ME 493 Final Design Report

Robot Treadmill

Sponsor/Client: Dr. Hunt

Dominic Breiling, Ruben Contreras, Tiago Denczuk, Yura Kutsurenko, Isaac Miller

Due Date: 6/11/18

Table of Contents

Team Member Roles/Contributions

- Dominic Breiling
    - Treadmill Assembly
    - Documentation
    - Networking Tasks

- Ruben Contreras
    - Project Website
    - Circuit Diagram and wiring
    - Treadmill Assembly

- Tiago Denczuk
    - Matlab Interface
    - Arduino Control Code

- Yura Kutsurenko
    - Treadmill Design in CAD
    - Design Pulley System
    - Treadmill Assembly

- Isaac Miller
    - Part Machining
    - Motor Control Circuit Analysis/Testing
    - Treadmill Assembly
    - Source Necessary Parts
    - Weekly Meetings Leader

Executive Summary

Project Objective Statement

Dr. Hunt would like a computer controlled treadmill for use in testing quadruped robots modeled after dogs. The treadmill must be easily controlled using a desktop computer and must be able to achieve a range of speeds and belt tensions. The treadmill must also fit his specific needs in terms of minimum size, power requirements and reliability.

Final Status of Design Project

The key requirements from our customer were that the treadmill had to be controlled from a computer, be able to achieve speed of ~ 13 mph, and be table height. We went through multiple design concepts before settling on our current design. We thought about making a carouselle style treadmill, however we scrapped that idea due to multiple moving parts which can lead to reduced reliability. We settled on a regular belt and roller treadmill because parts were easy to get and it was simpler to build. To control the treadmill, we are using an Arduino DUE that is connected to a computer, which has a Matlab interface to input speed profiles into the treadmill. To achieve the table height requirement we used 80/20 extruded aluminum to build the right size frame.

The main systems in the treadmill are the frame, powertrain, and controller. The frame consists of 80/20 extruded aluminum, fasteners, a subframe for the motor, and machined plates that we made to hold the rollers. The powertrain consists of a 100V DC motor, a motor controller and a pulley system. The controller system has an Arduino DUE, an opto isolator circuit, as well as the Matlab code.

Key Performance Metrics

We have achieved all requirements our sponsor gave us. 80/20 extruded aluminum let us build the frame which has the exact dimensions we wanted. It is also easy to work with and attach subsystems to.

To meet the speed criteria, we had to create a new pulley system to achieve the desired speed, which is higher than a regular treadmill. First, we calculated the pulley sizes we needed. Then, we designed a pulley in SolidWorks and 3D printed it with a filament, which has Kevlar fibers in it to make the pulley very durable and strong.

The last important requirement was to make it possible to control the treadmill from a computer. To do that, we got a motor controller and connected it to an Arduino DUE, which sends it a PWM signal. The Arduino receives the signal from Matlab into which needed speed or speed profiles are entered.

Client Requirements

Requirements Written in Voice of the Customer

- The treadmill should be about table height
    - Easy to work around
- The treadmill should be sturdy
    - Can support the weight of the robot dog
    - Is a semi-permanent installation
- Should have good control of speed
    - Can accept command speeds through computer
- Should be able to display the actual speed of the treadmill
- Should have the ability to easily change the backboard of running surface
- Should be easy to start up
- Should have the ability to connect with a computer through a serial port

Table 1: Importance Values for Primary Requirements

Importance	**Primary Need**	**Secondary Need**
6	Safe	● Must have emergency shut-offs ● Must feel sturdy and be cable of supporting robot weight ● Belt must be stable and have minimal vibration when running ● Keep operators away from all moving parts
3	Economical	● Must have a maximum fabrication cost of $500
9	Reliable	● Must perform reliably for 5 years without maintenance ○ Bearings can be a source of wear ○ Roller surfaces might need to be replaced ● The final microcontroller should be free of memory leaks
3	Communications	● Must be able to communicate with a computer via serial connection for control and data acquisition ● The microcontroller must be able to accept a speed parameter ● The microcontroller must send the actual speed of the treadmill, the desired speed of the treadmill and the clock time to the computer
3	Speed control	● Must be capable of belt speeds from 0 to 15 mph ● Musl be capable of a linear speed ramp up as well as maintaining a constant speed

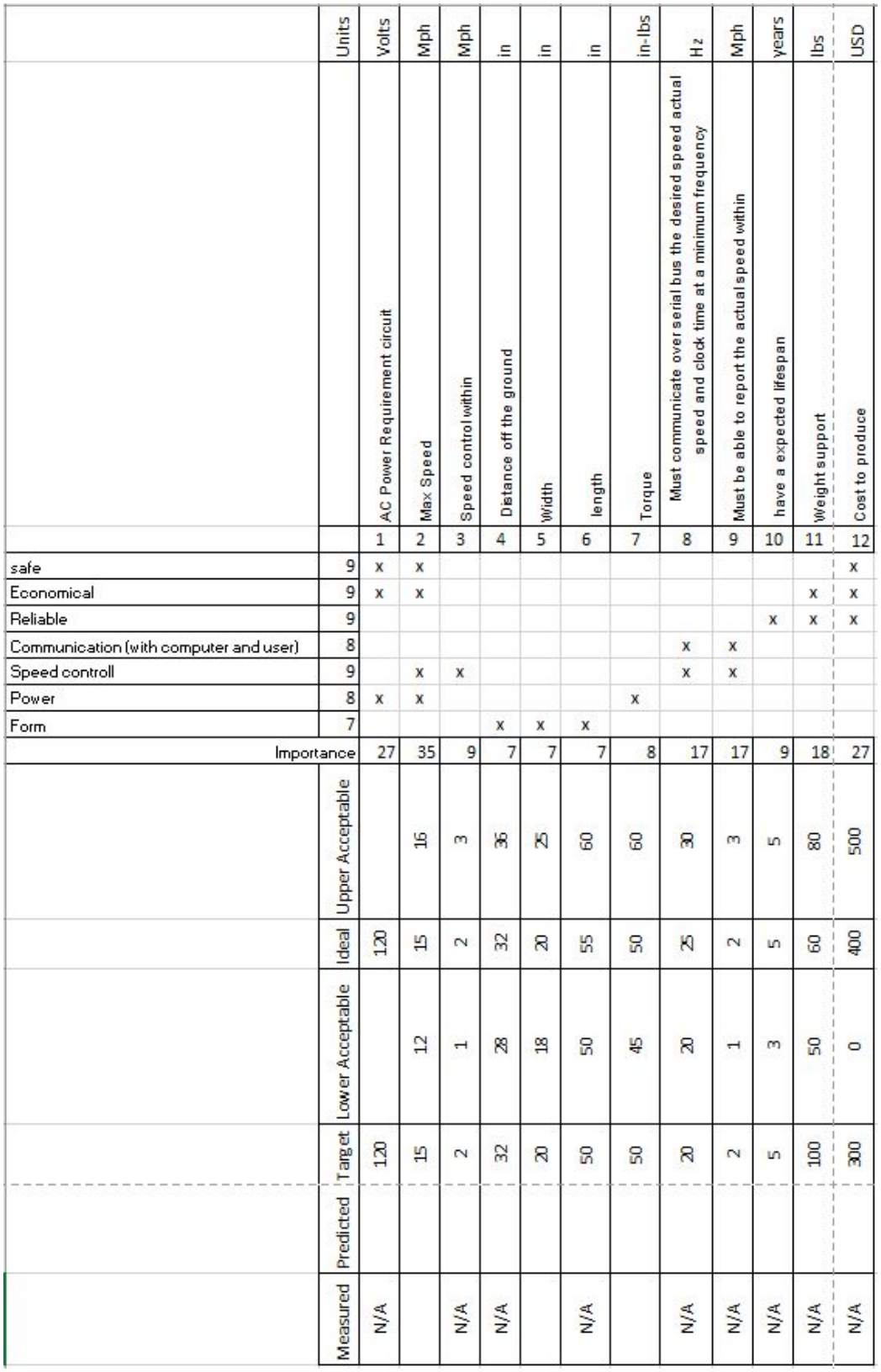
6	Power	● 120 VAC
6	Form	● Must accommodate puppy robot ● The belt surface must be at table height to allow for ergonomic use

Quantitative Evaluation:

Table 2. Important performance metrics

Client Need	**Performance Metric**
Safe	● On/Off switch ● All moving parts are covered
Economical	● Fabrication must be less than 500 $
Reliable	● Life span of 5 years ● Replaceable rollers ● Replaceable motor
Communications	● Arduino controlled ○ 2Hz or greater ○ At least two channels ● Matlab interface ● Serial port
Speed control	● Speed of ~ 13 mph ● Accurate reading of ∓2 mph ● Ability to input a speed profile
Power	● 120 V ● Minimum 40 in-lb torque
Form	● Frame 55 in x 20 in x 12 in

Performance Matrix:



		Units	Volts	Mph	Mph	in	in	in	in-lbs	Hz	Mph	years	lbs	USD
			AC Power Requirement circuit	Max Speed	Speed control within	Distance off the ground	Width	length	Torque	Must communicate over serial bus the desired speed actual speed and clock time at a minimum frequency	Must be able to report the actual speed within	have a expected lifespan	Weight support	Cost to produce
			1	2	3	4	5	6	7	8	9	10	11	12
safe		9	x	x										x
Economical		9	x	x									x	x
Reliable		9										x	x	x
Communication (with computer and user)		8								x	x			
Speed controll		9		x	x					x	x			
Power		8	x	x					x					
Form		7				x	x	x						
Importance			27	35	9	7	7	7	8	17	17	9	18	27
	Upper Acceptable			16	3	36	25	60	60	30	3	5	80	500
	Ideal		120	15	2	32	20	55	50	25	2	5	60	400
	Lower Acceptable			12	1	28	18	50	45	20	1	3	50	0
	Target		120	15	2	32	20	50	50	20	2	5	100	300
	Predicted													
	Measured		N/A		N/A	N/A		N/A		N/A	N/A	N/A	N/A	N/A

Figure 1: Performance matrix of client needs

Conceptual Design Summary

The figure below shows the necessary systems that make up the treadmill. Starting at the top, there are three main functions of the treadmill, a computer interface, speed control, and variable surface. Moving down to the next level of each system. The computer interface sends inputs to the system controller, which communicates with the motor controller and controls speed. On the final level, the speed sensor sends data to the system controller, while the DC motor is controlled by the motor controller. On the right there are two ways to generate a variable belt surface, changing the belt tension and a removable support for the belt.

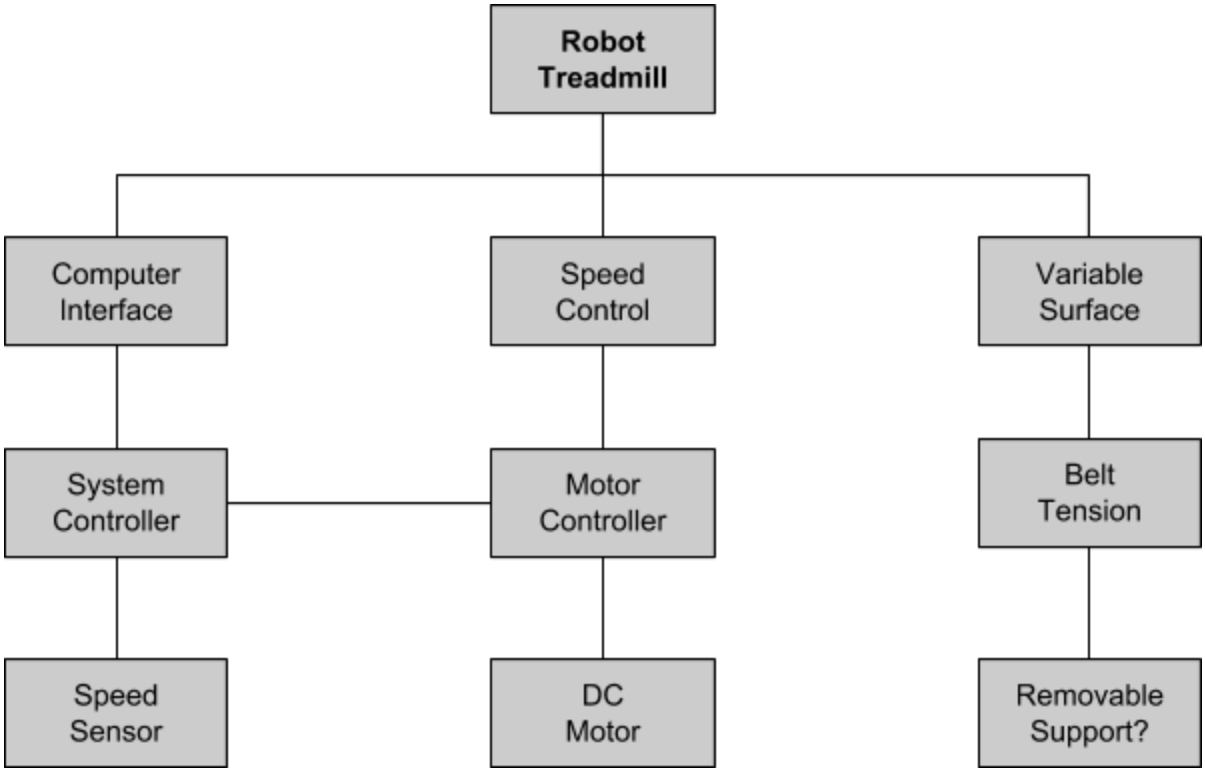


Figure 2: Robot treadmill systems

Subsystem Highlights

The treadmill has a custom made frame, as well as custom machined belt roller holders. The frame supports the DC motor which rotates the rollers through a custom made pulley system. The pulley system translates the rotational motion into linear motion of the belt. The motor is powered by a DC motor controller. The speed is measured at the DC motor's rotor with a rotary encoder. Arduino DUE gets the input from the rotary encoder and converts it to linear speed. That same Arduino DUE is connected to the DC motor controller through an opto isolated circuit. Arduino DUE sends the PWM signal to the DC motor controller, then the motor controller sends higher voltage PWM to the DC motor.

Frame

The main frame is made out of 80/20 15 - series extruded aluminum. Dr. Hunt had some extruded aluminum available from previous projects, which we were able to use. However, we had to buy more due to the large size of the treadmill frame. We cut the 80/20 in the PSU machine shop, we also filed the edges to appear smooth to the touch and not cut anyone. To fasten the 80/20 together we used the special 80/20 bolts and anchors that lodge into the 80/20 slots. We used 4 - bolt L brackets, instead of the more common 2 bolt ones because rigidity was an important part of our project criteria.

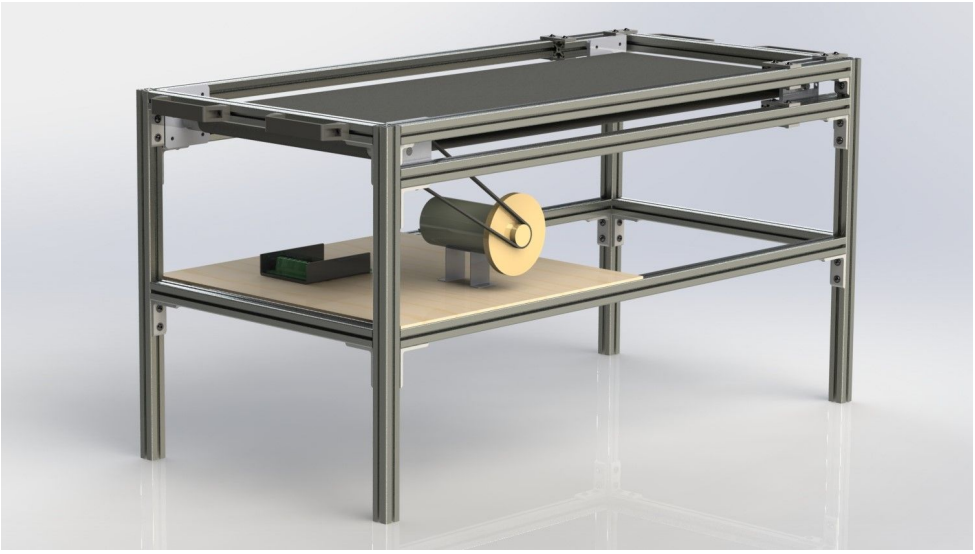


Figure 3: Final CAD design render

The powertrain and the microcontroller sits inside of the treadmill frame on a piece of plywood that is fastened to the frame. The DC motor is attached to the plywood with the subframe it was installed on from the free treadmill we got, not having to manufacture the subframe saved us time and money. The final design can be seen in figure 3.

We had to manufacture new roller holder plates in order for the rollers we salvaged from a treadmill to work, which can be seen in figure 4. First, we designed them in CAD so that we know the exact dimensions we need, then we machined them in the PSU machine shop out of aluminum. Each plate is attached to two different 80/20 pieces with four bolts. The rear roller

plates have a hole where a set screw sits, which allows to change tension in the treadmill's belt. This is also used to align the rollers. The rear roller on the treadmill is shorter than the front roller. To accommodate the difference in lengths of rollers, we fastened a short piece of 80/20 to the rear end of the frame. That extra piece extended the plates inward which allowed for correct fitment of rollers.

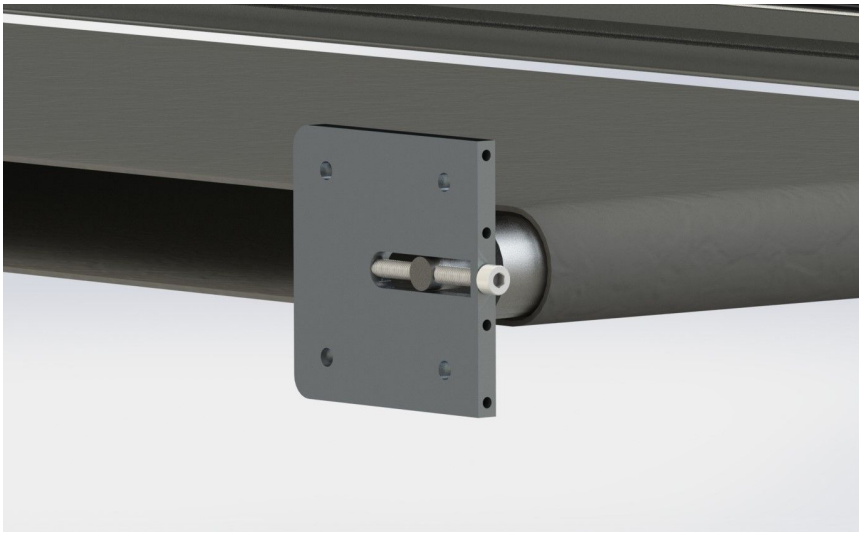


Figure 4: Aluminum belt roller plates with adjustment bolt

Powertrain

The powertrain system consists of a 95 VDC motor, a pulley system, a motor controller and an opto isolator circuit (figure 5) . We were able to salvage the motor from a free treadmill we got. The motor is rated to 95 VDC 21.4 A and achieves a max speed of 5000 RPM.

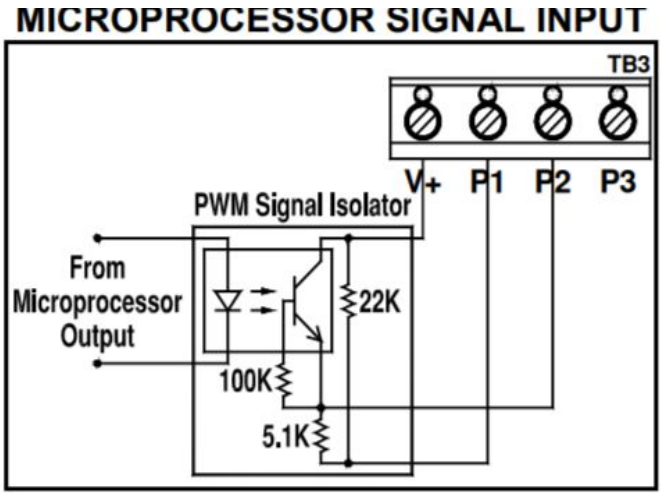


Figure 5: Opto isolator circuit

We wanted to use the pulley system that was in the treadmill we got, however after some testing we found out that the max speed was only about 10 mph. To fix that issue we had to design a new, smaller pulley. We took measurements of the existing pulley, such as the inner and outer

diameter, the width, size of the belt grooves, and scaled it down to what we calculated would work best to achieve our desired speed. We used the pulley formula (1)

$$d_1 n_1 = d_2 n_2 \qquad (1)$$

to calculate the size of the pulley we would need to achieve the speed of ~13 mph. Where $d_1 n_1$ is the diameter and the rotational speed of the first pulley, and $d_2 n_2$ is the diameter and the rotational speed of the second pulley. We found out that our pulley had to be almost the same diameter as the roller (2in.) in order to achieve that speed. We used a 3D printer that Dr. Hunt has in his lab to print the pulley. Kevlar filament was used to 3D Print the pulley because we wanted the pulley to be as strong as possible. Even though the existing pulley was measured with a set of calipers, the 3D printed pulley (figure 6) was accurate enough to achieve the desired press fit on the roller.

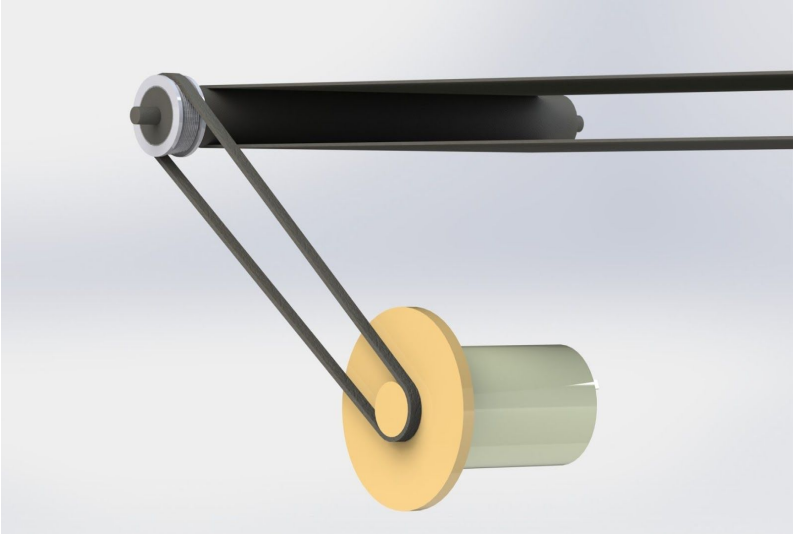


Figure 6: 3D printed pulley, motor and belt

Motor Controller

The motor is powered by a high voltage DC motor controller KBWT-112 (figure 7), which receives PWM signal from Arduino and then sends 100V PWM to the motor. We wanted to use the motor controller that was in the treadmill we got for free. However, it was difficult to work with because it had very little documentation, was not user friendly and was meant for only one application. We burnt multiple motor controllers we were able to get for free from Craigslist. After that, we gave up and decided to purchase a user friendly motor controller that had a lot of supporting documentation on how to use it. The motor controller we purchased had two options of controlling the output signal to the motor. One of the options was to use a rotary potentiometer. We used that option at first while testing the DC motor. The second option was to send a PWM signal to the motor controller. The signal was sent to the motor controller from an Arduino.

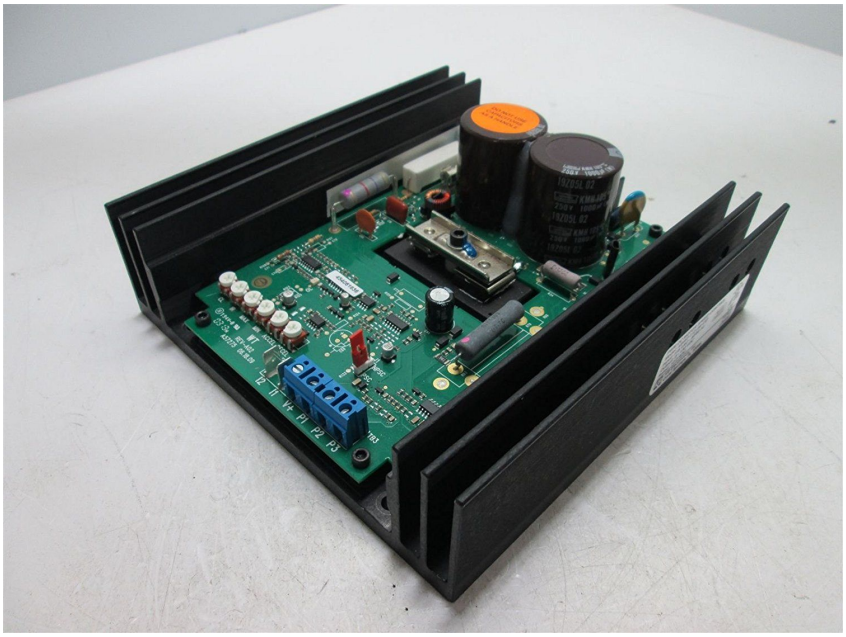


Figure 7: KBWT-112 DC motor controller

Control System

The heart of our treadmill is its control system. The control system allows for easy control of the speed profile using a matlab app which communicates over serial to the arduino microcontroller. The figure below (figure 8) is a flow chart which shows the data flow with in the treadmill control program and how data is passed between the matlab graphical user interface and the backend arduino programming.

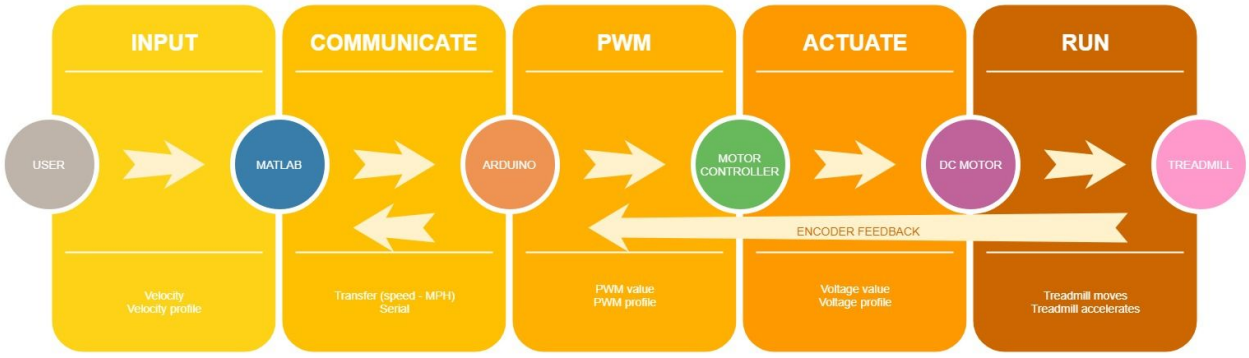


Figure 8: Control Flow Diagram

Performance Summary

Frame

Size and rigidity was an important criteria in our project. We chose to make the frame out of commonly used and easily available 80/20 extruded aluminum. Our team, as well as our mentor, had concerns about the rigidity of the frame and vibrations produced due to the large motor and many rotating parts. We designed and analyzed the frame before building to make sure that we

could construct a rigid frame. We tested multiple motors that we were able to acquire for free to see which one was the quietest and runs smoothest. We acquired belt rollers with high quality bearing, which helps with low noise and smooth operation.

We used the motor subframe from the treadmill that we acquired for free to attach to the main frame that we made out of 80/20. That subframe has a motor position adjustment which allows for easy motor replacement as well as the pulley belt tensioning. Before the final assembly, we ran the motor up to the max speed of 5000 RPM on the frame and observed if there would be any vibrations or impulses that would upset the frame. We concluded that there were none.

Powertrain

After accidentally burning multiple motor controllers, we decided to acquire a more common and easy to work with controller. The controller that we got is powered by 120VAC, so no need for a complex power supply or a converter. The motor can be controlled by either a rotary potentiometer or PWM signal from an arduino. The controller has good documentation which allows for easy setup and troubleshooting.

We designed and made the pulley system for our treadmill to achieve higher than normal treadmill speeds. We calculated and designed the right size pulley in SolidWorks and then 3D printed it with a special Kevlar filled filament, which made a very high strength part.

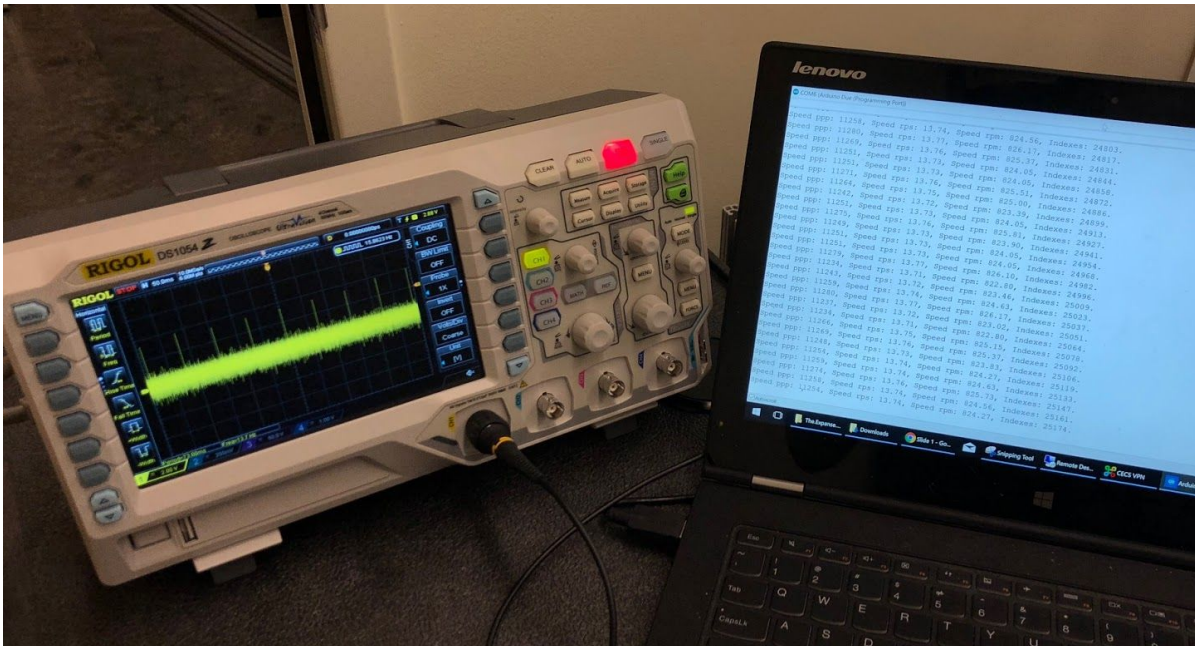
Final Status

Ultimately we brought the project to a close with the treadmill working in good order. We can input a speed profile and the treadmill does exactly what it is supposed to. The treadmill slowly ramps up to the desired speed, controls the speed with a PID control loop and outputs the data from the run to a user friendly CSV file for further analysis.

We did however leave a few tasks undone which could be further pursued by another group. The treadmill has a upcycled treadmill motor which is designed to be used in a low speed treadmill. In order to increase the max speed we geared the motor up. While this achieved our speed goal it robbed the system of low speed torque which in turn created a nonlinear rotational speed vs voltage input curve for the motor. This impedes control because the lower speed and higher speeds should have different PID tunings. This type of control band solution, if implemented would likely help increase control stability throughout the speed range. An additional way to address this issue would be through the use of a motor with a steeper torque curve at low RPM.

Appendices

A. Supplemental Images



Testing the rotary encoder on Arduino and Oscilloscope

B. Catalog of Design Artifacts

a. CAD Design Assembly and Parts
b. Arduino and Matlab code

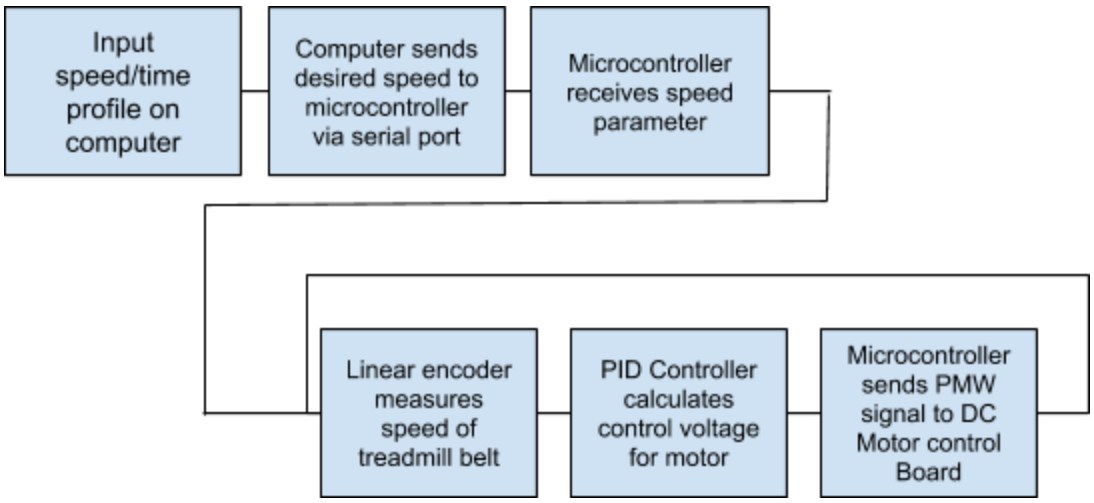
C. Concept Analysis

Project Objective Statement

Dr. Hunt would like a computer controlled treadmill for use in testing quadruped robots modeled after dogs. The treadmill must be easily controlled using a desktop computer and must be able to achieve a range of speeds and belt tensions. The treadmill must also fit his specific needs in terms of minimum size, power requirements and reliability.

Overview and Diagram Description

The treadmill is broken down into three main requirements speed control, variable surface and physical properties. The physical properties of the treadmill are the least restrictive requirements and really depend upon the size and implementation of the mechanical systems. The variable surface requirement is also open ended allowing us to implement any method to tension the belt and remove the support bed from under the belt. Speed control is the driving requirement for the project with specific details on how it will be obtained.


Input speed/time profile on computer
Computer sends desired speed to microcontroller via serial port
Microcontroller receives speed parameter
Linear encoder measures speed of treadmill belt
PID Controller calculates control voltage for motor
Microcontroller sends PMW signal to DC Motor control Board

Powertrain System

The powertrain system is made up of four items Speed input, microcontroller, DC motor and speed sensor. For this document we will focus on the DC Motor and how to measure speed of the treadmill.

Powertrain System Concept Analysis

In order to settle upon the final design for our treadmills drive and speed control system completed both internal and external searches. First we began with an internal search which included brainstorming ways to create the desired effect. Next we performed external searches which included reading published papers, looking in a few robotics books, and disassembling two treadmills to see how their designers solved the problem. Also a Lego scaled prototype was built to have a simple model to reference during our concept design stage seen in appendix 2. From there we sought to integrate solutions from the whole process into our final design. In order to finalize our design we considered permutations of the following items and selected the most pragmatic options that would fulfill our requirements.

1. Motor type
    - AC motor
    - DC motor
2. Power transmission
    - Direct drive: dive wheel coupled directly to the motor
    - Belt drive: drive wheel powered by a tensioned belt
    - Gear drive: drive wheel powered by geared transmission box
3. Gear ratio
    - 1:1
    - 1:2
    - 1:3
    - etc.

Our final design utilized a high torque DC motor, gear ratio slightly higher than 1 to 3 , and a belt drive system to drive the walking surface. We chose a DC motor because they are more readily available and the motor controllers are less expensive. We chose a belt driven system because belts we required a gear ratio to meet our speed goals and machining belt pulley is within our ability in the shop. Table 1 shown below are set of configurations that must be met in order to satisfy the 15 mph requirement. The table's purpose is to specify the minimum requirements for any given motor/gear ratio. For example, if we wanted a gear ratio of 1:4 we would need to make sure that the motor can run at at least 3151 rpm and also have 1.5 HP.

DC Motor

Design Configurations			
Transmission Ratio	**Power (HP)**	**Torque (in-lb)**	**RPM**
1	1	100.02	630
2	1.5	50.01	1260
3	2	66.7	1890
4	2.5	62.513	2520
5	3	100.02	3151

Table 1: Different design configurations needed for motor. Calculations for table values in appendix 1

Measuring Speed

A few ideas were considered for measuring the running speed of the treadmill belt, including:

- Linear encoder that is mounted onto the side of the treadmill belt. The basic idea would resemble what printers have inside of them to locate the printing head position.
- Magnetic sensor (hall effect sensor) that is mounted onto the gear that moves the rollers. Treadmills use hall sensor to measure speed, so it would be relatively easy to implement it into our system.
- Wheel mounted reflective optical encoder. An easy to build shaft mounted encoder that involves a white disk with black stripes. An infrared LED illuminates the rotating disk and the white stripes reflect into a phototransistor, whereas the black stripes don't. The phototransistor then produces a corresponding train of output pulses.
- Light sensor that is pointed to a treadmill's belt and white marks painted on the belt at a defined spacing. The light sensor would count the marks and convert the number to speed.

The quality criteria to select which concept to use in the design were: inexpensive, reliable, durable, readily available, ease of implementation, and most importantly, accurate. A phug chart was designed to help with the decision process, and the belt reflective encoder ended up ranking the highest. It was also the preferred choice, since it combined many of the characteristics of the other concepts in one. Another side benefit of this choice is that it will allow the team to experience building a custom encoder.

		Speed Measurement Concepts			
Selection Criteria	**Weight**	**Linear Encoder**	**Hall Effect Sensor**	**Wheel Reflective Encoder**	**Belt Reflective Encoder**
Economic	1	1	1	1	1
Reliability	3	1	1	0	1
Durability	2	1	0	0	1
Availability	1	1	1	1	1
Accuracy	3	1	0	-1	1
Ease to implement	2	0	0	1	1
Weighted sum		10	5	1	12
Rank		2	3	4	1

Table 2: Phug chart for speed control measurement

For example prototype design ideas, please refer to appendix 2.a

Conclusion

Q: How does the analysis presented in this report fit into the entire design project?

The subsystem we covered in this report is the most important part of our project. Our sponsor needs to test dynamic properties of quadrupedal robots, a robust drive train is critical for that application.

Q: How certain are you that the subsystem concept you have selected will be feasible and successful in achieving requirements for the final design?

After performing an in depth research of how other universities solve the issue of testing quadrupedal robots, it was clear that the best way would be to mimic a regular treadmill. We believe going with a tested solution would be the least risky way.

Q: What additional work needs to be done to increase your certainty of success with this subsystem?

We would need to perform another in depth analysis to choose a suitable controller to make sure our subsystems work well together and perform robustly.

Q: What is your back-up plan if this subsystem does not perform well enough, or is not feasible?

We are confident that the subsystem we research is going to work. If our original plan fails we would have to research other concept ideas like a carousel style treadmill or moving walkway style treadmill.

Q: What additional big risks lie ahead? (Those risks may involve other subsystems.)

Making all subsystems work together is going to be a risk. Also, our budget is very limited so we are planning on using used parts and taking apart old treadmills. We might encounter parts that are in the end of their life and that might fail within the near future.

Q: What additional resources or guidance do you need?

We don't think we need any more guidance or help other than our sponsor as well as our mentor.

Motor Calulations

$Max_{speed} := 15$ mph $Hp_1 := 1$ Power in HP

$n1 := 1$ transmition ratio

$R := 4$ radius of spindel (in)

$r := R \cdot 0.0254 = 0.102$ m

$$Vel := \frac{Max_{speed} \cdot 1609}{60^2} = 6.704 \text{ m/sec}$$

$$w_1 := \frac{Vel \cdot n1}{r} = 65.986$$

$$RPM_1 := \frac{w_1 \cdot 60}{2 \cdot \pi} = 630.119 \qquad Torque_1 := \frac{Hp_1 \cdot 63025}{RPM_1} = 100.021 \text{ in-lb}$$

$n2 := 2$ transmition ratio $Hp_2 := 1.5$ Power in HP

$$w_2 := \frac{Vel \cdot n2}{r} = 131.972$$

$$RPM_2 := \frac{w_2 \cdot 60}{2 \cdot \pi} = 1.26 \times 10^3 \qquad Torque_2 := \frac{Hp_2 \cdot 63025}{RPM_2} = 75.016 \text{ in-lb}$$

$n3 := 3$ transmition ratio $Hp_3 := 2$ Power in HP

$$w_3 := \frac{Vel \cdot n3}{r} = 197.958$$

$$RPM_3 := \frac{w_3 \cdot 60}{2 \cdot \pi} = 1.89 \times 10^3 \qquad Torque_3 := \frac{Hp_3 \cdot 63025}{RPM_3} = 66.681 \text{ in-lb}$$

$n4 := 4$ transmition ratio $Hp_4 := 2.5$ Power in HP

$$w_4 := \frac{Vel \cdot n4}{r} = 263.944$$

$$RPM_4 := \frac{w_4 \cdot 60}{2 \cdot \pi} = 2.52 \times 10^3 \qquad Torque_4 := \frac{Hp_4 \cdot 63025}{RPM_4} = 62.513 \text{ in-lb}$$

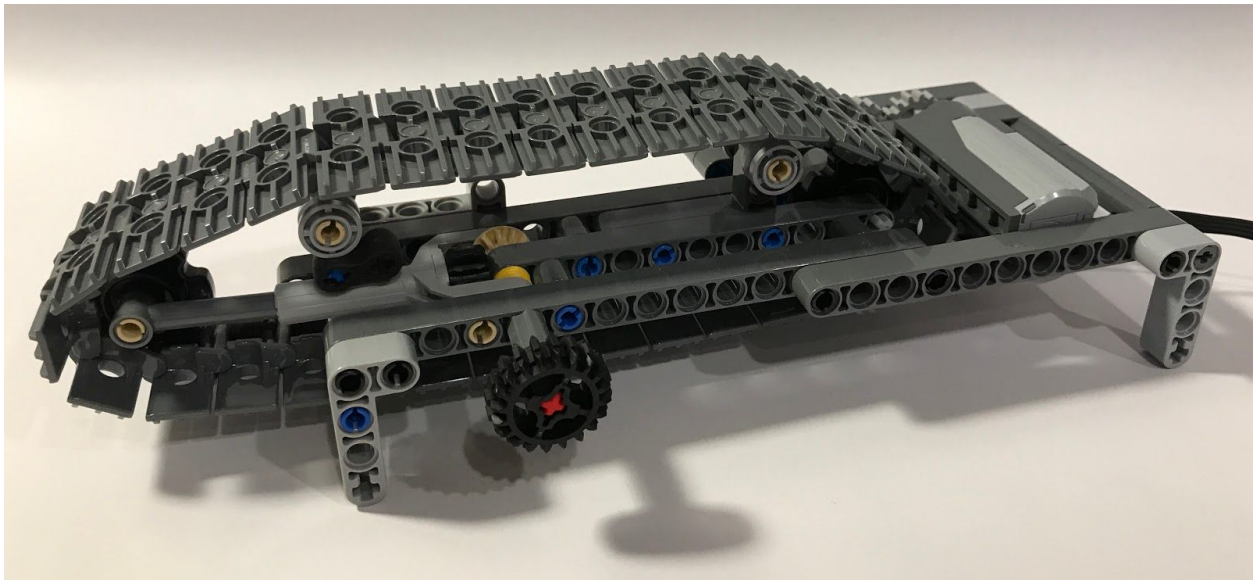
$n5 := 5$ transmition ratio $Hp_5 := 5$ Power in HP

$$w_5 := \frac{Vel \cdot n5}{r} = 329.929 \text{ rad/sec}$$

$$RPM_5 := \frac{w_5 \cdot 60}{2 \cdot \pi} = 3.151 \times 10^3 \qquad Torque_5 := \frac{Hp_5 \cdot 63025}{RPM_5} = 100.021 \text{ in-lb}$$

Lego Scale Prototype Model

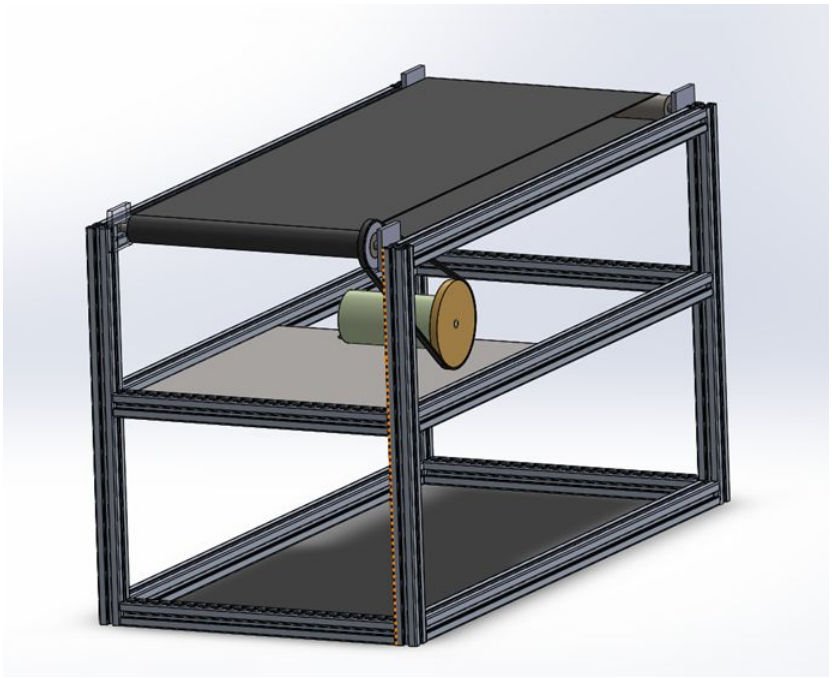
Below is a picture of a Lego prototype treadmill. This prototype was developed to have a simple mechanical model to use as reference while developing the final treadmill. The prototype has a screw mechanism to adjust belt tension and an electric motor to drive the belt.



Bibliography

Clark, Dennis, and Michael Owings. *Building Robot Drive Trains*. McGraw-Hill, 2003.

D. Frame Design Concepts



D. System Level Requirements Matrix
E. Computer Interface:
F. System Controller:
G. Speed Sensor:
H. Motor Controller
I. DC Motor
J. Belt Tension

E. Computer Interface

MATLAB Code (Treadmillv03.mlapp):

The first part of the code is generated and defines all the properties. MATLAB App Designer creates this part of the code based on the elements added in the graphic interface. It is not directly editable in App Editor.

```
classdef Treadmillv03 < matlab.apps.AppBase
        % Properties that correspond to app components
        properties (Access = public)
        RobotTreadmillUIFigure      matlab.ui.Figure
        Plot                    matlab.ui.control.UIAxes
        ConnectionSetupPanel        matlab.ui.container.Panel
        ConnectButton               matlab.ui.control.Button
        DisconnectButton            matlab.ui.control.Button
        SerialPortEditFieldLabel  matlab.ui.control.Label
        SerialPortField     matlab.ui.control.EditField
        Port                matlab.ui.control.Label
        PortUnav            matlab.ui.control.Label
        ControlTypePanel            matlab.ui.container.Panel
        Label               matlab.ui.control.Label
    InputType               matlab.ui.control.Switch
        ManualControlPanel          matlab.ui.container.Panel
        SpeedSliderLabel            matlab.ui.control.Label
        SpeedSlider         matlab.ui.control.Slider
        MPH               matlab.ui.control.Label
        InputSpeed          matlab.ui.control.NumericEditField
        RunPanel            matlab.ui.container.Panel
        RunButton           matlab.ui.control.Button
        StopButton            matlab.ui.control.Button
        SpeedProfilePanel           matlab.ui.container.Panel
        ChooseFile          matlab.ui.control.Button
        FileName            matlab.ui.control.EditField
        LoadProfile         matlab.ui.control.Button
        ErrorPanel          matlab.ui.container.Panel
        CloseButton         matlab.ui.control.Button
        WrongFile           matlab.ui.control.Label
        HidePlot            matlab.ui.container.Panel
        end
```

Next, a function is called to execute the created components. Upon execution of the app, and before any action is taken by the user the startupFcn(app) function makes sure that no serial port is open (instrreset). Additionally, this function determines the initial outlook of the app, by turning off the visibility of the panels, tabs and buttons that will be called on as the user interfaces with the app.

```
        methods (Access = private)
        % Code that executes after component creation
          function startupFcn(app)
        instrreset  % Disconnect and delete all instrument objects

        % Visible Tabs
```

```
app.PortUnav.Visible = 'off';
app.ControlTypePanel.Visible = 'off';
app.ManualControlPanel.Visible = 'off';
app.RunPanel.Visible = 'off';
app.SpeedProfilePanel.Visible = 'off';

global stp;
stp = 1;

end
```

The following functions are callbacks to user interactions with the app. Pushing a button, moving a slider, opening a file are some of the examples of actions that will call certain functions in the code. These functions appear in the code in the order they were written, since this is the standard way that App Editor generates them. There is no relationship between the order that a function will be called and the order it appears in the code.

The first callback functions to appear in the code is the RunButton. When the Run button is pushed, this function will first initialize a file to export the data that will eventually be serially imported from Arduino.

```
% Button pushed function: RunButton
function RunButtonPushed(app, event)
app.RunButton.Enable = 'off';
global fileExport;
fileExport = fopen(['expspeed_',datestr(now,'mm_dd_yyyy_HH_MM'),'.csv'],'w');
global s;
global stp;
flushinput(s);
```

Next, the function looks into which sort of speed input the user will choose (Manual Control or Speed Profile), and how it will send the information to the microcontroller. The first character '<' will signal the Arduino program that something meaningful is coming through serial. The next digit defines what kind of command is being sent. When "Run" is pushed, "2" means that the control will be manual, and "2" means that the control will be a speed profile. Commas will separate the digits and '>' will signal the end of that transmission.
In manual control, the second digit will represent the desired speed, in miles per hour (MPH).

```
switch app.InputType.Value
case 'Manual Control'
        app.SpeedSlider.Enable = 'on';
        fwrite(s,'<1,0>');
        pause(1);
    table = zeros(1,2);
        i = 1;
        stp = 0;
        while stp == 0
        app.HidePlot.Visible = 'off';
```

Once the information is sent to the serial port, the program will read the feedback from Arduino, and write the data to the initialized file, as well as plot it in a live graph in the GUI.

```
            b = s.BytesAvailable;
            while b > 0
            tsec = str2double(fscanf(s));
        sp = str2double(fscanf(s));
            table(i,1) = tsec;
            table(i,2) = sp;
            fprintf(fileExport,'%f,%g\n',tsec,sp);
            pause(0.5);
            b = s.BytesAvailable;
            plot(app.Plot, table(:,1), table(:,2))
            title(app.Plot, 'Current Speed')
            i = i+1;
            end
            end
```

In speed profile mode, the second digit will be the flat part of the speed profile speed. The third will represent time in seconds for the end of the ramp up time, the fourth digit will be the beginning of the ramp down time, and the last and fifth digit will be the time for the end of the ramp down. These values will be parsed from the input file.
Similarly to the manual speed mode, once the data is sent to the serial port, the program will read the feedback input from Arduino and write to the export file, as well as plot.

```
    case 'Speed Profile'
            global profile;
            maxSpeed = 12;
            speedInMPH = profile(2,2);
            if speedInMPH >= maxSpeed;
            speed = maxSpeed;
            elseif speedInMPH <= 0;
            speed = 0;
            else
            speed = speedInMPH;
            end
            t1 = profile(2,1);              % end of ramp up time
        t2 = profile(3,1);          % end of flat time
            t3 = profile(4,1);              % end of ramp down time
            sendData = sprintf('<2,%d,%d,%d,%d>',speed,t1,t2,t3);
            fwrite(s,sendData);
          pause(1);
            table = zeros(1,2);
            i = 1;
            b = s.BytesAvailable;
            while b > 0
            tsec = str2double(fscanf(s));
            sp = str2double(fscanf(s));
            table(i,1) = tsec;
            table(i,2) = sp;
            fprintf(fileExport,'%f,%g\n',tsec,sp);
            pause(0.5);
            b = s.BytesAvailable;
          plot(app.Plot, table(:,1), table(:,2))
            title(app.Plot, 'Current Speed')
```

```
                i = i+1;
                end
                fclose('all');
        end

        end
```

The next callback function is the StopButton, which defines what happens once the red Stop button is pressed. The message "<0>" is sent to serial, and some of the apps interface features are reset.

```
% Button pushed function: StopButton
        function StopButtonPushed(app, event)
        global s;
        global stp;
        stp = 1;
        flushinput(s);
        fclose('all');
        fwrite(s,'<0>');

        app.RunButton.Enable = 'on';
        switch app.InputType.Value
        case 'Manual Control'
                app.SpeedSlider.Enable = 'off';
                app.SpeedSlider.Value = 0;
                app.InputSpeed.Value = 0;

        case 'Speed Profile'

        end
        end
```

The callback function SpeedSlider will read the value from the graphical speed slider and send that information to the serial port in the same way as the previously described "Run" button on Manual Mode.

```
        % Value changing function: SpeedSlider
        function SpeedSliderValueChanging(app, event)
        global s;
        flushinput(s);
        global fileExport;
        speed = event.Value;
        app.InputSpeed.Value = speed;
            sendData = sprintf('<1,%d>',speed);
        fwrite(s,sendData);
        end
```

The InputType function reads from the switch interface (Manual Control or Speed Profile) and sends information to the serial port to stop the motor. It also stores the value of the input for

other functions to understand in which mode the speed will be controlled. It also changes the GUI interface to show the panels, buttons, sliders, etc, relevant to the chosen mode.

```
% Value changed function: InputType
function InputTypeValueChanged(app, event)
global s;
flushinput(s);
fclose('all');
global stp;
stp = 1;
fwrite(s, '<0>');  % If mode is changed, this automatically stops the motor

if app.InputType.Value == "Manual Control"
app.SpeedSlider.Value = 0;
app.InputSpeed.Value = 0;
app.SpeedProfilePanel.Visible = 'off';
app.ManualControlPanel.Visible = 'on';
app.SpeedSlider.Enable = 'off';
app.RunPanel.Visible = 'on';
    app.RunButton.Enable = 'on';
app.HidePlot.Visible = 'on';

elseif app.InputType.Value == "Speed Profile"
app.ManualControlPanel.Visible = 'off';
app.SpeedProfilePanel.Visible = 'on';
app.ErrorPanel.Visible = 'off';
app.RunButton.Enable = 'off';
app.HidePlot.Visible = 'off';
end

end
```

When the function ChooseFile is called, the input file is opened. The function only allows for csv type files to be opened, and it will show an error if a different kind of file is called.

```
% Button pushed function: ChooseFile
function ChooseFileButtonPushed(app, event)
[filename, pathname, filerindex] = uigetfile('.csv');
filetype = '.csv';
if filename ~= 0
if endsWith(filename,filetype) == 0
        app.ErrorPanel.Visible = 'on';
else
        app.FileName.Value = filename;
        app.LoadProfile.Enable = 'on';
end
end
end
```

LoadProfile will be called when the Load Profile button is pushed. This function will read the opened file and make it available for other functions. It will also plot the profile in the GUI window for a preview before its sent to the serial port.

```
% Button pushed function: LoadProfile
```

```
function LoadProfileButtonPushed(app, event)
filename = app.FileName.Value;
global profile;
profile = csvread(filename);
plot(app.Plot, profile(:,1), profile(:,2))

app.RunButton.Enable = 'on';
end
```

CloseButton is the function that closes the error message that appears if the wrong kind of file is loaded.

```
  % Button pushed function: CloseButton
function CloseButtonPushed(app, event)
app.ErrorPanel.Visible = 'off';
end
```

The function ConnectButton is called when the Connect button is pushed. It first resets all serial ports, so the app can't try to open a port that is already opened by MATLAB. It then checks whether the port being opened is available, and if so, it opens the port.

```
% Button pushed function: ConnectButton
function ConnectButtonPushed(app, event)
instrreset  % Disconnect and delete all instrument objects

global s;
serialport = app.SerialPortField.Value;
avports = seriallist;  % Available ports

validport = find(strcmp(avports,serialport));  % Check if input port is available

if size(validport,2) == 0
app.PortUnav.Visible = 'on';
else
% Connect to serial
s = serial(app.SerialPortField.Value,'BaudRate', 115200);
fopen(s);
flushinput(s);
% Confirm Connection
app.PortUnav.Visible = 'off';
app.Port.Visible = 'on';

% Visible Tabs
  app.ControlTypePanel.Visible = 'on';
app.SpeedProfilePanel.Visible = 'off';
app.ManualControlPanel.Visible = 'on';
app.InputType.Value = 'Manual Control';
app.SpeedSlider.Enable = 'off';
app.RunPanel.Visible = 'on';
app.ConnectButton.Enable = 'off';


end
end
```

The function DisconnectButton is called when the button Disconnect is pushed. It closes off the open port and resets the GUI interface.

```
% Button pushed function: DisconnectButton
function DisconnectButtonPushed(app, event)
instrreset  % Disconnect and delete all instrument objects

% Visible Tabs
app.PortUnav.Visible = 'off';
app.Port.Visible = 'off';
app.ControlTypePanel.Visible = 'off';
app.ManualControlPanel.Visible = 'off';
app.RunPanel.Visible = 'off';
app.SpeedProfilePanel.Visible = 'off';
app.ConnectButton.Enable = 'on';
app.HidePlot.Visible = 'on';

end
```

The function InputSpeed is very similar to SpeedSlider, but it reads its input from a direct input.

```
% Value changed function: InputSpeed
function InputSpeedValueChanged(app, event)
speed = app.InputSpeed.Value;
global fileExport;
global s;
flushinput(s);
app.SpeedSlider.Value = speed;
    sendData = sprintf('<1,%d>',speed);
fwrite(s,sendData);
end
end
```

The following functions are generated by App Designer. They configure all the GUI components and initialize them. It's where the components characteristics such as size, color, alignment, and initial value are defined.

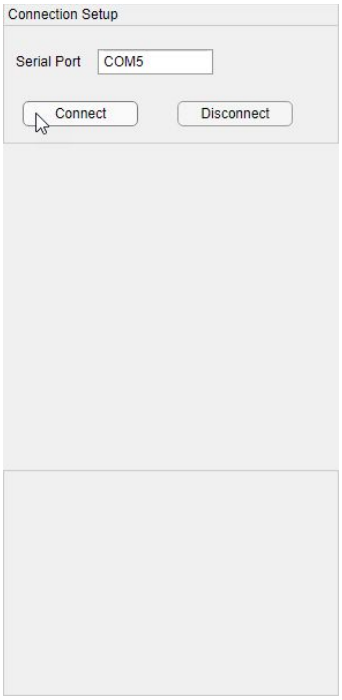
```
% App initialization and construction
methods (Access = private)
% Create UIFigure and components
function createComponents(app)
  % Create RobotTreadmillUIFigure
app.RobotTreadmillUIFigure = uifigure;
app.RobotTreadmillUIFigure.Position = [100 100 291 601];
app.RobotTreadmillUIFigure.Name = 'Robot Treadmill';
app.RobotTreadmillUIFigure.Resize = 'off';
% Create Plot
app.Plot = uiaxes(app.RobotTreadmillUIFigure);
title(app.Plot, 'Loaded Profile')
xlabel(app.Plot, 'Time (seconds)')
ylabel(app.Plot, 'Velocity (MPH)')
app.Plot.Position = [4 1 288 196];
```

```
    % Create ConnectionSetupPanel
    app.ConnectionSetupPanel = uipanel(app.RobotTreadmillUIFigure);
    app.ConnectionSetupPanel.Title = 'Connection Setup';
    app.ConnectionSetupPanel.Position = [1 480 291 122];
    % Create ConnectButton
    app.ConnectButton = uibutton(app.ConnectionSetupPanel, 'push');
    app.ConnectButton.ButtonPushedFcn = createCallbackFcn(app, @ConnectButtonPushed, true);
    app.ConnectButton.Position = [17 15 100 22];
    app.ConnectButton.Text = 'Connect';
    % Create DisconnectButton
    app.DisconnectButton = uibutton(app.ConnectionSetupPanel, 'push');
app.DisconnectButton.ButtonPushedFcn = createCallbackFcn(app, @DisconnectButtonPushed, true);
    app.DisconnectButton.Position = [151 15 100 22];
    app.DisconnectButton.Text = 'Disconnect';
    % Create SerialPortEditFieldLabel
    app.SerialPortEditFieldLabel = uilabel(app.ConnectionSetupPanel);
app.SerialPortEditFieldLabel.HorizontalAlignment = 'right';
app.SerialPortEditFieldLabel.VerticalAlignment = 'top';
app.SerialPortEditFieldLabel.Position = [5 64 62 15];
    app.SerialPortEditFieldLabel.Text = 'Serial Port';
    % Create SerialPortField
    app.SerialPortField = uieditfield(app.ConnectionSetupPanel, 'text');
    app.SerialPortField.Position = [82 60 100 22];
    app.SerialPortField.Value = 'COM5';
    % Create Port
    app.Port = uilabel(app.ConnectionSetupPanel);
    app.Port.VerticalAlignment = 'top';
    app.Port.FontColor = [0 1 0];
    app.Port.Enable = 'off';
    app.Port.Position = [218 59 56 24];
    app.Port.Text = '';
    % Create PortUnav
    app.PortUnav = uilabel(app.ConnectionSetupPanel);
    app.PortUnav.HorizontalAlignment = 'center';
    app.PortUnav.VerticalAlignment = 'top';
app.PortUnav.FontColor = [1 0 0];
    app.PortUnav.Enable = 'off';
    app.PortUnav.Position = [79 41 106 15];
    app.PortUnav.Text = 'Port unavailable';
    % Create ControlTypePanel
    app.ControlTypePanel = uipanel(app.RobotTreadmillUIFigure);
    app.ControlTypePanel.Title = 'Control Type';
    app.ControlTypePanel.Position = [1 406 291 75];
    % Create Label
    app.Label = uilabel(app.ControlTypePanel);
    app.Label.HorizontalAlignment = 'center';
    app.Label.VerticalAlignment = 'top';
    app.Label.Position = [127 -7 25 15];
    app.Label.Text = '';
    % Create InputType
    app.InputType = uiswitch(app.ControlTypePanel, 'slider');
    app.InputType.Items = {'Manual Control', 'Speed Profile'};
    app.InputType.ValueChangedFcn = createCallbackFcn(app, @InputTypeValueChanged, true);
    app.InputType.Position = [117 23 45 20];
    app.InputType.Value = 'Manual Control';
    % Create ManualControlPanel
    app.ManualControlPanel = uipanel(app.RobotTreadmillUIFigure);
    app.ManualControlPanel.Title = 'Manual Control';
    app.ManualControlPanel.Position = [1 264 291 143];
    % Create SpeedSliderLabel
```

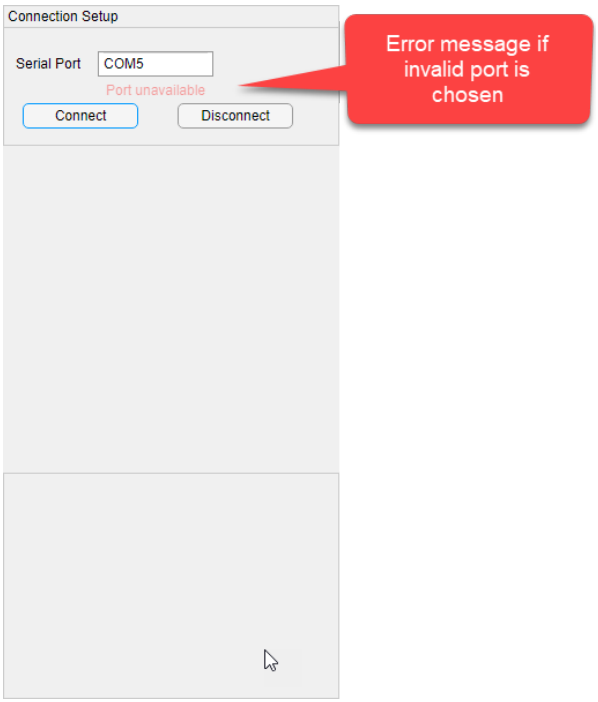
```
    app.SpeedSliderLabel = uilabel(app.ManualControlPanel);
app.SpeedSliderLabel.HorizontalAlignment = 'right';
app.SpeedSliderLabel.VerticalAlignment = 'top';
     app.SpeedSliderLabel.Position = [10 103 41 15];
    app.SpeedSliderLabel.Text = 'Speed';
    % Create SpeedSlider
    app.SpeedSlider = uislider(app.ManualControlPanel);
    app.SpeedSlider.Limits = [0 12];
    app.SpeedSlider.ValueChangingFcn = createCallbackFcn(app, @SpeedSliderValueChanging, true);
    app.SpeedSlider.Position = [72 109 150 3];
    % Create MPH
    app.MPH = uilabel(app.ManualControlPanel);
    app.MPH.VerticalAlignment = 'top';
    app.MPH.Position = [130 41 32 15];
    app.MPH.Text = 'MPH';
    % Create InputSpeed
    app.InputSpeed = uieditfield(app.ManualControlPanel, 'numeric');
    app.InputSpeed.Limits = [0 12];
        app.InputSpeed.ValueDisplayFormat = '%.2f';
    app.InputSpeed.ValueChangedFcn = createCallbackFcn(app, @InputSpeedValueChanged, true);
    app.InputSpeed.HorizontalAlignment = 'left';
    app.InputSpeed.FontSize = 24;
       app.InputSpeed.Position = [16 33 100 30];
    % Create RunPanel
    app.RunPanel = uipanel(app.RobotTreadmillUIFigure);
    app.RunPanel.Position = [1 196 291 68];
    % Create RunButton
    app.RunButton = uibutton(app.RunPanel, 'push');
    app.RunButton.ButtonPushedFcn = createCallbackFcn(app, @RunButtonPushed, true);
    app.RunButton.BackgroundColor = [0 1 0];
    app.RunButton.FontWeight = 'bold';
    app.RunButton.FontColor = [1 1 1];
    app.RunButton.Position = [17 24 100 22];
    app.RunButton.Text = 'Run';
    % Create StopButton
    app.StopButton = uibutton(app.RunPanel, 'push');
    app.StopButton.ButtonPushedFcn = createCallbackFcn(app, @StopButtonPushed, true);
    app.StopButton.BackgroundColor = [1 0 0];
    app.StopButton.FontWeight = 'bold';
    app.StopButton.FontColor = [1 1 1];
    app.StopButton.Position = [161 24 100 22];
    app.StopButton.Text = 'Stop';
    % Create SpeedProfilePanel
    app.SpeedProfilePanel = uipanel(app.RobotTreadmillUIFigure);
    app.SpeedProfilePanel.Title = 'Speed Profile';
    app.SpeedProfilePanel.Position = [1 263 291 143];
    % Create ChooseFile
    app.ChooseFile = uibutton(app.SpeedProfilePanel, 'push');
    app.ChooseFile.ButtonPushedFcn = createCallbackFcn(app, @ChooseFileButtonPushed, true);
    app.ChooseFile.Position = [8 91 100 22];
    app.ChooseFile.Text = 'Choose File';
    % Create FileName
    app.FileName = uieditfield(app.SpeedProfilePanel, 'text');
    app.FileName.Editable = 'off';
    app.FileName.BackgroundColor = [0.9412 0.9412 0.9412];
    app.FileName.Position = [8 53 271 22];
    % Create LoadProfile
    app.LoadProfile = uibutton(app.SpeedProfilePanel, 'push');
    app.LoadProfile.ButtonPushedFcn = createCallbackFcn(app, @LoadProfileButtonPushed, true);
    app.LoadProfile.Enable = 'off';
```

```
            app.LoadProfile.Position = [8 17 100 22];
            app.LoadProfile.Text = 'Load Profile';
            % Create ErrorPanel
            app.ErrorPanel = uipanel(app.SpeedProfilePanel);
               app.ErrorPanel.Title = 'Error';
            app.ErrorPanel.Position = [14 23 260 82];
            % Create CloseButton
            app.CloseButton = uibutton(app.ErrorPanel, 'push');
            app.CloseButton.ButtonPushedFcn = createCallbackFcn(app, @CloseButtonPushed, true);
            app.CloseButton.Position = [80 15 100 22];
            app.CloseButton.Text = 'Close';
            % Create WrongFile
            app.WrongFile = uilabel(app.ErrorPanel);
            app.WrongFile.HorizontalAlignment = 'center';
            app.WrongFile.VerticalAlignment = 'top';
            app.WrongFile.Position = [40 43 185 15];
            app.WrongFile.Text = 'Wrong file type. Upload a csv file.';
            % Create HidePlot
            app.HidePlot = uipanel(app.RobotTreadmillUIFigure);
            app.HidePlot.Position = [1 1 291 196];
            end
            end
            methods (Access = public)
            % Construct app
            function app = Treadmillv03
            % Create and configure components
            createComponents(app)
            % Register the app with App Designer
            registerApp(app, app.RobotTreadmillUIFigure)
            % Execute the startup function
            runStartupFcn(app, @startupFcn)
            if nargout == 0
            clear app
            end
            end
            % Code that executes before app deletion
            function delete(app)
            % Delete UIFigure when app is deleted
            delete(app.RobotTreadmillUIFigure)
            end
            end
end
```

Upon opening the Treadmill app, the user will have the option to set up a serial connection, required to communicate information to and from the Arduino board. The first action will be to choose a Serial Port, and click “Connect”.



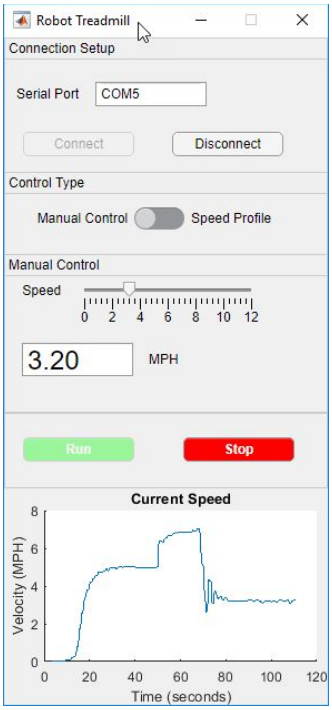
An error message will be displayed if the user chooses a port that



is unavailable.

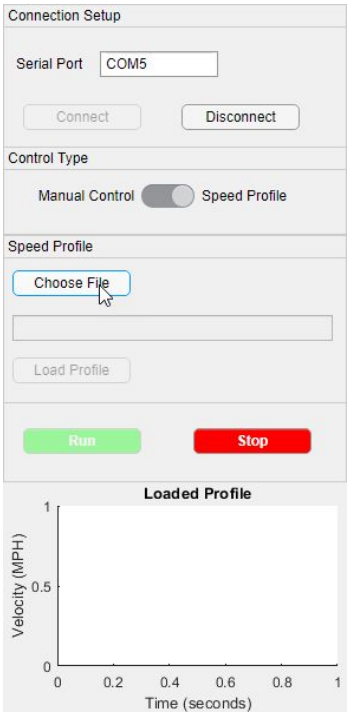
If a valid port is selected and the connection is successful, the app will then present a switch to select what kind of speed input will be provided (Manual Control or Speed Profile), the default choice being “Manual Control”.

To start Manual Control, the user must click on the green “Run” button. This action will initiate the data transmission to and from the Treadmill. The data sent will correspond to the desired speed of the treadmill in miles per hour. The data received will be visualized as a graph of the speed versus time (in seconds). The same data will be recorded in an external csv file.



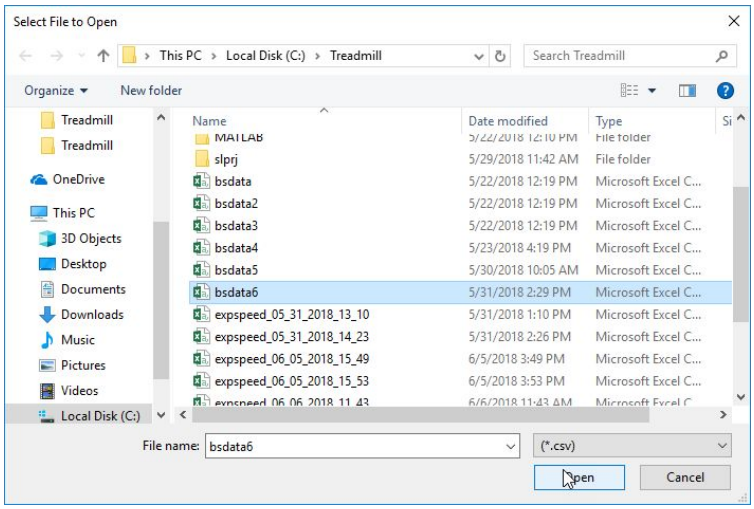
Another option is for the user to choose Speed Profile. This option allows the user to load a pre-programmed profile for speed over time, that, once sent out, will run the treadmill accordingly, without any further adjustments.

Once the switch to Speed Profile is made in the app, the Speed Profile panel becomes visible. The first required action is to chose the file with the profile. Clicking the Choose File button will open a window for the user to browse the file in their computer.

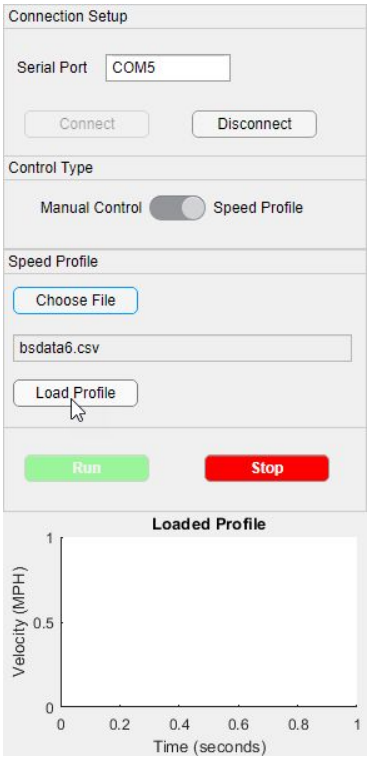


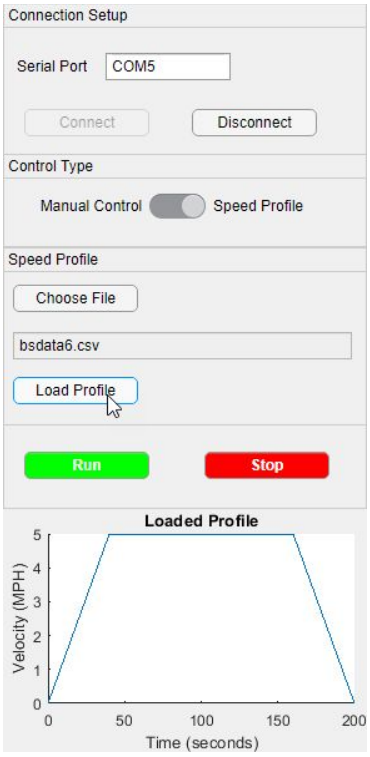
In the current version of the app, only CSV type files are accepted. The profile data should have 2 columns and 4 rows. The first column corresponds to the time, in seconds, and the second to the desired speed.

	A	B
1	0	0
2	40	5
3	160	5
4	200	0



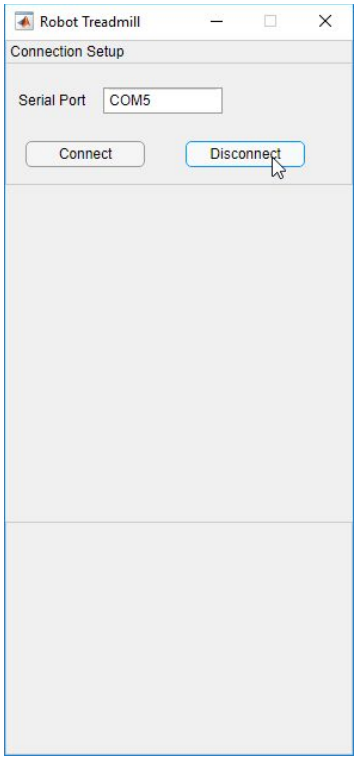
This version of the software only allows a speed profile that starts at 0 MPH, linearly increases its speed to a steady state value, and then linearly decreases back to a stop.
After choosing a file, the user shall press the Load Profile button in order for the profile to be parsed into the program. At that point, a preview of the loaded profile will be plotted to allow a visualization of the data loaded.





Clicking on the green Run button at this point will initiate communication with the treadmill system. The profile will be uploaded to the Arduino board, and real-time speed results will start being plotted in the apps window. The same data will also be exported to an external file.

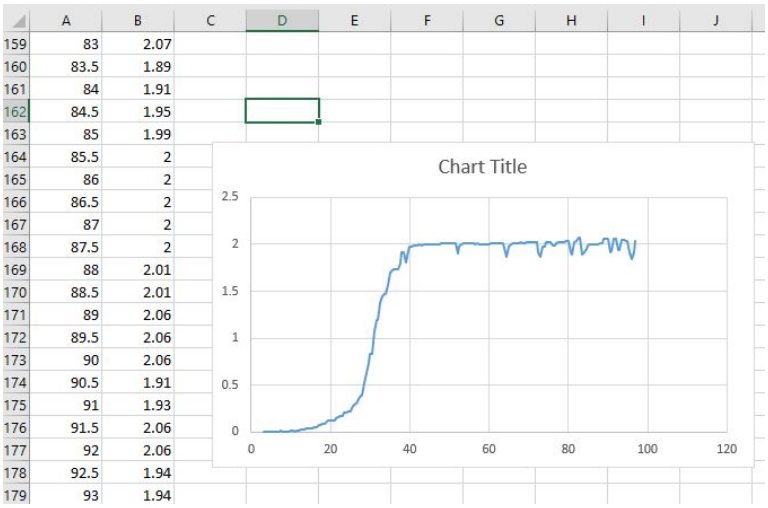
At any point, clicking on the red Stop button will stop the treadmill and also stop serial communication. If the user wants to record the wind down data during manual control, they will have to select a speed of 0 and wait until that speed is achieved before pressing Stop.



Selecting Disconnect will also stop the treadmill and quit communication, and it will also close the serial port.

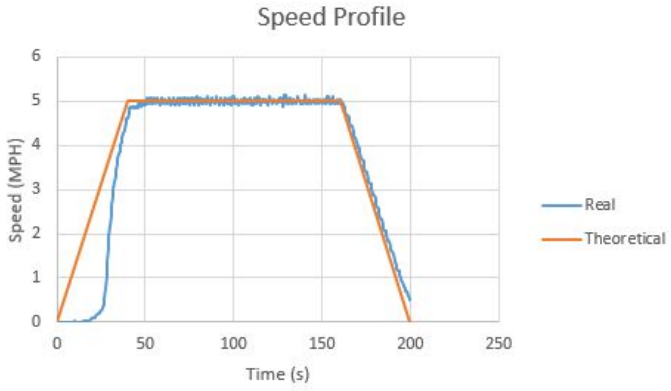
The data collected during the runs will be saved in the same directory from where the Treadmill app is running. It will be saved to a file named “expspeed[date and time].csv”.

The data will be recorded in 2 columns, the first one being the time, in seconds, and the second one the speed in MPH. The data can be easily plotted for visualization, and can also be used for analysis, etc.



	A	B
159	83	2.07
160	83.5	1.89
161	84	1.91
162	84.5	1.95
163	85	1.99
164	85.5	2
165	86	2
166	86.5	2
167	87	2
168	87.5	2
169	88	2.01
170	88.5	2.01
171	89	2.06
172	89.5	2.06
173	90	2.06
174	90.5	1.91
175	91	1.93
176	91.5	2.06
177	92	2.06
178	92.5	1.94
179	93	1.94

For example, the desired speed profile can be directly compared to the actual speed of the treadmill.



F. Arduino Control

Arduino Code (Treadmill.ino)

The Arduino code is preloaded into the onboard Arduino Due board.

The PID control used is being called using a pre-existing Arduino PID library. It's a simple, but useful PID control. The documentation for this library can be found at https://playground.arduino.cc/Code/PIDLibrary .

```
#include <PID_v1.h>   // use PID library
```

At this point the code also declares which Arduino pin will be used for sending the PWM data out, and it also declares the global variables to be used throughout the program.
The values for Kp and Ki were determined experimentally, and this will be the point in the program for any changes in these values. These Kp and Ki are used in the PID loops throughout the program.

```
#define PWM_PIN 5

//From MATLAB:
//'0' means STOP
//'1' means MANUAL SPEED INPUT
//'2' means SPEED PROFILE

unsigned long startMillis;  //some global variables available anywhere in the program
unsigned long currentMillis;
double prevMillis = 0;
//double timeChange = 0;
int sampleTime = 5;
// variables to hold values for the encoder. The inputs are 2 and 13 (and optionally A6 for the index).
const int ENCODER_EDGES_PER_ROTATION = 2048 * 4;
const int ENCODER_SAMPLES_PER_SECOND = 2;
const int LOOP_DELAY_MS = 0;

const byte numChars = 32;
char receivedChars[numChars];
char tempChars[numChars];        // temporary array for use when parsing

// variables to hold the parsed data
int optionFromPC = 0;
float sFromPC = 0;
float t1FromPC = 0;
float t2FromPC = 0;
float t3FromPC = 0;

// variables to be used for the closed loop
float desiredSpeed = 0;
//float Error = 0;
//float errSum = 0;
double desiredHz = 0;
double actualHz = 0;
```

```
float actualMPH = 0;
double PWMOutput = 0;
int INPUT_TIMING_PIN = 11;
int InputTime = 0;
double  Kp = .5, Ki = .25, Kd = 0;

PID motorPID(&actualHz, &PWMOutput, &desiredHz, Kp, Ki, Kd, DIRECT); // specify parameters for controller

boolean newData = false;

//variables to define the transformations (based on measurements)
float dShaft = 1; // inches
float dWheel = 2.4; //inches
float dRoller = 1.92; //inches
```

The setup() function is used to initialize the serial communication, setup the PID controller, initialize the time of running the program, and to setup the quadrature encoder.
The settings for the encoder are given by the manufacturer, and the parts included in this code are the ones necessary to register the frequency of the motor in rotations per second, or Hz.
Running time will be measured using millis(). startMillis will be the initial time, and any further measurement will be done by calculating the difference between the current millis() and startMillis.
A few of the PID setup definitions include the frequency in which the PID control will be calculated, and its limits for output (0 - 255).

```
void setup() {
  Serial.begin(115200);
  pinMode(INPUT_TIMING_PIN, OUTPUT);
  pinMode(PWM_PIN, OUTPUT);

  // Setup Quadrature Encoder
  REG_PMC_PCER0 = PMC_PCER0_PID27
                | PMC_PCER0_PID28
                | PMC_PCER0_PID29;

  // Setup a channel in waveform mode
  REG_TC0_CMR2 = TC_CMR_TCCLKS_TIMER_CLOCK4
               | TC_CMR_WAVE
               | TC_CMR_ACPC_TOGGLE
               | TC_CMR_WAVSEL_UP_RC;

  //Define the sample period
  REG_TC0_RC2 = F_CPU / 128 / ENCODER_SAMPLES_PER_SECOND;

  // Setup a channel in capture mode
  REG_TC0_CMR0 = TC_CMR_ABETRG
               | TC_CMR_LDRA_EDGE
               | TC_CMR_LDRB_EDGE
               | TC_CMR_ETRGEDG_EDGE
               | TC_CMR_CPCTRG;

  // Enable features, noting Speed not Position is chosen
  REG_TC0_BMR = TC_BMR_QDEN
              | TC_BMR_SPEEDEN
```

```
            | TC_BMR_EDGPHA;

  // Set everything going
  REG_TC0_CCR0 = TC_CCR_CLKEN | TC_CCR_SWTRG;
  REG_TC0_CCR1 = TC_CCR_CLKEN | TC_CCR_SWTRG;
  REG_TC0_CCR2 = TC_CCR_CLKEN | TC_CCR_SWTRG;

  startMillis = millis();
  motorPID.SetSampleTime(sampleTime);
  motorPID.SetOutputLimits(0, 255);
  motorPID.SetMode(AUTOMATIC);
}
```

The main loop will read for data incoming through the serial port, it will then parse this data, and run or stop the treadmill accordingly.

```
void loop() {
  recvWithStartEndMarkers();
  if (newData == true) {
    strcpy(tempChars, receivedChars);
    // this temporary copy is necessary to protect the original data
    //   because strtok() used in parseData() replaces the commas with \0
    parseData();
    defineOption();
    newData = false;
  }
}
```

The function recvWithStartEndMarkers() is used to receive data from the serial buffer. All characters are discarded until the start-marker (<) is detected. At that point, the function will write everything coming in into a temporary string, until the end-marker (>) is detected.

```
void recvWithStartEndMarkers() {
  static boolean recvInProgress = false;
  static byte ndx = 0;
  char startMarker = '<';
  char endMarker = '>';
  char rc;

  while (Serial.available() > 0 && newData == false) {
    rc = Serial.read();
    if (recvInProgress == true) {
      if (rc != endMarker) {
        receivedChars[ndx] = rc;
        ndx++;
        if (ndx >= numChars) {
          ndx = numChars - 1;
        }
      }
      else {
        receivedChars[ndx] = '\0'; // terminate the string
        recvInProgress = false;
```

```
        ndx = 0;
        newData = true;
      }
    }
    else if (rc == startMarker) {
      recvInProgress = true;
    }
  }
}
```

The data stored in the receivedChars string will then be parsed with the function parseData().

```
void parseData() {      // split the data into its parts
  char * strtokIndx; // this is used by strtok() as an index

  strtokIndx = strtok(tempChars, ","); // Gets the first part of the data
  optionFromPC = atof(strtokIndx);     // convert this part to an integer

  strtokIndx = strtok(NULL, ",");
  sFromPC = atof(strtokIndx);     // speed input is in MPH

  strtokIndx = strtok(NULL, ","); //this continues where the previous call left off
  t1FromPC = atof(strtokIndx);

  strtokIndx = strtok(NULL, ",");
  t2FromPC = atof(strtokIndx);

  strtokIndx = strtok(NULL, ",");
  t3FromPC = atof(strtokIndx);
}
```

The function defineOption() reads what kind of input option is being sent, and then calls out the according actuating function. Option 0 stops the motor, option 1 receives continuous manual control from the computer, and option 2 receives data to be used as a speed profile.

```
void defineOption() {
  switch (optionFromPC) {
    case 0 :
      stopRunning();
      break;
    case 1 :
      manualControl();
      break;
    case 2 :
      speedProfile();
      break;
  }
}
```

All the function stopRunning() does is it sends 0 to the PWM pin, effectively stopping the motor.

```
void stopRunning() {
  analogWrite(PWM_PIN, 0);
```

```
}
```

The function manualControl() reads the desired speed, converts it to a motor rotation frequency in Hz, and uses this parameter in a PID loop. It also reads the current speed, and sends out to the computer the current time and speed, read from the encoder and converted to MPH. It will run this loop until a new signal is sent in from the computer.

```
void manualControl() {

  currentMillis = millis() - startMillis;
  int prevMillis = -1;
  while (Serial.available() == 0) {
```

The next part of the code is used to read the pulses from the encoder, and then it transforms that number into a motor frequency in Hz.

```
InputTime = InputTime ^ 0x01; //Toggle pin when input signal starts over.
    digitalWrite(INPUT_TIMING_PIN, InputTime);

    //Rotations per second, from encoder
    int iSpeedPPP = REG_TC0_RA0;    // Pulses Per sample Period
    actualHz = ((iSpeedPPP / (ENCODER_EDGES_PER_ROTATION * 1.0)) * ENCODER_SAMPLES_PER_SECOND);
```

Once the current speed is known, the PID() function is called. It uses the current and desired speeds in Hz, the values of Ki and Kp, and it outputs a PWM value, which is then sent out to run the motor.

```
    desiredHz = convertToHz(sFromPC);

    motorPID.Compute(); //call out the PID() function

    // Write the PID output to the pin
    if (PWMOutput > 255) {
      analogWrite(PWM_PIN, 255);
    }
    else if (PWMOutput < 0) {
      analogWrite(PWM_PIN, 0);
    }

    else {
      analogWrite(PWM_PIN, PWMOutput);
    }
```

The program will then send out the information (time, speed) to the computer. The following part of the code limits it so it sends this data every 0.5 seconds (500 milliseconds).

```
 actualMPH = convertToMPH(actualHz);

    //Write the output to serial every .5 seconds
    float timeInSec = currentMillis;
```

```
    timeInSec /= 1000;
    if (currentMillis % 500 == 0 && prevMillis != currentMillis) {
      Serial.println(timeInSec);
      Serial.println(actualMPH);
    }
    prevMillis = currentMillis;
    currentMillis = millis() - startMillis;
  }
}
```

The speedProfile() function reads in the input data and converts it into a profile over time. It calculates the slopes of ramping up and down and then run a loop similar to the Manual Control loop, with the exception that the speed changes as a function of time.
The first thing it does after calculating the slopes is to reset the clock, so the treadmill run time starts at zero seconds.

```
void speedProfile() {
  double slope1 = sFromPC / t1FromPC;
  double slope2 = -sFromPC / (t3FromPC - t2FromPC);

  int    t1 = t1FromPC * 1000; //convert times to ms
  int    t2 = t2FromPC * 1000;
  int    t3 = t3FromPC * 1000;

  startMillis = millis();
  currentMillis = millis() - startMillis;
```

The program then starts reading the current speed from the encoder, and it changes the desired speed as time goes on. The function ends when the programmed profile time runs out.

```
while (currentMillis <= t3) {
    //Rotations per second, from encoder
    int iSpeedPPP = REG_TC0_RA0;    // Pulses Per sample Period
    actualHz = ((iSpeedPPP / (ENCODER_EDGES_PER_ROTATION * 1.0)) * ENCODER_SAMPLES_PER_SECOND);

    if (currentMillis <= t1) {
      desiredSpeed = currentMillis * slope1 / 1000;
    }
    else if (currentMillis >> t1 && currentMillis <= t2) {
      desiredSpeed = sFromPC;
    }
    else if (currentMillis >= t2) {
      desiredSpeed = (sFromPC + (currentMillis - t2) * slope2 / 1000);
    }

    desiredHz = convertToHz(desiredSpeed);
    motorPID.Compute(); //call out the PID() function

    // Write the PID output to the pin
    if (PWMOutput > 255) {
      analogWrite(PWM_PIN, 255);
    }
    else if (PWMOutput < 0) {
      analogWrite(PWM_PIN, 0);
```

```
    }
    else {
      analogWrite(PWM_PIN, PWMOutput);
    }

    actualMPH = convertToMPH(actualHz);

    //Write the output to serial every .5 seconds
    float timeInSec = currentMillis;
    timeInSec /= 1000;
    int everyHalfSec = currentMillis % 500;
    if (everyHalfSec == 0 && prevMillis != currentMillis) {
      Serial.println(timeInSec);
      Serial.println(actualMPH);
    }
    prevMillis = currentMillis;
    currentMillis = millis() - startMillis;
  }
}
```

The last two functions are simply converting the speeds in MPH into motor rotation frequency, and vice versa. They use the previously defined values for the diameter of the motor shaft, diameter of the gear wheel and diameter of the roller.

```
float convertToMPH(float Hz) {
  float MPH = PI / 17.6 * (Hz * dShaft * dRoller / dWheel); //MPH
  return  MPH;
}

///||\\\||///||\\\||///||\\\||///||\\\

float convertToHz (float MPH) {
  float vInHz = 17.6 / PI  * (MPH * dWheel / (dShaft * dRoller)); //Hz
  return vInHz;
}
```

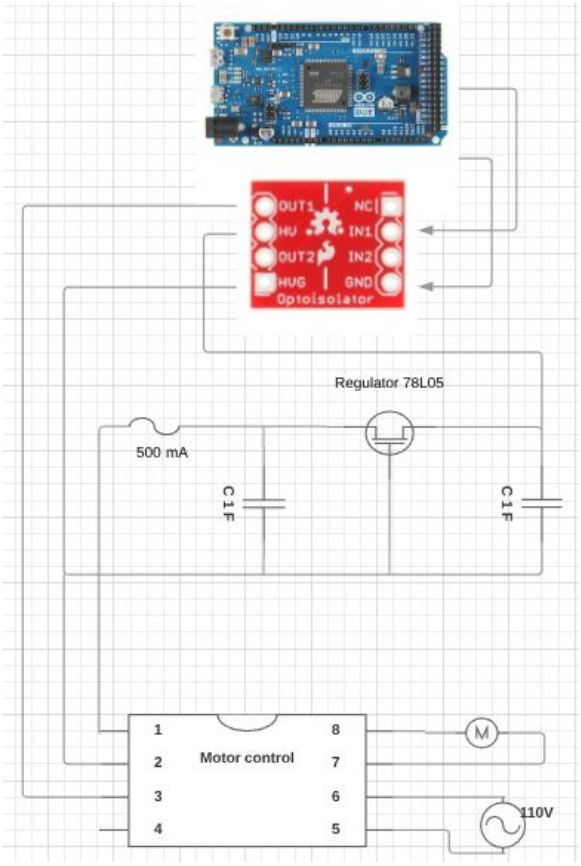
///||\\\||///||\\\||///||\\\||///||\\\

Bill of Materials

Part	Quantity	Price
54.3 in. 80/20 15 - series	6	Provided by sponsor
39 in. 80/20 15 - series	4	Provided by sponsor
25.2 in. 80/20 15 - series	4	Provided by sponsor

80/20 15 - series		$76
Belt rollers	2	Taken from a free treadmill
Belt	1	Taken from a free treadmill
Front roller plates	2	Machined
Rear roller plates	2	Machined
Plywood 25in. x 25 in	1	Free
100V DC motor	1	Taken from a free treadmill
Pulley belt	1	Taken from a free treadmill
Small pulley	1	3D printed
DC Motor Controller	1	$155
Breadboard	1	$5.90
Wires	Many	Taken from the lab
Fasteners - FBHSCS & T-Nut	48	$26.85
3 in.Bolts (1/4"-20)	6	$10
Arduino DUE	1	$37.40
Wooden belt support plate	1	Taken from a free treadmill
DC motor subframe		Taken from a free treadmill
Total:		$311.15

Electrical Schematic



1 = Voltage source on the motor control
2 = Ground signal input
3 = Positive signal input
4 = No connection
5,6 = Wall power input
7,8 =Motor DC output