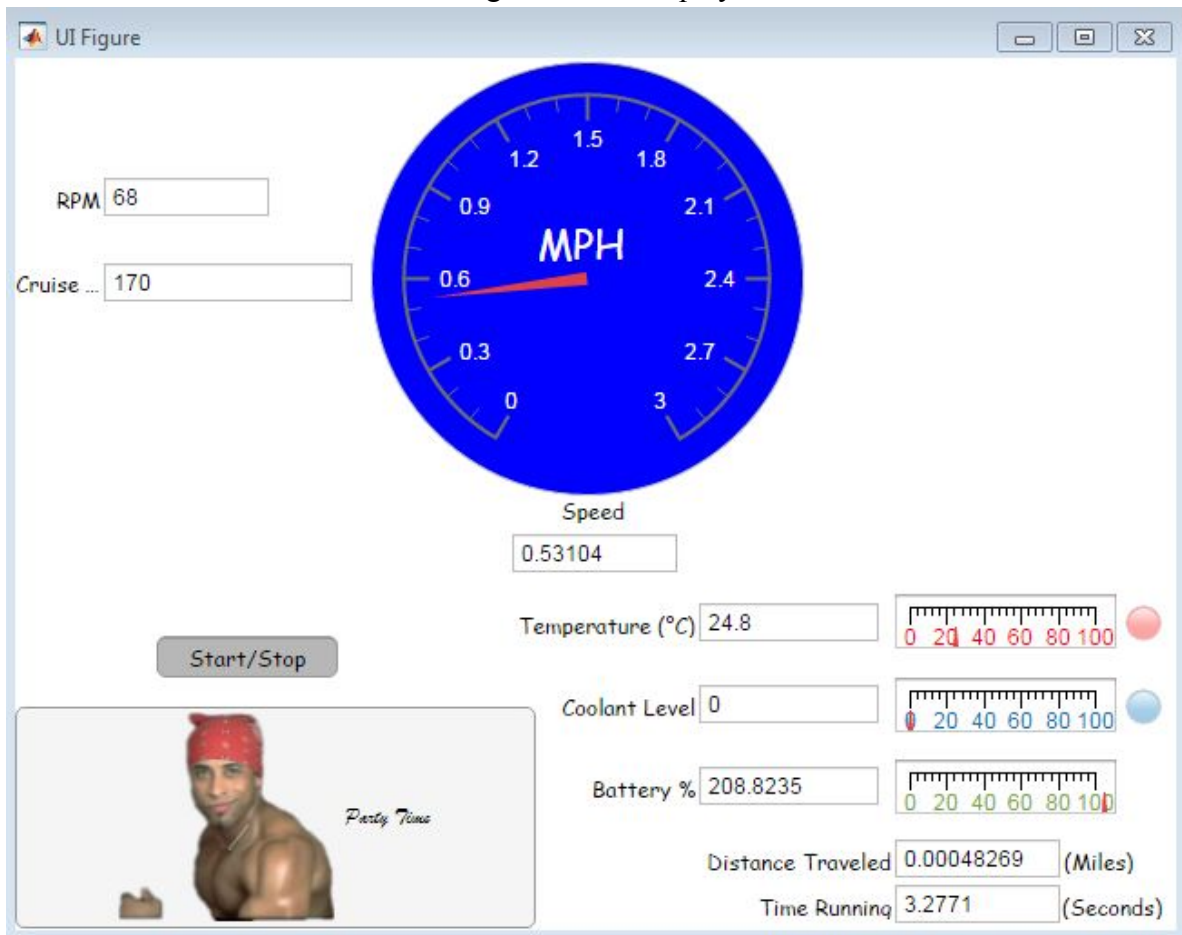


Figure II: Gui

```

1 scope=visa('agilent', 'USB0::0x0957::0x1799::MY56100822::0::INSTR');
2 %DMM=visa('agilent', 'USB0::0x2A8D::0xB318::MY56150015::0::INSTR');
3
4 powsup=visa('agilent', 'USB0::0x2A8D::0x1202::MY56170308::0::INSTR');
5 a=serial('COM5','baudRate',9600);
6
7
8 open=1;
9
10 % fopen(powsup);
11 % fopen(scope);
12 % fopen(DMM);
13 % fopen(a);
14
15 data=[];
16
17 tic
18 while app.StartStopButton.Value==start
19
20 if open ==1
21
22 fopen(powsup);
23 fopen(scope);
24 %fopen(DMM);
25 fopen(a);
26 open = 0;
27 end
28
29 fprintf(powsup,'APPL CH2,12,DEF');
30 drawnow()
31
32 for i=1:4
33 data(i,:)=str2double(fscanf(a));
34 end
35
36 if length(data) == 4
37 rpm=data(1);
38 Temp=data(2);
39 CoolantAna=data(3);
40 Cruise=data(4);
41 end
42
43 %battery lvl
44 V=str2double(query(scope,'MEAS:VMAX?'));
45 Batterylvl=((V-5)/3.6);
46 datastruc.Batterylvl=num2str(Batterylvl*100);
47
48 datastruc.RPM=num2str(rpm);
49 %datastruc.RPM='350';

```

```

49 %datastruc.RPM='350';
50
51 %rpm=str2double(datastruc.RPM);%turning RPM into a numerical value
52
53 %speed calculation
54 circum=0.621*pi; %diameter of wheel is 2 5/8 inches
55 Speed=(rpm*circum*60)/3360; %rpm*circumference*minutes in hour/inches in a mile
56 datastruc.Speed=num2str(Speed);
57
58 %Distance Traveled
59 hoursLaps=toc/3600;
60 DistTrav=Speed*hoursLaps;
61 datastruc.DistTrav=num2str(DistTrav);
62
63
64 %Temperature handling
65 %Temp=26;
66 datastruc.Temp=num2str(Temp);
67 if Temp >= 27
68 app.TempLED.Enable='on';
69 else
70 app.TempLED.Enable='off';
71 end
72
73 %Coolant handling
74 %CoolantAna=190;
75 maxcoolant=1023;
76 if CoolantAna <= 200
77 app.CoolantLED.Enable='off';
78 else
79 app.CoolantLED.Enable='on';
80 end
81 Coolant=CoolantAna*100/maxcoolant;
82 datastruc.Coolant=num2str(Coolant);
83
84 %Cruise
85 if Cruise>0
86 datastruc.Cruise=num2str(Cruise);
87 else
88 datastruc.Cruise='Turned OFF';
89 end
90
91 %Time elapsed
92 datastruc.TimeRun=num2str(toc);
93
94 app.RPMTextArea.Value=datastruc.RPM;
95 app.SpeedTextArea.Value=datastruc.Speed;
96 app.SpeedGauge.Value=Speed;
97 app.DistanceTraveledTextArea.Value=datastruc.DistTrav;

```

```

100
101 app.CoolantLevelTextArea.Value=datastruc.Coolant;
102 app.CoolantGauge.Value=Coolant;
103 app.CruiseControlRPMTextArea.Value=datastruc.Cruise;
104 app.TimeRunningTextArea.Value=datastruc.TimeRun;
105 app.BatteryGauge.Value=Batterylvl*100;
106 app.BatteryTextArea.Value=datastruc.Batterylvl;
107
108
109
110
111 pause(0.5)
112
113 end
114 if open == 0
115 fprintf(powsup,'APPL CH2,0,DEF');
116
117 fclose(a);
118 fclose(powsup);
119 fclose(scope);
120
121 end
122
123 %fclose(DMM);
124 delete(a);
125 %delete(scope);
126 delete(powsup);
127 delete(DMM);

```

Figure III. Gui Code



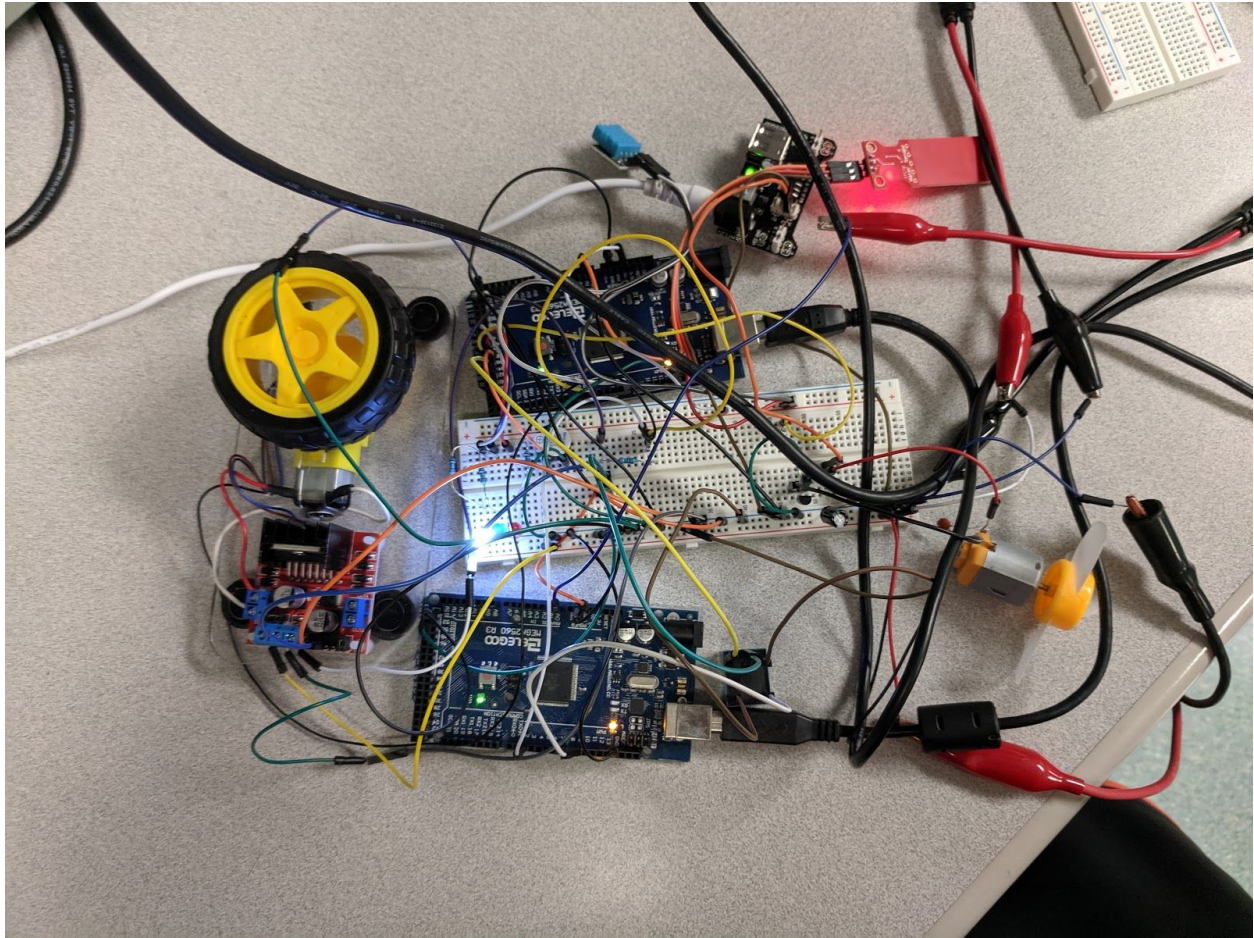


Figure IV. Image of final setup

### **Discussion and Conclusion**

App designer has a pretty interesting coding methodology, as students were forced to put most the programming in the callback of the start and stop button. There were a lot of issues regarding opening all the lab equipment communications to keep the GUI running after multiple starts and stops. A lot of innovations in the Gui design were cancelled or adapted due to the restrictions in place by the serial communication channels and the App Designer methodology.