# Lab 6: System Analysis 2

**Submitted to**

Dr. Jolanta Janiszweska

**Prepared by**

The Grinders

Heath Myers

Nate Johnson

Jason Kibler II

**Engineering 1182**

The Ohio State University

College of Engineering

Newark, OH

**March 5, 2020**

The AEV design and code underwent multiple test runs, with multiple scenarios. The first scenario tested was to go 15ft forward, brake, and then return 10 feet in the opposite direction. This was done in two runs. The first run was conducted by converting the distance needed to travel into marks and then using the “goToRelativePostion” command to in the code to complete the run. Once that run was completed, the data was uploaded to and excel spreadsheet, from there, the raw data was converted using MATLAB code. This new information was then utilized to construct another code using the “celerate” command to complete the same run as before.

The next scenario required that the AEV traverse the whole track, while stopping at several destinations along the way. The run started at the maintenance bay, went to the passenger pickup (Grand Canyon), then the Caribbean Waves, and finally arrived at its final destination, Alaska. With plenty of tinkering and around 20-25 runs later the run was successfully completed. The data from the successful runs were then uploaded to an excel spreadsheet and reduced to real numbers such as time, distance, position, current, voltage, supplied power, and incremental energy. The data obtained from these test runs is and will be used to improve and implement different design and program options.

The first code, which relied on the “goToRelativePostion” function, can be seen in the appendixes under “Flat rail test using relative position function.” The AEV was successfully ran on the track using this code, after several runs of having to alter the engine power and mark values. The EEPROM data (the data stored on the Arduino) from the run was then extracted from the Arduino, uploaded to a spread sheet, and then reduced to physical, numerical values using a MATLAB code which can be seen in the appendixes under “MATLAB Code.” The equations used to reduce the data are as follows:

$$t = \frac{t_E}{1000}$$

Where $t_E$ is the time of the run in milliseconds and $t$ is the time of the run in seconds.

$$I = \frac{I_E}{1024} * V_R * \frac{1\ Amp}{0.185\ Vots}$$

Where $I_E$ is the value of the current given by the EEPROM data, $V_R$ is the Arduino reference voltage with a value of 2.46 volts and $I$ is the current in amps.

$$V = \frac{15 * V_E}{1024}$$

Where $V_E$ is the given EEPROM voltage and $V$ is the voltage in volts.

$$d = 0.0124 * marks$$

Where $marks$ is the given EEPROM accumulated wheel count and d is the distance in meters.

$$s = 0.0124 * pos$$

Where $pos$ is the wheel count and $s$ is the displacement of the AEV from the starting point in meters.

$$E_j = \frac{P_j + P_{j+1}}{2} * (t_{j+1} - t_j)$$

Where $P_j$ is the power at a given point, $P_{j+1}$ is the power at the next point, $t_j$is the time at a given point, $t_{j+1}$ is the time at the next point, and $E_j$ is the incremental power in joules.

$$E = \sum_{n=1}^{N-1} E_n$$

$$v = \frac{s_i - s_{i-1}}{t_i - t_{i-1}}$$

Where $s_i$ is the displacement at a given point, $s_{i-1}$ is the last point's displacement, $t_i$ is the time at a given point, $t_{i-1}$ is the time at the last point, and $v$ is velocity in meters per second.

$$KE = \frac{1}{2}mv^2$$

Where $KE$ is the kinetic energy of the AEV and $m$ is the mass of the AEV which is .258 kg.

The data form this run is depicted in figure 1 and figure 2. Figure 2 depicts the power supplied vs the distance traveled and figure 1 depicts the power supplied vs time with a phase brake down. The phase breakdown shows the code in each phase, the time it took for the code to execute, and the total energy used during each phase can as well as a total energy, be seen in table 1.

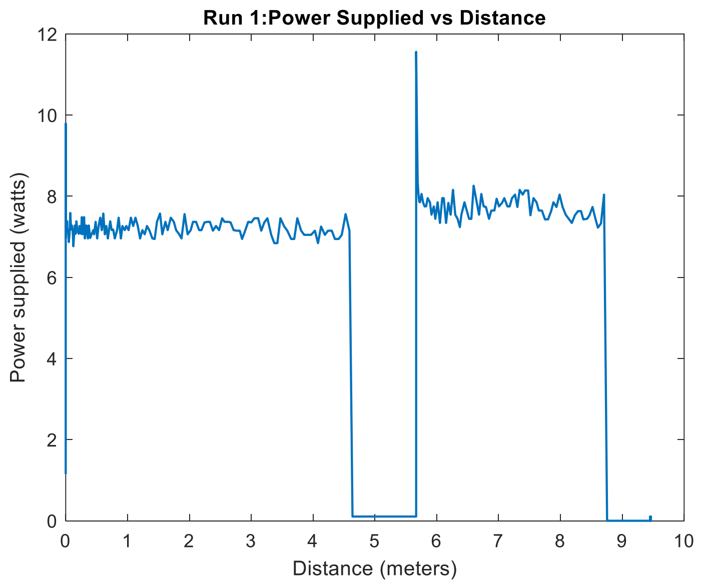

Figure 1:  Power supplied vs distance for the relative position command flat track run.

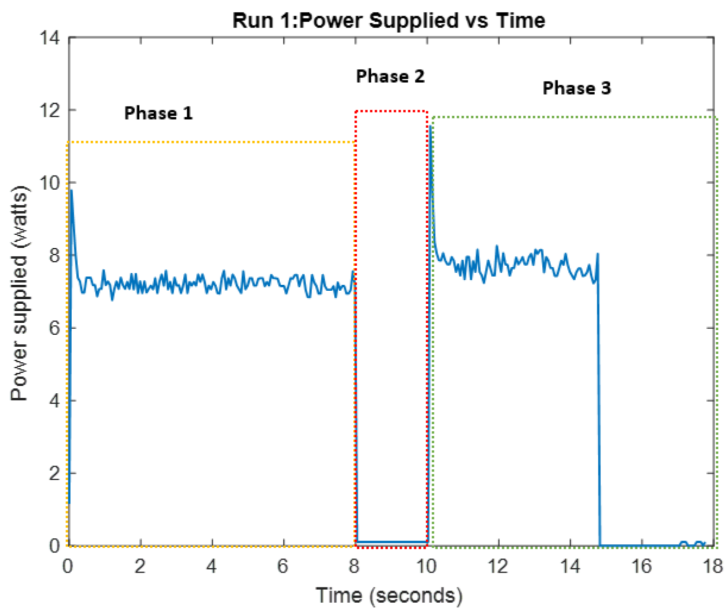

Figure 2: Plot of power supplied vs time for the first flat track run, with a phase breakdown.

| Phase | Arduino Code | Time (Seconds) | Total Energy (Joules) |
|---|---|---|---|
| 1 | motorSpeed(4,25)/ goToRelativePosition(370) | 7.863 | 221.941 |
| 2 | brake(4)/goFor(2) | 2.16 | 7.191 |
| 3 | Reverse(4)/motorSpeed(4,25)/goToRelativePositon(247)/brake(4) | 7.557 | 142.151 |
| Total | | | 371.283 |

Table 1: Phase breakdown for flat rail test using relative position command

A second code was then made, using the data from the first run, that would cause the AEV to travel the same distances as the first code but relied on the “celerate” command. This code can be seen in the appendixes under “Flat rail test using accelerate function.” The data, once again, was extracted from the Arduino, uploaded to an excel spreadsheet, and then reduced to physical, numerical values using the MATLAB code. The data from this run can be seen in figures 3 and 4 which show power vs distance and power vs time respectively. The phase breakdown, showing the code in each phase, the time it took for the code to execute, and the total energy used during each phase can, be seen in table 2.

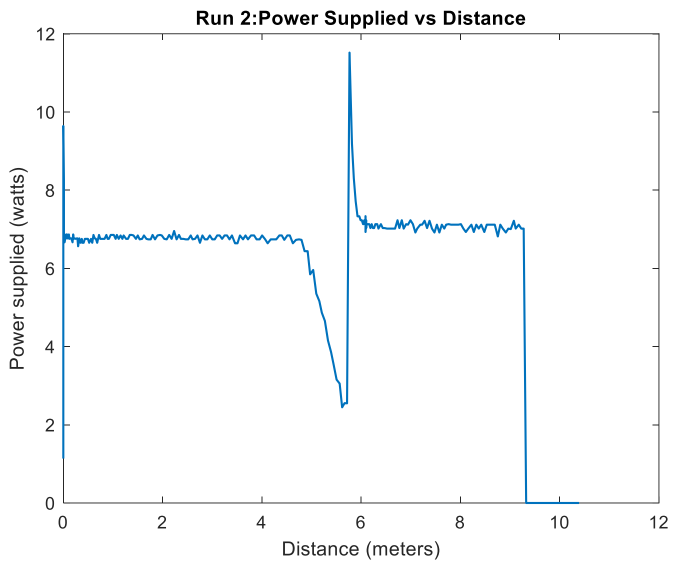

Figure 3: Power supplied vs distance for the flat rail test using accelerate command.

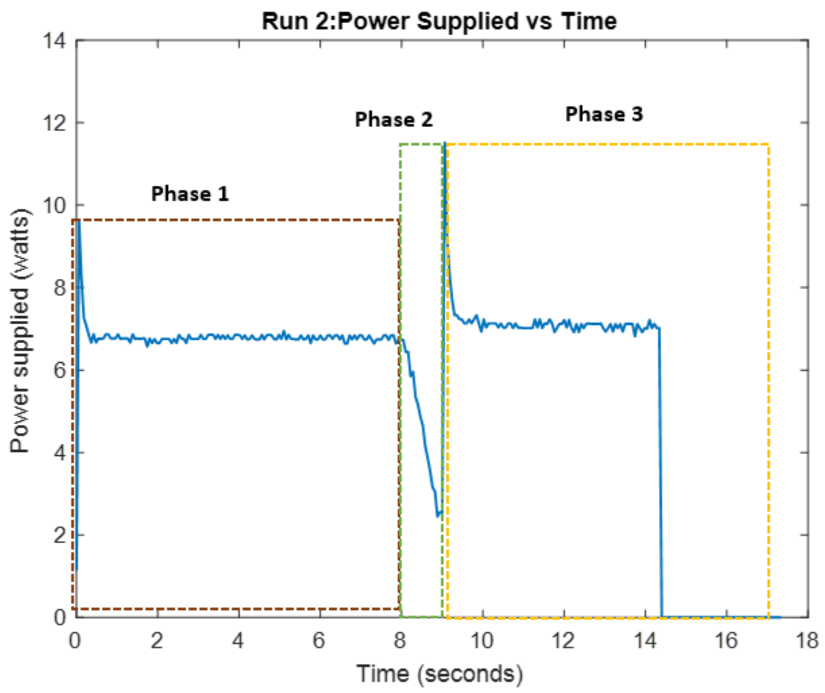

Figure 4: Power supplied vs time for the flat rail test using accelerate command

| Phase | Arduino Code | Time (Seconds) | Total Energy (Joules) |
|---|---|---|---|
| 1 | celerate(4,35,35,7.98) | 7.92 | 210.397 |
| 2 | celerate(4,35,0,1) | 1.08 | 22.387 |
| 3 | Reverse(4)/ celerate(4,35,35,5.32) | 8.1 | 147.339 |
| Total | | | 380.123 |

Table 2: Phase breakdown for the flat rail test using accelerate command

Once the initial two runs were completed and the AEV track behavior was predictable, a halftrack run was conducted. This halftrack run consisted of the AEV traveling form the maintenance station to the Grand Canyon, to Waves, then then to Alaska. To complete this task a much more involved Arduino code had to be created. Moreover, this code took several runs in order to develop into one that allowed for the AEV to not only stop within the parameters at each location but to also travel safely and smoothly between each stop. The final Arduino code can be seen in appendixes under "Half rail test."

Once the half run was completed successfully, the EEPROM data was extracted from the AEV, uploaded into an excel spreadsheet, and then reduced into physical, numerical values. These values were then used to create figures 5, 6, 7, 8, 9 which show supplied power vs distance, velocity vs distance, kinetic energy vs distance, propulsion efficiency vs distance, and supplied power vs time (with a phase breakdown) respectively. Continually, a phase breakdown table can be seen in table 3 with the Arduino code, time elapsed, distance covered, and total energy used for each phase of the AEV halftrack trip.

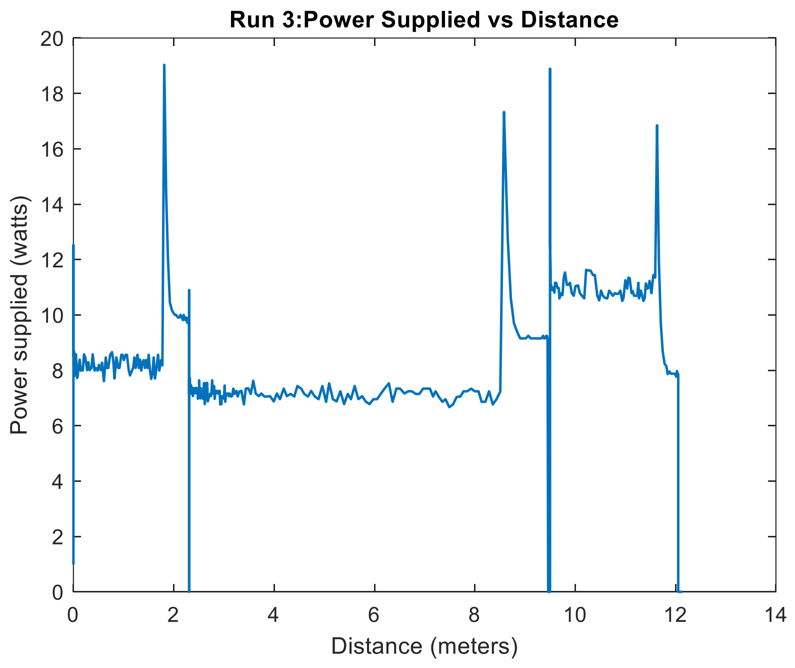

Figure 5: Graph of power supplied vs distance for halftrack run

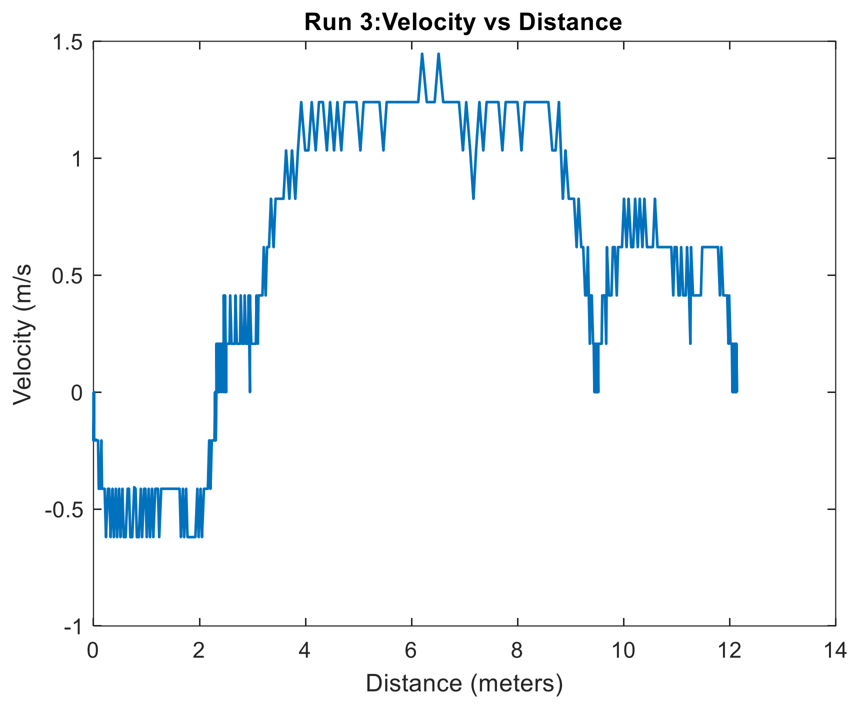

Figure 6: Graph of velocity vs distance for halftrack run

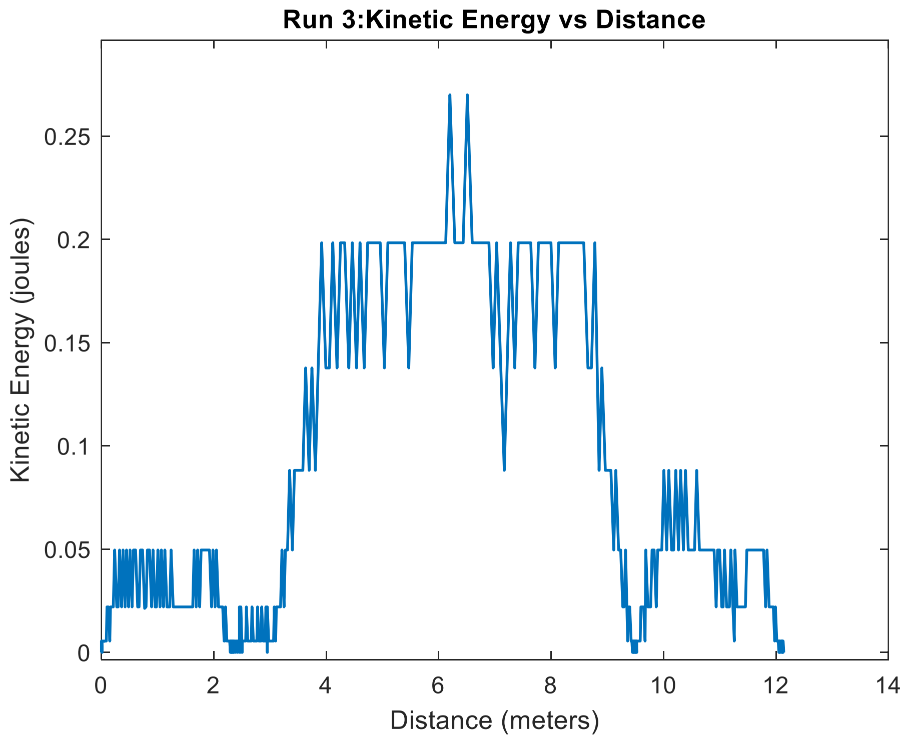

Figure 7: Graph of kinetic energy vs distance for halftrack run

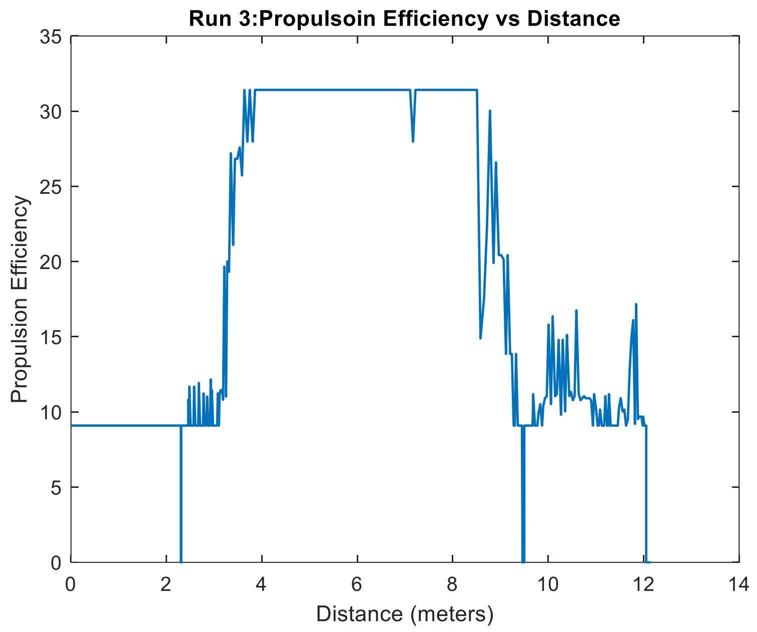

Figure 8: Graph of propulsion efficiency vs distance for halftrack run

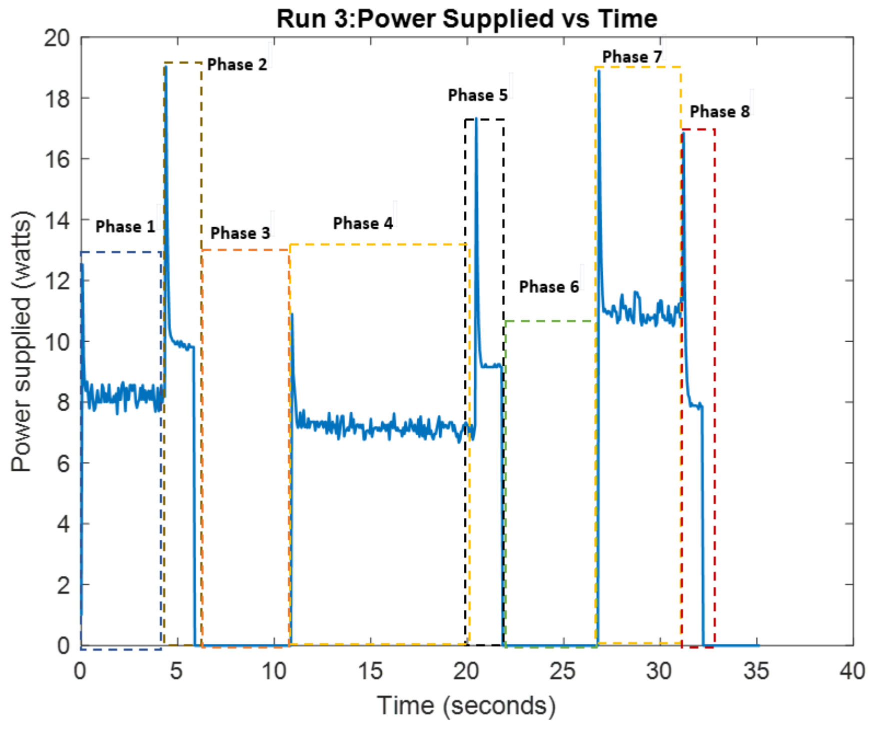

Figure 9: Graph of power suppled vs time with a phase breakdown for the halftrack run

| Phase | Arduino Code | Time (Seconds) | Distance (meters) | Total Energy (Joules) |
|---|---|---|---|---|
| 1 | motorSpeed(4,35)/ goToRelativePosition(-143)/brake(4) | 4.26 | 1.773 | 139.418 |
| 2 | reverse(4);motorSpeed(4,40)/goFor(1.5) | 1.50 | .532 | 62.294 |
| 3 | brake(4)/goFor(5) | 5.10 | 0 | 4.714 |
| 4 | motorSpeed(4,30)/goToReleativePositon(496)/brake(4) | 9.48 | 6.2 | 267.446 |
| 5 | reverse(4)/motorSpeed(4,40)/goFor(1.35) | 1.38 | .942 | 53.130 |
| 6 | brake(4)/goFor(5) | 5.04 | 0 | 4.714 |
| 7 | reverse(4)/motorSpeed(4,46)/goToRelativePosition(170)/brake(4) | 4.32 | 2.145 | 188.786 |
| 8 | Reverse(4)/motorSpped(4,35)/goFor(1)/brake(4) | 1.02 | .4588 | 36.245 |
| Total | | 32.1 | 12.049 | 756.746 |

Table 3: Halftrack phase breakdown

The figures 1 and 2 supplied a visualization of the power supplied to the AEV in relation to both time and distance. The take-away from these figures is that the power supplied spikes when starting the motors, meaning it would be more effective to have as few stoppages and changes in speed as possible. By comparing the two figures, it was also possible to determine the time it took to get certain distances, which will be used to consider future power levels if the AEV needs to run faster or slower.

The two initial runs were helpful in developing the code for the half-track run. The key components used from the initial two codes were the time it took as well as the marks traversed. Using these two values, it was possible to successfully determine the marks necessary to complete the specific parts of the run. The motor speed was also used as a reference to determine how fast the AEV should travel when not on an incline, and how much to adjust going up and incline.

Just like any lab several issues presented themselves as time when on. One of the major issues that came about during the lab was how to move the AEV up the incline and then till have enough speed to get the destination but not have too much to where the AEV would fly off the rail. This issue was solved by staring off by just getting the AEV to roll over the arch of the incline. Then once the power needed to traverse the incline was known it was slowly increased until the AEV started to roll, using its momentum from the accent, to the destination. Then a brake, reverse, and a motorSpeed function were added that allowed for the AEV to stop exactly where it needed to. If after this point the AEV did not stop were intended, the marks on the goToRelativePositon were increased or decreased until it did reach the location. Another challenge came about was how to deal with the AEV traveling down a hill and then rounding a turn in a safe manner. This was overcome by lowering the power of the motors to give the AEV less speed before it entered the decline so that it would be able round the turn slower. Another major issue that arose was the inconsistency of the stopping point at the first stop. This was overcome by making sure that the wheel positioning relative to the external sensors, and the AEV's location were in the same place for each run.

With the knowledge gained over the course of this lab, there is now a better overall understanding of the capabilities of the AEV, the behavior of the AEV on the track and the extent at which the Arduino code needs to be written in order to have a successful run. The values, tables, and figures form the three runs also provide more insight on how long each phase, and in turn an entire run should take; the distance that a specific function will cause the AEV to travel; and how much energy is used for each run and for each command. This information will be come pertinent in future designs, codes, and runs so that the cost and energy efficiency can be driven down with each test. Moreover, the knowledge of what percent power the motors need to be set to unorder to gain in elevation along the track as well as how to deal with turns and decreases in elevation.

## Appendixes

### Sample Calculations

**Nathan Johnson: Half-track run during time 300 ms**

Time: 300ms/1000 = 0.30 seconds (Done in line 10)

Current: (1amp/0.185volts) *(2.46volts)*( 91(ADC counts)/1024) =1.18 amps (Done in line 11)

Voltage: (15*501(ADC counts))/1024 = 7.34 volts (Done in line 12)

Distance: 0.0124*1(mark (accumulated)= 0.0124 meters (Done in line 13)

Position: 0.0124*-1(mark (from start)) = -0.0124 meters (Done in line 14)

Power: 1.18 amps * 7.34 volts = 8.66 watts (Done in line 15)

Incremental Energy: (8.66 + 8.19)/2*(0.36-.3)=0.504J (Done in line 29)

**Heath Myers:** Halftrack run during time 1861 ms

Time: Calculated in line 11 of MATLAB code

$$t = \frac{t_E}{1000}$$

$$\frac{1861}{1000} = 1.861\, seconds$$

Distance: Calculated in line 14

$$d = 0.0124 * marks$$

$$0.0124 * 49 = 0.6076\, meters$$

Position: Calculated in line 15

$$s = 0.0124 * pos$$

$$0.0124 * -49 = -0.6076\, meters$$

Current: Calculated in line 12

$$I = \frac{I_E}{1024} * V_R * \frac{1\, Amp}{0.185\, Vots}$$

$$\frac{80}{1024} * 2.46 * \frac{1}{0.185} = 1.04\, Amps$$

Voltage: Calculated in line 13

$$V = \frac{15 * V_E}{1024}$$

$$\frac{15 * 500}{1014} = 7.32\,Volts$$

Supplied power: Calculated in line 16

$$P = IV$$

$$1.04 * 7.32 = 7.61\,Watts$$

Incremental energy: Calculated in line 29

$$E_j = \frac{P_j + P_{j+1}}{2} * (t_{j+1} - t_j)$$

$$\frac{7.61 + 8.46}{2} * (1.921 - 1.861) = .482\,Joules$$

**Jason sample calculations for datapoint 25**:

Time: t = timems / 1000
.961 = 961/ 1000

Current: I = (current / 1024) * 2.46 * (1/0.185)
.844 = (85 / 1024) * 2.46 * (1/0.185)

Voltage: V = (15 * voltage) / 1024
7.324 = (15 * 500) / 1024

Distance: d = 0.0124*marks
.1612 = 0.0124*13

Position: s = 0.0124 * markspos
-.1612 = 0.0124 * -13

Power Suppled: P = V*I
P = 7.324*.844

**Arduino Code**

**Flat rail test using relative position function:**

```
motorSpeed(4,25);// powers all motors to 25% power
```

```
 goToRelativePosition(370);// maintains power at 25% until it travels 370 marks
 brake(4);// brakes all motors
 goFor(2);// executes brake for 2 seconds
 reverse(4);// reverses all motors
 motorSpeed(4,25);// powers all motors to 25% power
 goToRelativePosition(247);// maintains power at 25% until to travels 247 marks
 brake(4);// brakes all motors
```

**Flat rail test using accelerate function:**

```
celerate(4,35,35,7.98);// accelerates all motors from 35% power to 35% power for 7.98 seconds
celerate(4,35,0,1);// decelerates all motors form 35% power to 0% power
reverse(4);// reverses all motors
celerate(4,35,35,5.32);//accelerates all motors from 35% power to 35% power for 5.32 seconds
```

**Half rail test:**

```
motorSpeed(4,35); //changes all motor powers to 35%
goToRelativePosition(-143); // allows the AEV to travel 143 marks
brake(4); //brakes all motors
reverse(4); //reverses all motors
motorSpeed(4,40); //changes all motor speeds to 40%
goFor(01.5); //continues the motors at 40% for 1.5 seconds
brake(4); // brakes all motors
goFor(5);// continues brake for 5 seconds
motorSpeed(4,30); // sets motors to 30%
goToRelativePosition(496); // allows the AEV to travel 496 marks
brake(4); // brakes all motors
reverse(4); //reverses all motors
motorSpeed(4,40); //sets all motors speeds to 40%
goFor(1.35);// continues to have the motors speeds at 40% for 1.35 seconds
brake(4);// brakes all motors
goFor(5);// continues brake for 5 seconds
```

reverse(4);// reverses all motors

motorSpeed(4,46); // sets all motors to 46%

goToRelativePosition(170);// allows the AEV to travel 170 marks

brake(4);// brakes all motors

reverse(4);// reverses all motors

motorSpeed(4,35);// sets all motor speeds to 35% power

goFor(01);// continues the motor speed command for 1 second

brake(4);// brakes all motors

**Matlab code for halftrack run**

```
clc;clear
num = xlsread("hahaitworkedvictoryforvegeta.xlsx");
num([1:6],:) =[];
timems = num(:,1); %time vector
current = num(:,2); %current vector
voltage = num(:,3); %voltage vector
marksc = num(:,4); %marks vector
markspos = num(:,5); %position vector
m = .258;% mass in kg
for i = 1:length(timems)
    t(i) = timems(i) / 1000; %time
    I(i) = (current(i) / 1024) * 2.46 * (1/0.185); %current
    V(i) = (15 * voltage(i)) / 1024; %voltage
    d(i) = 0.0124*marksc(i); %distnace
    s(i) = 0.0124 * markspos(i); %position
    P(i) = V(i)*I(i); %power
    RPM(i)=-17.64*I(i)^2+690.375*I(i)+99.77; %Revs per min
end
for q=1:length(timems)-1
        v(q) = (s(q) - s(q+1)) / (t(q) - t(q+1)); %velocity
        ke(q) = .5 * m * v(q)^2; %kinetic energy
end
for n=1:length(v)
J(n)=(v(n))/((RPM(n)/60)*.0819); %Advance ration
PE(n)=-454.37*J(n)^3+321.58*J(n)^2+22.603*J(n); %Propulsion
Efficiency
end
for k = 1:length(timems)-1
```

```
    E(k) = ((P(k) + P(k+1)) / 2) * (t(k+1) - t(k));
end

Etot=sum(E); %Energy sum
plot(t,P,'LineWidth',1.15); %plot of time vs position
xlabel('Time (seconds)')
ylabel('Power supplied (watts)')
title('Run 3:Power Supplied vs Time')
ylim([0 20]);



xR=4.322;
xL=.06;
t=t';
iL=knnsearch(t,xL);
iR=knnsearch(t,xR);
E_phase_1=sum(E(iL:iR))/m;
fprintf('E phase 1: %8.3f \n',E_phase_1)

xL2=4.322;
xR2=5.822;
iL2=knnsearch(t,xL2);
iR2=knnsearch(t,xR2);
E_phase_2=sum(E(iL2:iR2))/m;
fprintf('E phase 2: %8.3f \n',E_phase_2)

xL3=5.822;
xR3=10.922;
iL3=knnsearch(t,xL3);
iR3=knnsearch(t,xR3);
E_phase_3=sum(E(iL3:iR3))/m;
fprintf('E phase 3: %8.3f \n',E_phase_3)

xL4=10.922;
xR4=20.402;
iL4=knnsearch(t,xL4);
iR4=knnsearch(t,xR4);
E_phase_4=sum(E(iL4:iR4))/m;
fprintf('E phase 4: %8.3f \n',E_phase_4)

xL5=20.402;
xR5=21.782;
iL5=knnsearch(t,xL5);
```

```
iR5=knnsearch(t,xR5);
E_phase_5=sum(E(iL5:iR5))/m;
fprintf('E phase 5: %8.3f \n',E_phase_5)

xL6=21.782;
xR6=26.822;
iL6=knnsearch(t,xL3);
iR6=knnsearch(t,xR3);
E_phase_6=sum(E(iL6:iR6))/m;
fprintf('E phase 6: %8.3f \n',E_phase_6)

xL7=26.822;
xR7=31.142;
iL7=knnsearch(t,xL7);
iR7=knnsearch(t,xR7);
E_phase_7=sum(E(iL7:iR7))/m;
fprintf('E phase 7: %8.3f \n',E_phase_7)

xL8=31.142;
xR8=32.162;
iL8=knnsearch(t,xL8);
iR8=knnsearch(t,xR8);
E_phase_8=sum(E(iL8:iR8))/m;
fprintf('E phase 8: %8.3f \n',E_phase_8)

Enet=E_phase_1+E_phase_2+E_phase_3+E_phase_4+E_phase_5+E_ph
ase_6+E_phase_7+E_phase_8;
fprintf('Run 1 total is: %8.3f',Enet)
```

## Lab Responsibilities

Nate Johnson wrote the introduction and discussion question 2d and 1c. Heath Myers wrote the conclusion, discussion question 3, discussion question 2c and made all figures and tables for question 1. Jason Kibler II wrote discussion question 2a and 2b and commented on the Arduino code.