Simulation and Control System of a Railroad Track Power Harvesting Device

Kyle J. Phillips
University of Nebraska-Lincoln, kyle.phillips22@huskers.unl.edu

Follow this and additional works at: http://digitalcommons.unl.edu/mechengdiss
Part of the Mechanical Engineering Commons

http://digitalcommons.unl.edu/mechengdiss/26

This Article is brought to you for free and open access by the Mechanical & Materials Engineering, Department of at DigitalCommons@University of Nebraska - Lincoln. It has been accepted for inclusion in Mechanical (and Materials) Engineering -- Dissertations, Theses, and Student Research by an authorized administrator of DigitalCommons@University of Nebraska - Lincoln.
SIMULATION AND CONTROL SYSTEM OF A RAILROAD TRACK POWER HARVESTING DEVICE

by

Kyle Phillips

A THESIS

Presented to the Faculty of

The Graduate College of at the University of Nebraska

In Partial Fulfillment of Requirements

For the Degree of Masters of Science

Major: Mechanical Engineering

Under the Supervision of Professor Carl A. Nelson

Lincoln, Nebraska

August 2011
With the vastness of existing railroad infrastructure, there exist numerous road crossings which are lacking warning light systems and/or crossing gates due to their remoteness from existing electrical infrastructure. Along with lacking warning light systems, these areas also tend to lack distributed sensor networks used for railroad track health monitoring applications. With the power consumption required by these systems being minimal, extending electrical infrastructure into these areas would not be an economical use of resources. This motivated the development of an energy harvesting solution for remote railroad deployment.

This thesis describes a computer simulation created to validate experimental on-track results for different mechanical prototypes designed for harvesting mechanical power from passing railcar traffic. Using the Winkler model for beam deflection as its basis, the simulation determines the maximum power potential for each type of prototype for various railcar loads and speeds. Along with calculating the maximum power potential of a single device, the simulation also calculates the optimal number and position of the devices needed to power a standard railroad crossing light signal. A control system was also designed to regulate power to a battery, monitor and record power production, and make adjustments to the duty cycle of the crossing lights.
accordingly. On-track test results are compared and contrasted with results from simulations, discrepancies between the two are examined and explained, and conclusions are drawn regarding suitability of arrays of such energy harvesting systems for powering high-efficiency LED lights at railroad crossings and powering track-health sensor networks.
ACKNOWLEDGEMENT

The author would like to thank my advisor, Dr. Carl Nelson, for his guidance, direction, and support during this project. Thanks to Mahmood Fateh and Gary Carr of the Federal Railroad Administration’s Track Research Division for providing technical support and direction. Special thanks to Richard Kotan of the Omaha Public Power district for ongoing cooperation on this project and for allowing access to the Nebraska City grade crossing test site. Also the author would like to thank Abolfazl Pourghodrat who also worked on this project. This work has been performed under a grant from the Federal Railroad Administration (FRA).
Table of Contents

ACKNOWLEDGMENTS iv
TABLE OF CONTENTS v
LIST OF FIGURES vii

Chapter 1. INTRODUCTION 1

Chapter 2. BACKGROUND 5

2.1 Existing Technologies .............................................................. 6
2.2 Power Requirements ............................................................... 9
2.3 Power Harvesting Device .......................................................... 9

Chapter 3. SIMULATION 13

3.1 Graphical User Interface .......................................................... 15
3.2 Equations of Motion ............................................................... 16
3.3 Track Deflection and Power Generation ..................................... 20
  3.3.1 Track Deflection ............................................................... 20
  3.3.2 Generator Speed ............................................................. 26
  3.3.3 Power Production ............................................................. 32
3.4 Device Positioning ................................................................. 37
3.5 Train Simulation ................................................................. 39
  3.5.1 Power Losses Due to Resistance ......................................... 42
  3.5.2 Duty Cycle and Battery Life ............................................... 49
  3.5.3 Pass/Fail Criteria ............................................................. 51

Chapter 4. SIMULATION RESULTS 52

4.1 Device Variables ................................................................. 52
4.2 Device Position ................................................................. 56

Chapter 5. CONTROL SYSTEM 64

5.1 Electrical System ................................................................. 64
5.2 Charge Controller ............................................................... 67
LIST OF FIGURES

2.1 First generation single one-way clutch power harvester design………………... 12
2.2 Second generation double one-way clutch power harvester design…………….. 12

3.1 Flow diagram of overall simulation…………………………………………… 14
3.2 MATLAB graphical user interface……………………………………………. 15
3.3 Track deflection profile for a loaded train at 55 mph………………………… 24
3.4 Track deflection profile for an unloaded train at 55 mph…………………… 25
3.5 Track velocity profile for a loaded train at 55 mph…………………………… 27
3.6 Track velocity profile for an unloaded train at 55 mph……………………… 28
3.7 Generator shaft revolutions per minute for first generation design…………… 30
3.8 Generator shaft revolutions per minute for second generation design……….. 31
3.9 Power production for second generation design with a gear ratio of 1:50…… 35
3.10 Power production for first generation design with a gear ratio of 1:50……… 36
3.11 Configurations of device positioning………………………………………… 37
3.12 Start and end times of devices for spread option…………………………….. 40
3.13 Power production through sections of wire………………………………….. 43
3.14 Flow diagram for calculating power losses through the sections of wire…… 48

4.1 Simulation results for a train traveling at 55 mph with different values for gear ratio, load and device design……………………………………… 53
4.2 Simulation results for a train traveling at 10 mph with different values for gear ratio, load and device design……………………………………… 53
4.3 Comparison of instantaneous generator speed for a first generation device with 1:50 gear ratio and second generation device with a gear ratio of 1:100………………………………………………………… 55
4.4 Power production and battery life of spread out, second generation devices, for a loaded train traveling at 55 mph……………………………………… 58
4.5 Power production and battery life of centered, second generation devices, for a loaded train traveling at 55 mph................................. 59

4.6 Power production and battery life of spread out, first generation devices, for a loaded train traveling at 55 mph........................................ 61

4.6 Power production and battery life of spread out, first generation devices, for a loaded train traveling at 55 mph with a length of 40 railcars...... 61

5.1 Power harvesting devices connected in parallel with blocking diodes......... 66

5.2 Wiring configuration for powering the single board computer.................. 69

5.3 Wiring configuration for control of the warning lights........................... 71

5.4 Rabbit single board computer with dimensions...................................... 74

6.1 Setup of laboratory testing with motor, voltage divider, single board Computer, charge controller, and battery all connected.......................... 83

6.2 Equivalent circuit for laboratory testing.................................................. 84

6.3 Voltage drop across a 10 Ω resistor for various generator speeds as measured by the single board computer............................................. 87

6.4 Lab testing results for both the single board computer and multimeters over various generator speeds......................................................... 88

6.5 Comparison between lab results and simulation equations...................... 89

6.6 Plotted comparison between lab results and simulation calculations......... 90

6.7 Four first generation device, spread out, for a loaded train traveling at 55 mph................................................................. 93

6.8 Single second generation device, centered, for a loaded train traveling at 55 mph................................................................. 93
Chapter 1. INTRODUCTION

With the significant number of deaths each year occurring at railroad crossings, particularly crossings which lack warning lights, considerable efforts are being made to improve public safety. Deploying warning light systems at these crossings is the logical answer to improving safety; however, many of these crossings are in remote areas lacking electrical infrastructure. Extending existing electrical infrastructure into many of these areas is impractical due to the high associated costs. Along with lacking warning light systems in these remote areas, they also tend to lack distributed track health monitoring systems. Because of this, derailments due to improperly maintained tracks can result. Installation of both these systems would significantly increase public safety around railroad tracks. With the power requirements for both systems being minimal, development of a low cost, low maintenance, and long-term electrical power generation device would be ideal for these areas.

The vast majority of unmarked railroad crossings are typically several and sometimes hundreds of miles away from any type of electrical infrastructure that could offer the needed electrical power. Because of the remoteness of the majority of unmarked crossings, there are currently no widespread solutions to providing warning lamp systems to them. Even crossings that are close to a source of electrical power are not typically considered for these services due to the high associated costs of providing them with warning lights, building infrastructure to reach the grade crossings, and servicing and providing maintenance operations that would be necessary. With these associated costs it has become widely accepted to not provide these remote railroad crossings with electrical infrastructure as the costs outweighs the benefits.
To monitor track health, the railroad industry frequently utilizes networks of distributed sensors. These networks tend to be wireless and integrate various sensors to monitor physical and/or environmental conditions about a particular section of track. Rail deflection, rail wear, and other phenomena can all be monitored by these sensors. Strain gauges, accelerometers, and other devices can be used to detect bending and wear of the rail, and other conditions that could cause a section of track to fail. These wireless distributed sensor networks are integrated into sections of track that often span very remote areas. This, coupled with thousands of miles of railway track across the United States, means the vast majority of distributed sensor networks are located on spans of railroad track that are tens or hundreds of miles from populated centers.

Some variety of power harvesting is currently the most feasible and most frequently utilized option for providing the necessary power for both warning systems at railroad grade crossing and track health safety sensor implementations. The most common form of power harvesting used for these applications involves a bank of large solar cells mounted in close proximity to the electrical load that collects solar energy whenever available. With this setup a sufficient amount of power can be produced indefinitely for grade crossings as long as the sun is out to provide the necessary solar energy. However, since this is not always the case, an electrical storage system composed of several batteries is also needed. This electrical storage system can provide the necessary power to the grade crossing whenever solar energy is unavailable and can be replenished whenever there is an excess of solar energy. Because of the size of the solar panels and associated battery banks, this form of power harvesting often becomes a victim of theft and vandalism.
To provide power to the sensors and the wireless receivers and transmitters used by distributed track health monitoring systems, batteries, and more specifically rechargeable batteries, are typically utilized. Because the monitoring system can operate on tens of mW, batteries become an ideal option to use because of the low associated costs. Similarly to the solar panels, an electrical storage system of one or multiple batteries can be integrated with the monitoring system and supply power as needed. As long as the battery network has sufficient amounts of electrical energy, the track health monitoring system can freely operate. However, once the battery network has been drained of its available energy, they must be replaced, forcing maintenance out to the track site.

Because of recent advances in power production and efficiency of piezoelectric materials, they are increasingly being investigated for applications in deployed wireless sensor networks. Along with improvements to piezoelectric materials, sensors used for monitoring track health are also being improved such that they can operate on lower amounts of power. While these improvements in technologies would not put piezoelectric materials anywhere near the same power levels as solar power harvesting, and they would not be capable of powering railroad crossing warning systems, they are slowly becoming a viable solution, when used in conjunction with batteries, to provide power to wireless sensors used for monitoring railroad track health.

With each of the previously mentioned methods of providing power to warning lamp systems at railroad grade crossings and wireless sensor networks in remote areas, there exist drawbacks, be it reliability or sheer quantity of power production. Because of this, an alternative method needs to be developed capable of producing the necessary
electrical energy to reliably accommodate the power needs for both systems. This alternative method of harvesting power would have the potential to vastly increase railroad safety.

To achieve the power requirements necessary for warning light systems and track health monitoring, power harvesting devices of different designs were fabricated. The premise of these designs was to harness the vertical displacement of the rail and ties due to passing railcar traffic and translate it into electrical energy. The goal of these designs was to achieve power production on the order of 10 W per device. This power production would be sufficient to illuminate a bank of high-efficiency crossing lamps along with powering a track health monitoring system.

With these designs, a computer simulation is necessary to validate the test results. Also an electrical control system is needed to allow for remote deployment, both of which are presented in this thesis. Chapter 2 presents a literature review of current work in power harvesting techniques that relate to railroad track safety enhancements. The different power harvester designs used for this thesis are also discussed. Chapter 3 describes the simulation engine created to analyze the different prototypes and determine the maximum rotational speeds of the devices, the expected power that the devices are capable of harvesting, optimal positioning of the devices, and the duty cycle of the lights. Simulation results are discussed in Chapter 4. The electrical control system designed to allow for on-track deployment, power and data collection, and control of the crossing lights is discussed in Chapter 5. Chapter 6 presents lab test procedures and comparison of lab data to simulation data. Summary and conclusions drawn from our results are discussed in Chapter 7.
Chapter 2. BACKGROUND

Railway systems are a crucial part of today’s infrastructure with rail traffic continually increasing over the years. This increase in traffic has resulted in more wear on the railroad track and more interactions with the general public. With this increase, train accidents have also increased. This has created the need for reliable power in remote areas which could be used by safety devices such as track health monitoring systems and warning light systems. Improvements are being continually made to sensors and wireless network technologies allowing them to become smaller and more efficient than their predecessors. With these improvements, deployments of these systems become more practical and cost-effective.

Data has been collected and studied by the Federal Railroad Administration (FRA) examining railroad track accidents and the safety of humans involved. In 2010, the FRA calculated there were 687 accidents and 102 fatalities at unmarked railroad grade crossings [1]. The FRA also calculated there were 656 accidents resulting from track failure in the same year [2]. It was concluded that there was a strong relationship between the number of accidents and the type of railroad track grade crossing at which they occurred. The highest number of accidents most often occurred at unmarked railroad grade crossings, with unmarked railroad grade crossing referring to a public or private roadway intersecting a section of railroad track where there are no cross-arms or warning lamps but may have static warning signs. These crossings can be found throughout the United States with the greatest occurrence in remote areas.
A study determined that by installing warning lamp systems at the thousands of unmarked grade crossings in the United States, the majority of accidents, both fatal and non-fatal, could be eliminated [3]. The cost of installing traditional electrical infrastructure in remote areas to provide power for warning light systems and rail monitoring systems is too high to be practical. This motivated the development of a power harvesting scheme that could be deployed in grade crossings in remote areas of the country to provide power for these systems.

2.1 Existing Technologies

Energy harvesting is a field of study that is growing quickly with new applications being created each day. Advances in electrical components, methodologies, and other technologies are reducing power consumption in electrical devices while also increasing the efficiency of power harvesters and the amount of power they can harness. Possible applications for this type of research range from medical, to consumer, to remote data monitoring, and so forth.

There are numerous types of sources for energy harvesting, including mechanical, thermal, electromagnetic, wind, solar, and human energy sources. Mechanical energy can be harvested from mechanical stress and strain along with vibrations. Thermal energy can be harvested from waste energy from heaters and furnaces, or from various friction sources. Electromagnetic energy can be harvested from coils, transformers and inductors in various ways. Human energy can be harvested from actions such as walking, bending or sitting.
Many energy harvesters tend to utilize piezoelectric materials for generating power. In 2003, Ottman, et al. [4] were capable of harvesting power by mechanically exciting a piezoelectric element at a rate of tens of Hz. Under laboratory conditions this circuit was capable of generating 31 mW. In 2003, using a piezoelectric device implanted in an artificial knee, Platt et al. [5] were able to generate 5 mW which was used to power a small microprocessor. This design subjected three 1.2 cm³ piezoelectric actuators to loading of typical human movement. In 2008, Feenstra et al. [6] developed a backpack that generated electrical power via changes in the tension in the strap. The backpack was modified with a mechanically amplified piezoelectric stack actuator, which replaced the strap buckle, and produced a mean power of 0.4 mW. In 2001, Shenck and Paradiso [7] described using piezoelectrics in shoe inserts to generate an average power of 8.4 mW under typical walking conditions.

In 2005, Rome et al. [8] used a rack and pinion gear to convert the vertical movement of a backpack into electrical energy. By using a set of springs, the backpack load was allowed to move relative to the frame of the backpack. The linear motion created was transformed into rotary motion for an electric generator, producing a maximum power of 7.4 W.

In 2008, using piezoelectric and inductive coil techniques, Nelson et al. [9] were able to harvest power from passing railcars. Using the bending in the rail created by loaded railcars passing, a piezoelectric element mounted on the bottom of the rail experienced a time-varying longitudinal strain, producing an average power of 1 mW. Similarly, an inductive voice-coil device was also tested that utilized the vertical
This technique also produced an average power of 1 mW.

These various techniques could be used to power sensors currently used in monitoring track health (e.g., strain gauges, temperature sensors) as they only require power in the range of a few mW. Because of this, piezoelectric and/or inductive voice-coil devices could be potential solutions to long-term, low-maintenance power supplies for distributed wireless networks. While these techniques could satisfy low power needs they are not adequate to meet the higher power demands of safety mechanisms such as lighted grade crossing warning systems.

There are others also pursuing harvesting power from railcar traffic. In 2010, Nagode [10] developed two prototypes that harvested power from railcar suspensions. The first design was based on a linear generator concept and was capable of producing approximately 1 W. The other design transformed translational motion into rotational to power a generator. This design was capable of producing peak powers of up to 70 W. Both prototypes were lab tested around the 1 Hz range with a maximum displacement of 0.75 inches. The following simulation could easily be modified to quickly predict power production over a larger range of variables.

In 2010, Hansen et al. [11] demonstrates the potential of a power harvester utilizing the mechanical energy of railroad track deflections to power crossing lights. It was determined that multiple devices of this type could generate the needed power for the lights in areas of relatively high-speed train traffic. This design is the basis for the computer simulation and control system discussed in this thesis.
2.2 Power Requirements

Current railroad crossing warning systems typically consist of eight 12-inch diameter low-power light emitting diode warning lamps. These high efficiency lamps satisfy the federal requirements for illumination and draw power on the order of 10 W per lamp [12].

For remote areas that have the necessary electrical infrastructure for warning lights, the setup is such that there are four lamps per direction of traffic with two sets of two lamps that alternate off and on. With this arrangement, four lights will be illuminated the entire time the crossing is active. According to the American Railway Engineering and Maintenance-of-Way (AREMA) standards, to meet the mandated intensity the nominal power necessary by the lights is approximately 8 W at 9 VDC [15]. The total power required whenever a grade crossing warning lamp system is operational becomes approximately 40 W.

2.3 Power Harvesting Device

To achieve these power requirements two power harvesting devices were designed and tested. Both devices tested operate in very similar manners. The devices span across two rail ties and are directly driven by the vertical displacement of the rail and tie(s) due to passing railcar traffic. The devices utilize this displacement and translate the linear motion into rotational motion. This rotational motion is then used to rotate a permanent magnet direct current (PMDC) generator which produces power for our electrical system.

To convert the linear motion of the rail deflection into rotational motion that would be used by the PMDC generator, a rack and pinion gear system was used. The rack
is mounted in the ground and kept stationary while the pinion gear along with the entire device move up and down with the rail and tie(s). Depending on the design, the rotary motion of the pinion gear was either rectified by a single one-way clutch or a combination of two one-way clutches and a system of gears. Without these clutches, as the rail deflects down and then back up, the pinion gear would oscillate between clockwise and counterclockwise motion. With the generator being designed for one way motion the oscillation between the two would have a cancelling effect on power production.

The single one-way clutch prevents this alternating clockwise and counterclockwise rotation of the generator and only allows the rotary motion created from the downward deflection of the rail to be translated through the clutch and into the generator. Because this design only utilizes the downward motion of the track it is only taking advantage of half the power that could be generated. Utilizing a combination of two one-way clutches, both the downward and upward motion of the track can be harnessed, maximizing power production.

With the deflection of the rail being minimal, approximately 0.05 to 0.75 inches, amplification of the rotary motion is necessary to rotate the generator with enough speed to produce a sufficient amount of power. To achieve this amplification a planetary gearhead was used. The single one-way clutch system utilized a 1:50 planetary gearhead whereas the double one-way clutch system used a 1:100 planetary gearhead. With every rotation of the pinion gear the generator would rotate either 50 or 100 times depending on the design.
The amplified rotational motion of the shaft is then transferred into a PMDC generator to produce the required electrical energy. Direct current generators allow for easy integration of multiple power harvesting devices together. Also integration with an electrical storage system and the crossing lights is much simpler as opposed to alternating current. These generators are small, reliable, and durable.

Two different generators of similar design were used with the power harvesters. Both generators come from the same manufacturer and are typically used in small renewable energy projects such as wind turbines or bicycle power. For the first generation single one-way clutch system a generator with smaller power production capacity was used [16]. This generator operates within the range of zero to 5,000 revolutions per minute and produces a current in the range of zero to 2.5 amps. For the second generation double one-way clutch a larger capacity generator was used [17]. This generator operates within the range of zero to 5,000 revolutions per minute and produces a current in the range of zero to 8 amps.

The overall design of the first generation single one-way clutch power harvester is shown in Figure 2.1 [13], while the overall design of the second generation double one-way clutch power harvester is shown in Figure 2.2 [14].
Figure 2.1 – First generation single one-way clutch power harvester design.

Figure 2.2 – Second generation double one-way clutch power harvester design.
Chapter 3. SIMULATION

With the associated high forces and unique oscillatory motion of the rail all being translated into the power harvester, controlled lab tests on the different prototypes become difficult to accurately conduct. Along with this, on-site tests are also difficult due to uncontrollable variables in train weight, speed, and track conditions. Because of these problems a simulation was needed to help validate the results from the different prototypes and to also determine favorable characteristics such that improvements could be made to the design. To calculate the expected results of different prototypes MATLAB code was used for the simulation. The MATLAB code was split into seven separate files corresponding to specific tasks.

Depending on user inputs the code first calculates maximum track deflection. Then, depending on the type of gear ratios, the simulation determines the average power production for the generator. With that the program calculates the minimum number of devices needed to power the warning lights along with the appropriate positioning for the devices along the section of track. Power losses through the wiring are then determined and an adjustable duty cycle is applied to the warning lights. With all these calculations taken into consideration the simulation goes through a pass/fail criterion where it tests to see if the lights have enough power to remain on the entire time it would take the train to pass, while also checking that there is minimal power loss to a rechargeable battery which is used in conjunction with an electrical control system.

The flow diagram of Figure 3.1 shows the steps for a typical simulation with each step described in detail later in this chapter. The positioning loop represents the addition
of more devices as the simulation runs until the pass criteria are met. The train simulation loop represents the total time it would take the train to pass over the area of interest, which varies with train speed and other variables. If the user were to choose the acceleration option, the process would be altered slightly such that the deflection of the rail and power generation for a single device would be calculated for every cycle in the train simulation loop.

Figure 3.1 – Flow diagram of overall simulation.
3.1 Graphical User Interface

The first file used in the simulation is for a graphical user interface (GUI), labeled Work1 and shown in Appendix A. This file builds the GUI, as shown in Figure 3.2, and is how the user will interact with the simulation and input the numerous variables that are available. This GUI allows for quick and easy changes be made to the simulation settings without having to go into the code itself to make changes. This is useful for users less familiar with the programming.

![MATLAB graphical user interface](image)

Figure 3.2 - MATLAB graphical user interface
For this simulation the user has control over train variables such as speed, changes in acceleration, number of railcars, and the average weight for a single rail car. These variables directly affect power production for the power harvesting device and the overall amount of train-device interaction time the simulation will cover.

Along with control over train variables, the user can also control variables associated with the build details of the power harvesting device and how each device will be used. These variables include the gear ratio of the planetary gearhead with options of 1:25, 1:50, 1:75, and 1:100, the option of harvesting only the downward motion of the rail and tie(s) or both the downward and upward motion, how the devices are positioned (i.e., spread out along the track or centered at the railroad crossing), and the minimum number of devices to start the simulation.

Finally the user has the option to control the amount of power the device can produce, alter the time at which the warning lights turn on, and control which figures are displayed at the conclusion of the simulation (e.g., displacement, velocity, acceleration, revolutions per minute, power production, battery life, and duty cycle). When the user chooses to control the amount of power the device produces, all calculations related to the deflection of the rail are ignored. The simulation uses this inputted value to determine the minimum number of devices needed assuming the device actually produced this much power.

3.2 Equations of Motion

After the user has entered all the necessary variables into the GUI and clicked “Simulate,” the next MATLAB file, labeled fconversions and shown in Appendix A,
converts all the variables into similar units. Also by using the equations of motion, calculations are done to determine the distance the train will cover for the entire simulation, the final velocity of the train, and the amount of time the warning lights will be on.

For the simulation the distance the train will cover is dependent on train speed and whether or not the user chooses to alter the distance at which the warning lights turn on. For a standard simulation in which no changes are made to the timing of the warning light system, the warning lights turn on 20 seconds before the train will reach the crossing as per the minimum FRA regulations [15]. With this setup, the distance between the front of the train and the crossing when the lights turn on is calculated by Equation 3.1.

\[ d = v \times t_c \]  

(3.1)

where:

- \( d \) is the distance from the crossing when the light turns on
- \( v \) is the velocity,
- \( t_c \) is the time it takes for the train to reach the crossing

If the user inputs a certain distance at which the lights turn on, then \( d \) becomes that distance and the time \( t_c \) is calculated from Equation 3.1. For both cases the distance \( d \) is multiplied by 2 to take into account train traffic being able to go both ways, which will create symmetry about the railroad crossing; this will be called the total distance hereafter.
Because the user has the ability to simulate either a train at constant speed or one which is accelerating, the velocity of the last railcar as it passes both the railroad crossing and the point at which it the train traveled the total distance become critical. These velocity values determine the how long the crossing lights remain on and the total time for the simulation to run. To calculate these velocities Equations 3.2 and 3.3 were used,

\[
\begin{align*}
v_{fc} &= (v_i^2 + 2 \cdot a \cdot (d + l))^{1/2} \\
v_f &= (v_i^2 + 2 \cdot a \cdot (2 \cdot d + l))^{1/2}
\end{align*}
\]  

(3.2) 

(3.3)

where:

\(v_{fc}\) is the train’s velocity as the final railcar passes the crossing,

\(v_i\) is the train’s initial velocity,

\(a\) is the acceleration of the train,

\(l\) is the length of the train,

\(v_f\) is the train’s velocity as the final railcar passes the total distance.

These equations work both when the train remains at a constant speed and when nonzero acceleration is experienced. If there is no value entered for acceleration in the GUI, then the default value of \(a\) is zero and the final velocities are the same as the initial velocity given.
To calculate the total time the crossing lights remain on and the total time for the simulation to run, two sets of equations are required. The first set of equations, Equations 3.4 and 3.5, are used if there is no acceleration occurring.

\[ t_t = \frac{(2d + l)}{v_i} \]  \hspace{1cm} (3.4)

\[ t_l = t_c + \frac{l}{v_i} + 12 \] \hspace{1cm} (3.5)

where:

- \( t_t \) is the total time of the simulation,
- \( t_l \) is the total time the light remains on.

For a typical simulation the total time the warning lights must remain on to meet FRA regulations is 20 seconds before the train reaches the crossing, plus the time it takes the entire train to pass, plus an additional 12 seconds after the final car has passed the crossing [15].

For an accelerating train, a different set of equations must be used, shown as Equations 3.6 and 3.7.

\[ t_t = \frac{(v_f - v_i)}{a} \] \hspace{1cm} (3.6)

\[ t_l = \frac{(v_{fc} - v_i)}{a} + 12 \] \hspace{1cm} (3.7)
3.3 Track Deflection and Power Generation

With the values determined for the total distance, the time the simulation will cover, and the total time the lights remain on, the next file executed is named fdeflection and is shown in Appendix A. This file is responsible for determining the total track deflection, generator rotational speed, and the average power generation for a single device. If the train is not accelerating this file is only executed once, corresponding to this time, and the value calculated for the average power generation is used throughout the entire simulation. If the train is accelerating then this file is continuously executed and values continuously updated, which will be discussed in a later section.

3.3.1 Track deflection

With vertical track displacement being the driving force for the prototype, it becomes the basis of the simulation. To accurately simulate the vertical deflection of the railroad track due to loading by railcar wheels, the Winkler model of vertical track deflection was chosen, as it is a validated model for track displacement [18]. This model describes the deflection of a beam resting on a continuous, uniform elastic foundation. More specifically, the deflection of the beam under an applied load is linearly proportional to the pressure between the base of the rail and the foundation [19]. The model linearly relates rail deflection to a single point load and has a non-linear relationship to track modulus. This linearity allows for the combination of multiple axle loads through the property of superposition, which is advantageous. The Winkler model of vertical track deflection is shown in Equation 3.8.

\[
y(x) = \frac{p\beta}{2\mu} e^{-\beta x} [\cos(\beta x) + \sin(\beta x)]
\]  

(3.8)
where:

\[
\beta = \left( \frac{u}{4EI} \right)^{1/4}
\]

\(y(x)\) is the track deflection,

\(P\) is the applied load to the rail,

\(u\) is the track modulus,

\(E\) is the modulus of elasticity of the rail,

\(I\) is the second moment of area of the rail,

\(x\) is the distance between the applied load and the point of measured deflection.

With a simple substitution for \(x\), Equation 3.8 becomes a function of the time, \(t_d\), the wheel contact is away from crossing over the device. Throughout the entire simulation the values for \(u\), \(E\), and \(I\) remain constant at 3,000 psi, 30,000,000 psi and 93.7 in\(^4\) respectively [20], while \(P\) and \(t_d\) are dependent on the user input for the total weight of a single railcar and velocity of the train. Because \(P\) is the applied load at the point of contact for a single wheel, the total weight is broken up evenly across each of the eight wheels associated with a single car.

To determine the total deflection experienced by the power harvesting device the \textit{fdeflection} file simulates two connected railcars, with dimensions of a standard coal car quad-hopper [21], passing over the device. Because the device only uses the deflection
from a single rail, eight points of contact will be used to represent the wheels of the two cars.

Before the file begins to simulate the passing of two railcars, time offsets for each of the wheels must be calculated. The time offsets represent the amount of time it takes a specific wheel to reach the device after the front edge of the first railcar passes over. By knowing distances between each of the wheels in relation to the very front of the railcar these offsets can be calculated using Equation 3.9.

\[
offset_i = \frac{d_i}{v} \quad (3.9)
\]

where:

- \(offset_i\) is the time for a specific wheel to reach the device,
- \(d_i\) is the distance said wheel is from the device,
- \(v\) is the velocity of the train.

Using the Winkler model and appropriate time offsets for each of the eight wheel contacts, the deflection can be calculated for each single contact at a given point. The eight contacts can then be superimposed to determine the total deflection of the rail experienced at the device’s position due to all the contacts. This summation can then be done over the period of time that it takes the front of the first railcar and the back of the second railcar to pass over the device. The total summation over a given period of time is shown in Equation 3.10.
where:

\[ y(t) = \sum_{i=1}^{i=8} \left[ \frac{P\beta}{2\mu} e^{-\beta(t-\text{offset}_i)} \left( \cos(\beta(t-\text{offset}_i)) + \sin(\beta(t-\text{offset}_i)) \right) \right] \quad (3.10) \]

\[ t \] is the time it takes for one railcar to pass,

\[ \text{offset}_i \] is the time it takes for each specific wheel to reach the device.

With Equation 3.10 the total deflection for varying train speeds and weights can be easily determined. For the simulation, values for the train’s velocity can range anywhere between 10 mph and above with a probable maximum train speed being approximately 60 mph, while the railcar weight can range from 58000 lbs for an unloaded railcar up to 280000 lbs for a loaded railcar [21]. It should be noted that the speed of the train does not affect the distance the track deflects but only determines the frequency at which it does so. Within the range of a typical train velocity the rail will always experience the same amount of deflection independent of speed.

For a fully loaded railcar of 280000 lbs traveling at 55 mph the deflection experienced by the rail at a given point as a function of time is shown in Figure 3.3.
Figure 3.3 – Track deflection profile for a loaded train at 55mph.

The maximum vertical track deflection predicted for this setup was 0.25 inches. The maximum deflection occurred when the coupler of the two railcars was directly over the device. The vertical blue lines in Figure 3.3 represent the moment in time when an individual wheel crosses directly over the device.

The vertical track deflection for an unloaded train of 58000 lbs was also determined by using the Winkler model with the same constant values. Values used were 3,000 psi for the track modulus (representing the stiffness of the composite structure supporting the rail), $u$, 30,000 psi for the modulus of elasticity of the rail, $E$, and 93.7 in$^4$ for the second moment of area, $I$, as previously. For an unloaded railcar of 58000 lbs
traveling at 55 mph the deflection experienced by the rail at a given point as a function of time is shown in Figure 3.4

The maximum vertical track deflection predicted for this setup was 0.05 inches. The maximum deflection occurred when the coupler of the two railcars was directly over the device. The vertical blue lines in Figure 3.4 represent the moment in time when an individual wheel crosses directly over the device.

As shown in Figure 3.3 and Figure 3.5 the deflection of the rail rebounds close to its original position between the set of axles for a single railcar. This rebounding of the rail will occur for all train velocities.

Figure 3.4 – Track deflection for an unloaded train at 55 mph.
3.3.2 Generator Speed

To determine the average power of a single device the generator speed must be calculated from the vertical track deflection. To do this the Winkler model was differentiated, and the speed at which the track deflects was determined. The result of differentiating Equation 3.10 is Equation 3.11.

\[ v(t) = \sum_{i=1}^{l=8} \left[ -\frac{P\beta^2}{\mu} e^{-\beta(t-offset_i)} \ast \sin(\beta(t-offset_i)) \right] \]  

(3.11)

where:

\( v(t) \) is the velocity at which the track is deflected.

Using the same constant values for \( u, E, \) and \( I \) as before in section 3.3.1, the velocity profile for a given train velocity and weight can be determined.

The plot for a loaded train traveling at 55 mph is shown in Figure 3.5.
The maximum vertical velocity for a loaded train at 55 mph was predicted to be 3.5 inches per second. The maximum downward velocity occurs right before the first wheel of the back axle crosses the device and the maximum upward velocity occurs right after the last wheel of the front axle crosses the device. Unlike the deflection profile, the velocity of the train affects the maximum velocity of the rail motion. Change in deflection occurs more quickly the faster the train travels, and thus results in a faster rail velocity. The vertical blue lines in Figure 3.5 represent the moment in time when an individual wheel crosses directly over the device.

The plot for an unloaded train traveling at 55 mph is shown in Figure 3.6.
The maximum vertical velocity predicted was 0.7 inches per second and it also occurred when the coupler of the two railcars was directly over the device.

With the vertical track velocity determined, the MATLAB file then calculates the speed at which the generator would rotate due to the velocity of the rail. With both prototypes the rack gear is stationary in the design of the power harvesting device but it is moving up and down relative to the pinion gear since the device translates vertically with the track system. The angular speed of the pinion gear with respect to time as the railcars pass over is calculated by using Equation 3.12.

$$\omega(t) = \frac{v(t)}{r} \tag{3.12}$$
where:

\[ \omega(t) \] is the angular velocity of the pinion gear,

\[ v(t) \] is the vertical velocity of the rack gear relative to the pinion gear,

\[ r \] is the pitch radius of the pinion gear.

For both prototypes, the rack and pinion gear remain the same size with the pinion gear having a pitch radius of 0.79 inches. The angular velocity is then converted into revolutions per minute (rpm) using Equation 3.13

\[ \text{rpm} (t) = \omega(t) \times \frac{1 \text{revolution}}{2\pi \text{radians}} \times \frac{60 \text{ sec}}{\text{min}} \times G_r \times M_t \quad 3.13 \]

where:

\[ G_r \] is the gear ratio of the planetary gearbox,

\[ M_t \] is the type of motion.

Depending on user inputs for the planetary gearbox ratio the value for \( G_r \) can vary from 1:25, 1:50, 1:75, and 1:100. The value for \( M_t \) can either be 1, corresponding to when the power harvester only harnesses the downward motion of the rail, or 1.33 for when the power harvester harnesses both the downward and upward motion of the rail. Because the second generation power harvester has a slightly different design there is an extra gear connection and the value of 1.33 comes from the gear reduction between these two gears.

Because the single one-way clutch design can only utilize the downward velocity of the rail and the double one-way clutch rectifies the direction of the rail such that it can
utilize both directions alterations must be made to the revolutions per minute to represent these designs accurately.

With the first generation design only the downward motion of the rail can be utilized as mentioned earlier. Because of this any upward motion of the rail is useless. To represent this in terms of actual revolutions of the generator, any values for the revolutions per minute that correspond to the upward motion are set to zero. For a loaded train traveling at 55 mph, the revolutions per minute profile of the generator for a single one-way clutch device with a gear ratio of 1:50 is shown in Figure 3.7.

![Figure 3.7 – Generator shaft revolutions per minute for first generation design.](image)

With the second generation design the upward motion is rectified such that it can utilize both the downward and upward motion. To represent this, the absolute value of the
revolutions per minute as a function of time is used. By taking the absolute the revolutions per minute will all be in the same direction. For a loaded train traveling at 55 mph, the revolutions per minute profile of the generator for a double one-way clutch device with a gear ratio of 1:50 is shown in Figure 3.8.

Figure 3.8 – Generator shaft revolutions per minute for second generation design.

For the single one-way clutch of a loaded train traveling at 55 mph the average rpms was calculated to be 284.6, while for the double one-way clutch of the same train the average rpms was calculated to be 758.9.
3.3.3 Power production

With the instantaneous revolutions calculated the simulation continues and calculates the average power produced per single device. The program first calculates the period it takes for one car to pass. It then examines the data for the revolutions over the course of this time period directly in the middle of the total available data. This area corresponds to the time from when the middle of the first car passes over the device to when the middle of the second car passes. This area contains the most complete data when analyzing the two cars.

From available data specifications for the generators used the instantaneous open-circuit voltage for each specific generator can be calculated. This is done by taking the slope from the open-circuit voltage curve provided and multiplying it by the instantaneous rotational speed. Starting with an equation for a simple voltage divider shown as Equation 3.14 an equation for current can be derived and is shown in Equation 3.15.

\[ V_{out}(t) = \frac{V_s(t) \cdot R_t}{R_t + R_s} \]  \hspace{1cm} (3.14)

\[ I(t) = \frac{(V_s(t) - V_b)}{R_t + R_s} \]  \hspace{1cm} (3.15)

where:

- \( V_{out}(t) \) is the voltage measured across the generator terminals
- \( V_s(t) \) is the open circuit voltage,
$R_t$ is the resistance across the generator terminals (including resistance due to the battery).

$I(t)$ is the current,

$V_b$ is the voltage of the battery,

$R_l$ is the resistance across the terminal without the battery,

$R_s$ is the internal resistance of the generator.

For the simulation the values for the battery and the resistance of the load are constant. The battery that will be charged from the power harvesting device operates at approximately 12.6 volts. While the voltage of the battery can vary anywhere between approximately 11-13.4 volts, 12.6 was used because that is the battery’s nominal voltage. As for the resistance of the load a voltage divider will be used for on track deployment to allow for voltage readings and thus is taken into account for the simulation. This voltage divider plus any other resistances in the electrical circuit will have a total resistance equal to the internal resistance of the generator. Matching the total resistance of the circuit to the internal resistance of the generator maximizes power production. For the generator used with the first generation device the internal resistance is 21 ohms. For the generator used in with the second generation device the internal resistance is 7.7 ohms. The simulation will match the total resistance across the terminals $R_t$ with the internal resistance of the generator chosen. Depending on the generator, the values for the slope of the open-circuit voltage curve vary resulting in different values for the current.
With the current determined, the power being produced as a function of time can be calculated. Starting with Equation 3.16 to calculate power for a circuit, a simple substitution of known values is done such that it becomes Equation 3.17.

\[ P = V_{tot} \times I \]  \hspace{1cm} (3.16) \\
\[ P(t) = I(t) \times (V_b + I(t) \times R_I) \]  \hspace{1cm} (3.17)

where:

\[ V_{tot} \] is the total voltage drop.

For a loaded train traveling at 55 mph with a gear ratio of 1:50, the power as a function of time for the double one-way clutch design is shown in Figure 3.9. The average power production is calculated to be 92.03 watts with instantaneous power peaking at 520 watts.
For a loaded train traveling at 55 mph with a gear ratio of 1:50, the power as a function of time for the single one-way clutch design is shown in Figure 3.10. The average power production is calculated to be 11.9 watts with instantaneous power peaking at 141 watts.
Figurer 3.10 - Power production of first generation design with 1:50 gear ratio.

With these results the average generator speed, average current, and average power are all calculated by summing the respective functions over a period of one car and dividing by the total number of samples. For the case shown in Figure 3.9 the average generator speed was 758.9 rpms and the average current was 1.67 amps. For the case shown in Figure 3.10 the average generator speed was 284.6 rpms and the average current was .248 amps. These values are then outputted at the conclusion of the simulation.
3.4 Device Positioning

With all the calculations completed in sections 3.1 and 3.2 the program executes the next file labeled *fspace* as shown in Appendix A. By default the number of devices the simulation begins with is 1, but the user has the option to change this starting value to the number of devices they would like per symmetric half-length of simulated track. As the simulation progresses, this file is continuously called and the number of devices is increased by 1, as shown earlier in Figure 3.1. This loop is repeated until the simulation achieves the pass criteria, to be discussed later, or a maximum of 10 devices is reached.

The user has two options of how the devices will be placed along the track. The first option is to have the power harvesters evenly spaced out along the track over the total distance, which was calculated in the *fconversion* file, excluding having one at the crossing itself. The second option is to have all the devices placed directly at the crossing. Sample configurations for both are shown in Figure 3.11.

![Figure 3.11 – Configurations of device positioning.](image)

If the spread option is chosen for the simulation, the file only examines the distance that the train covers in 20 seconds, which was calculated as $d$ in section 3.1 and is half of the total distance. The file utilizes the symmetry about the crossing to complete
the positioning over the total distance. A row vector is created and stores the position of the devices by using Equation 3.18 is used.

\[ P_d[1, i] = d - (i - 1) \frac{d_s}{N_d} \]  

where:

\( P_d \) is a row vector of device positions,

\( d_s \) is the determined spacing,

\( N_d \) is the number of devices,

\( i \) is the matrix index, starting at 1 and increasing to \( N_d \).

As stated previously this file either starts with a single device or a number specified by the user. With the spread option, the position of device is mirrored on both sides of the crossing so the minimum starting number of devices for this option is two. No matter the number of devices chosen there will always be a device placed at the distance from the crossing that the warning lights turn on. This is to maximize the amount of time power is being produced without an overproduction. With this matrix of the device positions covering half the total distance, the file copies, reverses the order, and multiplies the matrix by -1. With the original matrix and the reverse matrix, a new matrix is created by concatenating the two. This new matrix contains the distances (ft) of each device from the crossing, with the devices on the starting half taking positive values and the devices on the ending half negative. For a simulation of a train traveling at 55 mph where 3 devices are used per half, the matrix would be \([1613, 1075, 538, -538, -1075, -1613]\). For
the spread option a row vector filled with zeros, corresponding with their position at the crossing is created, with the number of columns being equal to the number of devices.

3.5 Train Simulation

While the simulation loops through the *fspace* M-file and calculates the new positioning for each loop, a new M-file labeled *ftest*, shown in Appendix A, is executed. This file is responsible for determining whether or not with this specific set of user inputs can a set number of devices generate enough power to keep the warning lights on for the proper amount of time.

This file first uses the matrix of device positions created in *fspace* and creates a second row vector containing the times at which the train will cross over each device, which will be the time they start to produce power. This starting time for device power production is the time it takes the train to reach the device after the simulation begins, which is time \( t = 0 \). Along with creating a matrix for the starting times, another similar matrix is created containing the time at which the train finishes crossing each device and power production for that device stops. Depending on the user’s input for train acceleration the start and stop times for the devices are calculated by two different methods.

For a train traveling at a constant velocity the start and stop times for each device are calculated using Equations 3.19 and 3.20.

\[
\begin{align*}
    t_{ds}[1,i] &= \frac{(d-P_d[1,i])}{v_i} \\
    t_{df}[1,i] &= t_{ds}[1,i] + \frac{t}{v_i}
\end{align*}
\]
where:

\[ t_{ds} \] is a row vector of times the devices start to produce power,

\[ t_{df} \] is a row vector of times the devices finish producing power.

A simple diagram of this depicting the spread option is shown in Figure 3.12, where \( d_2 \) and \( d_3 \) are the corresponding distances from the first device to the subsequent two devices, \( V \) is the velocity of the train, \( S_{t_i} \) is the start time for each device, and \( L \) is the length of the train.

![Figure 3.12 – Start and end times of devices for spread option.](image)

For a train accelerating along the railroad track, the start time for each device is calculated using Equation 3.21.

\[
t_{ds}[1,i] = \frac{(v_i^2 + 2a(d - P_d[1,i]))^{1/2} - v_i}{a}
\]  

(3.21)
To calculate the time at which each device stops producing power the velocity of the train as it begins and finishes passing over the device must be determined using Equations 3.22 and 3.23.

\[
v_s[1, i] = v_i + a * t_{ds}[1, i] \quad (3.22)
\]

\[
v_f[1, i] = (v_i[1, i]^2 + 2 * a * l) \quad (3.23)
\]

where:

\( v_s[1, i] \) is a row vector of the velocity of the train as it begins passing over each device,

\( v_f[1, i] \) is a row vector of the velocity of the train as it finishes passing over each device.

With these velocity matrices determined the time at which each device stops producing power is calculated from Equation 3.24.

\[
t_{df}[1, i] = t_{ds}[1, i] + \frac{(v_f[1, i] - v_s[1, i])}{a} \quad (3.24)
\]

With start and end times for each device calculated the file begins to simulate the entire length of the train passing over the entire area of interest. Starting at time \( t = 0 \), corresponding to the time the crossing light first comes on, and continuing in increments of 1 sec through the total time it takes for the train to pass over the total distance, \( d \), the program begins to calculate the total power being produced, the total power being lost due to resistance in the wires, and the average power being drawn by the crossing lights.
As discussed previously, for a train traveling at a constant velocity the average power generation for a single device is only calculated once towards the beginning of the simulation and that value is used for each iteration of the power production simulation. However, if the train is accelerating the average power generation for a single device must be continuously calculated with updated velocity through the iterations. This average power value is only valid for a single iteration before it must be updated. This adds a significant amount of time onto the total time it takes for the simulation to complete.

For each 1-second increment the total power being produced by all the devices is calculated. The simulation checks to see how many devices are producing power at that specific time. This is done by determining if the value for time for that specific iteration falls in between the time a specific device starts producing power, $t_{ds}$, and stops producing power, $t_{df}$, if this is the case then that device is producing power. This is done for each device in the simulation. If it is determined that a device is producing power the average power production that was determined for a single device is added to the total power production. Along with this, a row vector is created that keeps track of which devices are producing power and in what order, which is used in determining power losses.

3.5.1 Power Losses Due to Resistance

Once the total power is calculated for a specific iteration, the amount of power lost due to resistances in the wires is determined. For the centered option, where all the devices would be located at the railroad crossing, power losses due to resistances in the
wires are considered negligible in comparison to the total power being produced and thus are not calculated. For the spread option, since the devices can be such a large distance from the crossing, up to approximately 1600 feet for a train traveling 55 mph, the power lost through the wires cannot be ignored.

Power losses due to the resistance of the wires vary depending on the number of devices producing power and at which position along the wire is being examined. Because the flow of current travels inwards towards the crossing and the current from each device is summed together as it goes, each section of wire between devices will have a different amount of current flowing through it and thus a different amount of power for that section of wire. An example of the different power values across each section of wire for one half of the overall positioning is shown in Figure 3.13.

```
<table>
<thead>
<tr>
<th>ON</th>
<th>8 W</th>
<th>ON</th>
<th>16 W</th>
<th>ON</th>
<th>24 W</th>
<th>OFF</th>
<th>24 W</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td>B</td>
<td></td>
<td>C</td>
<td></td>
<td>D</td>
<td></td>
</tr>
</tbody>
</table>

Figure 3.13 – Power production through sections of wire
```

With this example the average power production for a single device with no power losses to the wire is 8 watts and the three devices positioned farthest from the crossing are producing power while the device closest to the crossing is not producing power. For the outermost section of wire labeled A, only the current being produced for the outermost device is flowing through it and thus the power for that section is 8 watts. For the section of wire labeled B, the current being produced for the two outermost devices is flowing through it and thus the power for that section is 16 watts. This pattern
continues until reaching the crossing where the current flowing through the closest section of wire will be the total current produced for all active devices and thus the power for that section will be the sum of the powers which in this example is 24 watts.

To simulate the power lost the M-file labeled *fpowerlosses*, shown in Appendix A, is executed for each iteration of time during which the train passes over the total distance. The file first separates the positions of the devices into two halves. The first half corresponds to having the outermost device turn on first and then subsequent devices turning on working inwards towards the crossing, which represents a train approaching the railroad crossing. The second half corresponds to having the innermost device turning on first and then subsequent devices turning on working outwards away from the crossing, which represents a train leaving the railroad crossing. This is done to keep track of how much power is flowing through each section of wire between devices. The power between each section will vary depending on how many devices are producing power and in which order. Towards the end of the simulation as devices stop producing power, starting with the outermost and working inwards, the current flowing from each device as they stop must be removed from the subsequent sections of wires. The power losses through each section are then determined and summed for that half. To calculate power loss through a length of wire Equation 3.25 is used.

\[ P_l = L \times R \times I^2 \]  

(3.25)

where:

- \( P_l \) is the power lost in watts,
- \( L \) is the length of wire,
$R$ is the resistance of the wire in ohms per foot,

$I$ is the current flowing through the wire.

The wire used to connect the devices is 8 gauge wire and it has a resistance of 0.0006282 ohms per foot.

Before the file begins to simulate either half of the crossing it first determines the total power that that particular half is producing. This is simply done by counting how many devices are producing power and multiplying it by the average power produced per single device. The order in which the devices are producing power is not important for this calculation.

For the first half of devices, the simulation begins to search for the nearest device to the crossing that is producing power. This is done by looping through the matrix that contains the power production indicators for each device that was created in the previous section. Once the simulation finds the innermost device that is producing power it then can calculate the power losses across the section of wire between the device and the crossing lights. Because this is the device closest to the crossing lights the current flowing through this section must be equal to the total current being produced for that half regardless of which other devices are generating current. With power across this section known, the power loss for that section of wire can be calculated using a variation of Equation 3.25 shown as Equation 3.26

$$P_t = P_d[1, i] \ast R \ast \left(\frac{R_s}{V}\right)^2$$

where:
$P_s$ is the power for a section of wire,

$V$ is the voltage.

The simulation then continues, and finds the next closest device to the crossing that is producing power, which would be the next adjacent device. The total power across this section and any subsequent sections becomes the power for the previous section minus the average power production of a single device shown in Equation 3.27.

$$P_s = P_{sp} - P_{pd}$$

(3.27)

where:

$P_{sp}$ is the power for the previous section of wire,

$P_{pd}$ is the power production for a single device.

Because the previous device will always be located closer to the crossing the current being produced from it does not flow through this section of wire. With the current and power known for each section wire, the power loss through each section after the first can be calculated using Equation 3.28.

$$P_l = (P_d[1,i] - P_d[1,i-1]) * R * \left(\frac{P_s}{V}\right)^2$$

(3.28)

Each time the power loss for a section of wire is calculated it is added to all the previous power losses. Upon completion of the loop the total power losses for a specific iteration is calculated for this half of devices.
The second half of devices is calculated through the exact same manner as the first half with the only difference being that the positions of these devices have a negative value which is corrected for these calculations.

The total power losses calculated for each half are then summed up and output back into \textit{fiest} file. The total power losses are then subtracted from the total power produced giving the total power available to the lights and battery for that specific iteration. The file \textit{fiest} then continues to execute calling back to the \textit{fpowerlosses} file for each iteration of entire simulation time. A flow diagram for calculating power losses is shown in Figure 3.14.

The total amount of power lost due to resistance of the wires varies depending on the overall setup of the simulation. The amount of current flowing through a section of wire is the biggest determinant of power losses because power losses are dependent on the square of the current. For example, if several devices are being used, with each device producing a small amount of current, the power losses are going to remain relatively low through the sections of wire farthest from the crossing since current is lowest at points farthest from the crossing and power loss grows with the square of current. Only the few sections of wire closest to the crossing will have significant current flowing through them and these sections will experience the most power losses. If only a few devices are being used that can produce large amounts of current the power losses are going to be significantly higher. As discussed before, for the spread option there will always be a device positioned at the point where it would take the train 20 seconds to reach the crossing. Because of the higher amount of current and because of the significant length of wire this current will be flowing through the power losses are going to be much higher.
Figure 3.14 – Flow diagram for calculating power losses through sections of wire.
3.5.2 Duty Cycle and Battery Life

If the value of time for an iteration is less than the total time the lights are on, calculated as $t_i$ in section 3.3.1, then the power required for the lights to be on must be subtracted from the total power available. To minimize the power consumption of the lights an adjustable duty cycle is implemented. The simulation begins with the assumption that at any point in time half of the warning lights at the crossing will be on while the other half of the lights are off alternating every 1 second. However, if power production does not meet the required power for the lights, the simulation can make adjustments to the timing of the warning lights, to minimize the required power.

If the power production is less than the power required by the lights, approximately 40 watts, the “dead time” between flashes is increased. The dead time refers to the amount of time between flashes when neither set of lights is illuminated. With the beginning assumption, the dead time is zero but if the power production is low the dead time will increase proportionally up to 0.5 seconds, half the illumination period, at which point the dead time between flashes will no longer increase as this is a reasonable limit on the duty cycle.

To calculate optimum dead time between flashes the M-file labeled \textit{fdutycycle}, as shown in Appendix A, is executed for every iteration that the lights are on. This file keeps a running average of the total power available to the lights over the last five iterations, which would be 5 seconds. This total power is after power losses are subtracted. The average power available is divided by the power required by the lights resulting in the amount of time the light will remain on. If the value is greater than one
there is enough power being generating such that there is no dead time between the lights flashing. If the value is less than one, then that value becomes the amount of time the lights will remain on in seconds, with the dead time being the difference between that time and one. The minimum amount of time the lights will be on is 0.5 seconds and if the value goes below this the simulation will autocorrect and raise it to 0.5. With the amount of time calculated that the lights will remain on for each iteration the M-file `fdutycycle` is exited and the simulation returns back to where it was in the M-file `ftest` and begins to check the amount of power remaining in the batteries.

While the warning lights are on, for each iteration, the amount of power drawn from the battery must be calculated. This is done by multiplying the amount of power required by the warning lights to remain on by the time the lights are on. If the warning lights are on for the minimum of 0.5 seconds than only 20 watts are required. The total power available is then subtracted from this value to determine how much power is required by the battery. The total power remaining in the battery is then calculated by subtracting the needed power from the previous power remaining in the battery. These calculations can be combined and shown in Equation 3.29.

\[
P_b = P_{bp} - P_{ta} - 40 \times t_L
\]  

(3.29)

where:

- \( P_b \) is the power remaining in the battery,
- \( P_{bp} \) is the power remaining in the battery from the previous iteration,
- \( t_L \) is the amount of time the lights remain on per flash,
$P_{ta}$ is the amount of power available to the lights produced by the devices.

The calculations for the remaining battery life along with the duty cycle of the lights are only calculated while the time of the iteration is less than the total time the lights are on. Once the iteration reaches the point in time where the lights are no longer on all the available power produced thereafter is added to the power remaining in the battery.

3.5.3 Pass/ Fail Criteria

If at any point during the train simulation the power remaining in the battery goes below a certain threshold the file immediately stops. This threshold can be set to zero corresponding to when the battery is completely drained and the lights are no longer able to remain on or some value that would represent that there is not enough power being produced and the battery would be completely drained after a few trains passed by. If this threshold is reached the train simulation stops, exits the M-file $ftest$ and returns to the M-file $fspace$. Upon returning to the M-file $fspace$ another device is added into the calculations. New spacing and positioning of the devices are determined before the file $ftest$ is executed again. If the number of devices reaches 5 per half the simulation is deemed unsuccessful and the user inputs should be altered.

If the train simulation is able to run to completion, which corresponds to the value of time for the iterations reaching the total time it takes the train to cover the entire distance, then the simulation is determined to be successful. If successful the overall simulation ends and the desired information is displayed to the user.
Chapter 4. SIMULATION RESULTS

Various simulations were conducted such that the effects that different variables have on power production could be analyzed along with how positioning of the devices affects the overall success rate of the simulated device installation.

4.1 Device Variables

Several simulations were completed that tested the different parameters affecting power production with the results for average generator speed, average current and average power shown in Figures 4.1 and 4.2. Figure 4.1 shows the simulation results for a train traveling at 55 mph with different values for gear ratios, load, and type of device design while Figure 4.2 shows the simulation results for a train traveling at 10 mph with the same variables.
Figure 4.1 – Simulation results for a train traveling at 55 mph with different values for gear ratio, load, and device design.

Figure 4.2 - Simulation results for a train traveling at 10 mph with different values for gear ratio, load and device design.
From these results it was determined that the average generated speed is directly proportional to the gear ratio, the load, and the speed of the train. As the gear ratio doubles so does the generator speed, while a loaded train increases the generator speed by a factor of 4.8 over that of an unloaded train. The factor of 4.8 comes directly from the ratio in the weight of a loaded train to an unloaded train. Similarly the generator speed increases by the same factor as the ratio between the speed of a train traveling at 55 mph to 10 mph, which is 5.5. It was also determined that the double one-clutch design increases the generator speed by a factor of 2.66 over that of the single one-clutch with the other variables held constant. It would seem that the double one-way clutch would simply double the generator speed of the single one-way clutch but this is not the case since along with harvesting both the downward and upward motion of the rail, there is a slight design difference between the two designs as mentioned previously. With the double one-way clutch there is an additional gear ratio of 1.33.

The results for average amperage do not have a direct relationship with the average speed as one might expect. Examining the results for a loaded train traveling at 10 mph, shown in Figure 4.2, with the single one-way clutch design and a gear ratio of 1:50, the average generator speed is 51.38 rpm with an average current of 0.01 amps. Next examining the same loaded train traveling at 10 mph with the double one-way clutch design with a gear ratio of 1:25 the average speed is 69.14 rpm while the average current is 0. Even though the average speed for this case is more than the first case the amperage is lower. This is due to the fact that the generator does not produce current at all speeds. When charging a battery, a generator must first reach the same voltage as the battery before any current will flow.
With the simulation the battery voltage is set at 12.6 volts so the voltage across
the generator terminals must reach this before the current will flow. From the open-circuit
curve for each generator and from previous calculations the double one-way clutch must
first reach 284 rpm before it will start producing current while the single one-way clutch
will start producing current at 252 rpm. While the resultant average speed is higher for
the double one-way clutch it never reaches the minimum of 284 rpm to produce current.
A comparison of the instantaneous speed for the two cases is shown in Figure 4.3.

![Figure 4.3 – Comparison of instantaneous generator speed for a first generation device
with 1:50 gear ratio and second generation device with a gear ratio of 1:100](image-url)
Because of the need for generators to reach a specific shaft speed before they will start producing current a direct relationship between the average speed and average amperage is not possible.

The results from the various simulations help predict the power production of the power harvesting device under different conditions for optimizing future designs. It is clear from the results that the double one-way clutch is superior to the single one-way in terms of generating the most power. For a loaded train traveling at 55 mph the double one-way clutch is capable of producing a maximum 276.87 watts while the single one-way clutch can only produce a maximum of 48.81 watts. While 48.81 watts would be sufficient to power the warning lights with a single device, if the train were to travel slightly slower or was not as heavy as the assumed loaded train, or the track system stiffer than assumed, multiple devices could be needed.

Also from the results it should be noted that for an unloaded train traveling at 10 mph neither device design is capable of producing any power and that the warning lights would have to rely completely on the rechargeable battery.

### 4.2 Device Position

With the double one-way clutch being determined as the superior design, analysis can be done with how the device should be positioned. Both options of either being spread out along the track or centered at the crossing have their pros and cons.

If the spread option is chosen, power production when the lights first turn on will be low because only a single device is producing power. If the center option is chosen, there are no devices producing power until the train reaches the crossing. With both cases
the initial drain on the battery will be large because of the lack of power being generated by the power harvesters for the lights.

With the spread option as the train approaches the crossing and while it is passing more devices will become active. This results in a gradual gain in power production, whereas with the centered option all the devices begin to produce power simultaneously resulting in a large spike in power. A more gradual increase is typically desirable because if there is more power being produced than what both the warning lights and the battery can use, which can occur if the devices are all centered at the crossing, then a charge controller must divert that extra power to ground such that the lights and the battery are not damaged. The power that gets diverted is essentially wasted. Along with the gradual increase the power harvesters are positioned such that after the warning lights turn off the devices continue to produce power charging the battery. This is unlike the centered option where once the train passes the crossing the lights still remain on another 12 seconds drawing power from the battery alone. This makes the spread devices the better option; however, as discussed in Chapter 3 there are large associated power losses due to the resistance in the wires. To examine how each option affects power production and battery life the power plots were graphed over the course of the entire simulation.

From the previous simulation results it was determined that a double one-way clutch is capable of producing the most power at 276.87 watts. Using the variables to achieve this power production the simulation was run with the spread option chosen to determine the minimum number of devices needed. The simulation determined that a minimum of 2 devices were needed; a single device would have been sufficient with this
example but there must be symmetry. The result of this simulation is plotted in Figure 4.4.

Figure 4.4 – Power production and battery life of spread out, second generation devices, for a loaded train traveling at 55 mph option

Examining the plot in Figure 4.3 it is obvious that with this device design there is more than enough power for the crossing lights. For the first 40 seconds only a single device is producing power. After the 40 seconds the device located at the opposite end also begins to produce power during the time that the train is passing over both devices so that the power production is doubled. After 80 seconds the train has completely passed by the first device and only the second device is producing power. While the maximum power calculated for this device was 276.87 watts, because of power losses the warning
lights and battery only receive 142 watts from a single device. Because the lights only require 40 watts the rest of the power is used to charge the battery. From the simulation the battery is continuously being charged which results in the battery remaining at full power.

Running the same simulation with the exception that the centered option is chosen results in Figure 4.5.

Figure 4.5 - Power production and battery life of centered, second generation device, for a loaded train traveling at 55 mph.

With this simulation only a single device was needed to provide the necessary power to the lights. As shown in Figure 4.5 at the beginning of the simulation there are no devices producing power. This causes a large drain on the battery. Once the train
reaches the crossing the power harvester begins to produce power. Because it is located right at the crossing power losses are negligible and it can produce the full 276.87 watts that the device was calculated to be capable of. Because there is such an excess of power the battery becomes completely charged almost immediately. Once the battery becomes fully charged any excess power is wasted because neither the lights nor the battery can use it. Once the train passes the power harvester stops producing power while the lights continue to stay on for 12 seconds drawing power. With the centered option the battery will never finish completely charged because of these 12 seconds when there is consumption but the power harvester is inactive.

Simulating a device that produces far less power than previously mentioned resulted in a much more gradual power production. Also the battery maintained a much more consistent charge. Power production for a single one-way clutch device with a gear ratio of 1:50 was simulated for a train traveling at 55 mph. This device was calculated to produce an average power of 11.83 watts. With the spread option chosen only a total of four devices were need to power the lights and keep the battery fully charged. The results of the simulation are shown in Figure 4.6.
Figure 4.6 - Power production and battery life of spread out, first generation devices, for a loaded train traveling at 55 mph.

Similarly to the centered option there is an initial decrease in battery charge due to the limited power production of a single device. As the simulation progresses and more devices begin to produce power the battery slowly becomes fully charged. As the number of devices producing power starts to decrease the warning lights begin to draw power from the battery again. Once the lights turn off the remaining device that is producing power charges the battery back to full.

Running the same simulation as previously with the exception that the centered option is chosen produces similar results as in Figure 4.5. To provide enough power to the lights and to keep the battery as close to fully charged as possible a total of four
devices were needed. Comparing this to the results of the spread option reveals that the spread option is the better choice. Because it requires the same number of devices to power the crossing lights and charge the battery the spread option is better because the battery is fully charged at the completion of the simulation.

The simulation results shown in Figures 4.4, 4.5, and 4.6 are conducted for a train with a length of 120 railcars, which is a typical length. The length of the train, however, affects the overall power production for a simulation. Train length determines how many power harvesters are producing power at any point in time. The more railcars in a train increases the total distance it covers and thus also increases the number of devices producing power at any point in time. The converse is true for a train with fewer railcars. However, with a shorter train the lights remain on for a shorter period since it takes less time for the entire train to pass the crossing which also affects the overall charge of the battery.

With the same setup as for Figure 4.6 (a first generation device with a gear ratio of 1:50 for a train traveling at 55 mph) the minimum number of railcars needed to fully recharge the battery with only four devices being used was 40, with the result shown in Figure 4.7. Any fewer than this and the battery will not be fully charged at the completion of the simulation.
Figure 4.7 - Power production and battery life of spread out, first generation devices, for a loaded train traveling at 55 mph with a length of 40 railcars.
Chapter 5. CONTROL SYSTEM

For remote deployment and on-track testing an electrical control system was designed. This system consisted of a charge regulator, a rechargeable battery, and a Rabbit single board computer.

5.1 Electrical System

In the future, to deploy the power harvesters out in the field a reliable electrical system will be needed. This electrical control system needs to be able to handle the estimated average power production along with the significant power spikes that are predicted. Because the power harvesting devices produce power in bursts rather than a consistent flow of power and assuming the possibility that all the devices could have their peak power production at once the maximum amperage that could possibly be produced must be taken into consideration.

The maximum amperage that can be produced by the generators for either design in short bursts is 8 amps. Considering three devices all hitting 8 amps simultaneously a total of 24 amps would be experienced by the system. There exists the possibility of exceeding 24 amps with four or more devices being used; however, due to available electrical equipment and the high probability of this not happening it was decided to design the electrical control system capable of handling power spikes up to 25 amps. To prevent any damage if more than 25 amps was produced a 25 amp fuse is incorporated to protect the system.

The overall electrical setup consists of having the power harvesters, however many that might be needed to power the crossing lights, connected in parallel with each
other. The parallel connection consists of connecting all the positive terminals of the generators onto the same wire while all the ground terminals are all connected on a separate wire. By connecting the generators in parallel the amperage of each generator can be summed up while keeping the voltage at each generator independent of each other. The voltage at each generator is determined by the total resistance in the circuit that the current must flow through. This includes the extra resists providing by the wire the farther away the device is from the crossing.

Each generator will be wired with its own blocking diode between the generator’s positive terminal and the main positive wire that all the generators are connected to. A diode is an electronic component that allows electrical current to flow in one direction and prevents it from flowing in the reverse direction. By using diodes, they will prevent the flow of electric current back into the generator when the generator is not producing any power. If electrical current did flow back into the generator it would begin to act like a motor and attempt to rotate the shaft of the power harvesting device. This would have damaging effects on the generator and possibly the power harvesting device as well. The diodes used would be able to handle up to 8 amps in the forward direction which corresponds to the maximum amperage that a single generator could produce. The connection of the generators and diodes is shown in Figure 5.1
Figure 5.1 – Power harvesting devices connected in parallel with blocking diodes.

To wire the generators’ positive terminal to the diodes a minimum of 12 gauge wire will be used to handle the maximum of 8 amps that may flow through it. Each diode will then be connected to the main positive wire that connects all the devices back to the electrical control system located at the railroad crossing. This main wire will need to be a minimum of 8 gauge wire to handle the maximum of 25 amps. While using larger gauge wire (values less than 8) would reduce the power losses due to resistance in the wire, costs can become an issue.

Also, as described earlier in section 3.4.1, the maximum amperage is experienced in the section of wire that runs from the railroad crossing to the nearest device. The current through any section of wire connecting two devices decreases as the distance from the crossing, and thus the number of devices between said section and the crossing increases. Because of this, smaller gauge wire could be used for the different sections farther away from the crossing. However for simplicity and the fact that wire resistance would increase with smaller gauge wire, 8 gauge wire is used for the entire length.
5.2 Charge Controller

The main positive wire that connects all the power harvesters will be connected to a charge controller located at the railroad crossing. With the large spikes in power production that are experienced by the system, a charge controller is needed in order to efficiently charge the battery without overcharging and causing damage to it. A charge controller controls the rate at which electric current flows into electric batteries [22].

The charge controller chosen was manufactured for smaller alternative energy purposes such as solar panels or wind turbines. Because smaller wind turbines typically use PMDC generators and power production can be very sporadic, somewhat similar to our railroad application, this device is ideal to use. This charge controller is a type of switching regulator which switches between on/off, with ‘on’ corresponding to fully charging the battery, while ‘off’ corresponds to zero charge being applied to the battery.

A switching regulator operates by monitoring the rechargeable battery’s voltage and determining when it should charge the battery. Once the voltage of the battery dips below a certain threshold the controller turns on and uses all available power to charge the battery. Once the voltage of the battery reaches an upper threshold the controller turns off and waits until the voltage drops again.

A linear regulator on the other hand is continuously charging the battery to hold it at a certain voltage. If the voltage begins to rise, the linear regulator compensates by dissipating power in the form of heat. This heat loss makes the linear regulator less efficient than the switching regulator. With the switching regulator charge efficiency can reach upwards of 98% [22].
This charge controller is capable of handling up to 25 amps and can begin charging with as little as 0.005 amps. Power consumption by the charge controller is very minimal with it requiring 0.002 amps in standby mode and 0.007 amps while operating. Additional operating characteristics, operating manual, and wiring configuration of the power harvesters and battery connected to the charge regulator are shown in Appendix C.

5.3 Automation

To control the duty cycle of the warning lights and to turn the Rabbit single board computer on and off a set of electromagnetic relays were utilized. An electromagnetic relay is a type of electrically operated switch that directs the flow of current between two different circuit paths. It typically operates by creating a magnetic field capable of moving a metal armature between two contacts, with each contact being a different path in the circuit. The armature first begins by touching one of the contacts and current is allowed to flow through the relay along this path. A magnetic field is then created by passing a separate current through a coil of wiring surrounding an iron core. This magnetic field forces the armature to move and begin touching the second contact. This then directs the incoming current down a different path through the relay. Once the current is removed from the coils the magnetic field dissipates and a spring pulls the armature back to its original position touching the first contact. The current and voltage that the relay supports, and the current and voltage required to create the magnetic field, vary depending on the electromagnetic relay.
The relays are broken up into two different groups dependent on their function. The first set of relays controls the current to the Rabbit single board computer while the second group controls current to the warning lights.

To minimize power consumption it was desired that the single board computer only be on when there was actually something for it to do, which is when a train passes by the railroad crossing. If this wasn’t the case, power would be continuously drained from the rechargeable battery while the single board computer was in standby mode waiting for a train. With a set of three electromagnetic relays, the single board computer is capable of turning on and executing its program once any kind of current begins to flow through the electrical system. The wiring configuration to accomplish this is shown using Figure 5.2

![Figure 5.2 – Wiring configuration for powering the single board computer](image-url)
where:

\( RL \) are electromagnetic relays,

\( D \) are diodes,

\( Ig \) is the current from the generators,

\( Ir \) is the current from charge regulator,

\( SBC \) is the current to the single board computer,

\( D \ O I \) is current from the single board computer digital pin 1.

As the generators first start producing power, current begins to flow in at \( Ig \), then through the diode, \( D1 \), and into the coils of the electromagnetic relay, \( RL2 \). The diode \( D2 \) prevents the current from flowing into any other part of the circuit except for the coils of \( RL2 \). The incoming current causes \( RL2 \) to switch contacts allowing current from the battery to flow to \( SBC+ \). As the current flows to \( SBC+ \) part of it branches off and flows through the coils of both the electromagnetic relays of \( RL1 \) and \( RL2 \). The diode \( D1 \) prevents the current from flowing back towards the generators. As the current flows through the coils of \( RL1 \) the contact in the relay switches, cutting off the current flowing from the generators to \( RL2 \). At the same time the current from the generators is cut off, the current from the battery is flowing through the coils of \( RL2 \) forcing the relay to remain switched. This creates a loop in which the relay will remain switched until the current from the battery discontinues.

With continuous power flowing from the battery the single board computer turns on and begins to execute its programming. Upon completion of its programming the
The single board computer sends a digital signal, D I/O 1, through the coils of RL3 causing the relay to switch contacts and thus cutting off the current flowing from the battery. With the current cut off from the battery, relays RL1 and RL2 switch back to their original contacts, the single board computer turns off, stopping the digital signal D I/O 1, and thus the relay RL3 also switches back to its original contact. The electrical circuit returns back to original wiring and is ready for the next time the generators begin producing power.

The second group of electromagnetic relays is used to control the duty cycle of the warning lights by directing the flow of the current. The single board computer sends out timed digital pulses that control the rate at which the relays switch back and forth. The wiring configuration to control the warning lights is shown in Figure 5.3

![Figure 5.3 – Wiring configuration for control of the warning lights.](image-url)
where:

F is a fuse,

R is the load applied by a set of lights.

Once a train approaches and the generators begin to produce power, current begins to flow from the battery into the circuit at SBC +, illustrated in Figure 5.3. This current splits into two with part of the current being consumed by the Rabbit single board computer while the other part of the current is consumed by the warning lights. The current being consumed by the single board computer must first pass a fuse which protects the single board computer from power spikes. The current that flows to the lights must first pass through the electromagnetic relay RL4. This relay determines which half of the lights receives the current and thus is illuminated. The relay RL4 switches contacts every second by receiving a digital signal, D I/O2, from the single board computer, through its coils. As the current is switched back and forth between the lights R1 and R2, the electromagnetic relays RL5 and RL6 control the amount of dead time there will be between flashes. The resting positions of the relays are such that when RL4 directs the current to them the current goes straight through the relay and into the lights.

Depending on the amount of dead time the single board computer determines there should be, the relays will receive a digital signal to switch contacts and stop the current flowing to the lights. If the dead time was calculated to be zero, then no digital signal is sent to the coils of RL5 or RL6 and the lights remain on the entire second that RL4 is switched towards them. If the dead time was calculated to be 0.3 seconds, then the relays RL5 or RL6 would receive a digital signal to cut the power to the lights 0.7
seconds after RL4 switched towards them. After the 1 second that RL4 is switched towards them the single board computer stops sending the digital signal RL5 or RL6 and they return back to their original positions.

The entire circuit used, both Figures 5.2 and 5.3, for the automation of the warning lights, was connected and powered by a 12 volt DC battery. All the current required by the circuit is drawn from this battery. The electromagnetic relays used for this circuit are rated up to 6 amps at 30 volts through the contacts, while the coils have a pickup value of 210 milliwatts at a nominal voltage of 5 volts and a maximum continuous voltage of 12.6 volts. At 12 volts these coils require 0.0175 amps. These coils are all connected to the battery’s ground. The diodes used were also rated up to 6 amps. The Rabbit single board computer has a maximum amperage of 0.375 amps at 12 volts so a fast acting fuse was used rated at 0.35 amps to prevent damage. The lights are connected in parallel such that if one light malfunctions the rest of the lights can remain on. By connecting them in parallel the required amperage is more than if they were in series. The maximum amperage required for one set of the lights is 3.33 amps at 12 volts. The total amperage drawn from the battery is the summation of RL1, RL2, the single board computer and one set of the lights for a total of 3.71 amps which is well within the range of the relays and the diodes. The single board computer is capable of having its digital signals pulled from 5 volts to 12 volts to provide the necessary power for each of the remaining relays. Data sheets for each of the electrical components are shown in Appendix C.
5.4 Single Board Computer

For overall monitoring of the power harvesters and control of the railroad crossing lights a Rabbit single board computer is implemented. The components of the single board computer consist of a Rabbit microprocessor, 10 analog I/Os and 40 digital I/Os, both an analog to digital converter and a digital to analog converter, an SD mount. All these components make the single board computer optimal for the electrical control system and for remote deployment at railroad crossings, since it can manage all the necessary I/O and perform remote data logging. The entire Rabbit single board computer and dimensions are shown in Figure 5.3.

![Rabbit single board computer with dimensions.](image)

The single board computer is programmed in a Dynamic C environment, which is very similar to programming with C++. The code is first written in this environment on a
PC and is then transferred to the single board computer via a USB cable and stored in its memory. Once disconnected from the PC the program will begin to run every time the single board computer is powered up.

A unique and powerful enhancement of Dynamic C over C++ is its simplicity to allow cooperative parallel processes to be simulated in a single program by Costatements. Because the single board computer is only equipped with a single processor, which can only execute a single instruction at a time, Dynamic C cannot achieve true parallel processing, where multiple instructions can be executed at the exact same time. However, it is capable of appearing to execute multiple instructions at once in a simpler manner than C++.

In a multitasking environment, more than one task, a sequence of operations, can appear to execute in parallel. Typically at some point within task the program is told to stop and wait until a certain condition is satisfied before it continues along with the remaining executions. When this occurs, the program exits the current task and goes into another task and begins to execute operations within it. The program continues to execute operations in this second task until the condition in the first task is met or the second task is told to stop. In this manner each task can do some of its executions while the other tasks are waiting which gives the appearance that they are executing in parallel.

To accomplish this type of multitasking coding can become very lengthy, complicated, and difficult for the programmer to achieve. In most programming environments keeping track of which execution is next for the different tasks when the program is switching between them is very difficult if not impossible. With Dynamic C
and the use of Costatements it is achieved without any extra programming. A comparison of sample code for C++ and Dynamic C achieving the same multitasking is shown in Appendix B.

5.4.1 Programming

The programming for the single board computer has 5 main tasks that are all multitasked using Costatements. The order in which the tasks are listed is the order of their priority. The first task will execute until it has to wait, then the second task will begin to execute until it has to wait or if the first task is done waiting. If both are waiting the third task will begin to execute until it has to wait, etc.

To achieve as close to real-time performance as possible the first task is a set of commands that calculate time as the program runs. The next task measures the voltage drop across a resister via analog pins. The third task executed controls the duty cycle of the crossing lights via digital outputs. After that, the fourth task executes which saves the measured voltage and the associated time at which it was taken into a continuously growing temporary file. The final task saves the completed temporary file to the SD card and ends the program. As the programs executes downward through the tasks keeping track of time becomes less and less important.

The single board computer turns on and begins to execute the saved program each time current begins to flow through the electrical system. The board will then remain on until the program reaches completion. The program starts by initializing various components of the single board computer, loading specific command libraries that are
used throughout the code, and creating sub functions that are used by the A/D converter to measure voltage.

After these initializations the program creates and defines all the variables that will be used, and sets their types. Initial values are given to some of these variables, while others will be determined later in the program.

Next the program initiates and defines the mode of operation for both the digital and analog pins that will be used. The digital pins are configured as outputs with an initial state of off. The analog pins are configured to be inputs with a mode of operation being single-ended polar. The single-ended uni-polar voltage mode measures the difference in voltage between two analog pins with respect to the same ground. With the uni-polar mode only positive voltages can be measured with a range of 0-20 volts.

The program then proceeds to mount the SD card making it available to the single board computer for read and write operations. In order for the single board computer to interact with the SD card, a FAT 32 file system has to be used. The FAT 32 file system is a typical file system used on many PCs for memory and must be used if the data from the single board computer is to be transferred to a PC. The commands to operate a FAT 32 system are one of the libraries initialized at the very beginning of the program.

To write data to the SD card the program must first check to see if the SD card is formatted properly and that there is available space. It then creates a new temporary file for the data to be stored in until the program is done collecting data. Chunks of data are continuously written to this temporary file while the program is running and upon completion it all gets transferred to the SD card.
Because data from multiple runs need to be recorded, this file must have a unique name each time the device is turned on. To accomplish this, a simple naming convention is used where the files are labeled ‘voltage1’, ‘voltage2’, ‘voltage3’, and so on. Each time the single board computer is turned on and the program runs it checks to see if the file ‘voltage1’ already exists on the SD card. If it doesn’t exist the program creates it; if it does exists the program increments the file name to ‘voltage2’ and checks if that exists. This continues iteratively until the program comes to the lowest available file name at which point it creates the new file.

With the SD card mounted and a temporary file created the program enters the main loop containing all the Costatements. The program will continuously execute through these Costatements until the program reaches completion.

As discussed earlier the first Costatement to execute is the one that keeps track of time. As the program enters this task it first must wait until a voltage is being recorded in the second Costatement before it continues. By doing this the time stamp will match the voltage history most accurately. After a voltage is measured the Costatement is allowed to continue. The next command is to wait for 0.1 seconds before it continues. Once this waiting period is complete, a counter variable, representing one tenth of a second, is incremented by 1 and the task begins again. Once this first variable reaches a value of 10, a second variable, representing 1 second, is incremented by 1, and again the task is restarted. Once this second variable reaches a value of 60, a third variable, representing 1 minute, is incremented by 1 and the task is again restarted. This process is continuously repeated. Because this is the first Costatement in the program all execution within it takes
precedent over the rest of the Costatements. The only time the other Costatements execute is when this task is waiting during each 0.1 second increment.

While the first task is waiting for 0.1 seconds to elapse, the second Costatement is allowed to execute. This Costatement contains the execution commands for measuring the voltage difference between two analog pins. This first task is to measure the raw data reading from the analog inputs. If the raw data exceeds the maximum range or if there is any kind of error the program will immediately stop. If the raw data is within the normal range the value is then put into a running average buffer containing the last 100 values. The program then checks to see if the calibration values for the inputs have been properly loaded during the initialization process. Once it is determined that the calibrations values have been properly initialized the program uses this data and converts the raw data into the proper voltage. Also every time a raw data value is converted to a voltage a counter is incremented. This counter keeps track of how many voltage values have been taken and is used in determining when the program should finish. If this counter reaches a certain value it is assumed that the train has completely passed and the program no longer needs to run. After converting the voltage, if this value is above 0.1 volts it triggers the timing task to start up again, the save data Costatement to begin, the light control Costatement to begin, and finally to reset the counter back to zero.

The next Costatement to execute takes the measured voltage and saves it into the temporary file. As mentioned previously once a voltage is detected above 0.1 volts this Costatement is now allowed to execute. It begins by creating a string containing the voltage and the time stamp. While this Costatement doesn’t execute at the same rate as the time stamp and the voltage reading tasks do, whenever it does execute it will have the
correct timestamp for a given voltage. With the string created containing the voltage, the task continues to execute by inserting the string into the temporary file. This temporary file continues to grow as new strings are appended to it.

With the voltage measured and recorded the next Costatement regulates the flow of current to the lights by controlling the electromagnetic relays. As discussed previously these relays receive digital pulses from the single board computer to dictate which way they are switched. This Costatement first must wait for the trigger from the voltage reading before it can begin to execute. Once a voltage is measured the task can begin its operations. It begins by waiting 1 second, which is the period at which the lights are flashing, after which it triggers the first of the digital pins to turn on. This pin controls the relay that directs power between the two sets of lights. After this the tasks calculates an appropriate dead time between the lights by dividing the average power measured by the full power required by the lights. Next the task enters a switching operation. This switch operation has two separate cases containing different pieces of code that the task switches between each time the Costatement iterates. Each case contains executions for separate digital pins that control the dead time for each set of lights. As discussed earlier, the resting state of each relay controlling the dead time of the lights is such that once there is available power into the relay the lights will turn on. As the Costatement enters the first case power is already flowing to the set of lights it controls. This case then waits for a time equal to 1 second minus the calculated dead time.

The final Costatement to execute contains the task of determining when the program should shut off, saving the temporary file to the permanent file on the SD card, and sending a digital signal to a relay that will cut the power off to the system. This
Costatement requires no trigger to begin and is always running. It begins by checking the counter to see if it has reached its threshold or not. If it has, then it is assumed the train has passed and the program should begin to shut down. The program begins by discontinuing the Costatment controlling the duty cycle of the lights, this frees up processing power which is used then used for saving the data to the SD card. Next the Costatement must wait until all the data have been saved to the temporary file. Once ready it moves the temporary file onto the SD card and upon completion of the save unmounts the SD card. Finally a digital signal is sent to a relay controlling the power to the entire circuit and all power is cut off. With no power the program is completed and the single board computer shuts off and waits to be turned back on.
Chapter 6. LAB AND FIELD TESTS

To validate simulation results and to test the electrical control system, lab tests were conducted. In-lab testing provided a safer and much more controlled environment than testing on the actual track. While in-lab testing conditions could not necessarily replicate the oscillatory motion of the actual rail deflecting, it was still suitable for characterizing the system. This testing provided information on the relationship of power output to speed and load and helped assist in the theoretical modeling of the simulation. Previous testing had already been accomplished for the single one-way clutch [13] so only testing of the double one-way clutch was conducted.

6.1 Setup

The power harvester was tested in the laboratory at various speeds and the current and voltages were measured at different positions in the circuit. Two multimeters and the Rabbit board were used to take measurements. One multimeter measured the current flowing through the circuit, the Rabbit single board computer measured the volatage drop across a resistor, and the final multimeter measured the total voltage drop across the terminals of the generator. A hand-held tachometer was also used to measure the rotational speed of the generator shaft.

To achieve the desired input speed a motor was attached to the end of the power harvester device. The motor had a maximum of 83 rpm with 155 in-lbs of torque. The motor provided a fairly constant rotation to the generator. While this does not duplicate the exact motion that the power harvesting device will experience from railroad track deflection it provided information on how the overall circuitry would operate.
The output terminals of the generator were attached to the overall circuit, with the positive terminal of the generator connected in series to the positive cable of a multimeter, and the negative terminal of the generator connected to the battery’s negative terminal. The multimeter was set to measure the total current flowing in the circuit. The negative cable of the multimeter was attached to a voltage divider containing various resistors. Two analog pins of the Rabbit board were connected across one of the resistors to measure the overall voltage drop. The voltage divider was then connected to the charge regulator. The charge regulator was then connected to the battery. The overall setup is shown in Figure 6.1 with an equivalent circuit diagram shown in Figure 6.2.

Figure 6.1- Setup of laboratory testing with motor, voltage divider, single board computer charge controller, and battery all connected.
Figure 6.2 – Equivalent circuit for laboratory testing where $V=$voltmeter, $V_s =$ generator, $A =$ ammeter, $I =$ current, $R_1 =$ series of resistors, $R_2 =$ resistor across which voltage drop is measured, $SBC =$ single board computer, $CC =$ charge controller, $V_b =$ battery.

6.2 Data Collection

Data were collected for several different generator speeds while the load applied to the generator was kept constant throughout testing. Measurements were taken for the rotational speed, the current, and the voltage across a resistor and across the generator terminals. As previously mentioned, to measure current a multimeter was connected in series with the other circuit elements and a multimeter was connected across the terminals of the generator to measure the total voltage. To measure the rotational speed a tachometer was used. There were some fluctuations ($\pm 20$ rpm) with the tachometer readings so an approximate value was used for each range.

To measure the voltage drop across the resistors the Rabbit single board computer was used. It sampled the analog voltage signal at a rate of 100 Hz. The data collected were saved to an SD card and uploaded to a PC to be analyzed. Using single ended uni-
polar as the mode of operation for the analog pins the single board computer was capable of measuring the voltage for a range of 0 – 20 volts. This voltage range was much larger than what the predicted values would be so the signal gain was adjusted within the programming of the single board computer such that the voltage range became 0 – 2 volts resulting in a more precise reading. From Ohm’s law the current flowing through the circuit can be calculated from knowing the resistance across which the voltage is measured. The current calculated can then be compared to the readings on the multimeter.

For testing, the single board computer was run continuously collecting data as the rotational speed was periodically increased. With each change in speed, measurements were recorded for both multimeters and the tachometer.

Lab testing initially began with a voltage divider consisting of a 20 Ω and a 1 Ω resistor connected in series, represented by R₁ and R₂ respectively in Figure 6.2. This voltage divider was needed such that voltage drop across the 1 Ω resistor was within the acceptable range of the single board computer. The overall load applied to the generator was 21 Ω. This overall load did not match the internal impedance of the generator of 7.7 Ω, but with the available resistors and attempting to keep the voltage drop across R₂ small it was deemed sufficient. If the internal resistance of the load matched that of the internal impedance of the generator maximum power production would have been achieved.

However, due to limitations of the motor, data collection with the 21 Ω load was limited to a very small range. As discussed in section 4.1, depending on the applied load the generator must reach a certain voltage before it will start to produce power. With an
applied load of 21 $\Omega$ and with the battery connected, the voltage required to produce any current was 22.47 volts which corresponds to 509 rpm for the generator shaft.

The maximum torque that the motor could apply to the generator was 155 in-lbs. Because the rotational input from the motor first goes through a 1:100 gearbox before reaching the generator the output speed is increased but the applied torque is reduced. The maximum torque that the motor could apply to the generator with the inclusion of the gearbox was 1.55 in-lbs, or 0.164 Nm.

While testing with the 21 $\Omega$ applied load, the maximum speed achieved by the generator shaft was approximately 620 rpm or 64.9 radians per second. At this speed the generator produced a measured current of 0.44 amps at 24.1 volts with the total power produced being 10.6 watts. Dividing the power produced by the rate at which the shaft rotates, yields that the torque required to rotate the shaft was 0.163 Nm which is at the maximum of what the motor can provide. Because of the limitations of the motor the load applied to the generator was increased to provide a larger range of operating speeds.

The total load applied to the generator was increased to 242 $\Omega$ with the single board computer measuring the voltage drop across a 10 $\Omega$ resistor. By increasing the load the range of output speeds became 565 to 1220 rpm.

6.3 Results

The single board computer recorded continuously for four different ranges of generator speeds as shown in Figure 6.3. The speeds, as measured by the tachometer, were 565, 714, 790, and 952 revolutions per minute. As the speed increased so did the voltage drop across the resistor as expected.
Figure 6.3 – Voltage drop across a 10-Ω resistor for various generator speeds as measured by the single board computer.

For each region corresponding to a different speed the voltage was averaged. As the speed increased, the voltages also increased, taking values of 0.53, 0.83, 0.99, and 1.26 volts.

Using Ohm’s law, current flowing through the resistor and thus the circuit can be calculated. The measured resistance of the resistor was not perfectly 10 Ω but rather 10.2 Ω. Dividing the measured voltage drops by the resistance results in respective currents of 0.052, 0.082, 0.098, and 0.124 amps.

The power produced can now be calculated by multiplying the square of the current by the total resistance of the circuit. The total resistance is equal to the resistance
of the load, 242 Ω, plus the resistance due the battery, which is calculated by dividing the voltage of the battery by the current. The corresponding powers for the different speeds are 1.32, 2.64, 3.53, 5.29 watts.

The values for current and power derived from the data collected by the single board computer can be compared to the data measured by both multimeters as shown in Figure 6.4. To calculate the power from the multimeter measurements, the total voltage is multiplied by the current.

![Lab Results](image)

**Figure 6.4 – Lab testing results for both the single board computer and multimeters over various generator speeds.**

where:

- $MM V_{tot}$ is the total voltage across the generator as measured by the multimeter,

- $MM Amps$ is the current through the circuit as measured by the multimeter,

- $SBC Amps$ is the current calculated from the single board computer measurement,

- $MM Power$ is the power calculated from the multimeter results,
**SBC Power** is the power calculated from the single board computer measurements.

The single board computer only measured the voltages for four different speed levels while the multimeters recorded an additional two speed levels. The results from the two measurement methods are comparable, with only 4% to 10% difference. Inconsistency could result from small variations in multimeter readings. Even at these values the accuracy of the single board computer measuring voltages is still valid.

The simulation results were then compared to the lab results to validate their accuracy. But because the simulation uses the oscillatory motion of the track, direct comparison between the lab tests, where the speeds are fairly constant, to the simulation, where speeds vary, are not possible. Instead the equations used by the simulation to calculate power were tested. From the equations used in the simulation and substituting in appropriate values for load and a constant value for the input speed a comparison can be made as shown in Figure 6.5.

![Figure 6.5- Comparison between lab results and simulation calculations.](image)
The results for both the measured and calculated values of volts, current, and power were plotted and are shown in Figure 6.6.

Figure 6.6- Plotted comparison between lab results and simulation calculations.

Analyzing the results for voltage, current, and power that were calculated from the simulation equations, there is maximum difference in values of about 14% and ranging as low as a 1.5% difference. The voltages for both the lab results and simulation equations matched up closely with about a 3% difference while the values for current and power had about 10% difference between the lab results and simulation results. Considering the variations associated with the tachometer measurements, results from the equations used in the simulation are comparable to the lab results.

To more accurately validate the simulation, on-track testing is desirable. On-track testing of the double one-way clutch has not been completed but previous field testing
had already been completed for the single one-way clutch design and the results [13] were used for comparison with the simulation. Data was collected using a load resistance of 37.3 Ω with the voltage drop measured across a 1-Ω resistor. From the data, it was concluded that for a loaded train traveling at 11.5 mph an overall average power of 0.22 watts was produced while for an unloaded train traveling at 13.5 mph an overall average power of 0.01 watts was produced.

Using the correct inputs for speed, 11.5 mph, load, 280000 lbs, gear ratio, 1:50, and design, downward motion, the simulation calculated an average generator shaft rpm of 394.8 with an average power production of 0.34 watts. Changing the inputs for train speed and load to 13.5 mph and 58000 lbs respectively, the simulation calculated an average generator shaft speed of 14.5 rpm with an average power production of 0.03 watts.

For both simulations the average power production was higher than the field test results. A possible contribution to this is the overall efficiency of the device. For this prototype it was previously calculated that because of efficiencies of the generator (85%), gearhead (85%), and backlash (90%), the total equivalent efficiency of the power harvester was 65% [13]. When this efficiency is applied to the simulation results the average power for a loaded train traveling at 11.5 mph becomes 0.221 watts while for an unloaded train traveling at 13.5 mph the average power becomes 0.019 watts. These results are very comparable to the field test and validate the simulation accuracy.

Using these results to validate the simulation, predicting power production for higher speeds can be done. With the single one-way clutch design with applied
efficiencies, for a train traveling at 55 mph the average power production per device was calculated to be 7.41 watts for a loaded train and 0.318 watts for an unloaded train. With the double one-way clutch design, for a train traveling at 55 mph the average power production per device was calculated as 215.2 watts for a loaded train and 9.23 watts for an unloaded train.

From these calculations the minimum number of devices needed to power the railroad crossing for a loaded train traveling at 55 mph can be determined. With the single one-way clutch design, a minimum of two devices are needed to power the lights while a total of four are needed to power the lights and maintain the charge of the battery as shown in Figure 6.6. With the double one-way clutch design, a single device is sufficient at powering the lights while maintaining charge in the battery as shown in Figure 6.7. These results give promise to the functionality of the power harvesting device in the field.
Figure 6.7 – Four first generation devices, spread out, for a loaded train traveling at 55 mph.

Figure 6.8 – Single second generation device, centered, for a loaded train traveling at 55 mph.
Chapter 7. SUMMARY AND CONCLUSIONS

Power harvesting in the form of wind and solar power has already proven itself valuable in providing railroads with a reliable source of power for wireless track sensor networks and other railroad applications. However, the need for improved railroad safety at unmarked grade crossings creates a demand for more power generation. Improvements to unmarked grade crossings would consist of installing a warning light system. These warning light systems would consist of a set of eight high-efficiency light emitting diode lamps with each light requiring approximately 8 watts at 9 VDC. Overall the system would have two sets of alternating lamps such that a maximum of 40 watts of power is provided at any given time.

While wireless sensors require a small amount of power, on the order of tens of milliwatts, and do not necessarily need to be active at all times, the warning light systems must be active any time a train approaches, with failure to do so having significant consequences. With wind and solar power dependent on the weather conditions they cannot guarantee that there will always be sufficient power for the lights. This motivated the Federal Railroad Administration to seek out other power harvesting solutions that could reliably provide the necessary power to the warning lights. The conceptual goal was to develop a power harvesting setup capable of reliably powering the warning light system, being low-maintenance such that it could function without human intervention for several months and beyond, and without interfering with railway traffic.

To accomplish these goals two power harvesting devices were designed to generate power on the order of 10 watts. The driving force behind both power harvesting
devices was the vertical railroad track deflection from passing railcars. Both power harvesters harnessed this vertical track deflection to translate it into rotational motion to drive a PMDC generator. The first generation design only harnessed the downward motion of the track while the second generation design harnessed both the downward and upward motions.

With the associated high forces and unique oscillatory motion of the rail all being translated into the power harvester, controlled lab tests on the different prototypes become difficult to accurately conduct. Along with this, on-site tests are also difficult due to uncontrollable variables in train weight, speed, and track conditions. Because of these problems a simulation was needed to help validate the results from the different prototypes and to also determine optimal characteristics such that improvements could be made to the design.

The simulation was used to perform analysis on both device designs by determining the maximum and average rotational speeds experienced by the device, the expected power production of the device, the minimum number of devices required to power the crossing lights, and the most favorable positioning of the devices between the options of either being spread out along the track or centered at the crossing.

The maximum power output calculated for the first design, consisting of a single one-way clutch, for a loaded train traveling at 55 mph was 7.41 watts. With this design it was determined that there would need to be a minimum of four power harvesting devices spread out evenly along the track to supply the necessary power to the warning lights.
For the second design, consisting of two one-way clutches, the maximum power output calculated for a loaded train traveling at 55 mph was 215.2 watts, with only a single device needed positioned at the crossing to supply the necessary power to the warning lights.

Along with developing the simulation engine, an electrical control system was also designed. The motivation for this electrical system was such that the power generated by the power harvesters could be used to power the lights and any excessive power could be stored in a rechargeable battery. This battery acts like a failsafe providing power to the warning lights whenever there is not enough being produced by the power harvesters. This electrical control regulates power to the rechargeable battery, protecting it from any power spikes that may occur increasing its lifetime. This electrical system consisted of a charge regulator and a rechargeable battery. The entire electrical control system was designed such that it could handle power spikes from the power harvesters up to a maximum of 25 amps.

The electrical control system was also capable of controlling the duty cycle at which the warning lights flashed along with measuring and recording power production. Controlling the duty cycle of the lights minimized the power requirements while measuring and recording power production allowed for remote deployment and data collection. Both tasks were accomplished by utilizing a Rabbit single board computer that was integrated into the electrical system. The Rabbit single board computer also controlled several electromagnetic relays that directed the flow of current to the two sets of warning lamps along with the flow of current to the Rabbit single board computer itself.
The results from lab and field tests were used to validate the simulation results and to confirm the functionality of the electrical control system. The lab tests conducted consisted of attaching a motor to the second generation prototype. The motor provided a consistent torque and rotational speed to the power harvesting device. It was noted that this setup did not replicate the oscillatory motion of the rail deflecting but it was satisfactory for validation purposes.

Various speeds of the generator shaft were tested with the current, the voltage across the generator terminals, and the voltage drop across a resistor all being recorded. Multimeters were used to measure the current and the voltage across the generator while the Rabbit single board computer was used to measure the voltage drop across the resistor. The data recorded by the Rabbit single board computer were used to calculate power production for each range of generator speed. These results were then compared to the power production calculated from the measurements of the multimeters. The difference between the two calculations ranged from 4% to 10% with some of the inconsistency being contributed to fluctuations in the multimeter readings. These results validate the Rabbit board capability of measuring and recording voltages.

Results from lab tests were also used to validate the simulation’s accuracy. While a straight comparison between lab results and simulation results could not be done due to the steady speed of the lab test as opposed to the oscillating speed used in the simulation the equations used to calculated power in the simulation could be validated. For the same electrical load and speed as the lab tests, the simulation equations calculated the average power production. The results from simulation equations were compared to the results calculated from the multimeter. The difference between the two calculations for power
ranged from 8% to 16%. Again these inconsistencies could be the result of possible fluctuations in both the multimeter and tachometer readings. Taking the possibility of these inconsistencies into consideration the simulation equation results are fairly comparable to lab test results.

To further validate the simulation previous field test data were used for comparison. From the field tests it was concluded that for the first generation power harvester under a loaded train traveling at 11.5 mph the average power production was 0.22 watts while for an unloaded train traveling at 13.5 mph the average power production was 0.01 watts. Using the same variables for train weight, speed, gear ratios, and applied load to the generator the simulation calculated an average power production of 0.34 watts for the loaded train and 0.03 watts for the unloaded train. After applying a previously calculated generator system efficiency of 65% these values become 0.221 watts for the loaded train and 0.019 watts for the unloaded train. These results validated the simulation’s accuracy, and predicting power production through extrapolation to higher speeds was done.

For a loaded train traveling at 55 mph the average power production of the first generation prototype was 7.41 watts while the second generation prototype had an average power production of 215.2 watts. The most favorable number and position of both devices were then determined. For the first generation device a total of four power harvesters spread out evenly along the track were needed to provide power to the warning lights while also keeping the battery close to fully charged. For the second generation prototype a single device positioned at the crossing was able to accomplish this.
For an unloaded train traveling at 55 mph the average power production of the first generation prototype was 0.318 watts while the second generation had an average of 9.23 watts. For this case the first generation prototype as it was originally designed was not capable of producing enough power for the warning lights while the second generation prototype was able to provide the necessary power with a total of four devices spread out along the track. If the gear ratio of the first generation prototype was increased from 1:50 to 1:100 the first generation device was still not capable of producing the necessary power.

From these results it is determined that the second generation prototype is capable of meeting the power requirements for both loaded and unloaded trains traveling at 55 mph requiring a minimal amount of devices. It is also determined that for an unloaded train traveling at 10 mph neither prototype would be capable of producing the necessary power for the lights and thus large amounts of power must be drawn from the battery.

Examining the various results of the simulation it can be determined that certain combinations of power harvesting device variables are ideal for certain conditions. If the environment, referring to train speeds and weight, in which the power harvester is deployed is relatively constant, a power harvesting device can be tailored to obtain maximum power. In most cases the preferred combination of variables would be the second generation prototype as originally designed with the 1:100 gear ratio. However, in areas where train traffic is relatively slow, changing from the generator used in the second generation device to the generator used in the first generation prototype would be beneficial because it can generate power at a lower input speed. As train speed increases
and the generator shaft speed increases, the second generation generator becomes the better option.

Also by examining the various simulation results it can be determined that certain train conditions can drastically affect the overall effectiveness of both power harvesting prototypes. In areas where there are significant variations in the train speed, weight and length, worst case scenarios must be planned for. These worst case scenarios would come from trains traveling approximately 15 mph or slower, unloaded trains, short trains, or any combination of the three. Improvements made to the power harvester design that could account for these worst case scenarios would greatly improve the overall effectiveness and safety of the system.

Future work will include making improvements and additions to the simulation engine. A third generation prototype is currently being developed that will be included into the simulation such that power production for the device can be predicted for various train speeds and weights. Another inclusion to the simulation is developing a third option for determining the spacing of the devices. This third option would calculate and test all the possibilities of how the devices can be positioned along the track until the best possible positioning is determined. This third option would greatly increase the time for the simulation to run to completion. The overall efficiency of the simulation will also be examined such that improvements can be made to reduce the amount of runtime. Other future work includes field testing the second generation prototype along with the electrical control system. The results from these field tests will be compared to the simulation results to help further validate their accuracy. These field tests will also
provide insight on the functionality of the control system and help address any problems in its design.
BIBLIOGRAPHY


A.1 M-File Work1

```matlab
function varargout = Work1(varargin)
% WORK1 M-file for Work1.fig
% WORK1, by itself, creates a new WORK1 or raises the existing
% singleton*.
% H = WORK1 returns the handle to a new WORK1 or the handle to
% the existing singleton*.
% WORK1('CALLBACK',hObject,eventData,handles,...) calls the local
% function named CALLBACK in WORK1.M with the given input
% arguments.
% WORK1('Property','Value',...) creates a new WORK1 or raises the
% existing singleton*. Starting from the left, property value
% pairs are
% applied to the GUI before Work1_OpeningFcn gets called. An
% unrecognized property name or invalid value makes property
% application
% stop. All inputs are passed to Work1_OpeningFcn via varargin.
% *See GUI Options on GUIDE's Tools menu. Choose "GUI allows only
% one
% instance to run (singleton)".
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help Work1
% Last Modified by GUIDE v2.5 01-Jul-2010 11:57:36

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
 gui_State = struct('gui_Name',       mfilename, ... 'gui_Singleton', gui_Singleton, ... 'gui_OpeningFcn', @Work1_OpeningFcn, ... 'gui_OutputFcn', @Work1_OutputFcn, ... 'gui_LayoutFcn', [] , ... 'gui_Callback', []);

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

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before Work1 is made visible.
function Work1_OpeningFcn(hObject, eventdata, handles, varargin)
```
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Work1 (see VARARGIN)

% Choose default command line output for Work1
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes Work1 wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = Work1_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

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

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

% Hint: get(hObject,'Value') returns toggle state of AccelCheck
if (get(hObject,'Value') == get(hObject,'Max'))
    % Checkbox is checked-take appropriate action
    set(handles.AccelValue, 'visible', 'on')
else
    % Checkbox is not checked-take appropriate action
    set(handles.AccelValue, 'visible', 'off')
end

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

% Hint: get(hObject,'Value') returns toggle state of PowerCheck
if (get(hObject,'Value') == get(hObject,'Max'))
    % Checkbox is checked-take appropriate action
    set(handles.PowerValue, 'visible', 'on')
else
    % Checkbox is not checked-take appropriate action
    set(handles.PowerValue, 'visible', 'off')
end

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

% Hint: get(hObject,'Value') returns toggle state of LightCheck
if (get(hObject,'Value') == get(hObject,'Max'))
    % Checkbox is checked-take appropriate action
    set(handles.LightValue, 'visible', 'on')
else
    % Checkbox is not checked-take appropriate action
    set(handles.LightValue, 'visible', 'off')
end

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

% Hints: get(hObject,'String') returns contents of SpeedValue as text
% str2double(get(hObject,'String')) returns contents of SpeedValue as a double
user_entry = str2double(get(hObject,'string'));
if isnan(user_entry)
    errordlg('You must enter a numeric value','Bad Input','modal')
    uicontrol(hObject)
    return
end
% --- Executes during object creation, after setting all properties.
function SpeedValue_CreateFcn(hObject, eventdata, handles)
% hObject    handle to SpeedValue (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

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

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

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

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

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

% Hints: get(hObject,'String') returns contents of CarValue as text
% str2double(get(hObject,'String')) returns contents of CarValue as a double
user_entry = str2double(get(hObject,'string'));
if isnan(user_entry)
    errordlg('You must enter a numeric value','Bad Input','modal');
uicontrol(hObject)
function AccelValue_CreateFcn(hObject, eventdata, handles)
  % hObject    handle to AccelValue (see GCBO)
  % eventdata  reserved - to be defined in a future version of MATLAB
  % handles    structure with handles and user data (see GUIDATA)

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

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

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

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

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

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

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

  % Hints: get(hObject,'String') returns contents of AccelValue as text
  %        str2double(get(hObject,'String')) returns contents of
  %        AccelValue as a double
  user_entry = str2double(get(hObject,'string'));
  if isnan(user_entry)
    errordlg('You must enter a numeric value','Bad Input','modal')
    uicontrol(hObject)
    return
  end

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

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

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

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

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

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

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

function PowerValue_Callback(hObject, eventdata, handles)
  % hObject    handle to PowerValue (see GCBO)
  % eventdata  reserved - to be defined in a future version of MATLAB
  % handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of PowerValue as text
%       str2double(get(hObject,'String')) returns contents of
%       PowerValue as a double
user_entry = str2double(get(hObject,'string'));
if isnan(user_entry)
    errordlg('You must enter a numeric value','Bad Input','modal')
    uicontrol(hObject)
    return
end
% Proceed with callback...

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

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

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

% Hints: get(hObject,'String') returns contents of LightValue as text
%       str2double(get(hObject,'String')) returns contents of
%       LightValue as a double
user_entry = str2double(get(hObject,'string'));
if isnan(user_entry)
    errordlg('You must enter a numeric value','Bad Input','modal')
    uicontrol(hObject)
    return
end
% Proceed with callback...

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

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
% --- Executes on selection change in MotionBox.
function MotionBox_Callback(hObject, eventdata, handles)
% hObject    handle to MotionBox (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

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

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

% --- Executes on slider movement.
function slider1_Callback(hObject, eventdata, handles)
% hObject    handle to slider1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
% get(hObject,'Min') and get(hObject,'Max') to determine range
% of slider
sliderVal = get(hObject,'Value');
sliderStatus = num2str(sliderVal);
set(handles.textStatus,'string', sliderStatus)

% --- Executes during object creation, after setting all properties.
function slider1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to slider1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end
function pushbutton1_Callback(hObject, eventdata, handles)
    SpeedValue= str2double(get(handles.SpeedValue,'String'));
    CarValue = str2double(get(handles.CarValue,'String'));
    AccelValue = str2double(get(handles.AccelValue, 'String'));
    PowerValue = str2double(get(handles.PowerValue, 'String'));
    LightValue = str2double(get(handles.LightValue, 'String'));
    WeightValue = str2double(get(handles.textStatus, 'String'));
    DeviceValue = str2double(get(handles.Devices, 'String'));
    valRatios = get(handles.RatioBox,'value');
    valMotion = get(handles.MotionBox, 'value');
    valPosition = get(handles.DevicePos, 'value');

    %%%%%%%%%%%%%%%%%FIGURES%%%%%%%%%%%%%%%%%%%%%%%
    FigDisplace = get (handles.DisplaceFig,'value');
    FigAccel = get (handles.AccelFig,'value');
    FigVelocity = get (handles.VelocityFig,'value');
    FigRPM = get (handles.RPMFig,'value');
    FigPower = get (handles.PowerFig,'value');
    FigBattery = get (handles.BatteryFig,'value');
    FigDuty = get (handles.DutyFig,'value');
    %%To many arguments so have to use a matrix%%%
    %FigDeflection ( FigDisplace,FigVelocity,FigAccel, FigRPM, FigPower)
    FigDeflections = zeros (1,5);
    FigDeflections (1,1) = FigDisplace;
    FigDeflections (1,2) = FigVelocity;
    FigDeflections (1,3) = FigAccel;
    FigDeflections (1,4) = FigRPM;
    FigDeflections (1,5) = FigPower;
    %FigFinal (FigBattery, FigDuty);
    FigFinal = zeros (1,2);
    FigFinal(1,1) = FigBattery;
    FigFinal(1,2) = FigDuty;
    switch valRatios
    case 1
        GearRatio= 25;
    case 2
        GearRatio = 50;
    case 3
        GearRatio = 75;
    case 4
        GearRatio=100;
    end
switch valMotion
    case 1
        Motion = 1;
    case 2
        Motion = 2;
end

switch valPosition
    case 1
        Position = 1;
    case 2
        Position = 2;
end

fposition(SpeedValue,CarValue,AccelValue,PowerValue,LightValue,WeightValue,GearRatio,Motion,Position,DeviceValue,FigDeflections,FigFinal);

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

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

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

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

% --- Executes on button press in DisplaceFig.
function DisplaceFig_Callback(hObject, eventdata, handles)
% hObject    handle to DisplaceFig (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hint: get(hObject,'Value') returns toggle state of DisplaceFig

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

% Hint: get(hObject,'Value') returns toggle state of VelocityFig

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

% Hint: get(hObject,'Value') returns toggle state of AccelFig

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

% Hint: get(hObject,'Value') returns toggle state of RPMFig

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

% Hint: get(hObject,'Value') returns toggle state of PowerFig

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

% Hint: get(hObject,'Value') returns toggle state of BatteryFig

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

% Hint: get(hObject,'Value') returns toggle state of DutyFig

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

% Hint: get(hObject,'Value') returns toggle state of DeviceBox
% Hint: get(hObject,'Value') returns toggle state of PoweCheck
if (get(hObject,'Value') == get(hObject,'Max'))
    % Checkbox is checked-take appropriate action
    set(handles.Devices, 'visible', 'on')
else
    % Checkbox is not checked-take appropriate action
    set(handles.Devices, 'visible', 'off')
end

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

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

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

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject, 'BackgroundColor', 'white');
end
A.2 M-File fposition

function fposition(SpeedValue, CarValue, AccelValue, PowerValue, LightValue, WeightValue, GearRatio, Motion, Position, DeviceValue, FigDeflections, FigFinal)
clear global
% declare global variables

global vspeed_initial
global vtotal_time
global vpower_per_sec
global vpower_light
global vbattery_max
global vaccel
global vlength
global vloaded
global vlight_time
global vGearRatio
global vMotion

%Figure Globals
global FDeflections

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
FDeflections = FigDeflections;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Convert train speed to feet per second
vspeed_initial= SpeedValue*5280 / (60 * 60) ;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%E number of cars convert to feet
vlength = CarValue*53;

vloaded = WeightValue;

vaccel = AccelValue;

vGearRatio = GearRatio;

vMotion = Motion;
%Enter power of the lights (watts)
%**********Need to double check**********
\[v_{power\_light} = 40;\]

\[
% \text{20 seconds is the minimum distance from light}\]
% minimum distance from signal light comes on
\[\text{if } LightValue == 0 \]
\[v_{set\_light\_time} = 20;\]
\[v_{distance} = v_{speed\_initial} * v_{set\_light\_time};\]
\[\text{else}\]
\[v_{distance} = LightValue;\]
\[v_{set\_light\_time} = v_{distance} / v_{speed\_initial};\]
\[\text{end}\]

% velocity of train as the end passes final device
\[v_{final} = (v_{speed\_initial}^2 + 2 * \text{AccelValue} \times (2 \times v_{distance} + v_{length}))^{\frac{1}{2}};\]
%velocity of train as end passes crossing, wait 12 seconds then light will
%turn off
\[v_{final\_light} = (v_{speed\_initial}^2 + 2 \times \text{AccelValue} \times (v_{distance} + v_{length}))^{\frac{1}{2}};\]

%total time for train to be pass over all devices
\[\text{if } \text{AccelValue} == 0 \]
\[v_{total\_time} = (2 \times v_{distance} + v_{length}) / v_{speed\_initial};\]
\[v_{light\_time} = v_{set\_light\_time} + 12 + v_{length} / v_{speed\_initial};\]
\[\text{else}\]
\[v_{total\_time} = (v_{final} - v_{speed\_initial}) / \text{AccelValue};\]
\[v_{light\_time} = (v_{final\_light} - v_{speed\_initial}) / \text{AccelValue} + 12;\]
\[\text{end}\]

% enter size of battery being used (watts)
\[v_{battery\_max} = 10 \times 3600 / v_{light\_time} \times 12;\]

%Enter choice of power calculation
\[\text{if } \text{PowerValue} == 0 \]
\[v_{power\_per\_sec} = \text{fdeflection1}(v_{speed\_initial}, v_{loaded});\]
\[\text{else}\]
\[v_{power\_per\_sec} = \text{PowerValue};\]
\[\text{end}\]
%run function to calculate spacing
fspace(vdistance,Position,DeviceValue, FigFinal);
function fposition(SpeedValue, CarValue, AccelValue, PowerValue, LightValue, WeightValue, GearRatio, Motion, Position, DeviceValue, FigDeflections, FigFinal)
clear global
% declare global variables

global vspeed_initial
global vtotal_time
global vpower_per_sec
global vpower_light
global vbattery_max
global vaccel
global vlength
global vloaded
global vlight_time
global vGearRatio
global vMotion

%Figure Globals
global FDeflections

FDeflections = FigDeflections;

%Convert train speed to feet per second
vspeed_initial= SpeedValue*5280 / (60 * 60) ;

%E number of cars convert to feet
vlength = CarValue*53;

vloaded = WeightValue;
vaccel = AccelValue;
vGearRatio = GearRatio;

vMotion = Motion;
%Enter power of the lights (watts)
%***********Need to double check*********
vpower_light = 40;

% ******* 20 seconds is the minimum distance from light****
% minimum distance from signal light comes on
if LightValue == 0
    vset_light_time = 20;
    vdistance = vspeed_initial * vset_light_time;
else
    vdistance = LightValue;
    vset_light_time = vdistance / vspeed_initial;
end

% velocity of train as the end passes final device
vfinal = (vspeed_initial^2 + 2*AccelValue*(2*vdistance+vlength))^(1/2);
%velocity of train as end passes crossing, wait 12 seconds then light will
%turn off
vfinal_light = (vspeed_initial^2 + 2*AccelValue*(vdistance+vlength))^(1/2);

%total time for train to be pass over all devices
if AccelValue == 0
    vtotal_time = (2*vdistance + vlength) / vspeed_initial;
    vlight_time = vset_light_time + 12 + vlength / vspeed_initial;
else
    vtotal_time = (vfinal - vspeed_initial) / AccelValue;
    vlight_time = (vfinal_light - vspeed_initial) / AccelValue + 12;
end

% enter size of battery being used (watts)
vbattery_max = 10 * 3600 / vlight_time * 12;

%Enter choice of power calculation

if PowerValue == 0
    vpower_per_sec = fdeflection1(vspeed_initial, vloaded);
else
    vpower_per_sec = PowerValue;
end
%run function to calculate spacing
fspace(vdistance, Position, DeviceValue, FigFinal);
A.4 M-File fdeflection

function vpowersec = fdeflection1(vspeed,vloaded)

global vGearRatio
global vMotion

global FDeflections
global voutput

vvspeed = vspeed * 12;

vspeed = vspeed * 12;

http://www.freightcaramerica.com/quad-hopper.htm

http://www.nrel.gov/docs/fy02osti/32511.pdf

E = 30000000; \% ***** psi *****

I = 93.7; \% ***** in\(^4\) *****

p_load = vloaded / 8; \% ***** lbs *****

u = 3000; \% ***** psi *****

l_car = 53 * 12; \% ***** integer inches *****

L_wheels = 36; \% ***** integer inches *****

L_wheel_coupler = 60; \% ***** integer inches *****

B = (u / 4 / E / I) \(^.25\);

vbattery = 12.6;

vres1 = 21;

Vs = 0;

vtotal_amps = 0;

vtotal_rpms = 0;

vtotal_power = 0;
vtotal_amps1 = 0;
vtotal_rpms1 = 0;
y=1;

% internal resistance of generator
if vMotion == 1
   vires = 7.7;
   vslope = .0444;
else
   vires = 21;
   vslope = .05;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% %%%%%%%%
%*********************For Actual Power Calculation********
%********************** Wheel offsets************** ********
offset_1=round((L_wheels/vspeed)*1000);
offset_2=round((L_wheel_coupler/vspeed)*1000);
offset_3=round(((l_car-L_wheel_coupler)/vspeed)*1000);
offset_4=round(((l_car-L_wheels)/vspeed)*1000);
offset_5 =round((( l_car+L_wheels)/vspeed)*1000);
offset_6 =round((( l_car+L_wheel_coupler)/vspeed)*1000);
offset_7 = round(((2*l_car-L_wheel_coupler)/vspeed)*1000);
offset_8 =round((( 2*l_car-L_wheels)/vspeed)*1000);

%************************ period for one car ************
%displacement of rail inches (time)
for i=1:(l_car*4/vspeed)*1000,
   w(i) =(-p_load*B/2/u)*exp(-B*i*vspeed/1000)*(cos(B*i*vspeed/1000)+sin(B*i*vspeed/1000));
end

% velocity of displacement inches / second
for i=1:(l_car*4/vspeed)*1000,
   v(i) = -2*(-p_load*B/2/u)*(B*vspeed)*exp(-B*vspeed*i/1000)*sin(B*vspeed*i/1000);
end

% acceleration of displacement inches / sec^2
for i=1:(l_car*4/vspeed)*1000,
\[
a(i) = -2*(-p_{\text{load}}*B/2/u)*(B*\text{vspeed})^2*\exp(-B*\text{vspeed}*i/1000)*(\cos(B*\text{vspeed}*i/1000)-\sin(B*\text{vspeed}*i/1000))
\]
end

% superposition of all the wheel contacts
for i=1:l_car*2/vspeed*1000,
x_exteral(i)=i;

% ***** axle 1 *****
if i-offset_1 > 0
    w_1(i)=w(abs(i-offset_1));
v_1(i) = v(abs(i-offset_1));
a_1(i) = a(abs(i-offset_1));
elseif i-offset_1 <0
    v_1(i) = -v(abs(i-offset_1));
w_1(i)=w(abs(i-offset_1));
a_1(i) = a(abs(i-offset_1));
else
    w_1(i)= -(p_{\text{load}}*B/2/u)*(\cos(B*0)+\sin(B*0)) ;
v_1(i) = 0;
a_1(i)= -2*(-p_{\text{load}}*B/2/u)*(B*\text{vspeed})^2;
end

% ***** axle 2*****
if i-offset_2 > 0
    w_2(i)=w(abs(i-offset_2));
v_2(i) = v(abs(i-offset_2));
a_2(i) = a(abs(i-offset_2));
elseif i-offset_2 <0
    v_2(i) = -v(abs(i-offset_2));
w_2(i)=w(abs(i-offset_2));
a_2(i) = a(abs(i-offset_2));
else
    w_2(i)= -(p_{\text{load}}*B/2/u)*(\cos(B*0)+\sin(B*0)) ;
v_2(i) = 0;
a_2(i)= -2*(-p_{\text{load}}*B/2/u)*(B*\text{vspeed})^2;
end

% ***** axle 3 *****
if i-offset_3 > 0
    w_3(i)=w(abs(i-offset_3));
v_3(i) = v(abs(i-offset_3));
a_3(i) = a(abs(i-offset_3));
elseif i-offset_3 <0
    v_3(i) = -v(abs(i-offset_3));
w_3(i)=w(abs(i-offset_3));
a_3(i) = a(abs(i-offset_3));
else
    w_3(i)= -(p_{\text{load}}*B/2/u)*(\cos(B*0)+\sin(B*0)) ;
v_3(i) = 0;
a_3(i)= -2*(-p_{\text{load}}*B/2/u)*(B*\text{vspeed})^2;
end

% ***** axle 4 *****
if i-offset_4 > 0
    w_4(i)=w(abs(i-offset_4));

```
\text{v}_4(i) = \text{v}(\text{abs}(i-\text{offset}_4));
\text{a}_4(i) = \text{a}(\text{abs}(i-\text{offset}_4));
\text{elseif } i-\text{offset}_4 <0
\text{v}_4(i) = -\text{v}(\text{abs}(i-\text{offset}_4));
\text{w}_4(i)=\text{w}(\text{abs}(i-\text{offset}_4));
\text{a}_4(i) = \text{a}(\text{abs}(i-\text{offset}_4));
\text{else}
\text{w}_4(i)= - (\text{p}_\text{load}\cdot \text{B}/2/\text{u})\cdot (\cos(\text{B}\cdot 0)+\sin(\text{B}\cdot 0));
\text{v}_4(i) = 0;
\text{a}_4(i)= -2\cdot (\text{p}_\text{load}\cdot \text{B}/2/\text{u})\cdot (\text{B}\cdot \text{vspeed})^2;
\text{end}

%************************car next to it*************************

******** axel 5****
\text{if } i-\text{offset}_5 > 0
\text{w}_5(i)=\text{w}(\text{abs}(i-\text{offset}_5));
\text{v}_5(i) = \text{v}(\text{abs}(i-\text{offset}_5));
\text{a}_5(i) = \text{a}(\text{abs}(i-\text{offset}_5));
\text{elseif } i-\text{offset}_5 <0
\text{v}_5(i) = -\text{v}(\text{abs}(i-\text{offset}_5));
\text{w}_5(i)=\text{w}(\text{abs}(i-\text{offset}_5));
\text{a}_5(i) = \text{a}(\text{abs}(i-\text{offset}_5));
\text{else}
\text{w}_5(i)= - (\text{p}_\text{load}\cdot \text{B}/2/\text{u})\cdot (\cos(\text{B}\cdot 0)+\sin(\text{B}\cdot 0));
\text{v}_5(i) = 0;
\text{a}_5(i)= -2\cdot (\text{p}_\text{load}\cdot \text{B}/2/\text{u})\cdot (\text{B}\cdot \text{vspeed})^2;
\text{end}

******** axel 6
\text{if } i-\text{offset}_6 > 0
\text{w}_6(i)=\text{w}(\text{abs}(i-\text{offset}_6));
\text{v}_6(i) = \text{v}(\text{abs}(i-\text{offset}_6));
\text{a}_6(i) = \text{a}(\text{abs}(i-\text{offset}_6));
\text{elseif } i-\text{offset}_6 <0
\text{v}_6(i) = -\text{v}(\text{abs}(i-\text{offset}_6));
\text{w}_6(i)=\text{w}(\text{abs}(i-\text{offset}_6));
\text{a}_6(i) = \text{a}(\text{abs}(i-\text{offset}_6));
\text{else}
\text{w}_6(i)= - (\text{p}_\text{load}\cdot \text{B}/2/\text{u})\cdot (\cos(\text{B}\cdot 0)+\sin(\text{B}\cdot 0));
\text{v}_6(i) = 0;
\text{a}_6(i)= -2\cdot (\text{p}_\text{load}\cdot \text{B}/2/\text{u})\cdot (\text{B}\cdot \text{vspeed})^2;
\text{end}

******** axel 7
\text{if } i-\text{offset}_7 > 0
\text{w}_7(i)=\text{w}(\text{abs}(i-\text{offset}_7));
\text{v}_7(i) = \text{v}(\text{abs}(i-\text{offset}_7));
\text{a}_7(i) = \text{a}(\text{abs}(i-\text{offset}_7));
\text{elseif } i-\text{offset}_7 <0
\text{v}_7(i) = -\text{v}(\text{abs}(i-\text{offset}_7));
\text{w}_7(i)=\text{w}(\text{abs}(i-\text{offset}_7));
\text{a}_7(i) = \text{a}(\text{abs}(i-\text{offset}_7));
\text{else}
\text{w}_7(i)= - (\text{p}_\text{load}\cdot \text{B}/2/\text{u})\cdot (\cos(\text{B}\cdot 0)+\sin(\text{B}\cdot 0));
\text{v}_7(i) = 0;
a_7(i) = -2*(-p_load*B/2/u)*(B*vspeed)^2;

%***** axel 8
if i-offset_8 > 0
    w_8(i)=w(abs(i-offset_8));
v_8(i) = v(abs(i-offset_8));
a_8(i) = a(abs(i-offset_8));
elseif i-offset_8 <0
    v_8(i) = -v(abs(i-offset_8));
w_8(i)=w(abs(i-offset_8));
a_8(i) = a(abs(i-offset_8));
else
    w_8(i)= (p_load*B/2/u)*(cos(B*0)+sin(B*0));
v_8(i) = 0;
a_8(i) = -2*(-p_load*B/2/u)*(B*vspeed)^2;
end

% ***** Total_deflection *****
w_total(i)=(w_3(i)+w_4(i)+w_1(i)+w_2(i)+w_5(i)+w_6(i)+w_7(i)+w_8(i));
v_total(i)=(v_3(i)+v_4(i)+v_1(i)+v_2(i)+v_5(i)+v_6(i)+v_7(i)+v_8(i));
a_total(i)=(a_3(i)+a_4(i)+a_1(i)+a_2(i)+a_5(i)+a_6(i)+a_7(i)+a_8(i));

end

%new differentiation
q = diff(v_total);
z = diff(x_extra1)/1000;
for i=1:l_car*2/vspeed*1000-1,
x_extra2(i) =i;
da(i) = q(i)/z(i);
end
for i = 1 : l_car*2/vspeed*1000

%calculate power produced by devices due to deflection
if vMotion == 1
    v_revolution(i) = abs((v_total(i)/(2*pi()*.7874))*60*vGearRatio*(20/15));
else
    v_revolution(i) = (v_total(i)/(2*pi()*.7874))*60*vGearRatio;
end
if v_revolution (i) <0
    v_revolution(i)=0;
end

Vs(i) = vslope*v_revolution(i);
vout(i)= Vs(i)-vbattery;
if vout(i) <0
    vout(i) = 0;
end
vdevice_amps(i) = vout(i)/(vres1+vires);
vdevice_power(i) =
vdevice_amps(i)*(vbattery+vdevice_amps(i)*vres1);
if vdevice_amps(i) >0
  end
end

%generator graphs******************
% NEED TO INCLUDE DIFFERENT GENERATOR DATA and Max Rpms
%12 is load of circuit

x=0;
for (i = t_period_start:t_period_end)
  x=x+1;
  y_period(x) = vdevice_power(i);
  y_period_rpm(x) = 600;v_revolution(i);
  vtotal_rpms= vtotal_rpms + v_revolution(i);
  vtotal_power= vtotal_power + vdevice_power(i);
  vtotal_amps = vtotal_amps +vdevice_amps(i);
  vaverage_rpms = vtotal_rpms / x ;
  vaverage_power = vtotal_power /x;
  vaverage_amps = vtotal_amps / x ;
  vpowersec = vaverage_power;
  if v_revolution (i) > 252
    y=y+1;
    vtotal_amps1 = vtotal_amps1 +vdevice_amps(i);
    vtotal_rpms1 = vtotal_rpms1 + v_revolution(i);
    vaverage_rpms1 = vtotal_rpms1 /y;
    vaverage_amps1 = vtotal_amps1 / y;
  end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% %%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%Revolution%%%%%%%%%%%%%%%%
%x_period_rpm = x_extra1(t_period_start):x_extra1(t_period_end);
%v_rev = polyarea(x_period_rpm,y_period_rpm) /1000;
%v_rpm_generator = v_rev / (t_period_end - t_period_start)*1000
v_rpm_gear = vaverage_rpms/ vGearRatio
vaverage_amps
vaverage_power
vaverage_rpms
vaverage_rpms1
vaverage_amps1
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% %%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%POWER %%%%%%%%%%%%%%%%%%%%%%%%%%%% %%%%%%%%%%%%%%%
%x_period = x_extra1(t_period_start):x_extra1(t_period_end);
%Power produce as one car passes over
%v_power_car = polyarea(x_period,y_period)/1000;
%Power per second
%vpowersec= v_power_car / (t_period_end - t_period_start)*1000

voutput = zeros (1,5);
voutput (1,1) = vaverage_rpms;
voutput (1,2) = v_rpm_gear;
voutput (1,3) = vaverage_power;

if FDeflections(1,1) == 1
figure(1);
clear;
hold on;
plot(x_extra1,w_total,'r');
plot([offset_3,offset_3],[0.05,-0.2]);
plot([offset_4,offset_4],[0.05,-0.2]);
plot([offset_1,offset_1],[0.05,-0.2]);
plot([offset_2,offset_2],[0.05,-0.2]);
plot([offset_5,offset_5],[0.05,-0.2]);
plot([offset_6,offset_6],[0.05,-0.2]);
plot([offset_7,offset_7],[0.05,-0.2]);
plot([offset_8,offset_8],[0.05,-0.2]);

ylabel('y - displacement (in)');
xlabel('time - t (.001 sec)');
title('displacement vs. time ');
end

if FDeflections(1,2) == 1
figure(2);
clear;
hold on;
plot(x_extra1,v_total,'r');
plot([offset_3,offset_3],[0.001,5]);
plot([offset_4,offset_4],[0.001,5]);
plot([offset_1,offset_1],[0.001,5]);
plot([offset_2,offset_2],[0.001,5]);
plot([offset_5,offset_5],[0.001,5]);
plot([offset_6,offset_6],[0.001,5]);
plot([offset_7,offset_7],[0.001,5]);
plot([offset_8,offset_8],[0.001,5]);

ylabel('v- velocity (in/s)');
xlabel('time - t (.001 sec)');
title('velocity vs. time ');
end

if FDeflections(1,3) == 1
figure(3);
clear;
hold on;
plot(x_extra2,da,'r');
plot([offset_3,offset_3],[0.001,1]);
plot([offset_4,offset_4],[0.001,1]);
plot([offset_1,offset_1],[0.001,1]);
plot([offset_2,offset_2],[0.001,1]);
plot([offset_5,offset_5],[0.001,1]);
plot([offset_6,offset_6],[0.001,1]);
plot([offset_7,offset_7],[0.001,1]);
plot([offset_8,offset_8],[0.001,1]);
ylabel('a-- acceleration (in/s \(^2\))');
xlabel('time - t (.001 sec)');
title('acceleration vs. time ');
end

if FDeflections(1,5) == 1

figure(5);
clf;
hold on;
plot([x_extral],vdevice_power,'r');
ylabel('p --power (watts)');
xlabel('time - t (.001 sec)');
title('power vs. time ');
end

if FDeflections(1,4) == 1
figure(4);
clf;
hold on;
plot([x_extral],v_revolution,'r');
plot([offset_3,offset_3],[0.001,1]);
plot([offset_4,offset_4],[0.001,1]);
plot([offset_1,offset_1],[0.001,1]);
plot([offset_2,offset_2],[0.001,1]);
plot([offset_5,offset_5],[0.001,1]);
plot([offset_6,offset_6],[0.001,1]);
plot([offset_7,offset_7],[0.001,1]);
plot([offset_8,offset_8],[0.001,1]);
ylabel('r--revolutions per minute (rpms)');
xlabel('time - t (.001 sec)');
title('revolutions vs. time ');
end
A.5 M-File \textit{ftest}

\begin{verbatim}
function vpowersec = fdeflection1(vspeed,vloaded)
global vGearRatio
global vMotion

global FDeflections
global voutput

%vspeed in inches per second
vspeed=vspeed*12;
%http://www.freightcaramerica.com/quad-hopper.htm

%http://www.nrel.gov/docs/fy02osti/32511.pdf

%the modulus of elasticity of the rail
E=30000000;    \% ***** psi *****

%moment of inertia of the rail
I=93.7;        \% ***** in^4 *****

%loaded car weight spread out to the 8 wheels
p_load=vloaded/8;    \% ***** lbs *****

%******** Need to double Check******
%track modulus
u=3000;        \% ***** psi *****

%length of car
l_car=53*12;    \% ***** integer inches *****

%distance from front to first axel
L_wheels=36;     \% ***** integer inches *****

% distance from front to second axel
L_wheel_coupler=60; \% ***** integer inches *****

B=(u/4/E/I)^.25;

%battery volts
vbattery=12.6 ;

% resistance circuit divider
vres1 =21;
Vs= 0;
vtotal_amps = 0;
vtotal_rpms = 0;
vtotal_power =0;
\end{verbatim}
vtotal_amps1 = 0;
vtotal_rpms1 = 0;
y=1;

% internal resistance of generator
if vMotion == 1
    vires = 7.7;
    vslope = .0444;
else
    vires = 21;
    vslope = .05;
end

%**********************************************************************************************
% For Actual Power Calculation*********
% Wheel offsets**************************
offset_1 = round((L_wheels/vspeed)*1000);
offset_2 = round((L_wheel_coupler/vspeed)*1000);
offset_3 = round(((l_car-L_wheel_coupler)/vspeed)*1000);
offset_4 = round(((l_car-L_wheels)/vspeed)*10000);
offset_5 = round(((l_car+L_wheels)/vspeed)*1000);
offset_6 = round(((l_car+L_wheel_coupler)/vspeed)*1000);
offset_7 = round(((2*l_car-L_wheel_coupler)/vspeed)*1000);
offset_8 = round(((2*l_car-L_wheels)/vspeed)*1000);

% period for one car *****************
t_total_time = (l_car*2/vspeed)*1000;
t_period_start = round(t_total_time*.25);
t_period_end = round((t_period_start + l_car / vspeed)*1000));

% displacement of rail inches (time)
for i=1:(l_car*4/vspeed)*1000,
    w(i) =(-p_load*B/2/u)*exp(-B*i*vspeed/1000)*(cos(B*i*vspeed/1000)+sin(B*i*vspeed/1000));
end

% velocity of displacement inches / second
for i=1:(l_car*4/vspeed)*1000,
    v(i) = -2*(-p_load*B/2/u)*(B*vspeed)*exp(-B*vspeed*i/10000)*sin(B*vspeed*i/1000);
end

% acceleration of displacement inches / sec^2
for i=1:(l_car*4/vspeed)*1000,
a(i) = -2*(-p_load*B/2/u)*(B*vspeed)^2*exp(-B*vspeed*i/1000)*(cos(B*vspeed*i/1000)-sin(B*vspeed*i/1000));
end

%superposition of all the wheel contacts
for i=1:l_car*2/vspeed*1000,
    x_extra1(i)=i;

% ***** axle 1 *****
    if i-offset_1> 0
        w_1(i)=w(abs(i-offset_1));
        v_1(i) = v(abs(i-offset_1));
        a_1(i) = a(abs(i-offset_1));
    elseif i-offset_1 <0
        v_1(i) = -v(abs(i-offset_1));
        w_1(i)=w(abs(i-offset_1));
        a_1(i) = a(abs(i-offset_1));
    else
        w_1(i)= -(p_load*B/2/u)*(cos(B*0)+sin(B*0));
        v_1(i) = 0;
        a_1(i)= -2*(-p_load*B/2/u)*(B*vspeed)^2;
    end

% ***** axle 2*****
    if i-offset_2> 0
        w_2(i)=w(abs(i-offset_2));
        v_2(i) = v(abs(i-offset_2));
        a_2(i) = a(abs(i-offset_2));
    elseif i-offset_2 <0
        v_2(i) = -v(abs(i-offset_2));
        w_2(i)=w(abs(i-offset_2));
        a_2(i) = a(abs(i-offset_2));
    else
        w_2(i)= -(p_load*B/2/u)*(cos(B*0)+sin(B*0));
        v_2(i) = 0;
        a_2(i)= -2*(-p_load*B/2/u)*(B*vspeed)^2;
    end

% ***** axle 3 *****
    if i-offset_3> 0
        w_3(i)=w(abs(i-offset_3));
        v_3(i) = v(abs(i-offset_3));
        a_3(i) = a(abs(i-offset_3));
    elseif i-offset_3 <0
        v_3(i) = -v(abs(i-offset_3));
        w_3(i)=w(abs(i-offset_3));
        a_3(i) = a(abs(i-offset_3));
    else
        w_3(i)= -(p_load*B/2/u)*(cos(B*0)+sin(B*0));
        v_3(i) = 0;
        a_3(i)= -2*(-p_load*B/2/u)*(B*vspeed)^2;
    end

% ***** axle 4 *****
    if i-offset_4> 0
        w_4(i)=w(abs(i-offset_4));
v_4(i) = v(abs(i-offset_4));
a_4(i) = a(abs(i-offset_4));
elseif i-offset_4 <0
 v_4(i) = -v(abs(i-offset_4));
w_4(i)=w(abs(i-offset_4));
a_4(i) = a(abs(i-offset_4));
else
 w_4(i)=- (p_load*B/2/u)*(cos(B*0)+sin(B*0));
v_4(i) = 0;
a_4(i)= -2*(-p_load*B/2/u)*(B*vspeed)^2;
end

%************************car next to
it*******************************

%%%%**** axel 5****
if i-offset_5 > 0
 w_5(i)=w(abs(i-offset_5));
v_5(i) = v(abs(i-offset_5));
a_5(i) = a(abs(i-offset_5));
elseif i-offset_5 <0
 v_5(i) = -v(abs(i-offset_5));
w_5(i)=w(abs(i-offset_5));
a_5(i) = a(abs(i-offset_5));
else
 w_5(i)=- (p_load*B/2/u)*(cos(B*0)+sin(B*0));
v_5(i) = 0;
a_5(i)= -2*(-p_load*B/2/u)*(B*vspeed)^2;
end

%%%%**** axel 6
if i-offset_6 > 0
 w_6(i)=w(abs(i-offset_6));
v_6(i) = v(abs(i-offset_6));
a_6(i) = a(abs(i-offset_6));
elseif i-offset_6 <0
 v_6(i) = -v(abs(i-offset_6));
w_6(i)=w(abs(i-offset_6));
a_6(i) = a(abs(i-offset_6));
else
 w_6(i)=- (p_load*B/2/u)*(cos(B*0)+sin(B*0));
v_6(i) = 0;
a_6(i)= -2*(-p_load*B/2/u)*(B*vspeed)^2;
end

%%%%**** axel 7
if i-offset_7 > 0
 w_7(i)=w(abs(i-offset_7));
v_7(i) = v(abs(i-offset_7));
a_7(i) = a(abs(i-offset_7));
elseif i-offset_7 <0
 v_7(i) = -v(abs(i-offset_7));
w_7(i)=w(abs(i-offset_7));
a_7(i) = a(abs(i-offset_7));
else
 w_7(i)=- (p_load*B/2/u)*(cos(B*0)+sin(B*0));
v_7(i) = 0;
a_7(i) = -2*(-p_load*B/2/u)*(B*vspeed)^2;
end

%***** axel 8
if i-offset_8 > 0
w_8(i)=w(abs(i-offset_8));
v_8(i) = v(abs(i-offset_8));
a_8(i) = a(abs(i-offset_8));
elseif i-offset_8 <0
v_8(i) = -v(abs(i-offset_8));
w_8(i)=w(abs(i-offset_8));
a_8(i) = a(abs(i-offset_8));
else
w_8(i)= (p_load*B/2/u)*(cos(B*0)+sin(B*0));
v_8(i) = 0;
a_8(i) = -2*(-p_load*B/2/u)*(B*vspeed)^2;
end

% ***** Total_deflection *****
w_total(i)=(w_3(i)+w_4(i)+w_1(i)+w_2(i)+w_5(i)+w_6(i)+w_7(i)+
w_8(i));
v_total(i)=(v_3(i)+v_4(i)+v_1(i)+v_2(i)+v_5(i)+v_6(i)+v_7(i)+
v_8(i));
a_total(i)=(a_3(i)+a_4(i)+a_1(i)+a_2(i)+a_5(i)+a_6(i)+a_7(i)+
a_8(i));
end

%new differentiation
q = diff(v_total);
z = diff(x_extra1)/1000;
for i=1:l_car*2/vspeed*1000-1,
x_extra2(i) =i;
da(i) = q(i)/z(i);
end
for i = 1 : l_car*2/vspeed*1000

%calculate power produced by devices due to deflection
if vMotion == 1
  %%%%%%%20/15 is dependent on design of the second device
  v_revolution(i) =
  abs((v_total(i)/(2*pi()*0.7874))*60*vGearRatio*(20/15));
else
  v_revolution(i) = (v_total(i)/(2*pi()*0.7874))*60*vGearRatio;
end
if v_revolution (i) <0
  v_revolution(i)=0;
end

Vs(i) = vslope*v_revolution(i);
vout(i)= Vs(i)-vbattery;
if vout(i) <0
  vout(i) = 0;
end
\[ \text{vdevice_amps}(i) = \frac{\text{vout}(i)}{\text{vres}1 + \text{vires}}; \]
\[ \text{vdevice_power}(i) = \text{vdevice_amps}(i) \times (\text{vbattery} + \text{vdevice_amps}(i) \times \text{vres}1); \]
\[ \text{if } \text{vdevice_amps}(i) > 0 \]
\[ \text{end} \]
\[ \text{end} \]

% generator graphs

%%%%%%%%%%%%%%%%%%%%%% NEED TO INCLUDE DIFFERENT GENERATOR DATA and Max Rmps

%12 is load of circuit

\[ x = 0; \]
\[ \text{for } (i = \text{t_period_start}:\text{t_period_end}) \]
\[ x = x + 1; \]

\[ \%y_period(x) = \text{vdevice_power}(i); \]
\[ \%y_period_rpm(x) = 600; \%v_revolution(i); \]
\[ \text{vtotal_rpms} = \text{vtotal_rpms} + \text{v_revolution}(i); \]
\[ \text{vtotal_power} = \text{vtotal_power} + \text{vdevice_power}(i); \]
\[ \text{vtotal_amps} = \text{vtotal_amps} + \text{vdevice_amps}(i); \]
\[ \text{vaverage_rpms} = \text{vtotal_rpms} / x; \]
\[ \text{vaverage_power} = \text{vtotal_power} / x; \]
\[ \text{vaverage_amps} = \text{vtotal_amps} / x; \]
\[ \text{vtotal_amps1} = \text{vtotal_amps} + \text{vdevice_amps}(i); \]
\[ \text{vtotal_rpms1} = \text{vtotal_rpms} + \text{v_revolution}(i); \]
\[ \text{vaverage_rpms1} = \text{vtotal_rpms1} / y; \]
\[ \text{vaverage_amps1} = \text{vtotal_amps1} / y; \]

\[ \text{end} \]
\[ \text{end} \]

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Revolution
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

\[ \%x_period_rpm = \text{x_extreal(t_period_start):x_extreal(t_period_end)}; \]
\[ \%v_rev = \text{polyarea(x_period_rpm, y_period_rpm)} / 1000; \]
\[ \%v_rpm_generator = \%v_rev / (\text{t_period_end} - \text{t_period_start}) \times 1000 \]

\[ \text{v_rpm_gear} = \text{vaverage_rpms} / \text{vGearRatio} \]
\[ \text{vaverage_amps} \]
\[ \text{vaverage_power} \]
\[ \text{vaverage_rpms} \]
\[ \text{vaverage_rpms1} \]
\[ \text{vaverage_amps1} \]

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% POWER
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% $x_{\text{period}} = x_{\text{extral}(t_{\text{period_start}})}:x_{\text{extral}(t_{\text{period_end}})}$
% \text{Power produce as one car passes over}
% $v_{\text{power}_\text{car}} = \text{polyarea}(x_{\text{period}},y_{\text{period}})/1000$
% \text{Power per second}
% $v_{\text{powersec}} = v_{\text{power}_\text{car}} / (t_{\text{period}_{\text{end}}} - t_{\text{period}_{\text{start}}}) \times 1000$

voutput = zeros(1,5);
output (1,1) = vaverage_rpm;
output (1,2) = v_rpm_gear;
output (1,3) = vaverage_power;

if FDeflections(1,1) == 1
figure(1);
clf;
hold on;
plot(x_extra1,w_total,'r');
plot([offset_3,offset_3],[0.05,-0.2]);
plot([offset_4,offset_4],[0.05,-0.2]);
plot([offset_1,offset_1],[0.05,-0.2]);
plot([offset_2,offset_2],[0.05,-0.2]);
plot([offset_5,offset_5],[0.05,-0.2]);
plot([offset_6,offset_6],[0.05,-0.2]);
plot([offset_7,offset_7],[0.05,-0.2]);
plot([offset_8,offset_8],[0.05,-0.2]);

ylabel('y - displacement (in)');
xlabel('time - t (.001 sec)');
title('displacement vs. time ');
end

if FDeflections(1,2) == 1
figure(2);
clf;
hold on;
plot(x_extra1,v_total,'r');
plot([offset_3,offset_3],[0.001,5]);
plot([offset_4,offset_4],[0.001,5]);
plot([offset_1,offset_1],[0.001,5]);
plot([offset_2,offset_2],[0.001,5]);
plot([offset_5,offset_5],[0.001,5]);
plot([offset_6,offset_6],[0.001,5]);
plot([offset_7,offset_7],[0.001,5]);
plot([offset_8,offset_8],[0.001,5]);

ylabel('v- velocity (in/s)');
xlabel('time - t (.001 sec)');
title('velocity vs. time ');
end

if FDeflections(1,3) == 1
figure(3);
clf;
hold on;
plot(x_extra2,da,'r');
plot([offset_3,offset_3],[0.001,1]);
plot([offset_4,offset_4],[0.001,1]);
plot([offset_1,offset_1],[0.001,1]);
plot([offset_2,offset_2],[0.001,1]);
plot([offset_5,offset_5],[0.001,1]);
plot([offset_6,offset_6],[0.001,1]);
plot([offset_7,offset_7],[0.001,1]);
plot([offset_8,offset_8],[0.001,1]);
ylabel('a- acceleration (in/s ^2)');
xlabel('time - t (.001 sec)');
title('acceleration vs. time ');
end
if FDeflections(1,5) == 1

figure(5);
clf;
hold on;
plot((x_extra1),vdevice_power,'r');
ylabel('p -power (watts)');
xlabel('time - t (.001 sec)');
title('power vs. time ');
end
if FDeflections(1,4) == 1
figure(4);
clf;
hold on;
plot((x_extra1),v_revolution,'r');
plot([offset_3,offset_3],[0.001,1]);
plot([offset_4,offset_4],[0.001,1]);
plot([offset_1,offset_1],[0.001,1]);
plot([offset_2,offset_2],[0.001,1]);
plot([offset_5,offset_5],[0.001,1]);
plot([offset_6,offset_6],[0.001,1]);
plot([offset_7,offset_7],[0.001,1]);
plot([offset_8,offset_8],[0.001,1]);
ylabel('r-revolutions per minute (rpms)');
xlabel('time - t (.001 sec)');
title('revolutions vs. time ');
end
**A.6 M-File `fpowerlosses`**

```matlab
function y = fpowerlosses(vpositions,vpower_on,vpower_per_sec)

vdevices = length(vpower_on);
vpower_loss = 0;
x = 0;
y=0;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% %%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%% One half of total%%%%%%%%%%%%%%%% %%%%%%%%%%%%%%
for m = 1:vdevices/2
    if vpower_on(1,m) ==1
        x= x+1;
    end
end
% The total power being produced per iteration.
vpower_sec_total = x*vpower_per_sec;
vfirst =0;
% Starts with the the innermost device postion and loops itself outward
for n=vdevices/2:-1:1
    %checks to see if device is producing power
    if vpower_on(1,n)==1;
        %.006282 is resistance of wire in ohms per feet
        % check to see what the volatage of the system is
        if vfirst == 0
            vpower_loss = vpower_loss + vpositions(1,n)*.0006282*(vpower_sec_total/24)^2;
            vfirst =1;
        else
            vpower_sec_total= vpower_sec_total -  vpower_per_sec;
            vpower_loss = vpower_loss + (vpositions(1,n)-vpositions(1,(n+1)))*.0006282*((vpower_sec_total)/24)^2;
        end
    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% %%%%%%%%%%%%%%%%
%%%%%%%%%%%%%% Other Half%%%%%%%%%%%%%%%%%%%%%%%%%% %%%%%%%%%%%%%%
vpower_sec_total =0;
for m = (vdevices/2+1) : vdevices
    if vpower_on(1,m) ==1
        y= y+1;
    end
end
vpower_sec_total = y*vpower_per_sec;
vfirst1 = 0;

for n=(vdevices/2+1):1:vdevices
    if vpower_on(1,n)==1;
        if vfirst1 ==0
```
vpower_loss = vpower_loss + 
abs(vpositions(1,n))*.0006282*(vpower_sec_total/24)^2;
vfirst1 =1;
else
  vpower_sec_total = vpower_sec_total - vpower_per_sec;
vpower_loss = vpower_loss +  abs((vpositions(1,n) -
  vpositions(1,n-1)))*.0006282588*(vpower_sec_total/24)^2;

end

end

end

y = vpower_loss;
A.7 M-File \textit{fdutycycle}

\begin{verbatim}
function fdutycycle(vpower,t)
% period is the time between the lights flashing
% used to keep a moving average
global time1
global time2
global time3
global time4
global time5

global ygraph
global vgraph
global graphpower
global vtimelight
z=t+1;
ygraph(z) = z;
% sample powers are taken every 1 sec and average runs for
% after first loop no longer needed since variables will be defined
if t == 0
    time1 = 0;
    time2 = 0;
    time3 = 0;
    time4 = 0;
    time5 = 0;
end
%move average

    time5 = time4;
    time4 = time3;
    time3 = time2;
    time2 = time1;
    time1 = vpower;

    average = (time1 + time2 + time3 +time4 + time5)/5 ;

    vlight = average/40 ;
    graphpower(z) = average;
%time for light to be on
    vtimelight = 1 * vlight;
    if vtimelight <=.5
        vtimelight = .5;
    end
    if vtimelight >1
        vtimelight = 1;
    end
    vgraph(z) = vtimelight;
\end{verbatim}
APPENDIX B

RABBIT SINGLE BOARD COMPUTER
B.1 Programming

/****

************

Reads and records the voltage of a single-ended analog input Channels. The voltage is calculated from coefficients read from the reserved eeprom storage device. Controls electromagnetic relays via digital pins

************

***/

#class auto // Change local var storage default to "auto"

// include BLxS2xx series library
#use "BLxS2xx.lib"

// size of circular buffer
#define NUM_SAMPLES 2

// Call in the FAT filesystem support code.
#use "fat.lib"

// Map program to xmem if not compiling to separate I&D space.
#if !__SEPARATE_INST_DATA__
#memmap xmem
#endif

#define MAX_FILES 5

// Set file system to use forward slash as directory separator
#define FAT_USE_FORWARDSLASH

// Required control structure to operate the FAT filesystem
FATfile file;

// calibration values
calib_t adc_calibration[BL_ANALOG_IN];

// circular buffer
int adc_data_array[BL_ANALOG_IN][NUM_SAMPLES];

// running total of window
long adc_window_total[BL_ANALOG_IN];
// circular buffer position
int adc_data_pos[BL_ANALOG_IN];

// set the STDIO cursor location and display a string
void DispStr(int x, int y, char *s)
{
    x += 0x20;
    y += 0x20;
    printf ("\x1B%c%c%s", x, y, s);
}

// blank the stdio screen
void blankScreen(void)
{
    printf("\x1Bt");
}

// add rawdata to circular buffer and return average
int addToBufferAvg(int channel, int rawdata)
{
    auto int *pos;

    // store pointer to array for speedup
    pos = &adc_data_array[channel][adc_data_pos[channel];

    // adjust total value in window
    adc_window_total[channel] += rawdata - *pos;

    // add new value to circular buffer
    *pos = rawdata;

    // adjust position to be within bounds
    if (++adc_data_pos[channel] >= NUM_SAMPLES)
    {
        adc_data_pos[channel] = 0;
    }

    // return new average
    return (int) ((adc_window_total[channel] + (NUM_SAMPLES / 2)) / NUM_SAMPLES);
}

// convert from rawdata to volts
float convertToVolts(int channel, int rawdata)
{

rawdata = adc_calibration[channel].offset - rawdata;
return adc_calibration[channel].gain * rawdata;

int main()
{
#define REC_LEN 21
    int rawdata;
    int channel;
    int gaincode;
    float voltage;
    char p[128];
    char z[64];
    int sample;
    int period;
    int flip;
    int mount;
    // Toggle relays
    int duty_cycle;
    int percentage;
    float full_power;
    int relay;
    int light_on;

    // Digital vars
    char mask;
    char channel_bank;

    int i, rc, savedRC;
    int led;
        int h, m, s, ms;
    int filestate;
    int trial;
    long alloc;
    long readto;
        float val;
    char LEDDRShadow;
    char LEDDDRShadow;
    char LEDFRShadow;
    char *iptr, *optr;
    char ibuf[REC_LEN], obuf[REC_LEN];
    int icount, ocount;
    int record, turn_off;
    //static char * const name = "/voltage.txt";
    fat_part *first_part;  // Use the first mounted FAT partition.
m = s = ms = 0;                     // Clear system variables
filestate = 0;
led = 0;
readto = 0;
alloc = 1;
savedRC = 0;
record = 0;
turn_off = 0;
trial = 0;
mount = 1;
light_on = 0;

// Adjustable for duty cycle
// Period in Ms
period = 1000;
flip = 0;
full_power = 2;
relay = 2;
duty_cycle = 900;

// Initialize the controller
brdInit();

// Configure all outputs to be general digital outputs that are high
channel_bank = 0xFF;

// Set output to be general digital output
setDigOut(0, 1);

// Channels 0, 1, 2, are for duty cycle relays, channel 3 is relay for
// power to lights, channel 4 is power to shutoff relay.

// Configure channel 0 for Single-Ended unipolar mode of operation.
// (Max voltage range is 0 - 20v)
anaInConfig(0, SE0_MODE);

// initialize circular buffer index
adc_data_pos[0] = 0;

////////////////////////////////////////////////////////////////////////////////
while ((rc = fat_AutoMount(FDDF_USE_DEFAULT)) == -EBUSY);
    first_part = NULL;
for (i = 0; i < num_fat_devices * FAT_MAX_PARTITIONS; ++i)
{
    if ((first_part = fat_part_mounted[i]) != NULL)
    {
        // found a mounted partition, so use it
        break;
    }
}

// OK, filesystem exists and is ready to access. Let's create a file.
// Open the data file (Block until complete)
while (1)
{
    sprintf(z,"Voltage%d.TXT",trial);
    file.state = 0;  // Initialize filestate to Idle
    while ((rc = fat_Open( first_part, z, FAT_FILE,
        FAT_MUST_CREATE,
        &file, &alloc)) == -EBUSY);
        if (rc < 0)
        {
            if (rc == -17)
                trial+=1;
        }
    else
        {
            printf("fat_Open: rc = %d\n",rc);
            while ((rc = fat_UnmountDevice( first_part->dev )) == -
                EBUSY);
                return rc;
        }
    else
        break;
}
while (1)
{
    // prinrange();
    // printf(" Choose gain code (0-%d) = ", BL_MAX_GAINS - 1);
    // do
    // {
    //     while(!kbhit()); // wait for key hit
    //     gaincode = getchar() - '0';  // convert ascii key into number
    // } while ((gaincode < 0) || (gaincode >= BL_MAX_GAINS));
    // printf("%d", gaincode);

    // load calibration constants and init circular buffer
    gaincode = 7 ;
    _anaInEERd(0, SE0_MODE, gaincode, &adc_calibration[0]);

    // initialize circular buffer with zeros
    memset(adc_data_array, 0, sizeof(adc_data_array));
    // initialize total values with zeros
    memset(adc_window_total, 0, sizeof(adc_window_total));

    // blankScreen();
    //DispStr(1, 2, "A/D input voltage for channels 0 - 7");
    //DispStr(1, 3, "-------------------------------------");
    //DispStr(1, 14, "Press key to select another gain option.");

    while(1)
    {
        // update sinking drivers
        mask = 0x01;
        digOutBank(0, channel_bank);

        // Time Stamp data
        costate time always_on
        {
            waitfor (record==1);
            DelayMs(100);  // Wait for one tenth of second to elapse
            if (++ms == 10)  // Increment seconds
            {
                ms = 0;
                if (++s == 60)  // Increment minutes
                {
                    s = 0;
                    if (++m == 60)  // Increment minutes
                    {
                        m = 0;
                    }
                }
            }
        }
    }
}
s = 0;
if (++m == 60)  // Increment hours
{
    m = 0;
}
}

/////////////////////////////////////////////////////////////////////
/////////////////////// Read Voltage///////////////////////

costate read always_on
{
    DelayMs(100);  // Wait for one tenth of second to elapse to
                   // to allow other coded to run
    rawdata = anaIn(0, gaincode);
    if (rawdata > BL_ERRCODESTART)
    {
        sample = addToBufferAvg(0, rawdata);
        if (((long)(adc_calibration[0].gain)) != 0x80000000)
        {
            voltage = convertToVolts(0, sample);
            turn_off += 1;
        }
    }
    if (voltage < 0)
    {
        voltage = 0;
    }

    //There is a voltage reading so start recording
    if (voltage >= .1)
    {
        //restart clock time to match data

        record = 1;
        turn_off = 0;
    }
    sprintf(p, "Channel = %2d Raw = %5d Voltage = %.3fV   "
            "   %2d ", 0, sample, voltage, turn_off);
}
else
{
}
sprintf(p, "Channel = %2d Raw = %5d Voltage = Not Calibrated"
        " ", 0, sample);
}
else if (rawdata == BL_OVERFLOW)
{
    sprintf(p, "Channel = %2d Voltage = Exceeded Range ", 0);
}
else
{
    sprintf(p, "Channel = %2d Voltage = Error During Read ", 0);
}
DispStr(1,0 + 4, p);

 //////////////////////////////////////////////////////////////////////
 //////////////////////////////////////////////////////////////////////
 //Possibly test costate save and turn on when voltage >.1
 costate save always_on
 {
     if (record ==1)
     {
         // Create the fill string by inserting timestamp and reading
         sprintf(obuf, "%02d:%02d:%2d -- %6.3f\n", m, s, ms, voltage);
         ocount = 0; // Initialize output count and buffer pointer
         optr = obuf;
         waitfor (filestate == 0); // Wait until file is available
         filestate = 1; // Show file is being updated

         // Only do this at beginning to give power to lights
         if (light_on ==0)
         {
             channel_bank = channel_bank ^ (0x01 << 3);
         }
         // Begin Duty Cycle for lights
         light_on = 1;

         while (ocount < REC_LEN)  // Loop until entire record is written
         {

         }
```c
    waitfor((rc = fat_Write(&file, optr, REC_LEN - ocount)) != -EBUSY);
    if (rc < 0)
    {
        printf("fat_Write: rc = %d\n", rc);
        while ((rc = fat_UnmountDevice(first_part->dev)) == -EBUSY);
        return rc;
    }
    optr += rc;     // Move output pointer
    ocount += rc;   // Add number of characters written

    filestate = 0;  // Show file is idle
}

// This is to quit recording
const state quit always_on
{
    // either after fifteen seconds
    if (turn_off == 150)
    {
        // flips switch to lights
        channel_bank = channel_bank ^ (0x01 << 3);
        CoPause(&read);
        CoPause(&time);

        waitfor (filestate == 0);  // Wait until file is available
        savedRC = 0;

        // Unmount all of the mounted FAT partitions & devices before exit
        for (i = 0; i < num_fat_devices * FAT_MAX_PARTITIONS; ++i)
        {
            if (fat_part_mounted[i])
            {
                while ((rc = fat_UnmountDevice(fat_part_mounted[i]->dev)) == -EBUSY);
                if (!savedRC && rc)
                {
                    savedRC = rc;
                }
            }
        }
    }
```
channel_bank = channel_bank ^ (0x01 << 4);

} // or keyhit during debuggin
if (kbhit())
{
    // Flip switch to lights
    channel_bank = channel_bank ^ (0x01 << 3);
    CoPause(&read);
    CoPause(&time);

    waitfor (filestate == 0); // Wait until file is available
    savedRC = 0;

    // Unmount all of the mounted FAT partitions & devices before exit
    for (i = 0; i < num_fat_devices * FAT_MAX_PARTITIONS; ++i)
    {
        if (fat_part_mounted[i])
        {
            while ((rc =
                    fat_UnmountDevice(fat_part_mounted[i]->dev)) ==
                    -EBUSY);
            if (!savedRC && rc)
            {
                savedRC = rc;
            }
        }
    }

    // Flips switch to turn off power
    channel_bank = channel_bank ^ (0x01 << 4);
}

////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
// LIGHTS //////////////////////////////////////////////////////////////////////////////////

costate light always_on
{
    waitfor (light_on==1);
    flip++;
    waitfor (DelayMs(period));
    channel_bank = channel_bank ^ (0x01 << 0);
// duty_cycle = -period*(voltage / full_power);

sprintf(p, "Voltage = %.3fV % 2d Duty Cycle = %.3f", voltage, flip, duty_cycle);
    DispStr(1, 16, p);
switch (relay)
{
    case 1:
        channel_bank = channel_bank ^ (0x01 << 1);
        waitfor(DelayMs(period-duty_cycle));
        channel_bank = channel_bank ^ (0x01 << 1);
        relay =2;

        break;
    case 2:
        channel_bank = channel_bank ^ (0x01 << 2);
        waitfor(DelayMs(period-duty_cycle));
        channel_bank = channel_bank ^ (0x01 << 2);
        relay =1;

        break;
}

} //end main
B.2 Multitasking Example

Dynamic C Multi – Tasking

while(1){
    costate{ ... }     // task 1
    costate{           // task 2
        waitfor( buttonpushed() );
        turnondevice1();
        waitfor( DelaySec(60L) );
        turnondevice2();
        waitfor( DelaySec(60L) );
        turnoffdevice1();
        turnoffdevice2();
    }
    costate{ ... }     // task n
}

C++ Multi – Tasking

task1state = 1;           // initialization:
while(1){
    switch(task1state){
    case 1:
        if( buttonpushed() ){
            task1state=2; turnondevice1();
            timer1 = time; // time incremented every second
        }
        break;
    case 2:
        if( (time-timer1) >= 60L){
            task1state=3; turnondevice2();
            timer2=time;
        }
        break;
    case 3:
        if( (time-timer2) >= 60L){
            task1state=1; turnoffdevice1();
            turnoffdevice2();
        }
        break;
    }
    /* other tasks or state machines */
}
APPENDIX C

TECHNICAL DETAILS OF COMMERCIAL COMPONENTS
C.1 Grade Crossing Warning Lamp

**FEATURES / BENEFITS**

- Proven high-flux LED traffic signal technology
- AC/DC operation from 9 to 16 volts full output (operation down to 6 volts)
- Optional side light indicators reflect main beam signal to ensure safety
- Deeply red or red/green/gold or amber lens from lens with no beam alignment necessary
- Ring terminations easily attach to AREMA posts
- Red tinted, UV stabilized, and haze coated lens for extended life
- Better visibility and faster response time than incandescent
- Flicker resistant
- Reduced power consumption
- Years of maintenance-free operation

**APPLICATION**

The state-of-the-art 433 series LED rail crossing signal was developed by Dialight to lower power requirements and maintenance costs while increasing safety, life, and reliability in rail crossing applications. These lamps were designed utilizing Dialight's proven high-flux LED traffic signal technology to provide superior light performance for use on both existing and new crossing light applications. Safer, faster on/off response time and brilliant monochromatic color makes these LED lamps more noticeable than incandescent lamps. The modular lamps install easily without special tools and operate in both AC and DC applications.

The light is unique from other LED crossing lights in several ways: (1) Diffused lens and uniform appearance are similar to incandescent, (2) the lamp maintains the specified light output down to 9 volts, but operates down to as low as 5 volts, and (3) safety is increased by side lights that monitor the main light beam by reflecting light via patented light pipe technology, to ensure train engineers that the light aspect facing the street is actually flashing, providing two less failure points compared to any other competitor product.

**OPERATING CONDITIONS**

- Operating Temperature: -40°C to +70°C
- Operating Voltage: 5 to 16 volts AC/DC
- DC operating current @ 10 Volts: 1 Amp DC
- AC operating current @ 10 Volts: 1 Amp RMS
- Power Factor: > 98%
- Chromaticity: x = 0.330 and y ≤ 0.330

*EMI compliance to FCC specifications only

Dialight Corporation
15012 River Dr South - Farmingdale, NY 11735 USA
(800) 245-8866 / (516) 293-7440
www.dialight.com
ELECTRICAL AND PHOTOMETRIC SPECIFICATIONS

6 LED 12" Crossing Light V vs I and V vs Light Intensity Curves

Isotropic Light Intensity (cd) °

MECHANICAL DIMENSIONS

ORDERING INFORMATION

<table>
<thead>
<tr>
<th>Part Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>433-1R11-R111-1R08</td>
<td>With Sidelight, Rec. 12&quot; diameter, 6-14 Volts AC/DC, 2½&quot; leads, Ring lug, flat gasket (front), ARBMA compliant</td>
</tr>
</tbody>
</table>

Dialight Corporation
1501 Route 34 South - Flemington, N.J. 08822 USA
Tel (1) 908-464-8116 - Fax (1) 908-606-0790 - www.dialight.com
C.2 First Generation Design Generator

![Graph of Generator Open and Closed Circuit Curves]

- **Short Circuit Amperage Curve**: Shows the relationship between amperage and shaft RPM.
- **Open Circuit Voltage Curve**: Demonstrates the voltage variation with respect to shaft RPM.

![Graph of Generator Power and Current Output Curves]

- **Power Data**:
  - Intermittent Duty
  - Continuous Duty
  - **13.8 VDC**
- **Amperage Data**:
  - Intermittent Duty
  - **36.5 VDC**

**Maximum Continuous Duty**: 1.5 Amps
10 Minute Maximum at 1.5 Amps.

- **Warning**: When operating the generator above the continuous duty range, the unit may become hot to the touch.
C.3 Second Generation Design Generator

![Graphs showing generator performance curves](image-url)
C.4 Charge Controller Wiring Configuration
C.5 Electromagnetic Relays

## MINIATURE PC BOARD RELAY

### FEATURES
- Subminiature size
- High sensitivity, 110 mW pickup
- Coils to 48 VDC
- Hermetically sealed version available
- Epoxy sealed for automatic wave soldering
- Contacts rated at 3, 6 or 10 Amps
- Withstands 5 kV IEEE Lightning Surge (special order)
- Class B insulation (130°C) standard
- Class F insulation (155°C) version available
- UL, CUL file E44211
- VDE approved versions available (Class A insulation only)

### CONTACTS

<table>
<thead>
<tr>
<th>Arrangement</th>
<th>SPDT (1 Form C)</th>
<th>SPST (1 Form A) 10 A version only</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ratings</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Light Duty</td>
<td>Resistive load</td>
<td>Max., switched current: 100 W or 800 VA</td>
</tr>
<tr>
<td>Medium Duty</td>
<td>Max., switched voltage: 150 VDC or 300 VAC</td>
<td></td>
</tr>
<tr>
<td>Heavy Duty</td>
<td>Max., switched current: 10 A</td>
<td>Max., switched voltage: 150 VDC or 300 VAC</td>
</tr>
<tr>
<td>Material</td>
<td>Light duty: Silver</td>
<td>Medium duty: Silver nickel</td>
</tr>
<tr>
<td></td>
<td>Heavy duty: Silver cadmium oxide</td>
<td></td>
</tr>
<tr>
<td>Resistance</td>
<td>&lt; 100 milliseconds initially</td>
<td></td>
</tr>
</tbody>
</table>

### COIL

<table>
<thead>
<tr>
<th>Power</th>
<th>At Pickup Voltage (typical)</th>
<th>Standard coil: 210 mW</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sensitive coil: 140 mW</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Max., continuous dissipation</td>
<td>Class B: 2.0 W (65°F) ambient</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.4 W (40°C (104°F) ambient</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Class F: 2.6 W (20°C (68°F) ambient</td>
</tr>
<tr>
<td></td>
<td>Temperature rise</td>
<td>2.1 W (40°C (104°F) ambient</td>
</tr>
<tr>
<td></td>
<td>Max., continuous dissipation</td>
<td>At nominal coil voltage</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Standard coil: 35°C (95°F)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sensitive coil: 28°C (82°F)</td>
</tr>
<tr>
<td>Temperature</td>
<td>Max., 150°C (302°F) Class A</td>
<td>Max., 130°C (266°F) Class B</td>
</tr>
<tr>
<td></td>
<td>Max., 150°C (302°F) Class B</td>
<td>Max., 130°C (266°F) Class F</td>
</tr>
</tbody>
</table>

### GENERAL DATA

<table>
<thead>
<tr>
<th>Life Expectancy</th>
<th>Minimum operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mechanical Light Duty</td>
<td>9 x 10^4 at 1 A, 100 VAC</td>
</tr>
<tr>
<td>Medium Duty</td>
<td>1.8 x 10^4 at 6 A, 100 VAC</td>
</tr>
<tr>
<td>Heavy Duty</td>
<td>1 x 10^4 at 10 A, 120 VAC</td>
</tr>
<tr>
<td>Operate Time (typical)</td>
<td>5 ms at nominal coil voltage</td>
</tr>
<tr>
<td>Release Time (typical)</td>
<td>2 ms at nominal coil voltage (with no coil preheat)</td>
</tr>
<tr>
<td>Dielectric Strength (at sea level for 1 min)</td>
<td>750 Vrms contact to contact</td>
</tr>
<tr>
<td>Insulation Resistance</td>
<td>1000 megohms min. at 20°C, 500 VDC, 90% RH</td>
</tr>
<tr>
<td>Dripout</td>
<td>Greater than 5% of nominal coil voltage</td>
</tr>
<tr>
<td>Ambient Temperature Operating</td>
<td>At nominal coil voltage</td>
</tr>
<tr>
<td>Storage</td>
<td>+55°C (131°F) to -25°C (13°F) Class B</td>
</tr>
<tr>
<td>Vibration</td>
<td>0.35g/20g at 10–55 Hz, 10 g at 55–110 Hz</td>
</tr>
<tr>
<td>Shock</td>
<td>10 g</td>
</tr>
<tr>
<td>Enclosure</td>
<td>P.C.T. polyester</td>
</tr>
<tr>
<td>Terminals</td>
<td>Tinned copper alloy, P.C.</td>
</tr>
<tr>
<td>Max. Solder Temp.</td>
<td>270°C (518°F)</td>
</tr>
<tr>
<td>Max. Solder Time</td>
<td>3 seconds</td>
</tr>
<tr>
<td>Max. Solvent Temp.</td>
<td>90°C (176°F)</td>
</tr>
<tr>
<td>Max. Immersion Time</td>
<td>30 seconds</td>
</tr>
<tr>
<td>Weight</td>
<td>6 grams</td>
</tr>
</tbody>
</table>

### NOTES

1. All values at 23°C (73°F).
2. Relay may pull in with less than 'Must Operate' value.
3. Other coil resistances and sensitivities available upon request.
4. Unsealed relays should not be dip dressed.
5. Specifications subject to change without notice.
## RELAY ORDERING DATA

### COIL SPECIFICATIONS

<table>
<thead>
<tr>
<th>NOMINAL VDC</th>
<th>MAX. VDC</th>
<th>RESISTANCE %</th>
<th>MOST OPERATE VDC</th>
<th>LIGHT DUTY (3 Amp contact)</th>
<th>MEDIUM DUTY (6 Amp contact)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Unsoldered</td>
<td>Epoxy Sealed</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Unsoldered</td>
<td>Epoxy Sealed</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Unsoldered</td>
<td>Epoxy Sealed</td>
</tr>
</tbody>
</table>

### SENSITIVE RELAYS: 1 Form C (SPDT)

<table>
<thead>
<tr>
<th>NOMINAL VDC</th>
<th>MAX. VDC</th>
<th>RESISTANCE %</th>
<th>MOST OPERATE VDC</th>
<th>LIGHT DUTY (3 Amp contact)</th>
<th>MEDIUM DUTY (6 Amp contact)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Unsoldered</td>
<td>Epoxy Sealed</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Unsoldered</td>
<td>Epoxy Sealed</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Unsoldered</td>
<td>Epoxy Sealed</td>
</tr>
</tbody>
</table>

### STANDARD RELAYS: 1 Form C (SPDT)

<table>
<thead>
<tr>
<th>NOMINAL VDC</th>
<th>MAX. VDC</th>
<th>RESISTANCE %</th>
<th>MOST OPERATE VDC</th>
<th>LIGHT DUTY (3 Amp contact)</th>
<th>MEDIUM DUTY (6 Amp contact)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Unsoldered</td>
<td>Epoxy Sealed</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Unsoldered</td>
<td>Epoxy Sealed</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Unsoldered</td>
<td>Epoxy Sealed</td>
</tr>
</tbody>
</table>

### HEAVY DUTY (10 Amp contact)

<table>
<thead>
<tr>
<th>NOMINAL VDC</th>
<th>MAX. VDC</th>
<th>RESISTANCE %</th>
<th>MOST OPERATE VDC</th>
<th>LIGHT DUTY (3 Amp contact)</th>
<th>MEDIUM DUTY (6 Amp contact)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Unsoldered</td>
<td>Epoxy Sealed</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Unsoldered</td>
<td>Epoxy Sealed</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Unsoldered</td>
<td>Epoxy Sealed</td>
</tr>
</tbody>
</table>

*The number after the dash indicates the number of contacts.*

### SENSITIVE RELAYS: 1 Form C (SPDT)

<table>
<thead>
<tr>
<th>NOMINAL VDC</th>
<th>MAX. VDC</th>
<th>RESISTANCE %</th>
<th>MOST OPERATE VDC</th>
<th>LIGHT DUTY (3 Amp contact)</th>
<th>MEDIUM DUTY (6 Amp contact)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Unsoldered</td>
<td>Epoxy Sealed</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Unsoldered</td>
<td>Epoxy Sealed</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Unsoldered</td>
<td>Epoxy Sealed</td>
</tr>
</tbody>
</table>

### STANDARD RELAYS: 1 Form C (SPDT)

<table>
<thead>
<tr>
<th>NOMINAL VDC</th>
<th>MAX. VDC</th>
<th>RESISTANCE %</th>
<th>MOST OPERATE VDC</th>
<th>LIGHT DUTY (3 Amp contact)</th>
<th>MEDIUM DUTY (6 Amp contact)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Unsoldered</td>
<td>Epoxy Sealed</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Unsoldered</td>
<td>Epoxy Sealed</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Unsoldered</td>
<td>Epoxy Sealed</td>
</tr>
</tbody>
</table>

*The number after the dash indicates the number of contacts.*

### RELAY ORDERING DATA - VDE APPROVED VERSIONS

<table>
<thead>
<tr>
<th>NOMINAL VDC</th>
<th>MAX. VDC</th>
<th>RESISTANCE %</th>
<th>MOST OPERATE VDC</th>
<th>LIGHT DUTY (6 Amp contact)</th>
<th>MEDIUM DUTY (6 Amp contact)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Unsoldered</td>
<td>Epoxy Sealed</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Unsoldered</td>
<td>Epoxy Sealed</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Unsoldered</td>
<td>Epoxy Sealed</td>
</tr>
</tbody>
</table>

### SENSITIVE RELAYS: 1 Form C (SPDT)

<table>
<thead>
<tr>
<th>NOMINAL VDC</th>
<th>MAX. VDC</th>
<th>RESISTANCE %</th>
<th>MOST OPERATE VDC</th>
<th>LIGHT DUTY (6 Amp contact)</th>
<th>MEDIUM DUTY (6 Amp contact)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Unsoldered</td>
<td>Epoxy Sealed</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Unsoldered</td>
<td>Epoxy Sealed</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Unsoldered</td>
<td>Epoxy Sealed</td>
</tr>
</tbody>
</table>

*The number after the dash indicates the number of contacts.*

---

**American Zettler, Inc.**

76 Columbia, Aliso Viejo, CA 92655 - Phone: (949) 634-5400 - Fax: (949) 634-5442 - Email: sales@azettler.com

11/15/05W
UL CIR RATINGS

Light Duty
- 9 A at 28 VDC or 360 VAC
- 1/4 HP at 120 VAC
- 1/4 HP at 120/240 VAC (100,000 cycles)
- NEMA A at 120/240 VAC, Medium Duty
- NEMA A at 120/240 VAC General Use, 100,000 cycles

Medium Duty
- 10 A at 24 VDC / 120 VAC Resistor
- 1/4 HP at 120 / 125 VAC
- 1/2 HP at 230 VAC
- 1 A at 272 VAC Resistor

Heavy Duty
- 10 A at 24 VDC / 120 VAC Resistor
- NEMA A at 120/240 VAC General Use
- 1/2 HP at 272 VAC Resistor
- 1 A at 272 VAC Resistor

VDE RATINGS

NEMA
- 5 A at 250 VAC resistive, 10,000 cycles

US NEMA
- 5 A at 250 VAC resistive, 20,000 cycles

MECHANICAL DATA

Coil Temperature Rise

Maximum Switching Capacity

AMERICAN ZETTLER, INC. www.azetter.com
76 COLUMBIA - ALISO VIEJO, CA 92656 - PHONE: (949) 851-4690 - FAX: (949) 851-4642 - E-MAIL: SALES@AZETTLER.COM