Simulation, Design, and Hardware Implementation of a 4-axis Cable Suspended Robot

A thesis presented to
the faculty of
the Russ College of Engineering and Technology of Ohio University

In partial fulfillment
of the requirements for the degree
Master of Science

Collier R. Fais
April 2017
© 2017 Collier R. Fais. All Rights Reserved
This thesis titled
Simulation, Design, and Hardware Implementation of a 4-axis Cable Suspended Robot

by
COLLIER R. FAIS

has been approved for
the Department of Mechanical Engineering
and the Russ College of Engineering and Technology by

Robert L. Williams II
Professor of Mechanical Engineering

Dennis Irwin
Dean, Russ College of Engineering and Technology
ABSTRACT

FAIS, COLLIER R., M.S., April 2017, Mechanical Engineering

Simulation, Design, and Hardware Implementation of a 4-axis Cable Suspended Robot

Director of Thesis: Robert L. Williams II

A classroom-scale 4-axis cable-suspended robot (CSR) is developed with MATLAB software for versatile graphical user interface (GUI) computer simulations, SolidWorks software for computer aided design (CAD), construction and machining of basic stock materials, and Parker Automation Control (PAC) software for 2-axis straight line trajectories. The MATLAB simulation program provides results of active cable lengths and velocities and accelerations, active cable tensions, active motor positions and velocities and accelerations, active cable-winch positions and velocities and accelerations, end-effector kinematics (position, velocity, acceleration, jerk and snap), global Cartesian end-effector position, robot Cartesian stiffness, and robot singularity evaluation over a variety of 3-D and 2-D trajectories and static poses. Results gained from the MATLAB simulation are used to verify that a CAD model of the prototype system can withstand the predicted operational forces based on several failure criterion yielding a minimum factor of safety of 4.34. The prototype hardware system is assembled and used to complete a straight-line point-to-point trajectory executed via the PAC software demonstrating a maximum, minimum and average position error of 10.3”, 4.8” and 8.7”, respectively, given a desired position within a 10 by 4 foot planar workspace.
I dedicate this thesis to my advisor and mentor, Dr. Bob Williams, whose teachings and innovative vision have and continue to inspire the intellectual imaginations of many.
ACKNOWLEDGEMENTS

The author would like to acknowledge the support and guidance of his thesis committee members and thesis collaborators: Dr. Bob Williams, Dr. Muhammad Ali, Dr. David Bayless, Dr. Morgan Vis-Chiasson and Professor Jesus Pagan. Further gratitude is extended to the family and close friends of the author for their unwavering patience, love and support.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Abstract</strong></td>
</tr>
<tr>
<td><strong>Dedication</strong></td>
</tr>
<tr>
<td><strong>Acknowledgements</strong></td>
</tr>
<tr>
<td><strong>List of Tables</strong></td>
</tr>
<tr>
<td><strong>List of Figures</strong></td>
</tr>
<tr>
<td>Chapter 1: Introduction</td>
</tr>
<tr>
<td>Literature Review</td>
</tr>
<tr>
<td>Algae</td>
</tr>
<tr>
<td>Uses and Advantages</td>
</tr>
<tr>
<td>Cultivation</td>
</tr>
<tr>
<td>Harvesting</td>
</tr>
<tr>
<td>Cable Robots</td>
</tr>
<tr>
<td>Definition and Current Applications</td>
</tr>
<tr>
<td>4-Axis Cable-Suspended Robot Proof-of-Concept Prototype</td>
</tr>
<tr>
<td>Project Information</td>
</tr>
<tr>
<td>Chapter 2: Thesis Objectives</td>
</tr>
<tr>
<td>Chapter 3: Cable Robot Mechanics and Control</td>
</tr>
<tr>
<td>Kinematics</td>
</tr>
<tr>
<td>Pseudostatics</td>
</tr>
<tr>
<td>Trajectory Generation Kinematics</td>
</tr>
<tr>
<td>Method of Control</td>
</tr>
<tr>
<td>Chapter 4: MATLAB and SolidWorks</td>
</tr>
<tr>
<td>MATLAB Simulation</td>
</tr>
<tr>
<td>Purpose</td>
</tr>
<tr>
<td>How to Use</td>
</tr>
<tr>
<td>Snapshot Example</td>
</tr>
<tr>
<td>Trajectory Examples</td>
</tr>
<tr>
<td>SolidWorks Model</td>
</tr>
<tr>
<td>Physical Model</td>
</tr>
<tr>
<td>Failure Criterion</td>
</tr>
<tr>
<td>Chapter 5: Hardware Implementation and Control</td>
</tr>
</tbody>
</table>
LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1</td>
<td>Comparison of oil yields from biomass feedstocks [2].</td>
<td>13</td>
</tr>
<tr>
<td>Table 2</td>
<td>FEA stress results.</td>
<td>71</td>
</tr>
<tr>
<td>Table 3</td>
<td>Trajectory test position errors.</td>
<td>91</td>
</tr>
<tr>
<td>Table 4</td>
<td>Minimum, maximum, average and range of position error in inches.</td>
<td>91</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Algal photobioreactor [9]</td>
<td>14</td>
</tr>
<tr>
<td>2</td>
<td>Large-scale open algae raceway pond [13]</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>Schematic of algal biofuel process [14]</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>Direct installed capital cost allocation [14]</td>
<td>17</td>
</tr>
<tr>
<td>5</td>
<td>ABB IRB 3400L [18], Adept Quattro s650H [19]</td>
<td>18</td>
</tr>
<tr>
<td>6</td>
<td>SkyCam end-effector and stadium sized workspace [23]</td>
<td>20</td>
</tr>
<tr>
<td>7</td>
<td>4-Axis cable-suspended robot static diagram</td>
<td>27</td>
</tr>
<tr>
<td>8</td>
<td>Free body diagram of CSR end-effector in a static pose</td>
<td>30</td>
</tr>
<tr>
<td>9</td>
<td>Constant velocity profile</td>
<td>36</td>
</tr>
<tr>
<td>10</td>
<td>Constant velocity profile divided into geometric sections</td>
<td>37</td>
</tr>
<tr>
<td>11</td>
<td>Sub-plotted trajectory generation graphs for nominal example</td>
<td>39</td>
</tr>
<tr>
<td>12</td>
<td>CSR simulator, main menu</td>
<td>42</td>
</tr>
<tr>
<td>13</td>
<td>CSR simulator, RUN menu</td>
<td>43</td>
</tr>
<tr>
<td>14</td>
<td>Snap shot user input parameters</td>
<td>44</td>
</tr>
<tr>
<td>15</td>
<td>3-D and 2-D orthographic views of snap-shot example</td>
<td>45</td>
</tr>
<tr>
<td>16</td>
<td>Snap shot data table example</td>
<td>45</td>
</tr>
<tr>
<td>17</td>
<td>CSR simulator, trajectory menu</td>
<td>47</td>
</tr>
<tr>
<td>18</td>
<td>Straight line trajectory</td>
<td>47</td>
</tr>
<tr>
<td>19</td>
<td>Determinant of $[A \ A^T]$ vs. time for straight line</td>
<td>48</td>
</tr>
<tr>
<td>20</td>
<td>Robot Cartesian stiffness vs. time for straight line</td>
<td>49</td>
</tr>
<tr>
<td>21</td>
<td>Cartesian trajectory vs. time for straight line</td>
<td>49</td>
</tr>
<tr>
<td>22</td>
<td>Active cable tensions vs. time for straight line</td>
<td>50</td>
</tr>
<tr>
<td>23</td>
<td>Active cable lengths vs. time for straight line</td>
<td>50</td>
</tr>
<tr>
<td>24</td>
<td>Active cable velocities vs. time for straight line</td>
<td>51</td>
</tr>
<tr>
<td>25</td>
<td>Active cable accelerations vs. time for straight line</td>
<td>51</td>
</tr>
<tr>
<td>26</td>
<td>Active motor encoder pulse count vs. time for straight line</td>
<td>52</td>
</tr>
<tr>
<td>27</td>
<td>Active motor angular velocities vs. time for straight line</td>
<td>52</td>
</tr>
<tr>
<td>28</td>
<td>Active motor angular accelerations vs. time for straight line</td>
<td>53</td>
</tr>
<tr>
<td>29</td>
<td>Active cable reel/winch angular velocities vs. time for straight line</td>
<td>53</td>
</tr>
<tr>
<td>30</td>
<td>Active cable reel/winch angular accelerations vs. time for straight line</td>
<td>54</td>
</tr>
<tr>
<td>31</td>
<td>Completed circular trajectory</td>
<td>54</td>
</tr>
<tr>
<td>32</td>
<td>Determinant of $[A \ A^T]$ vs. time for circle</td>
<td>55</td>
</tr>
<tr>
<td>33</td>
<td>Robot Cartesian stiffness vs. time for circle</td>
<td>55</td>
</tr>
<tr>
<td>34</td>
<td>Cartesian trajectory vs. time for circle</td>
<td>56</td>
</tr>
<tr>
<td>35</td>
<td>Active cable tensions vs. time for circle</td>
<td>56</td>
</tr>
<tr>
<td>36</td>
<td>Active cable lengths vs. time for circle</td>
<td>57</td>
</tr>
<tr>
<td>37</td>
<td>Active cable velocities vs. time for circle</td>
<td>57</td>
</tr>
<tr>
<td>38</td>
<td>Active cable accelerations vs. time for circle</td>
<td>58</td>
</tr>
</tbody>
</table>
Figure 39: Active cable reel/winch angular velocities vs. time for circle. ............................ 58
Figure 40: Active cable reel/winch angular accelerations vs. time for circle. ....................... 59
Figure 41: Active motor encoder positions vs. time for circle. ........................................ 59
Figure 42: Active motor angular velocities vs. time for circle. ........................................... 60
Figure 43: Active motor angular accelerations vs. time for circle. ..................................... 60
Figure 44: Harvest trajectory example in sequential order. ................................................. 62
Figure 45: Initial and final positions of the planar configuration. ........................................ 63
Figure 46: Active cable tensions vs. time for planar configuration. .................................... 64
Figure 47: Active perpendicular force on tower vs. time. .................................................... 65
Figure 48: SolidWorks CSR prototype tower model assembly. ........................................... 66
Figure 49: Full CSR prototype assembly in the ISEE research greenhouse. .................... 67
Figure 50: Free body diagram of single tower. .................................................................. 69
Figure 51: FEA stress visualization of single tower. ......................................................... 71
Figure 52: Simplified FBD of winch/motor coupling.......................................................... 72
Figure 53: Hardware tower assembly with motor/winch, pulley and cable. ....................... 74
Figure 54: Static configuration of the 4-axis CSR prototype. ............................................ 75
Figure 55: Planar configuration of cable suspended robot. .............................................. 76
Figure 56: Motor and winch coupling subassembly. .......................................................... 77
Figure 57: Full planar hardware and software configuration. ......................................... 78
Figure 58: Encoder wheel with Arduino controller. .......................................................... 79
Figure 59: Backside of encoder wheel. ............................................................................. 80
Figure 60: PAC and P-drive............................................................................................. 81
Figure 61: Ideal position vs. time for planar trajectory. .................................................... 83
Figure 62: Test 1 position vs. time.................................................................................... 85
Figure 63: Test 2 position vs. time.................................................................................... 85
Figure 64: Test 3 position vs. time.................................................................................... 86
Figure 65: Test 4 position vs. time.................................................................................... 86
Figure 66: Test 5 position vs. time.................................................................................... 87
Figure 67: Test 6 position vs. time.................................................................................... 87
Figure 68: Final position of planar trajectory. .................................................................. 89
Figure 69: Long exposure image of planar trajectory....................................................... 90
CHAPTER 1: INTRODUCTION

As the human population continues to grow, so does our need for energy. It has become clear that the ubiquitous use of fossil fuels to support our energy demands throughout industry, transportation and power generation has helped to induce an alarming shift in the Earth’s climate. An estimated 98% of all carbon emissions are attributed to the combustion of non-renewable fossil fuels [1]. Although it is difficult to speculate if, how, and when we will sever our dependence on carbon-based fuels, there exists hope in producing similarly capable fuels in a renewable and more environmentally friendly manner. Biomass energy holds great potential as an alternative, renewable, and even pollution-mitigating solution to the energy dilemma. Algae is a promising contender as a biomass fuel-source as it not only boasts a superior oil yield compared to other biomass feedstocks [2], but is also considered to be among the fastest growing plants in the world [1]. However, in order to increase the accessibility and economic viability of algae-based fuel products, certain innovations must be implemented. This paper expands upon and further contributes to the novel concept of using cable-suspended robots to mechanize the harvesting process of the commercial algae cultivation operations. The overall aim of this paper is to provide a better understanding of cable-suspended robots by presenting, discussing and analyzing the simulation, design and hardware control of a 4-axis cable suspended robot.
Literature Review

*Algae*

*Uses and Advantages*

Although the main interest of commercially grown algae is biofuel production, there are many other ways in which algae can and have been used. Some of the commercially important uses of algae include food supplements, animal nutrition, cosmetic additives and even pharmaceuticals [3] [4]. Whatever the end product may be, cultivating algae requires a large uptake of carbon dioxide ($CO_2$), a byproduct of coal fired power plants. It has been suggested that flue gas exiting such power plants be fed into the algae growing medium as a cheap source of $CO_2$ which would effectively trap the compound inside the biomass [5]. In a sense, this would make algal biofuel nearly carbon neutral as it would take in the waste $CO_2$ during cultivation thus temporarily suspending it from circulation before releasing it when combusted later on.

Algae is considered a rather oily plant due to its relatively high lipid content which can reach upwards of 70% by weight depending on the type [6]. When compared to other oily plants, algae reigns supreme in its overall oil yield as illustrated by table 1.
Another advantage that algae have over similar biomass feedstock crops is the fact that it grows in virtually any type of aqueous environment from freshwater to saltwater [7][8]. Even though algae is widespread in natural systems, cultivating it for commercial use requires special consideration and equipment.

*Cultivation*

In order to successfully cultivate algae on a commercial scale, optimal growing conditions must be replicated in a controlled manner to produce large crop yields. The most common methods of cultivation include tubular photobioreactors, flat plate photobioreactors, column photobioreactors, and raceway ponds [8]. Photobioreactors provide a closed system consisting of either glass or plastic containers and are generally found in a relatively small-scale laboratory setting similar to the configuration seen below.
While the photobioreactor route has the advantage of being a closed system and therefore a reduced risk of being contaminated by environmental pollutants, it has the downfall of being expensive and difficult to scale up [8][9][10][2]. Open systems such as raceway ponds offer a cheaper and larger-scale growing operation alternative. The raceway pond circulates shallow water via paddle wheels through channels in an outdoor setting similar to the setup seen below [11].
Figure 2: Large-scale open algae raceway pond [13].

Besides being relatively cheap and large-scale, algae raceway ponds can be built on non-agricultural land, require low maintenance and energy input, and are easier to keep clean [8]. For these reasons, the open raceway pond is seen as the most attractive method of cultivating algae on a large-scale. However, cultivation is just one step in the algae based biofuel product pathway. The next major step to take into consideration is harvesting.

Harvesting

With respect to open algae raceway ponds, the current method of harvesting the crop remains arduous and inefficient. As illustrated by the infographic below, the
harvesting process involves the introduction of flocculants to help the algae cells begin to coagulate followed by centrifuge filtration in order to de-water the mixture [12][9].

Figure 3: Schematic of algal biofuel process [14].

When examining the algal biofuel process as a whole, it has been clearly documented that the harvesting step is the largest bottleneck due to its inefficiency and high capital costs [13][9][12][2]. Filtration by centrifuge demands high energy input and requires a large amount of the algae water mixture to be drained from the raceway, which halts the cultivation process. Yet, centrifugation remains the most efficient method thus far. In an economic analysis model created by Davis et al. [12], the allocation of capital costs associated with producing algae-based biofuel with a raceway pond scheme operation are compared using the following chart.
Clearly, the capital expenditure associated with harvesting and dewatering algae represents the highest relative cost assuming the current centrifuge techniques are employed. This notion is reinforced by Uduman et al. who claim that “Centrifugation is seen as the most efficient recovery technique, yet the energy and capital costs associated are unappealing” [13]. While the prospect of producing algae-based biofuel holds great potential, it seems as though the technical and economic status quo relegate the idea to mere wishful thinking. Researchers at the Office of Energy Efficiency and Renewable Energy are unsatisfied with this predicament as they express a desire for innovation stating, “Significant cost energy savings must be realized before any widespread implementation of this approach can be carried out” [2]. What current research fails to explore is the idea of changing the harvesting process as a whole and that re-imagining the approach might beget the breakthrough they have been looking for. Enter the proposition of using a cable-suspended robot to harvest algae from raceway ponds. The
Algae-Harvesting Cable-Suspended Robot first suggested by Williams et al. [14] presents a novel harvesting solution by mechanizing the process in a new way.

Cable Robots

Definition and Current Applications

In general, most robots can be classified as either a serial robot or a parallel robot. The fundamental difference between the two has to do with the configuration of the robot actuators and linkages. The linkages of serial robots are aligned in a sequential order from the base to the end-effector whereas parallel robots have multiple arms attached to the base and utilize both actuated and un-actuated joints to support the end effector and its load [15]. Basic examples of a serial and parallel robot are pictured below.

Serial robots are generally more capable of moving heavier loads within a larger workspace compared to the typical parallel robot. Parallel robots have the advantage of

Figure 5: ABB IRB 3400L [18], Adept Quattro s650H [19].
higher speeds and greater precision [15]. Cable-suspended robots are considered to be a special type of parallel robot with unique advantages and design challenges. Instead of rigid links, cable-suspended robots have flexible cables which cascade from overhead pulleys supported by towers or a fixed structural features to support the end-effector and its load. The active length of each cable is manipulated by a computer controlled reel which spools and unspools the cable in coordination as to guide the end-effector through a desired path of trajectory. Intuitively, cables are only capable of pulling objects while in tension and incapable of pushing objects in compression. A properly designed control system will prevent negative cable tension by avoiding all instances of robot singularities. Cable-suspended robot singularities occur as the end-effector approaches the reasonable workspace boundaries and can be quantified by a special Jacobian matrix product determinant described by Williams et al. [14]. Further information on singularities is discussed in later sections pertaining to the MATLAB simulation program. The proposed and practiced applications of cable-suspended robots include industrial material handling [14][16], search and rescue [17], and contour crafting construction [18] among others. Perhaps the most commonly known example of a large-scale cable-suspended robot is the SkyCam, the action sports aerial point-of-view broadcasting system [19]. The SkyCam uses a high definition camera as an end-effector which is levitated by four Kevlar-wrapped fiber optic data-transmitting cables. High precision reels located outside the stadium of interest simultaneously yet independently control the active cable lengths in order to position the camera at the perfect view point as depicted in the image seen below.
This researcher gained exclusive inside access to the SkyCam system operation at
the Ohio State University (OSU) Stadium located in Columbus, Ohio through a private
tour on the Friday evening preceding the OSU vs. Wisconsin football game during the
fall 2014 season. The following paragraph is a verbatim excerpt of the email
correspondence addressed to Dr. Bob Williams written by the author of this paper dated
September 8, 2014.

It wasn't until this past Friday just past noon that I was phoned by the SkyCam
team informing me that the onsite lead technician at OSU would be able to meet
with me that evening to give me a quick tour. I hit the road as soon as possible
and arrived in Columbus at the Shoe around 5 p.m. where one of the lead
coordinators, Micah, took me right out to the middle of the field where the
SkyCam camera was resting several feet off the ground. He spoke to the control
room through his walkie-talkie asking them to “send it” and the camera instantly
came to life as it began to zoom around the massive workspace smoothly and
silent. I was impressed with the unwavering stability of the camera platform even in the face of light cross-winds. The controllers brought the camera swooping in and maneuvered some elevated circles around us. As we walked off the field, the camera floated right in front of us the whole way until it approached the workspace boundary at which point it zoomed back high and center. The control room was merely a fold-out table with two operators toying with joy-sticks while fixated on several live feed monitors. One operator controlled the x-, y- and z-coordinates in real time while the other controlled the camera's yaw pitch and zoom. I watched as they practiced the opening kick-off scene until it looked right. We then moved outside the stadium to the parking lot where the Reels were located. I quickly realized that the Reel is the most important feature of the entire system. The Reel uses a method of precision spooling which tracks the current turn, layer of cable wrapped around the spool, among other critical input parameters. I am not sure how the system is initially calibrated. I also learned that the camera has an on-board stability system which manipulates the cables at their attachment points to handle small disturbances as to keep the camera platform level at all times. I was not allowed to takes pictures/videos of the control room or the camera close up, but the Reels are just sitting out there in the parking lot for anyone to see. I asked my tour guide about the cable sag and cable stretch which I learned had all been accounted for in the programming. All in all it was a wonderful experience which gave me a better sense of the necessary
design aspects required to design and build a successful cable-suspended robot system.

This experience proved to be a vital learning opportunity providing useful first-hand information regarding a similar robotic system that this project aims to decode. The key take-away here was the knowledge that cable-suspended robots can handle heavy end-effector loads, work in large variably-sized workspaces while also being a mobile and quickly deployed system. Since harvesting algae from a raceway pond presents challenges and requirements akin to that of broadcasting high definition game footage in the way that SkyCam does, it can be inferred that the algae-harvesting cable-suspended robot system is a worthwhile research and development endeavor. The following sections provide a closer look at the mechanics of the 4-axis cable-suspended robot system.

4-Axis Cable-Suspended Robot Proof-of-Concept Prototype

Williams et al. originally presented the concept of the Algae-Harvesting Cable-Suspended Robot and have gone into great detail regarding the kinematics, dynamics, and control method [14]. Further mechanical analysis and simulation has been performed by Needler [20] providing a solid theoretical foundation for a large-scale 4-axis cable-suspended robot. However, none of the collaborating predecessors have brought an actual working prototype into fruition which is where this project comes into play.

This document presents the simulation, design, and hardware implementation of a small-scale 4-axis cable suspended robot. This first of its kind prototype cable-suspended
robot (CSR) is intended to be a proof-of-concept system. It is worth noting that this prototype CSR is not expected to actually harvest algae.

Project Information

This section provides an opportunity to credit the inventors and collaborators involved with this project. The evolution of the cable-suspended robot knowledge base at Ohio University begins with Dr. Bob Williams, a world-renowned roboticist and educator who specializes in cable-suspended robotic systems among other facets of the study and design of robots. The invention of the Algae-Harvesting Cable-Suspended robot is credited to Dr. Bob Williams and Professor Jesus Pagan in collaboration with Dr. David Bayless. There have been many research publications focused on cable-suspended robots and various novel applications to come out of Ohio University. As for the algae-harvesting application, there remain gaps between how far the project has come and what needs to be accomplished to fully realize the Algae-Harvesting Cable-Suspended Robot. It is important to reiterate the importance, significance and impact such an innovation represents not only in our eventual complete understanding of this novel robot, but how it will be a tool to help unlock a cleaner energy future. So far, this is the first project to actually design and build a small-scale prototype and implement simple hardware control.
CHAPTER 2: THESIS OBJECTIVES

There are four main objectives for this project. They are as follows:

- Create an interactive, versatile and user-friendly computer simulation program of the 4-axis cable suspended robot using MATLAB and Simulink.
  - Modes of operation include static snapshot configuration, 3-D trajectory and 2-D planar trajectory.
  - User will have ability to manipulate, configure, review, and analyze all important physical results including: active cable lengths and velocities and accelerations, active cable tensions, active motor positions and velocities and accelerations, active cable-winch positions and velocities and accelerations, end-effector kinematics (position, velocity, acceleration, jerk and snap), global Cartesian end-effector position, robot Cartesian stiffness, and singularity evaluation for various 3-D and 2-D trajectories and static poses.

- Create a model of a fully functional hardware system designed to handle the maximum loads predicted by the MATLAB simulation using SolidWorks.
  - Prototype design will be subject to the maximum loads induced by tension as predicted by the simulation using finite element analysis to ensure structural stability.
  - Perform static analysis to ensure that expected cable tensions will not tip the structure over and to further verify that motor torque limits will not be exceeded.
- Construct the hardware system based on the design.
  - The physical embodiment of the SolidWorks design will consist mainly of stock materials and built with the tools and machines available in the Ohio University Stocker labs.
  - Configure robot in a planar testing configuration.

- Program the Parker PAC motor control system to independently yet simultaneously control two axes in order to successfully execute a pre-programmed smooth trajectory path in the planar workspace based on user-input of final time and position.
  - A 10-foot straight line trajectory of fixed height will be performed displaying linear motion and constant velocity with periods of smooth acceleration at the beginning and end.
  - Motion will be verified using a light-weight string attached to end-effector and rotary encoder such that when displacement occurs, the string will turn the encoder. The rotational position vs. time data will be translated to linear displacement vs. time and plotted in MATLAB as to be compared with the ideal case given by MATLAB in objective 1.
  - Test trajectory of the end-effector will be repeated enough times to ensure that the location tolerance falls within an acceptable margin.
CHAPTER 3: CABLE ROBOT MECHANICS AND CONTROL

Kinematics

The simplest way to introduce and study the motion of the 4-axis CSR is to first perform a position-only kinematic analysis wherein the forces which cause motion are temporarily ignored. In order to describe the relative position of the end-effector with respect to the ground or fixed base, a set of Cartesian reference frames are established. The scope of this analysis does not include orientation control of the end-effector which is conveniently modeled as a point mass.

As previously stated, much of the kinematic derivations and analysis have already been established by Williams, Pagan, Needler et al. in [14] and [20]. For this reason, some steps in the following derivations have been deliberately omitted as a similar approach is utilized to adapt the various kinematic equations in order to comply with the reference frame conventions used in this analysis. Figure 7 illustrates the 4-axis CSR in a static pose generated by MATLAB and is overlaid with color-coordinated text to identify the reference frames and parameters of interest.
The base Cartesian reference frame \(\{a\}\) places the origin at the bottom left corner of the reachable workspace to ensure that all end-effector positions within the workspace are expressed in a positive sense. The end-effector position is denoted as \(^AP_P = \{x\ y\ z\\}^T\) indicating the positive XYZ Cartesian displacement from the fixed base frame \(\{a\}\) measured in feet. Each vertical tower is offset a net distance of \(D = [D_x\ D_y\ 0]^T\) from each corner of the workspace where \(D_x = D_y\), a user-adjustable distance also measured in feet. This slight offset helps to make sure each cable remains in positive tension i.e. if the end-effector were commanded to a location directly beneath the \(ith\) tower that would mean that the \(ith\) cable would bear the entire load causing the remaining cable tensions to approach zero. Under these conditions, the location of the top of each tower is expressed as such:

\[
^aB_{11} = \{-D_x\ -D_y\ \ h_1\}^T \\
^aB_{21} = \{-D_x\ \ L+D_y\ \ h_2\}^T
\]
\[ a_{B_{31}} = \begin{bmatrix} W+Dx & L+Dy & h_{3j} \end{bmatrix}^T \quad a_{B_{41}} = \begin{bmatrix} W+Dx & -Dy & h_{4j} \end{bmatrix}^T \]

The syntax used implicates that the location of each tower peak is expressed by a \([3\times1]\) vector giving the XYZ Cartesian position relative to the origin where \(h_i\) is tower height measured in feet and the width \(W\) and length \(L\) of the workspace are aligned with \(\hat{X}_a\) and \(\hat{Y}_a\) respectively. These tower pulley positions are distinguished from the tower base by the first subscript of \(B_{ij}\) \((i = 0, 1)\) and the second subscript identifies the tower \((j = 1-4)\).

When developing a relationship between the active cable lengths and the position of the end-effector, it is important to specify which of these two parameters is defined by the user. If explicitly given the end-effector position and asked to calculate the active cable lengths, this is considered the inverse pose kinematics (IPK) problem. The forward pose kinematics (FPK) problem is stated conversely wherein the active cable lengths are given in order to determine the end-effector position. Since the user will always be controlling the end-effector position in this application, we focus mainly on the IPK solution set. Although the FPK solution is still important because it allows for a circular check as a way to independently confirm the original IPK solution.

Given the end-effector position, \(P\), the four active cable lengths \(L_i\) can be directly calculated as they are simply the Euclidean norm of the vector connecting the top of each tower \(B_{ij}\) to point \(P\) as expressed by equation 1.

\[
L_i = ||a_{B_{ij}} - a_P|| \quad j = 1, 2, 3, 4
\]  

The active cable lengths, measured here in feet, are especially pertinent when performing a snapshot simulation example. The hardware system is subject to several
physical limitations such as maximum cable length, maximum cable rate \( \dot{L} = [\dot{L}_1 \dot{L}_2 \dot{L}_3 \dot{L}_4]^T \) and maximum cable acceleration \( \ddot{L} = [\ddot{L}_1 \ddot{L}_2 \ddot{L}_3 \ddot{L}_4]^T \) - all which are both ultimately limited by the performance capabilities of the actuator/winch assembly anchored at the base of each tower. The actuators used in the hardware system of this project are DC brushless servo motors, which will be described in further detail in later sections. As for the equations used to solve for the cable rates and cable accelerations, they have been thoroughly described and derived by Williams et al. in [14] and also by Needler in [20]. These governing equations have been deliberately omitted from this paper as to focus primarily on the new contributions created in this project, including, but not limited to, the trajectory generation kinematics, MATLAB simulation program, SolidWorks design and finite element analysis (FEA), hardware construction and hardware control implementation. The aforementioned kinematic equations can also be located within the MATLAB simulation code scripts and functions written by the author, all of which are found in the appendix.

Pseudostatics

Since high end-effector velocity is not necessarily desired for this application, a pseudostatic assumption is used to simulate the approximate behavior of the 4-axis CSR at low speeds. This is to say that the motion of this 4-axis CSR will be slow enough that the dynamic effects are small enough to be ignored. Further, this implies that the active cable tensions can be evaluated under the assumption that the system is always in static equilibrium \( \sum F = 0 \). A general free body diagram (FBD) of the end-effector is shown below in figure 8.
Again, this static analysis covers translation only movement and does not consider rotational movement. The four active cable tension vectors $\{T\} = \{t_1 \ t_2 \ t_3 \ t_4\}^T$ support the weight $W$ [lbf] of the end-effector and are also influenced by the externally applied force $\{aF\} = \{f_x \ f_y \ f_y\}^T$, depicted here as a curvy vector to indicate that this force is constantly changing in magnitude and direction. In the real-world, this external force can be attributed to things such as wind resistance, drag force of the algae raceway pond water and even brief instances of physical contact with the raceway structure itself. In accordance with the static equilibrium assumption ($\sum F = 0$), the summation of all forces acting upon the end-effector in relation to the FBD of figure 8 are expressed below in equation (2).

$$\sum_{i=1}^{4} \{a t_i\} + \{a F\} + m \{a g\} = \{0\}$$
\[ \sum_{i=1}^{4} t_i \{^a \mathbf{L}_i \} + \{^a \mathbf{F} \} + m \{^a \mathbf{g} \} = \{0\} \] (2)

Here we see that the active cable tensions \( \{^a \mathbf{T}\} = \{t_1 \ t_2 \ t_3 \ t_4\}^T \) can be expressed as a vector incorporating both magnitude and direction or as the product of the cable tension magnitude \( t_i \) and the unit vector oriented in the direction of the corresponding cable. Note that all vectors used in equation (2) are represented with respect to the base frame of reference \( \{^a\} \). The third term of equation (2), \( m \{^a \mathbf{g} \} \) is the total weight of the end-effector in pounds where \( \{^a \mathbf{g} \} = \{0 \ 0 \ -g\} \) with \( g = -32.3 \text{ ft/s}^2 \). Williams et al. [14] simplify equation (2) and solve the IPK for the 4x1 matrix of active cable tensions. The result is shown by equation (3).

\[ \{^a \mathbf{T}\} = -[^a \mathbf{A} ]^\dagger \{^a \mathbf{F} + m \mathbf{g}\} \] (3)

Equation (3) utilizes the pseudoinverse of the statics Jacobian matrix \( [^a \mathbf{A}] \) which is denoted above as \( [^a \mathbf{A} ]^\dagger \) and define by:

\[ [^a \mathbf{A}]^\dagger = [^a \mathbf{A}]^T ([^a \mathbf{A}] [^a \mathbf{A}]^T)^{-1} \] (4)

Where \( [^a \mathbf{A}] \), the statics Jacobian matrix, is a non-square 3x4 matrix which essentially stores information regarding the instantaneous end-effector position as well as the active cable lengths which is defined as:

\[ [^a \mathbf{A}] = \begin{bmatrix}
\left[ \frac{a_{B11} - a_{PP}}{L_1} \right] & \left[ \frac{a_{B12} - a_{PP}}{L_2} \right] & \left[ \frac{a_{B13} - a_{PP}}{L_2} \right] & \left[ \frac{a_{B14} - a_{PP}}{L_3} \right]
\end{bmatrix} \in \mathbb{R}^{3x4} \] (5)

Keep in mind that one of the biggest challenges in designing an effective controller for a 4-axis cable-suspended robot is to ensure that positive cable tension be maintained at all times of operation. Further, to design a system which operates smoothly and efficiently, it is not only important to maintain positive cable tensions, but also to optimize active cables tensions as to minimize the required input energy. Minimizing the
input energy is the name of the game when attempting to re-imagine the entire algae harvesting process in a successful way. For this reason, it becomes necessary to consider the optimization of certain factors such as cable tension, end-effector velocity and acceleration among other which brings into consideration the dynamics of the 4-axis cable-suspended robot. Should these dynamic influences prove dominant upon implementing control while testing the hardware system under the pseudostatic assumptions, steps towards incorporating the dynamic forces shall be added to the control structure.

Trajectory Generation Kinematics

In this section, a new method of generating end-effector trajectory motion is described, derived and illustrated with an example. This aspect of trajectory planning for the algae-harvesting cable-suspended robot project as a whole has not yet been closely examined. The motion of the 4-axis algae-harvesting CSR end-effector has been simulated to move at a constant incremental rate which ignores the notion of starting and ending at rest while exhibiting some type of smooth motion profile in between said points. End-effector motion not only start and end at rest in between stages of a pre-planned or even manually controlled trajectories, but a maximum velocity be achieved and maintained between phases of exceptionally smooth acceleration and deceleration. This section will break down the development of the 8th order polynomial profile followed by the position versus time during the phases of acceleration and deceleration of the 4-axis CSR end-effector.
As previously mentioned, an inverse pose kinematic approach shall be used to control the behavior of the 4-axis algae-harvesting CSR alpha prototype. The user will always be in control of the end-effector position while the computer determines the cables lengths required. In order to create this exceptionally smooth motion, we use several actuator constraints to ensure that functions of position, velocity, acceleration and jerk are continuous. The ultimate goal is to match the initial and final position of the end-effector as defined by the user. As I approach the final position, it may be necessary to execute certain trajectory stages at a specified target velocity or even a certain time interval. These constraints, initial and final position as well as final time, are the only required input from the user. In the early stages of developing the motion trajectory, the author chose velocity [ft/s] to be the driving factor. It has been envisioned that a certain globally defined ratio of the final time shall be spent during the acceleration phase and an identical period of time be spent on the deceleration. This allows the majority of time needed to execute a certain point-to-point trajectory to travel at a constant maximum velocity, which is compliant with the actuator limitations. The eight smooth motion actuator constraints are:

\[
\begin{align*}
\nu(t = 0) &= \nu_0 \\
\nu(t = t_1) &= \nu_{\text{max}} \\
a(t = 0) &= 0 \\
a(t = t_1) &= 0 \\
j(t = 0) &= 0 \\
j(t = t_1) &= 0 \\
s(t = 0) &= 0 \\
s(t = t_1) &= 0
\end{align*}
\]

Where \(\nu, a, j, \) and \(s\) are velocity, acceleration, jerk, and snap respectively. Seen on the left are the initial conditions desired at the beginning of each trajectory and on the
right we see the conditions that should be met at the onset of the constant velocity period of the trajectory occurring at $t_1$ (where $t_1$ is a pre-specified fraction of the total time prescribed for the current trajectory). Obtaining $t_1$ involves multiplying the total time prescribed for any given individual trajectory path by a factor denoted as PR or phase ratio. By default, the PR is set to 1/5 within the MATLAB simulation program, described in further detail in later sections of this paper. Equation (10) provide the relationship between the user-defined final time, $t_f$, and $t_1$ – the time at which end-effector reaches a maximum cruising velocity.

$$t_1 = PR \times t_f$$

$$t_2 = t_f - t_1$$

The second part of equation (10) is used to solve for $t_2$ or the time at which the phase of negative acceleration begins as to bring the end-effector back to rest.

With a total of eight constraints, a seventh-order polynomial for velocity can be fit to drive the end-effector from rest to some target maximum velocity, $v_{\text{max}}$. Four linear polynomial equations are obtained with eight unknown coefficients $a_i$, $i = 0,1,2,3,4,5,6,7$.

$$v(t) = a_7t^7 + a_6t^6 + a_5t^5 + a_4t^4 + a_3t^3 + a_2t^2 + a_1t + a_0$$

$$a(t) = 7a_7t^6 + 6a_6t^5 + 5a_5t^4 + 4a_4t^3 + 3a_3t^2 + 2a_2t + a_1$$

$$j(t) = 42a_7t^5 + 30a_6t^4 + 20a_5t^3 + 12a_4t^2 + 6a_3t + 2a_2$$

$$s(t) = 210a_7t^4 + 120a_6t^3 + 60a_5t^2 + 24a_4t + 6a_3$$

To solve for the unknown polynomial coefficients, we apply the motion constraints outlined by equations (6-9). The initial conditions seen on the left of
equations (6-9) implicitly solve for the first four polynomial coefficients. The remaining four coupled coefficients are solved for by using a 4x4 matrix/vector equation at $t = t_1$ as follows:

$$
\begin{bmatrix}
    t_1^7 & t_1^6 & t_1^5 & t_1^4 \\
    7t_1^6 & 6t_1^5 & 5t_1^4 & 4t_1^3 \\
    42t_1^5 & 30t_1^4 & 20t_1^3 & 12t_1^2 \\
    210t_1^4 & 120t_1^3 & 60t_1^2 & 24t_1
\end{bmatrix}
\begin{bmatrix}
a_7 \\
a_6 \\
a_5 \\
a_4
\end{bmatrix}
= 
\begin{bmatrix}
v_{\text{max}} - v_0 \\
0 \\
0 \\
0
\end{bmatrix}
$$

(15)

Altogether, this yields the single seventh-order polynomial solution set:

$$
\begin{align*}
a_0 &= v_0 \\
a_1 &= 0 \\
a_2 &= 0 \\
a_3 &= 0 \\
a_4 &= \frac{35}{t_1^3} (v_{\text{max}} - v_0) \\
a_5 &= -\frac{84}{t_1^5} (v_{\text{max}} - v_0) \\
a_6 &= \frac{70}{t_1^6} (v_{\text{max}} - v_0) \\
a_7 &= -\frac{20}{t_1^7} (v_{\text{max}} - v_0)
\end{align*}
$$

(16)

These symbolic solutions shown by (16) have been solved for using MATLAB. Clearly, there is one very important equation that is missing which is the polynomial fit for end-effector position as a function of time. The position is simply the integral of velocity which is given by equation (11), therefore to obtain the eighth-order position equation we perform a relatively straight forward integration using reverse chain rule.

$$
\begin{align*}
x(t) = \frac{1}{8} a_7 t^8 &+ \frac{1}{7} a_6 t^7 + \frac{1}{6} a_5 t^6 + \frac{1}{5} a_4 t^5 + \frac{1}{4} a_3 t^4 + \frac{1}{3} a_2 t^3 + \frac{1}{2} a_1 t^2 + a_0 t + x_0
\end{align*}
$$

(17)

The integration constant tacked on to the end of equation (17) is $x_0$ which represents the initial position. Ideally, the user need only input the desired position for the end-effector to move to along with the time at which that position be reached.
However, the polynomial coefficients are solved for in part using a target velocity, $v_{\text{max}}$, as a critical motion constraint. This essentially sets the target velocity as an independent variable causing the final position to be a dependent variable. In order to switch the roles of these variables, a simple relationship between the desired final position, $x_f$, and the maximum velocity, $v_{\text{max}}$, of the 4-axis CSR end-effector is derived.

To set up the derivation for the aforementioned relation, an example of a nominal velocity profile is shown below created under the following conditions:

\[
\begin{align*}
t_0 &= 0 \, [\text{sec}] & \quad & t_f = 10 \, [\text{sec}] \\
v_0 &= 0 \, [\text{ft/s}] & \quad & v_{\text{max}} = 1 \, [\text{ft/s}] 
\end{align*}
\]

*Figure 9: Constant velocity profile.*

It can be seen that with $PR = \frac{1}{5}$, $t_1 = 2$ and $t_2 = 8$. The seventh-order velocity polynomial curve is present between $t_0$ and $t_1$ and mirrored between $t_2$ and $t_f$. As previously mentioned, position is the integral of velocity, which can be obtained by reverse chain rule on the velocity polynomial or by using geometry to solve for the area under the velocity curve seen in figure 9. The second option is not always so simple, but for this case we can use it to calculate the exact area under the curve. The velocity profile seen in figure 9 is broken up into three simple shapes whose areas are easily calculated.
Now, the simplified relationship between final position and maximum velocity is symbolically derived.

\[ x_f = area\ of\ region\ 1 + area\ of\ region\ 2 + area\ of\ region\ 3 \]  \hspace{1cm} (18)

By symmetry:

\[ area\ of\ region\ 1 = area\ of\ region\ 3 \]  \hspace{1cm} (19)

\[ x_f = 2 \times (area\ of\ region\ 1) + area\ of\ region\ 2 \]  \hspace{1cm} (20)

Where:

\[ area\ of\ region\ 1 = \frac{1}{2} \times t_1 \times v_{\text{max}} \]  \hspace{1cm} (21)

\[ area\ of\ region\ 2 = (t_2 - t_1) \times v_{\text{max}} \]  \hspace{1cm} (22)

\[ t_1 = PR \times t_f \]  \hspace{1cm} (23)

\[ t_2 = t_f - t_1 = t_f - PR \times t_f = t_f(1 - PR) \]  \hspace{1cm} (24)

By substitution, equation (20) becomes:

\[ x_f = 2 \times \left( \frac{1}{2} \times PR \times t_f \times v_{\text{max}} \right) + \left( t_f(1 - PR) - PR \times t_f \right) \times v_{\text{max}} \]

\[ x_f = PR \times t_f \times v_{\text{max}} + \left( t_f(1 - 2PR) \right) \times v_{\text{max}} \]

\[ x_f = v_{\text{max}} \times t_f(1 - PR) \]  \hspace{1cm} (25)

Upon solving (25) for \( v_{\text{max}} \):

\[ v_{\text{max}} = \frac{x_f}{t_f(1-PR)} \]  \hspace{1cm} (26)

\textit{Figure 10:} Constant velocity profile divided into geometric sections.
By equation (26), it can been seen that $x_f$ and $t_f$ are now the independent variables while $v_{max}$ has become the dependent variable. This means that a full trajectory can be generated supplemented only by the user-defined final position and final time. A numerical example is provided here.

**Given:**
$$x_0 = 0 \ [ft], \ x_f = 10 \ [ft], \ t_f = 10 \ sec$$

**Find:**
$x(t)$ for smooth trajectory generation. Plot $x(t), v(t), a(t), j(t), s(t)$

**Result:**

For $0 < t < t_1$ (phase 1)

$$x(t) = -24.41(10^{-3})t^8 + 195.31(10^{-3})t^7 - 546.88(10^{-3})t^6 + 546.88(10^{-3})t^5$$

$$v(t) = -194.31(10^{-3})t^7 + 1.37t^6 - 3.28t^5 + 2.73t^4$$

$$a(t) = -1.37t^6 + 8.20t^5 - 16.41t^4 + 10.94t^3$$

$$j(t) = -8.20t^5 + 41.02t^4 - 65.63t^3 + 32.81t^2$$

$$s(t) = -41.02t^4 + 164.06t^3 - 196.88t^2 + 65.63t$$

For $t_1 < t < t_2$ (phase 2)

$$x(t) = 1.25t + x(t_1)$$

$$v(t) = 1.25$$

$$a(t) = 0$$

$$j(t) = 0$$

$$s(t) = 0$$

For $t_2 < t < t_f$ (phase 3)
Graphically, the behavior of $x(t)$, $a(t)$, $j(t)$, and $s(t)$ for phase 3 is flipped vertically and horizontally and offset accordingly, whereas the behavior of $v(t)$ is only flipped horizontally as seen in the plot below.

![Sub-plotted trajectory generation graphs for nominal example.](image)

*Figure 11*: Sub-plotted trajectory generation graphs for nominal example.

Using this method of trajectory generation, the end-effector of the 4-axis CSR can be steered in any desired direction within the workspace while maintaining smooth motion. Examples of several different trajectory paths are shown later in the MATLAB simulation section of this paper. This method may require further development to ensure that any given trajectory does not push any of the actuators beyond their physical limits. These types of considerations should be researched and optimized in the future in order to create a more robust and reliable trajectory generation. For the time being, this method
represents a start to an aspect of the algae-harvesting cable-suspended robot which had not yet been explored.

Method of Control

Since the basic goal regarding motion control for this prototype system is to achieve point-to-point trajectories within a desired time, the method of control should be as simple as possible in terms of user input. Once the end-effector is jogged or placed into position and cable tension is enacted at a calibration point, this point shall be considered the Cartesian origin on the workspace. From here, the user will be able to specify as many points within the XYZ Cartesian workspace with a corresponding time at which the end-effector should arrive at said points. The rest of the calculations and control will be performed by the computer. A straight-line trajectory will be computed between each point based on the kinematic equations outlined in the previous section. Each individual motor position kinematic equations will be discretized into time steps by the controller which then executes the trajectory by simultaneously yet independently actuating each motor.
CHAPTER 4: MATLAB AND SOLIDWORKS

MATLAB Simulation

The MATLAB simulation created here represents the core of the theoretical basis for this project. Since the analysis of this objective is so important, it is presented in sections including purpose, how to use, snapshot examples and trajectory examples. Solved numerical examples are provided for the snapshot and trajectory sections.

Purpose

The purpose of this program is to simulate the kinematics and dynamics of a 4-axis cable suspended robot. As a whole, this simulation program (aptly named The Cable Suspended Robot Simulator, or CSR SIMULATOR) represents the fulfillment of one of the key objectives outlined by the Programmer’s Master Thesis. With this simulation, we explore the kinematic and dynamic limits and constraints that arise when the CSR is configured in a user-defined static pose or a pre-programmed or manually-controlled end-effector trajectory. 3-D and 2-D plots show equally scaled and life-like depictions of the real-world hardware system configured by user-defined parameters. In addition to the life-like visual aid, the simulator generates tables and graphs which present important physical results relating to the corresponding mode of operation. High end-effect velocity is not necessarily desired for this application and thus a pseudostatic assumption is made to simplify the calculations for the trajectory modes of operation. Results and findings obtained from the CSR SIMULATOR are intended to help drive the design process of the real-world hardware prototype Cable Suspended Robot also built by the Programmer.
How to Use

The CSR simulator is structured in a graphical user interface (GUI) scheme utilizing a network of user input menus. The "Run" button sends the user to the Run Menu at which point the mode of operation can be selected. Depending on which mode has been selected, the user will be prompted to select further options and will be given the opportunity to manipulate certain system parameters. To exit the program at any time, locate and click the EXIT button. Seen below is a screenshot of the main menu followed by the RUN menu.

Figure 12: CSR simulator, main menu.
Snapshot Example

The first mode of operation to explore is the snap shot mode. Here the user is first prompted to enter several adjustable configuration parameters such as workspace size, tower height, end-effector location, and end-effector weight. A screenshot of the snap shot menu is seen below.
The values shown are the default values which can be changed by the user. If the user inputs an end-effector position which lies outside the workspace, an error will inform them that doing so breaks the rules and will reset the position to the geometric center of the workspace automatically. Upon pressing ‘OK’, the user is presented with a 3-D view of their configuration.
Figure 15: 3-D and 2-D orthographic views of snap-shot example.

Note the options available at the bottom in the screen in figure 15. Showing the data table provides the user with important physical results pertaining to their specific configuration. The data table for this configuration is:

Figure 16: Snap shot data table example.
The snap-shot mode of operation provides the user with the means to visualize and physically analyze any reasonable configuration. This feature is easy to use and should be employed before actually configuring the real-world hardware system to check that cable lengths do not go beyond their physical limits and that the cable tensions do not cause failure in terms of tipping the towers over or breaking the wood or even require a holding motor torque which is beyond the motors ability. Further, this feature can be used to find locations within a workspace where the Cartesian stiffness is lowest and also if the robot is experiencing a singular condition as indicated by the determinant of \[A*A^T\], which should always be avoided. A singular condition will arise if any cable fails to maintain positive tension or if any cable approaches infinite cable tension (this would happen as the height of the end-effector approaches the height of the tower). To see how these and other physical parameters behave throughout a specific motion, a trajectory analysis feature is used.

Trajectory Examples

The second mode of operation brings the 4-axis CSR to life with several animated trajectory examples. A trajectory menu first prompts the user to choose from a list of available pre-programmed trajectories.
Perhaps the simplest of the pre-programmed trajectories is the straight line. A vector trail follows the end-effector to better illuminate the trajectory path in space.

Figure 17: CSR simulator, trajectory menu.

Figure 18: Straight line trajectory.
The user has the opportunity to view the report after watching the animated trajectory which produces the following plots which include: active cable lengths and velocities and accelerations, active cable tensions, active motor positions and velocities and accelerations, active cable-winch positions and velocities and accelerations, end-effector kinematics (position, velocity, acceleration, jerk and snap), global Cartesian end-effector position, robot Cartesian stiffness, and singularity evaluation.

*Figure 19: Determinant of \([A \ A^T]\) vs. time for straight line.*
Figure 20: Robot Cartesian stiffness vs. time for straight line.

Figure 21: Cartesian trajectory vs. time for straight line.
Figure 22: Active cable tensions vs. time for straight line.

Figure 23: Active cable lengths vs. time for straight line.
Figure 24: Active cable velocities vs. time for straight line.

Figure 25: Active cable accelerations vs. time for straight line.
**Figure 26**: Active motor encoder pulse count vs. time for straight line.

**Figure 27**: Active motor angular velocities vs. time for straight line.
Figure 28: Active motor angular accelerations vs. time for straight line.

Figure 29: Active cable reel/winch angular velocities vs. time for straight line.
Figure 30: Active cable reel/winch angular accelerations vs. time for straight line.

The CSR Simulator is also capable to commanding smooth motion in curved paths as illustrated by the circle trajectory. A screen shot of the completed circle trajectory is given below.

Figure 31: Completed circular trajectory.
Upon choosing to ‘View Report’, the user is again presented with the following plots which display all relevant physical data pertaining to this specific trajectory.

**Figure 32:** Determinant of $[A A^T]$ vs. time for circle.

**Figure 33:** Robot Cartesian stiffness vs. time for circle.
Figure 34: Cartesian trajectory vs. time for circle.

Figure 35: Active cable tensions vs. time for circle.
**Figure 36:** Active cable lengths vs. time for circle.

**Figure 37:** Active cable velocities vs. time for circle.
Figure 38: Active cable accelerations vs. time for circle.

Figure 39: Active cable reel/winch angular velocities vs. time for circle.
Figure 40: Active cable reel/winch angular accelerations vs. time for circle.

Figure 41: Active motor encoder positions vs. time for circle.
Figure 42: Active motor angular velocities vs. time for circle.

Figure 43: Active motor angular accelerations vs. time for circle.
Further, the CSR Simulator can also execute multiple legs of motion throughout the same trajectory as shown in the harvest trajectory example. There are 11 legs of travel incorporated for this example. It is meant to simulate lifting the end-effector from a pick-up/drop-off point, and directing it over a raceway pond and subsequently dragging it though the water to collect algae. The harvest example simulates dragging the end-effector through two raceways before returning to the pick-up/drop-off point. One cool feature of this trajectory is that the color of the end-effector smoothly transitions from white to blue-green as it ‘harvests’ the algae, and even transitions to a green color as it lifts itself out of the raceway to simulate the dewatering effect. Snap shots of the harvest trajectory in action are given below.
Figure 44: Harvest trajectory example in sequential order.
The final feature, beyond the other 3-D trajectories, is the 2-D planar configuration mode which shows only two towers moving the end-effector across a straight 10-foot-long horizontal trajectory. Due to only having two motors and drives available with the Parker PAC control system at the time of testing, this planar configuration is used for the real-world proof-of-concept trajectory. For this reason, the active cable tensions over time in the planar case are used for the model designed in objective 2 to ensure that system failure will not occur throughout the duration of the real-world trajectory test completed in object 4. Images of the initial and final positions followed by a plot of the cable tensions vs. time are shown below.

Figure 45: Initial and final positions of the planar configuration.
An end-effector weight of 15.6 pounds was used in the planar case since this is the weight of the bowling ball end-effector that is used for the real-world test. The magnitude of maximum cable tension in either cable is 15.9 pounds. When considering the failure modes of one of the towers tipping inward or yielding of the material due to bending stress it is more important to consider the perpendicular component of tension acting on the top of each tower which is a function of both tension and the angle between the tower and the cable. For this reason, the perpendicular force acting at the top of each tower is plotted versus time.
Interestingly, the perpendicular force acting on the top of each tower is equivalent for each tower with a maximum value of 13.4 pounds. These key parameters discovered in this section are used in the analysis of the design developed in objective 2.

**SolidWorks Model**

Before any real construction began on the 4-axis CSR prototype, many solid 3D models were generated through an iterative design process using SolidWorks. The overall size and scale of the final design is intended to be compatible with the small-scale raceway ponds at the research greenhouse of the Institute for Sustainable Energy and the Environment (ISEE) located at Ohio University. After many design iterations, a suitable tower assembly is chosen.
Physical Model

The design parameters considered for this design focused mainly on functionality, mobility, adjustability, and simplicity. The accepted prototype model is rendered below noted for its compliance with the aforementioned design parameters.

Figure 48: SolidWorks CSR prototype tower model assembly.

Figure 48 shows one fully assembled tower of which there are four required for the full robot configuration. The main material of choice is 2x3 dimensional stud lumber,
chosen for its strength and workability. The height of each tower is fixed at approximately six feet. There is a motor/winch assembly anchored to the base and a pulley mounted at the tower peak with a steel L-bracket whose height can be adjusted if desired. Angled side and back supports help to resist tipping and bending of the main vertical upright due to imposed stress attributed to cable tension. The back support provides a platform on which to place a counterweight as to prevent the tower from tipping inwards in the direction of the cable tension. The robot fasteners consist mainly of deck screws and ¼ inch all-thread secured with washers and nuts. A full robot assembly is shown below depicting the robot operating at the ISEE small-scale raceway ponds. The raceways are modeled to scale.

Figure 49: Full CSR prototype assembly in the ISEE research greenhouse.
Again, it may be worth mentioning that actually harvesting algae, even on a small-scale as depicted in figure 49, is outside the scope of objectives for this project.

**Failure Criterion**

Before construction of the prototype towers began, it was necessary to ensure that the hardware would not fail under normal operating conditions due to any of the following failure criterion: cable tension causing tower to tip inwards, cable tension causing wood to yield, or cable tension exceeding the torque capabilities of the motor. The normal operating conditions considered here are that of the trajectory used in objective 4: 2-axis straight line trajectory in the planar workspace. The first failure criterion considered is tipping of the tower due to the perpendicular component of cable tension. A simple static analysis is used to ensure that tipping will not occur. Note that point ‘A’ is assumed to be fixed to the ground. In reality, the weight of the bucket placed on the back of the tower induces a sufficient magnitude of frictional force between the tower base and floor as to prevent slipping. However, the ideal case is a tower base that is fully fixed to the ground which would prevent tipping and slipping. The free body diagram for this statics problem is shown below.
Figure 50: Free body diagram of single tower.

The maximum allowable perpendicular force induced by tension is calculated by summing the moments about point ‘A’ under the following assumptions:

\[ W = \text{weight of counterweight} = 100 \text{ pounds} \]

\[ N = \text{normal force} = 0 \text{ lbf (tipping condition)} \]

\[ L = \text{length of tower} = 42'' \]

\[ h = \text{height of tower} = 72'' \]

No rotation
The sum of moments statics equation is given as:

$$\sum M_A = (F_{1,max} \cdot h) + (N \cdot L) - (W \cdot L) = 0$$  \hspace{1cm} (27)

$$F_{1,max} = \frac{(W \cdot L) - (N \cdot L)}{h} = \frac{(100 \text{ lbf} \cdot 42") - (0 \text{ lbf} \cdot 42")}{72"} = 58.3 \text{ lb}$$  \hspace{1cm} (28)

Comparing the result of equation (28) with the result of the maximum predicted perpendicular force of 13.4 pounds of tension, this yields a factor of safety (FOS) of 4.35. This indicates that neither tower should tip over during the real-world trajectory test.

The next failure criterion explored is material failure or the breaking of the wooden upright due to bending stress imposed by cable tension. A finite element analysis (FEA) was performed on the key components of the structure using SolidEdge modeling software. The Von Mises failure criterion is used for this analysis. Further, wood is not an isotropic material, but for the purposes of this FEA analysis, it is assumed to be as such. The stress results of FEA are shown in the image below.
Figure 51: FEA stress visualization of single tower.

Table 2: FEA stress results.

### 11.2 Stress Results

<table>
<thead>
<tr>
<th>Extent</th>
<th>Value</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum</td>
<td>2.29e-007 ksi</td>
<td>43.125 in</td>
<td>-2.000 in</td>
<td>3.000 in</td>
</tr>
<tr>
<td>Maximum</td>
<td>0.199 ksi</td>
<td>11.000 in</td>
<td>1.000 in</td>
<td>31.177 in</td>
</tr>
</tbody>
</table>
According to table 2, the FEA report indicates that the largest stress encountered is 0.199 ksi or 1.37 MPa. It is well known that there is a lot of variance when it comes to the material properties of most wood. For this reason, an yield strength of pine wood is assumed to be approximately 1.44 ksi or 9.96 MPa [21]. Under this assumption, a FOS for material failure under these conditions is 7.27. Therefore, no material failure should occur under the normal operating conditions tested in this experiment.

The final failure criterion to be considered is that of cable tension causing enough resistance against the motor that the motor’s torque limit is reached. The motor will automatically shut down if too large of a torque value is experienced. A simplified FBD of the winch/motor coupling which relates cable tension to motor torque is displayed below.

![Figure 52: Simplified FBD of winch/motor coupling.](image)

Where \(T_{\text{max}}\) is the maximum predicted cable tension of 15.9 lb., \(r_w\) is the average winch radius of 1.5”, \(r_m\) is the motor axis radius of 0.25”, \(t_w\) is the torque acting
on the winch, \( t_m \) is the torque acting on the motor and \( n \) is the gear reduction ratio of 1:153. A static analysis is performed to solve for the motor torque with the following equations.

\[
\sum M = T_{\text{max}} r_w - \tau_w = 0
\]  

(29)

Where

\[
\tau_w = n \tau_m
\]  

(30)

Combining equations (29) and (30) gives:

\[
\sum M = T_{\text{max}} r_w - n \tau_m = 0
\]

\[
\tau_m = \frac{T_{\text{max}} r_w}{n} = \frac{15.9 \text{ lb*in}}{153} = 0.16 \text{ lb*in}
\]  

(31)

Comparing the result of equation (31) to the rated motor torque of approximately 16.93 [lb*in], this gives a torque factor of safety of 108.5. This indicates that the motors should have no problem dealing with the tension induced torque expected for the real-world trajectory test. Since the three failure criterion were cleared, the system was constructed.
CHAPTER 5: HARDWARE IMPLEMENTATION AND CONTROL

Hardware Construction and Configuration

Based on the final design scheme created in SolidWorks, the actual materials were purchased and constituent parts of each tower assembly were machined and assembled. Pictured below is fully assembled tower.

Figure 53: Hardware tower assembly with motor/winch, pulley and cable.
A counterweight in the form of a large bucket containing a 50-pound bag of Quikrete concrete or sand has been placed on the back support. The motor/winch assembly is anchored to a removable wooden insert piece which is secured with large deck screws and ¼ inch all-thread rod. This basic configuration has been manually controlled to perform jogging motions with a hand-held drill and also with a single-axis Galil motor controller. A basic lift/lower open-loop motor control program was written and used to manipulate the vertical height of a weighted end-effector. However, as stated in the objectives section, the preferred control system is the Parker PAC motor controller although the Galil controllers remain a potential option for executing very simple test trajectories such as the straight line.

A full static assembly has been configured in an outdoor setting for a workspace of 25 x 25 feet pictured below.

*Figure 54: Static configuration of the 4-axis CSR prototype.*
The cables were manually tensioned using a hand-held drill as the motor controllers were not yet ready for operation at the time this photo was taken. The end-effector used here is a large bucket containing a few bricks to provide adequate cable tension.

As previously mentioned, only two Parker motors and drives were available at the time of testing and therefore the planar case is used for the real-world trajectory test. The configuration assembled for the planar case is pictured below.

![Figure 55: Planar configuration of cable suspended robot.](image)

A total distance of 12 feet separates the bases of each tower leaving room for the 10 foot long straight line trajectory in between the towers. Since only two towers are
used here, an extra counterweight is used on either tower increasing the total weight to 100 pounds.

Another hardware problem was coupling the Parker motors to the winches. This problem was solved by first machining aluminum shaft couples to the appropriate inner diameters such that they could clamp tightly around the motor shaft and the winch shaft. Then, to prevent rotation of the motor itself while spinning, an outer coupling fixture was machined and welded together which joins the motor to the winch via their respective mounting holes. This subassembly is pictured below.

Figure 56: Motor and winch coupling subassembly.

With two motor and winch subassemblies created, the full planar testing configuration can be assembled. Caution tape was used to mark out the fixed locations of
each tower and also to help indicate the distance traveled by the end-effector. An image of the full planar configuration is provided here.

*Figure 57: Full planar hardware and software configuration.*
The last critical piece of hardware required for testing is the encoder wheel whose purpose is to hold a small rotary encoder at a fixed position while allowing it to rotate freely. Due to the complexity and uniqueness of this piece, it was modeled in SolidWorks and subsequently printed out on a 3-D printer. Several images of the encoder wheel are provided below.

*Figure 58: Encoder wheel with Arduino controller.*
Figure 59: Backside of encoder wheel.

With the entire hardware system and Parker PAC software configured in for the planar trajectory test, it is time to develop the control program which will bring the cable suspended robot to life.
Hardware Control

The control software used in this project is the Parker Automation Control (PAC) system.

There are many forms of programming languages that can be used for this software but the one chosen for this project is called Structured Text (ST). The main built-in function block utilized to execute the test trajectory is the MC_Position_Profile function block. This built-in function block takes a paired motor position / time increment array as input and control the motion of the motor by matching the desired position at each time increment. So, by virtue of the spacing between desired motor positions occurring at fixed time intervals, the motor kinematics are implied. The motion is made smooth by fitting the data with a polynomial in a similar way that the position

Figure 60: PAC and P-drive.
data was created initially using MATLAB. The position arrays for each motor axis were created within the ST programming environment and fed in the MC_Position_Profile function block. Upon enabling the drives, the position profiles were executed and the trajectory occurred. This was performed a total of six times.

For each test, the encoder wheel was calibrated and zeroed. As the end-effector moved away from the wheel, the attached string would unwind the encoder wheel in sync with the displacement of the end-effector. The data gathered by the encoder was processed by an Arduino Uno microcontroller which outputs the time stamp for every time that the encoder position changed. Since the encoder can only process 30 ticks per revolution, this method of trajectory verification was not meant to be extremely precise but rather confirm that the trajectory was maintaining a constant velocity and smooth periods of positive and negative acceleration at the beginning and end respectively. Once filtered and conditioned, the encoder data is presented as a position versus time plot using MATLAB. As a reminder, the ideal shape of the position versus time plot is given by the MATLAB simulation program in the following image.
Figure 61: Ideal position vs. time for planar trajectory.
CHAPTER 6: RESULTS

This chapter provides the data analysis of the position vs. time data gathered by the Arduion-powered encoder and tracking end-effector displacement vs. time information. Raw data gathered by the rotary encoder was originally displayed in the serial monitor of the Arduino programming environment which was copied into Microsoft Excel and read by MATLAB which filtered and plotted the data in an easy-to-interpret graphical scheme.

Straight Line Trajectory Results

The following plots display the experimentally gathered data. A linear fit was placed on the portion of data that is specifically truncated for each case only the linear range, i.e. the middle eighth of the time, since the first and last tenth of the time is meant for periods of positive and negative acceleration to ensure that the end-effect starts and end at rest. Further, the $R^2$ value is also shown which indicates how close the measured data is to the linear approximation. An $R^2$ value of 1 indicates a perfect fit.
Figure 62: Test 1 position vs. time.

Figure 63: Test 2 position vs. time.
Figure 64: Test 3 position vs. time.

Figure 65: Test 4 position vs. time.
Figure 66: Test 5 position vs. time.

Figure 67: Test 6 position vs. time.
The value of the linear slope is also displayed on the plot itself which represents the average constant velocity. In each case, the final position exceeds the target position of 10 feet which could be explained be the slight gain in elevation across the trajectory. This error could be tuned by making small adjustments to the control program. Overall, each trajectory did indeed hit its mark from a visual standpoint. A picture taken from the final position of one of the trajectory test, shown below, clearly shows the desired outcome.

Analysis of Trajectory Results

Visually, it can be seen from the plots above that the end-effector seemed to overshoot the desired net displacement of 10 feet. As a reminder, the data used for these position vs. time plots was gathered using a low-resolution (30 ticks per revolution) rotary encoder connected to a 3D printed spool. Fishing line was wound up around the spool with its free end fixed to the bowling ball end-effector. As the end-effector travels away from the spool, it unraveled at a rate similar to the velocity of the end-effector. Assuming that the end-effector traveled on a horizontal line, this crude method of motion verification would likely produce very accurate results. However, it was observed that over the course of the point-to-point straight-line trajectory, a gain in elevation was experienced. This error can be attributed to several factors, namely the sensitivity of the ratio between motor turns and length of cable let in or out as well as the slightly unpredictable manner in which the cable wraps itself around the spool. Because of this, the end-effector was able to travel a horizontal distance of 10 feet with great accuracy as
Figure 68, but due to the gain in elevation a longer net distance was traveled which is what the encoder has recorded.

Figure 68: Final position of planar trajectory.

Figure 68 shows the final position of the trajectory test which clearly depicts the bowling ball end-effector hovering directly over the final 10 foot mark. Similar to the vector tracing of an end-effector path that was seen in the MATLAB simulation, an attempt was made at conveying the motion in one image by attaching a small LED light to the end-effector and running a trajectory test in a dark room. The light trail was recorded using a long exposure camera application which yielded this stunning image:
Figure 69: Long exposure image of planar trajectory.

In figure 69, the slight gain in elevation can be seen. Further, it shows that the winch on the left hand side was wound less precisely than the right hand winch. As the end-effector moves from left to right, the tension load transfers more to the right hand winch resulting in nearly perfect straight motion. From this, it can be inferred that with higher-precision winches, one could expect nearly perfect straight motion (if desired) for all movements.

The maximum measured position error was recorded during the first trial run coming in at 4.4’ making this test the obvious outlier. Such a large error was not actually observed during this test run leading the source of this error to stem more from miscalibrating and poor handling of the sensor equipment. Because of this, results gathered in the first test trial will be omitted in the following brief position error analysis.
Table 3 summarizes the results of the position error obtained from each test trial assuming that the error is simply the difference between the measured position and the desired position.

Table 3: Trajectory test position errors.

<table>
<thead>
<tr>
<th>Trajectory Test</th>
<th>Position Error [inches]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10.3</td>
</tr>
<tr>
<td>2</td>
<td>10.3</td>
</tr>
<tr>
<td>3</td>
<td>8.6</td>
</tr>
<tr>
<td>4</td>
<td>9.4</td>
</tr>
<tr>
<td>5</td>
<td>4.8</td>
</tr>
</tbody>
</table>

The results shown in Table 3 are further simplified by presenting the minimum, maximum, average and range of position errors given in inches as shown in Table 4.

Table 4: Minimum, maximum, average and range of position error in inches.

<table>
<thead>
<tr>
<th>Minimum Error</th>
<th>Maximum Error</th>
<th>Average Error</th>
<th>Error Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.8</td>
<td>10.3</td>
<td>8.7</td>
<td>5.4</td>
</tr>
</tbody>
</table>
CHAPTER 7: DISCUSSION OF RESULTS

This section will provide a brief discussion of the results and finding discovered by completing each objective. The first objective focused on creating an all-in-one MATLAB simulation program for a 4- or 2-axis cable suspended robot. Objective 2 built upon the findings in objective 1 by developing a hardware system design capable of handling the loads and duties required for basic trajectory testing. The third objective was to actually build and configure the full hardware and software system. Finally, the fourth objective brought everything together by performing real-world trajectory tests in a planar configuration.

The versatility and capabilities of the MATLAB simulation program goes above and beyond any other CSR simulation program ever developed at Ohio University. The user-friendly GUI provide an intuitive platform to test nearly any CSR configuration with snapshot analysis and trajectory analysis. This program will serve as a starting point for future CSR research as its ability to provide physical results pertains not only to the application outlined in this thesis but to other conceivable applications big and small.

The overall results from objective 2 suggest that this prototype design is fully capable of functioning without mechanical failure under reasonable loading conditions and workspace sizes. The steps taken in this design process should be pursued by future prototype developments before construction begins to reaffirm that the CSR tower mechanism can function properly without mechanical failure.

The four prototype towers that have been constructed represent a physical platform to test 3-D trajectories and manual control. They have succeeded as viable
proof-of-concept mechanisms that are mobile and easily taken apart for storage or transportation. Once better methods of control are developed, this hardware system will provide the means necessary to test said methods.

The real-world trajectory tests proved that the position profiles created in this project are possible to execute with an acceptable position tolerance. Data recorded externally by a rotary encoder clearly show that the periods of positive and negative acceleration at the beginning and end, respectively, take approximately one tenth of the total time as desire. Further, it can be seen that the control program is able to maintain a constant velocity. This simple planar straight line trajectory represents the foundation of more complex 3-D trajectories required for the ultimate goal of a much larger scale algae-harvesting cable-suspended robot.
CHAPTER 8: FUTURE WORK

As a whole, this project represent a solid starting point for a much larger end goal: a mobile large-scale algae-harvesting cable-suspended robot whose purpose is harvest algae in a more efficient way when compared to current methods. With a more efficient method of harvesting algae, algae products such as biofuel become cheaper and more accessible to the consumer. But in order to reach that goal, several system improvements must be pursued.

The first point of improvement deals with the hardware itself, specifically the winches. The winches used in this project are intended to be used to pull heavy objects with brute force, like on the front of a truck – not for precision. To increase precision, a new winch system should be developed which better controls the manner in which the cable is wrapped about the spool. Doing so would increase the steadiness of the trajectory as seen in figure 68.

One way to potentially bypass the need for a more precise winch system would be to improve the method of end-effector position sensing. Instead of relying solely on the motor encoders, which are essentially blind, to sense the end-effector position more objectively, it would not matter so much as to how the winch is spooling because the position is known and the winches could compensate accordingly independent of how spooled or unspool they are. For this, it is suggested that a GPS system be incorporated to determine the active position of the end-effector.

Further, there remains a lot of room for improvements on the control program. A dedicated human computer interface would help greatly with trajectory planning with
easy user inputs. This future control program should focus on optimizing movements with a more continuous method of control and not just the MC_Position_Profile function block. In addition to pre-programmable harvest trajectory paths, a manual override feature should be incorporated should the robot need to avoid certain unforeseen obstacles.
REFERENCES


APPENDIX A: MATLAB CODE

%----------------------------------------------------------
%   CSR_SIMULATOR.m
%   Main Frame of 4-axis Cable Robot Simulation Program
%   Programmer: Collier Fais
%   Ohio University, Mechanical Engineering
%   All Rights Reserved, February 2016
%   
%   Scripts and Functions Needed to Run Program:
%   ABOUT.m
%   ACTIVE_PARAM.m
%   ACTIVE_PARAM_PLANAR.m
%   Circle.m
%   CIRCLE_DRIVER.m
%   CONST_VAR.m
%   CONVERSION.m
%   FRAMEPLOT.m
%   FRAMEPOSE.m
%   HARVEST_DRIVER.m
%   MENU0.m
%   MENU1.m
%   MOTIONPLOT_PLANAR.m
%   MOTIONPLOT3D.m
%   MOTIONPLOT4
%   OBJECTIVE4_DRIVER.m
%   PLANAR.m
%   PLANARPOSE.m
%   SNAP.m
%   SNAPCALC.m
%   SNAPPLOT.m
%   SNAPPOSE.m
%   SNAPTABLE1.m
%   STRAIGHT_LINE_DRIVER.m
%   STRAIGHTLINE.m
%   TRAJECTORY_DIR.m
%   TRAJECTORY_GENERATOR.m
%   TRAJECTORY_GENERATOR_H.m
%   TRAJECTORY_PLOTS.m
%   TRAJECTORY_PLOTS_PLANAR.m
%   TRAJECTORYMENU.m
%   
%   Image Files Used:
%   ISO_ML.jpeg
%   ISO_SW.jpeg
%----------------------------------------------------------
% Clear workspace, close all figure and table, clear command window
% short holds
close all; clear; clc;
format SHORTENG; format compact;

% Define useful conversion factors
CONVERSION
% Define system constants and variables, subject to change within certain program functions

CONST_VAR

% Main Menu, menu number 0
% Creates figure window displaying main-menu button options
% Returns 'choice0' based on users choice
choice0 = 99; % Initialize to avoid errors if menu0 is closed
choice1 = 99;
EXIT0 = 0;
EXIT1 = 0;
EXIT2 = 0;

MENU0

妈妈嘛妈妈妈妈妈妈妈妈妈妈妈妈妈妈妈妈妈妈妈妈妈妈妈妈妈妈妈妈妈妈妈妈妈妈妈妈妈妈妈妈妈妈妈

%----------------------------------------------------------------
% ABOUT.m
% Provides user with an information window, purpose and how-to
% Programmer: Collier Fais
% Ohio University, Mechanical Engineering
% All Rights Reserved, March 2016
%----------------------------------------------------------------
% close all;
disp('ABOUT SECTION')

% str = ['test ' char(10) 'test'];
str = sprintf('PURPOSE 

The purpose of this program is to simulate the kinematics and dynamics of a 4-axis cable suspended robot. As a whole, this simulation program (aptly named The Cable Suspended Robot Simulator, or CSR SIMULATOR) represents the fulfillment of one of the key objectives outlined by the Programmer's Master Thesis. With this simulation, we explore the kinematic and dynamic limits and constraints that arise when the CSR is configured in a user-defined static pose or a pre-programmed or manually-controlled end-effector trajectory. 3-D and 2-D plots show equally scaled and life-like depictions of the real-world hardware system configured by user-define parameters. In addition to the life-like visual aid, the simulator generates tables and graphs which present important physical results relating to the corresponding mode of operation. High end-effect velocity is not necessarily desired for this application and thus a pseudostatic assumption is made to simply the calculations for the trajectory modes of operation. Results and findings obtained from the CSR SIMULATOR are intended to help drive the design process of the real-world hardware prototype Cable Suspended Robot also built by the Programmer.

HOW TO USE

Clicking 'OK' at the bottom of this dialog box will return to the Main Menu. The 'Run' button sends the user to the Run Menu at which point the mode of operation can be selected. Depending on which
mode has been selected, the user will be prompted to select further options and will be given the opportunity to manipulate certain system parameters. To exit the program at any time, locate and click the EXIT button.

For further information, questions, comments or suggestions, please contact the Programmer:
Collier Fais, MSME in training
Ohio University, Mechanical Engineering
Email: cf277309@ohio.edu
Copyright 2016, All Rights Reserved.

% Display text using a help dialog box
AboutBox = helpdlg(str);

set(AboutBox,'position',[350 15 350 500])

waitFor(AboutBox);

%----------------------------------------------------------------
% ACTIVE_PARAM.m
% Calculates Active trajectory parameters
% Programmer: Collier Fais
% Ohio University, Mechanical Engineering
% All Rights Reserved, April 2016
%----------------------------------------------------------------

P = [X(k) Y(k) Z(k)];

% Active cables lengths
% Cable vectors
C1 = B11 - P;
C2 = B12 - P;
C3 = B13 - P;
C4 = B14 - P;

% Magnitude of cable lengths
L1(k) = norm(C1);
L2(k) = norm(C2);
L3(k) = norm(C3);
L4(k) = norm(C4);

% Length of cables relative to starting length [ft]
L1_rel(k) = L1(k) - L1(1);
L2_rel(k) = L2(k) - L2(1);
L3_rel(k) = L3(k) - L3(1);
L4_rel(k) = L4(k) - L4(1);

M1_rev(k) = L1_rel(k)*M_rev_conv;
M2_rev(k) = L2_rel(k)*M_rev_conv;
M3_rev(k) = L3_rel(k)*M_rev_conv;
M4_rev(k) = L4_rel(k)*M_rev_conv;

% Active cable rates, velocities [ft/s]
if k == 1
L1_dot(k) = 0;
L2_dot(k) = 0;
L3_dot(k) = 0;
L4_dot(k) = 0;
else
    L1_dot(k) = (L1(k)-L1(k-1))/dt;
    L2_dot(k) = (L2(k)-L2(k-1))/dt;
    L3_dot(k) = (L3(k)-L3(k-1))/dt;
    L4_dot(k) = (L4(k)-L4(k-1))/dt;
end

% Active cable accelerations [ft/s^2]
if k == 1
    L1_ddot(k) = 0;
    L2_ddot(k) = 0;
    L3_ddot(k) = 0;
    L4_ddot(k) = 0;
else
    L1_ddot(k) = (L1_dot(k)-L1_dot(k-1))/dt;
    L2_ddot(k) = (L2_dot(k)-L2_dot(k-1))/dt;
    L3_ddot(k) = (L3_dot(k)-L3_dot(k-1))/dt;
    L4_ddot(k) = (L4_dot(k)-L4_dot(k-1))/dt;
end

% Cable reel kinematics: angular velocity [rad/s], acceleration [rad/s^2]
w_reel1(k) = L1_dot(k)/Wr;  % Reel 1 angular velocity
w_reel2(k) = L2_dot(k)/Wr;
w_reel3(k) = L3_dot(k)/Wr;
w_reel4(k) = L4_dot(k)/Wr;

a_reel1(k) = L1_ddot(k)/Wr;  % Reel 1 angular acceleration
a_reel2(k) = L2_ddot(k)/Wr;
a_reel3(k) = L3_ddot(k)/Wr;
a_reel4(k) = L4_ddot(k)/Wr;

% Motor kinematics: angular velocity [rad/s], acceleration [rad/s^2]
w_motor1(k) = w_reel1(k)*Wn;
w_motor2(k) = w_reel2(k)*Wn;
w_motor3(k) = w_reel3(k)*Wn;
w_motor4(k) = w_reel4(k)*Wn;

a_motor1(k) = a_reel1(k)*Wn;
a_motor2(k) = a_reel2(k)*Wn;
a_motor3(k) = a_reel3(k)*Wn;
a_motor4(k) = a_reel4(k)*Wn;

% Encoder count, based on calibrating encoders at zero to correspond
% to an associated cable length of 'zero', can be changed later
pulse1(k) = L1(k)*pulse_per_foot_cable;
pulse2(k) = L2(k)*pulse_per_foot_cable;
pulse3(k) = L3(k)*pulse_per_foot_cable;
pulse4(k) = L4(k)*pulse_per_foot_cable;

% Static Jacobian matrix 'A'
% Not to be confused with Back Support Pose info 'Aij'
A = [C1(1)/L1(k) C2(1)/L2(k) C3(1)/L3(k) C4(1)/L4(k);
    C1(2)/L1(k) C2(2)/L2(k) C3(2)/L3(k) C4(2)/L4(k);
    C1(3)/L1(k) C2(3)/L2(k) C3(3)/L3(k) C4(3)/L4(k)];

% Transpose of Jacobian
A_trans = A';

% PseudoInverse of Jacobian
A_pseudoInv = pinv(A);

% Cable tension calculation
ActiveT = -A_pseudoInv*(F+m*g);
T1(k) = ActiveT(1);
T2(k) = ActiveT(2);
T3(k) = ActiveT(3);
T4(k) = ActiveT(4);

% Active motor torques [ft*lb]
Motor_torque1(k) = (T1(k)*Wr)/Wn;
Motor_torque2(k) = (T2(k)*Wr)/Wn;
Motor_torque3(k) = (T3(k)*Wr)/Wn;
Motor_torque4(k) = (T4(k)*Wr)/Wn;

% Robot Translational Stiffness
E = 12000000;   % Estimated cable modulus of elasticity [lbs per sq inch]
D_cab = 0.16;   % Cable diameter [in]
A_cab = (pi/4)*D_cab^2; % Cable cross-sectional area [in^2]
k_i = (E*A_cab)*[L1(k)^-1 L2(k)^-1 L3(k)^-1 L4(k)^-1]; % Individual-cable spring constant terms
k_diag = diag(k_i); % Diagonal matrix of k-values [lb/ft]
K = A*k_diag*A_trans; % 3x3 translational cartesian stiffness matrix
K_eig = eig(K); % Eigen values represent the principal stiffness components
K_tot = norm([K_eig]); % Euclidean norm yields a single stiffness number
Stiff(k) = K_tot;

% Determinant of [A A^T]
det_A(k) = det(A*A_trans);

-------------------------------------------------------------------
% ACTIVE_PARAM_PLANAR.m
% Calculates Active trajectory parameters
% Programmer: Collier Fais
% Ohio University, Mechanical Engineering
% All Rights Reserved, April 2016
-------------------------------------------------------------------
P = [X(k) Y(k)];

% Active cables lengths
% Cable vectors
C1 = B11 - P;
C2 = B12 - P;

% Magnitude of cable lengths
L1(k) = norm(C1);
L2(k) = norm(C2);

% Length of cables relative to starting length [ft]
L1_rel(k) = L1(k) - L1(1);
L2_rel(k) = L2(k) - L2(1);

M1_rev(k) = L1_rel(k)*M_rev_conv;
M2_rev(k) = L2_rel(k)*M_rev_conv;

% Active cable rates, velocities [ft/s]
if k == 1
    L1_dot(k) = 0;
    L2_dot(k) = 0;
else
    L1_dot(k) = (L1(k)-L1(k-1))/dt;
    L2_dot(k) = (L2(k)-L2(k-1))/dt;
end

% Active cable accelerations [ft/s^2]
if k == 1
    L1_ddot(k) = 0;
    L2_ddot(k) = 0;
else
    L1_ddot(k) = (L1_dot(k)-L1_dot(k-1))/dt;
    L2_ddot(k) = (L2_dot(k)-L2_dot(k-1))/dt;
end

% Cable reel kinematics: angular velocity [rad/s], acceleration [rad/s^2]
w_reel1(k) = L1_dot(k)/Wr; % Reel 1 angular velocity
w_reel2(k) = L2_dot(k)/Wr;

a_reel1(k) = L1_ddot(k)/Wr; % Reel 1 angular acceleration
a_reel2(k) = L2_ddot(k)/Wr;

% Motor kinematics: angular velocity [rad/s], acceleration [rad/s^2]
w_motor1(k) = w_reel1(k)*Wn;
w_motor2(k) = w_reel2(k)*Wn;

a_motor1(k) = a_reel1(k)*Wn;
a_motor2(k) = a_reel2(k)*Wn;
% Encoder count, based on calibrating encoders at zero to correspond
to an associated cable length of 'zero', can be changed later
pulse1(k) = L1(k)*pulse_per_foot_cable;
pulse2(k) = L2(k)*pulse_per_foot_cable;

Theta1(k) = atan((49/12)/X(k));
Theta2(k) = atan((49/12)/(D-X(k)));

T2(k) = Wt/(((cos(Theta2(k))/cos(Theta1(k)))*sin(Theta1(k))) +
sin(Theta2(k)));
T1(k) = (T2(k)*cos(Theta2(k)))/cos(Theta1(k));

% Perpendicular force acting at top of tower caused by Tension, T
F1(k) = T1(k)*cos(Theta1(k));
F2(k) = T2(k)*cos(Theta2(k));

% Active motor torques [ft*lb]
Motor_torque1(k) = (T1(k)*Wr)/Wn;
Motor_torque2(k) = (T2(k)*Wr)/Wn;

function h = circle(x,y,r)
% UNTITLED2 Summary of this function goes here
% Detailed explanation goes here
hold on
th = 0:pi/50:2*pi;
xunit = r * cos(th) + x;
yunit = r * sin(th) + y;
h = plot(xunit, yunit);
hold off
end

%----------------------------------------------------------------
% CIRCLE_DRIVER.m
% Coordinates a straight line trajectory
% Programmer: Collier Fais
% Ohio University, Mechanical Engineering
% All Rights Reserved, March 2016
%----------------------------------------------------------------

% Radius of trajectory circle, r [ft]
r = 5;

% User defined inputs
t0 = 0; % This shouldn't change for calculation simplicity
tf = 10; % Total time [s] for trajectory execution, change this
to what ever you want
posf = 2*pi*r;  % Total dist. [ft] of trajectory, change to whatever, or calculate by supplying two cartesian points and taking the vector length norm between them

% Assumed nominal parameters, would require optimization in future to ensure actuator limits are not breached
PR = (1/5);  % ratio of total time spent in the acceleration phase, Phase1

% Time stamps
\( t_1 = PR \cdot t_f \);  % Time stamp for end of Phase 1 / beginning of Phase 2 [s]
\( t_2 = t_f - t_1 \);  % Time stamp for end of Phase 2 / beginning of Phase 3 [s]
\( dt = PR/2 \);  % Time increment [s]

% Time array
\( t = [t_0:dt:t_f] \);

% Solving for cruising velocity, see notes for derivation
vel1 = posf/(t_f*(1-PR));

% Distance at which acceleration phase ends, cruising begins [ft]
pos1 = (1/2)*t_1*vel1;

% Distance traveled at constant velocity [ft]
CoastD = posf - 2*pos1;

% Distance at which cruising ends, negative acceleration begins [ft]
pos2 = posf - pos1;

% Time interval of phase 2 (constant velocity phase) [seconds]
t_coast = CoastD/vel1;

xyoff=[W/2,L/2];  % Center of circle
zcir = [ ];
ThetaArray = [ ];
\( v=\text{Pi:361/101:Pc} \);

TRAJECTORY_GENERATOR

FRAMEPOSE
\( \text{for } k = 1: \text{length}(t) \)
\( \text{lower case 'x' refers to position vector data} \)
\( \text{upper case 'X' refers to mode-specific end-effector catesian...} \)
\( \text{x-motion} \)
\( X(k) = \cos(2\pi*(x(k)/posf))*r + (W/2); \)
\( Y(k) = \sin(2\pi*(x(k)/posf))*r + (L/2); \)
\( Z(k) = 1; \)
\( \text{Calculate active parameters} \)
ACTIVE_PARAM

if view3D == 1
    MOTIONPLOT3D
end

if view4 == 1
    MOTIONPLOT4
end

end

%----------------------------------------------------------
%   CONST_VAR.m
%   Contains several constants and semi-constant variables
%   Programmer: Collier Fais
%   Ohio University, Mechanical Engineering
%   All Rights Reserved, February 2016
%----------------------------------------------------------

% Constants, may be set by user LATER ON
W = 25;      % Width of workspace [ft]
L = 25;      % Length of workspace [ft]
Ht = 6;      % Fixed height of tower [ft]
dx = .5;     % Distance from towers to workspace in the x-dir [ft]
dy = .5;     % Distance from towers to workspace in y-dir [ft]
Dx = 3;      % Distance from edge of plot in x-dir [ft]
Dy = 3;      % Distance from edge of plot in y-dir [ft]
Lm = (4/6)*Ht; % Length of Bottom Neam as ratio of Boom [ft]
Lr = ((34*ItF)/6)*Ht; % Length of Bunner as ratio of Boom [ft]
Ls = ((43*ItF)/6)*Ht; % Length of Side Support as ratio of Boom [ft]
Lb = ((30*ItF)/6)*Ht; % Length of Back Support as ratio of Boom [ft]
fx = 0;               % External forces exerted on end effector [lbf]
fy = 0;
fy = 0;
fx = 0;
%Matrix of external forces exerted on end-effector
F = [fx fy fz]';
g = [0 0 -32.2]';     % Acceleration due to gravity [ft/s^2]
Wt = 15.6;            % Weight of end-effector [lbf]
m = Wt*PtS;           % Mass of end-effector [slug]
Tlw = 3;              % Tower line width, units unclear
Clw = 0.5;            % Cable line width
Plw = 3;              % End-effector path line width

% Cable reel parameters, assumed
Wr = 1.25;            % Winch radius [in] (ASSUMED, THIS IS NOT CONSTANT)
Wn = 153;             % Winch gear ratio (ASSUMED)

% Motor encoder ticks per revolution, assumed
encd = 528400;        % Ticks per revolution
winch_rev_per_foot_cable = 12/Wr/(2*pi);

motor_rev_per_foot_cable = winch_rev_per_foot_cable*Wn;

M_rev_conv = motor_rev_per_foot_cable;

pulse_per_foot_cable = motor_rev_per_foot_cable*encd;

%-----------------------------------------------------------------------------
%   CONVERSION.m
%   Useful conversion factors for calculations
%   Programmer: Collier Fais
%   Ohio University, Mechanical Engineering
%   All Rights Reserved, February 2016
%-----------------------------------------------------------------------------

% Conversion Factors
ItF = (1/12); % Inches to feet
DR = pi/180; % Degrees to radians
PtS = 0.031081; % Pounds to slugs

%-----------------------------------------------------------------------------
%   SNAPLOT.m
%   Plots the snap-shot
%   Programmer: Collier Fais
%   Ohio University, Mechanical Engineering
%   All Rights Reserved, March 2016
%-----------------------------------------------------------------------------

nameSNAP_PLOT = 'Font, Top, Right and 3D View Snapshot';

% close all;

% Figure 1, 3D plot
% FourSnapView = figure('position',[10 10 1520 760],'name',nameSNAP_PLOT,'NumberTitle','off');
subplot(222)
% Plotting the Booms
plot3([B01(1) B11(1)],[B01(2) B11(2)],[B01(3) B11(3)], 'k',
'linewidth', Tlw)
hold on
plot3([B02(1) B12(1)],[B02(2) B12(2)],[B02(3) B12(3)], 'k',
'linewidth', Tlw)
plot3([B03(1) B13(1)],[B03(2) B13(2)],[B03(3) B13(3)], 'k',
'linewidth', Tlw)
plot3([B04(1) B14(1)],[B04(2) B14(2)],[B04(3) B14(3)], 'k',
'linewidth', Tlw)

% Edge of plot-space patch
patch([0-Dx 0-Dx W+Dx W+Dx],[0-Dy L+Dy L+Dy 0-Dy],[-0.1 -0.1 -0.1 -0.1],[0 1 0 1])
% Workspace footprint patch
patch([0 0 W W],[0 L L 0],[.73 1 .96])

% Plotting the Bottom Beams
plot3([M01(1) M11(1)],[M01(2) M11(2)],[M01(3) M11(3)], 'k', 'linewidth', Tlw)
plot3([M02(1) M12(1)],[M02(2) M12(2)],[M02(3) M12(3)], 'k', 'linewidth', Tlw)
plot3([M03(1) M13(1)],[M03(2) M13(2)],[M03(3) M13(3)], 'k', 'linewidth', Tlw)
plot3([M04(1) M14(1)],[M04(2) M14(2)],[M04(3) M14(3)], 'k', 'linewidth', Tlw)

% Plotting the Runners
plot3([R01(1) R11(1)],[R01(2) R11(2)],[R01(3) R11(3)], 'k', 'linewidth', Tlw)
plot3([R02(1) R12(1)],[R02(2) R12(2)],[R02(3) R12(3)], 'k', 'linewidth', Tlw)
plot3([R03(1) R13(1)],[R03(2) R13(2)],[R03(3) R13(3)], 'k', 'linewidth', Tlw)
plot3([R04(1) R14(1)],[R04(2) R14(2)],[R04(3) R14(3)], 'k', 'linewidth', Tlw)

% Plotting the Side Supports [a]
plot3([S01a(1) S11a(1)],[S01a(2) S11a(2)],[S01a(3) S11a(3)], 'k', 'linewidth', Tlw)
plot3([S02a(1) S12a(1)],[S02a(2) S12a(2)],[S02a(3) S12a(3)], 'k', 'linewidth', Tlw)
plot3([S03a(1) S13a(1)],[S03a(2) S13a(2)],[S03a(3) S13a(3)], 'k', 'linewidth', Tlw)
plot3([S04a(1) S14a(1)],[S04a(2) S14a(2)],[S04a(3) S14a(3)], 'k', 'linewidth', Tlw)

% Plotting the Side Supports [b]
plot3([S01b(1) S11b(1)],[S01b(2) S11b(2)],[S01b(3) S11b(3)], 'k', 'linewidth', Tlw)
plot3([S02b(1) S12b(1)],[S02b(2) S12b(2)],[S02b(3) S12b(3)], 'k', 'linewidth', Tlw)
plot3([S03b(1) S13b(1)],[S03b(2) S13b(2)],[S03b(3) S13b(3)], 'k', 'linewidth', Tlw)
plot3([S04b(1) S14b(1)],[S04b(2) S14b(2)],[S04b(3) S14b(3)], 'k', 'linewidth', Tlw)

% Plotting the Back Support
plot3([A01(1) A11(1)],[A01(2) A11(2)],[A01(3) A11(3)], 'k', 'linewidth', Tlw)
plot3([A02(1) A12(1)],[A02(2) A12(2)],[A02(3) A12(3)], 'k', 'linewidth', Tlw)
plot3([A03(1) A13(1)],[A03(2) A13(2)],[A03(3) A13(3)], 'k', 'linewidth', Tlw)
plot3([A04(1) A14(1)],[A04(2) A14(2)],[A04(3) A14(3)], 'k', 'linewidth', Tlw)
% Plotting the Cables
plot3 ([B11(1) X(k)], [B11(2) Y(k)], [B11(3) Z(k)], 'r', 'linewidth', Clw)
plot3 ([B12(1) X(k)], [B12(2) Y(k)], [B12(3) Z(k)], 'r', 'linewidth', Clw)
plot3 ([B13(1) X(k)], [B13(2) Y(k)], [B13(3) Z(k)], 'r', 'linewidth', Clw)
plot3 ([B14(1) X(k)], [B14(2) Y(k)], [B14(3) Z(k)], 'r', 'linewidth', Clw)

plot3 ([2.5 X(k)], [W/2 W/2], [1 1], 'c', 'linewidth', Clw)

% Plotting the Cone end effector
 cyl1 = [0 .35];
cyl2 = [.35 0];
[x1, y1, z1] = cylinder(cyl1);
[x2, y2, z2] = cylinder(cyl2);
surf(x1+X(k), y1+Y(k), z1+Z(k))
surf(x2+X(k), y2+Y(k), z2+Z(k)-1)

title('3D Plot')
xlabel('Width [ft]')
ylabel('Length [ft]')
zlabel('Height [ft]')

grid on;
grid minor;
p1 = gca;
set(gca, 'Xcolor', [0 0 0]); % Sets axis line color, not grid lines
set(gca, 'Ycolor', [0 0 0]); % Sets axis line color, not grid lines
set(gca, 'Zcolor', [0 0 0]); % Sets axis line color, not grid lines
set(gca, 'zlim', [0 10]); % Changes the color of the background

axis equal;

hold off;
pause(1/64);

%%

% Figure 2 x-y plane, top view
% figure(2);
subplot(221)
% Plotting the Booms
plot([B01(1) B11(1)], [B01(2) B11(2)], 'k', 'linewidth', Tlw)
hold on
plot([B02(1) B12(1)], [B02(2) B12(2)], 'k', 'linewidth', Tlw)
plot([B03(1) B13(1)], [B03(2) B13(2)], 'k', 'linewidth', Tlw)
plot([B04(1) B14(1)], [B04(2) B14(2)], 'k', 'linewidth', Tlw)

% Edge of plot-space patch
patch([0-Dx 0-Dx W+Dx W+Dx],[0-Dy L+Dy L+Dy 0-Dy],[-0.1 -0.1 -0.1 -0.1],[0 1 0])

%Workspace footprint patch
patch([0 0 W W],[0 L L 0],[.73 1 .96])

% Plotting the Bottom Beams
plot([M01(1) M11(1)],[M01(2) M11(2)], 'k', 'linewidth', Tlw)
plot([M02(1) M12(1)],[M02(2) M12(2)], 'k', 'linewidth', Tlw)
plot([M03(1) M13(1)],[M03(2) M13(2)], 'k', 'linewidth', Tlw)
plot([M04(1) M14(1)],[M04(2) M14(2)], 'k', 'linewidth', Tlw)

% Plotting the Runners
plot([R01(1) R11(1)],[R01(2) R11(2)], 'k', 'linewidth', Tlw)
plot([R02(1) R12(1)],[R02(2) R12(2)], 'k', 'linewidth', Tlw)
plot([R03(1) R13(1)],[R03(2) R13(2)], 'k', 'linewidth', Tlw)
plot([R04(1) R14(1)],[R04(2) R14(2)], 'k', 'linewidth', Tlw)

% Plotting the Side Supports [a]
plot([S01a(1) S11a(1)],[S01a(2) S11a(2)], 'k', 'linewidth', Tlw)
plot([S02a(1) S12a(1)],[S02a(2) S12a(2)], 'k', 'linewidth', Tlw)
plot([S03a(1) S13a(1)],[S03a(2) S13a(2)], 'k', 'linewidth', Tlw)
plot([S04a(1) S14a(1)],[S04a(2) S14a(2)], 'k', 'linewidth', Tlw)

% Plotting the Side Supports [b]
plot([S01b(1) S11b(1)],[S01b(2) S11b(2)], 'k', 'linewidth', Tlw)
plot([S02b(1) S12b(1)],[S02b(2) S12b(2)], 'k', 'linewidth', Tlw)
plot([S03b(1) S13b(1)],[S03b(2) S13b(2)], 'k', 'linewidth', Tlw)
plot([S04b(1) S14b(1)],[S04b(2) S14b(2)], 'k', 'linewidth', Tlw)

% Plotting the Back Support
plot([A01(1) A11(1)],[A01(2) A11(2)], 'k', 'linewidth', Tlw)
plot([A02(1) A12(1)],[A02(2) A12(2)], 'k', 'linewidth', Tlw)
plot([A03(1) A13(1)],[A03(2) A13(2)], 'k', 'linewidth', Tlw)
plot([A04(1) A14(1)],[A04(2) A14(2)], 'k', 'linewidth', Tlw)

% Plotting the Cables
plot([B11(1) X(k)],[B11(2) Y(k)], 'r', 'linewidth', Clw)
plot([B12(1) X(k)],[B12(2) Y(k)], 'r', 'linewidth', Clw)
plot([B13(1) X(k)],[B13(2) Y(k)], 'r', 'linewidth', Clw)
plot([B14(1) X(k)],[B14(2) Y(k)], 'r', 'linewidth', Clw)

Circle(X(k), Y(k), 0.5);

title('X-Y Plane, Top View')
xlabel('Width [ft]')
ylabel('Length [ft]')

grid on;
grid minor;
pl = gca;
set(gca,'Xcolor',[0 0 0]);  %Sets axis line color, not grid lines
set(gca,'Ycolor',[0 0 0]);  %Sets axis line color, not grid lines
set(gca,'Zcolor',[0 0 0]);  %Sets axis line color, not grid lines
whitebg([1 .94 .81]);       % Changes the color of the background
axis equal;
hold off;
pause(1/64);

%%

% Figure 3, x-z plane, front view
% figure(3);
subplot(223)
% Plotting the Booms
plot([B01(1) B11(1)], [B01(3) B11(3)], 'k', 'linewidth', Tlw)
hold on
plot([B02(1) B12(1)], [B02(3) B12(3)], 'k', 'linewidth', Tlw)
plot([B03(1) B13(1)], [B03(3) B13(3)], 'k', 'linewidth', Tlw)
plot([B04(1) B14(1)], [B04(3) B14(3)], 'k', 'linewidth', Tlw)

% Plot edge of plot-space with a line instead of a patch
plot([0-Dx W+Dx],[-0.1 -0.1],'g','linewidth', 1)
% Plot workspace with a line instead of patch
plot([0 W],[0 0],'c','linewidth', 2)

% Plotting the Bottom Beams
plot([M01(1) M11(1)], [M01(3) M11(3)], 'k', 'linewidth', Tlw)
plot([M02(1) M12(1)], [M02(3) M12(3)], 'k', 'linewidth', Tlw)
plot([M03(1) M13(1)], [M03(3) M13(3)], 'k', 'linewidth', Tlw)
plot([M04(1) M14(1)], [M04(3) M14(3)], 'k', 'linewidth', Tlw)

% Plotting the Runners
plot([R01(1) R11(1)], [R01(3) R11(3)], 'k', 'linewidth', Tlw)
plot([R02(1) R12(1)], [R02(3) R12(3)], 'k', 'linewidth', Tlw)
plot([R03(1) R13(1)], [R03(3) R13(3)], 'k', 'linewidth', Tlw)
plot([R04(1) R14(1)], [R04(3) R14(3)], 'k', 'linewidth', Tlw)

% Plotting the Side Supports [a]
plot([S01a(1) S11a(1)], [S01a(3) S11a(3)], 'k', 'linewidth', Tlw)
plot([S02a(1) S12a(1)], [S02a(3) S12a(3)], 'k', 'linewidth', Tlw)
plot([S03a(1) S13a(1)], [S03a(3) S13a(3)], 'k', 'linewidth', Tlw)
plot([S04a(1) S14a(1)], [S04a(3) S14a(3)], 'k', 'linewidth', Tlw)

% Plotting the Side Supports [b]
plot([S01b(1) S11b(1)], [S01b(3) S11b(3)], 'k', 'linewidth', Tlw)
plot([S02b(1) S12b(1)], [S02b(3) S12b(3)], 'k', 'linewidth', Tlw)
plot([S03b(1) S13b(1)], [S03b(3) S13b(3)], 'k', 'linewidth', Tlw)
plot([S04b(1) S14b(1)], [S04b(3) S14b(3)], 'k', 'linewidth', Tlw)

% Plotting the Back Support
plot([A01(1) A11(1)], [A01(3) A11(3)], 'k', 'linewidth', Tlw)
plot([A02(1) A12(1)], [A02(3) A12(3)], 'k', 'linewidth', Tlw)
plot([A03(1) A13(1)], [A03(3) A13(3)], 'k', 'linewidth', Tlw)
plot([A04(1) A14(1)], [A04(3) A14(3)], 'k', 'linewidth', Tlw)
% Plotting the Cables
plot ([B11(1) X(k)], [B11(3) Z(k)], 'r', 'linewidth', Clw)
plot ([B12(1) X(k)], [B12(3) Z(k)], 'r', 'linewidth', Clw)
plot ([B13(1) X(k)], [B13(3) Z(k)], 'r', 'linewidth', Clw)
plot ([B14(1) X(k)], [B14(3) Z(k)], 'r', 'linewidth', Clw)

% Patch triangles that look like the cones for the iso view
patch([X(k) X(k)-cyl1(2) X(k)+cyl1(2)], [Z(k) Z(k)+1 Z(k)+1], [0 0 0], 'g')
patch([X(k) X(k)-cyl1(2) X(k)+cyl1(2)], [Z(k) Z(k)-1 Z(k)-1], [0 0 0], 'g')

title('X-Z Plane, Front View')
xlabel('Width [ft]')
ylabel('Height [ft]')
axis equal;
grid on;
grid minor;
hold off;
pause(1/64);

%%

% Figure 4, y-z plane, Right view
% figure(4);
% subplot(224)
% Plotting the Booms
plot([B01(2) B11(2)], [B01(3) B11(3)], 'k', 'linewidth', Tlw)
hold on
plot([B02(2) B12(2)], [B02(3) B12(3)], 'k', 'linewidth', Tlw)
plot([B03(2) B13(2)], [B03(3) B13(3)], 'k', 'linewidth', Tlw)
plot([B04(2) B14(2)], [B04(3) B14(3)], 'k', 'linewidth', Tlw)

% Plot edge of plot-space with a line instead of a patch
plot([0 -Dy L+Dy], [-0.1 -0.1], 'g', 'linewidth', 1)
% Plot workspace with a line instead of patch
plot([0 L], [0 0], 'c', 'linewidth', 2)

% Patch triangles that look like the cones for the iso view
patch([Y(k) Y(k)-cyl1(2) Y(k)+cyl1(2)], [Z(k) Z(k)+1 Z(k)+1], [0 0 0], 'g')
patch([Y(k) Y(k)-cyl1(2) Y(k)+cyl1(2)], [Z(k) Z(k)-1 Z(k)-1], [0 0 0], 'g')

% Plotting the Bottom Beams
plot([M01(2) M11(2)], [M01(3) M11(3)], 'k', 'linewidth', Tlw)
plot([M02(2) M12(2)], [M02(3) M12(3)], 'k', 'linewidth', Tlw)
plot([M03(2) M13(2)], [M03(3) M13(3)], 'k', 'linewidth', Tlw)
plot([M04(2) M14(2)], [M04(3) M14(3)], 'k', 'linewidth', Tlw)

% Plotting the Runners
```matlab
plot([R01(2) R11(2)], [R01(3) R11(3)], 'k', 'linewidth', Tlw)
plot([R02(2) R12(2)], [R02(3) R12(3)], 'k', 'linewidth', Tlw)
plot([R03(2) R13(2)], [R03(3) R13(3)], 'k', 'linewidth', Tlw)
plot([R04(2) R14(2)], [R04(3) R14(3)], 'k', 'linewidth', Tlw)

% Plotting the Side Supports [a]
plot([S01a(2) S11a(2)], [S01a(3) S11a(3)], 'k', 'linewidth', Tlw)
plot([S02a(2) S12a(2)], [S02a(3) S12a(3)], 'k', 'linewidth', Tlw)
plot([S03a(2) S13a(2)], [S03a(3) S13a(3)], 'k', 'linewidth', Tlw)
plot([S04a(2) S14a(2)], [S04a(3) S14a(3)], 'k', 'linewidth', Tlw)

% Plotting the Side Supports [b]
plot([S01b(2) S11b(2)], [S01b(3) S11b(3)], 'k', 'linewidth', Tlw)
plot([S02b(2) S12b(2)], [S02b(3) S12b(3)], 'k', 'linewidth', Tlw)
plot([S03b(2) S13b(2)], [S03b(3) S13b(3)], 'k', 'linewidth', Tlw)
plot([S04b(2) S14b(2)], [S04b(3) S14b(3)], 'k', 'linewidth', Tlw)

% Plotting the Back Support
plot([A01(2) A11(2)], [A01(3) A11(3)], 'k', 'linewidth', Tlw)
plot([A02(2) A12(2)], [A02(3) A12(3)], 'k', 'linewidth', Tlw)
plot([A03(2) A13(2)], [A03(3) A13(3)], 'k', 'linewidth', Tlw)
plot([A04(2) A14(2)], [A04(3) A14(3)], 'k', 'linewidth', Tlw)

% Plotting the Cables
plot ([B11(2) Y(k)],[B11(3) Z(k)], 'r', 'linewidth', Clw)
plot ([B12(2) Y(k)],[B12(3) Z(k)], 'r', 'linewidth', Clw)
plot ([B13(2) Y(k)],[B13(3) Z(k)], 'r', 'linewidth', Clw)
plot ([B14(2) Y(k)],[B14(3) Z(k)], 'r', 'linewidth', Clw)

title('Y-Z Plane, Right View')
xlabel('Length [ft]')
ylabel('Height [ft]')

grid on;
grid minor;
pl = gca;
set(gca,'Xcolor',[0 0 0]); %Sets axis line color, not grid lines
set(gca,'Ycolor',[0 0 0]); %Sets axis line color, not grid lines
set(gca,'Zcolor',[0 0 0]); %Sets axis line color, not grid lines
% set(gca,'zlim',[0 10]); % Changes the color of the background
axis equal;
hold off;
pause(1/64);
```

%-----------------------------------------------
% FRAMEPOSE.m
% Vector pose of towers and workspace
% Programmer: Collier Fais
% Ohio University, Mechanical Engineering
% All Rights Reserved, March 2016
%----------------------------------------------------------------
% Static pose of End Effector, first attempt  
% This will change when moving...
% P = [7 19.5 5.9]; % Located at dead center of workspace, default, can be changed

% Pose of Tower Boom  
% Syntax ~ Bij where:
% i indicates the X,Y,Z cartesian START and END of the Boom and
% j indicates the ID
B01 = [0-dx 0-dy 0];  % Location of base of tower 1  
B11 = [0-dx 0-dy Ht];  % Location of top of tower 1  
B02 = [0-dx L+dy 0];  
B12 = [0-dx L+dy Ht];  
B03 = [W+dx L+dy 0];  
B13 = [W+dx L+dy Ht];  
B04 = [W+dx 0-dy 0];  
B14 = [W+dx 0-dy Ht];

% Pose of Bottom Beam, Mij  
% i = left, right from a back oriented view, respectively  
% j = Tower ID
M01 = [(Lm/2)*cos(135*DR)-dx (Lm/2)*sin(135*DR)-dy 0];  
M11 = [(Lm/2)*cos(315*DR)-dx (Lm/2)*sin(315*DR)-dy 0];  
M02 = [(Lm/2)*cos(45*DR)-dx ((Lm/2)*sin(45*DR))+L+dy 0];  
M12 = [(Lm/2)*cos(225*DR)-dx ((Lm/2)*sin(225*DR))+L+dy 0];  
M03 = [((Lm/2)*cos(315*DR))+W+dx ((Lm/2)*sin(315*DR))+L+dy 0];  
M13 = [((Lm/2)*cos(135*DR))+W+dx ((Lm/2)*sin(135*DR))+L+dy 0];  
M04 = [((Lm/2)*cos(225*DR))+W+dx ((Lm/2)*sin(225*DR))-dy 0];  
M14 = [((Lm/2)*cos(45*DR))+W+dx ((Lm/2)*sin(45*DR))-dy 0];

% Pose of Runner, Rij  
% i = front, back from a back oriented view, respectively  
% j = Tower ID
R01 = [0-dx 0-dy 0];  
R11 = [(Lr)*cos(225*DR)-dx (Lr)*sin(225*DR)-dy 0];  
R02 = [0-dx L+dy 0];  
R12 = [(Lr)*cos(135*DR)-dx ((Lr)*sin(135*DR))+L+dy 0];  
R03 = [W+dx L+dy 0];  
R13 = [((Lr)*cos(45*DR))+W+dx ((Lr)*sin(45*DR))+L+dy 0];  
R04 = [W+dx 0-dy 0];  
R14 = [((Lr)*cos(315*DR))+W+dx (Lr)*sin(315*DR)-dy 0];

% Pose of Side Support a (right support, back view) Sija  
% i = bottom, top  
% j = Tower ID  
% a = right Side Support, back view
S01a = [Ls*cos(60*DR)*cos(315*DR)-dx Ls*cos(60*DR)*sin(315*DR)-dy 0];  
S11a = [0-dx 0-dy Ls*sin(60*DR)];
\[ S02a = [Ls \cdot \cos(60 \cdot DR) \cdot \cos(225 \cdot DR) - dx, Ls \cdot \cos(60 \cdot DR) \cdot \sin(225 \cdot DR) + L + dy, 0]; \]
\[ S12a = [0 - dx, L + dy, Ls \cdot \sin(60 \cdot DR)]; \]
\[ S03a = [Ls \cdot \cos(60 \cdot DR) \cdot \cos(135 \cdot DR) + W + dx, Ls \cdot \cos(60 \cdot DR) \cdot \sin(135 \cdot DR) + L + dy, 0]; \]
\[ S13a = [W + dx, L + dy, Ls \cdot \sin(60 \cdot DR)]; \]
\[ S04a = [Ls \cdot \cos(60 \cdot DR) \cdot \cos(45 \cdot DR) + W + dx, Ls \cdot \cos(60 \cdot DR) \cdot \sin(45 \cdot DR) - dy, 0]; \]
\[ S14a = [W + dx, 0 - dy, Ls \cdot \sin(60 \cdot DR)]; \]

% Pose of Side Support b (left support, back view) Sijb
% i = bottom, top
% j = Tower ID
% b = left Side Support, back view
\[ S01b = [Ls \cdot \cos(60 \cdot DR) \cdot \cos(135 \cdot DR) - dx, Ls \cdot \cos(60 \cdot DR) \cdot \sin(135 \cdot DR) - dy, 0]; \]
\[ S11b = [0 - dx, 0 - dy, Ls \cdot \sin(60 \cdot DR)]; \]
\[ S02b = [Ls \cdot \cos(60 \cdot DR) \cdot \cos(45 \cdot DR) - dx, Ls \cdot \cos(60 \cdot DR) \cdot \sin(45 \cdot DR) + L + dy, 0]; \]
\[ S12b = [0 - dx, L + dy, Ls \cdot \sin(60 \cdot DR)]; \]
\[ S03b = [Ls \cdot \cos(60 \cdot DR) \cdot \cos(315 \cdot DR) + W + dx, Ls \cdot \cos(60 \cdot DR) \cdot \sin(315 \cdot DR) + L + dy, 0]; \]
\[ S13b = [W + dx, L + dy, Ls \cdot \sin(60 \cdot DR)]; \]
\[ S04b = [Ls \cdot \cos(60 \cdot DR) \cdot \cos(225 \cdot DR) + W + dx, Ls \cdot \cos(60 \cdot DR) \cdot \sin(225 \cdot DR) - dy, 0]; \]
\[ S14b = [W + dx, 0 - dy, Ls \cdot \sin(60 \cdot DR)]; \]

% Pose of Back Support, Aij
% Not to be confused with Static Jacobian 'A'
% i = bottom, top from a back oriented view, respectively
% j = Tower ID
\[ A01 = [Lb \cdot \cos(60 \cdot DR) \cdot \cos(225 \cdot DR) - dx, Lb \cdot \cos(60 \cdot DR) \cdot \sin(225 \cdot DR) - dy, 0]; \]
\[ A11 = [0 - dx, 0 - dy, Lb \cdot \sin(60 \cdot DR)]; \]
\[ A02 = [Lb \cdot \cos(60 \cdot DR) \cdot \cos(135 \cdot DR) - dx, Lb \cdot \cos(60 \cdot DR) \cdot \sin(135 \cdot DR) + L + dy, 0]; \]
\[ A12 = [0 - dx, L + dy, Lb \cdot \sin(60 \cdot DR)]; \]
\[ A03 = [Lb \cdot \cos(60 \cdot DR) \cdot \cos(45 \cdot DR) + W + dx, Lb \cdot \cos(60 \cdot DR) \cdot \sin(45 \cdot DR) + L + dy, 0]; \]
\[ A13 = [W + dx, L + dy, Lb \cdot \sin(60 \cdot DR)]; \]
\[ A04 = [Lb \cdot \cos(60 \cdot DR) \cdot \cos(315 \cdot DR) + W + dx, Lb \cdot \cos(60 \cdot DR) \cdot \sin(315 \cdot DR) - dy, 0]; \]
\[ A14 = [W + dx, 0 - dy, Lb \cdot \sin(60 \cdot DR)]; \]
%----------------------------------------------------------------
% HARVEST_DRIVER.m
% Coordinates a straight line trajectory
% Programmer: Collier Fais
% Ohio University, Mechanical Engineering
% All Rights Reserved, March 2016
%----------------------------------------------------------------

% User defined inputs

\[ t_0 = 0; \quad \text{This shouldn't change for calculation simplicity} \]
\[ \text{leg1t} = 5; \quad \text{Time spent on leg 1 of travel} \]
\[ \text{leg2t} = 10; \quad \text{Time spent on leg 2 of travel} \]
\[ \text{leg3t} = 5; \quad \text{Time spent on leg 3 of travel} \]
\[ \text{leg4t} = 20; \quad \text{Time spent on leg 4 of travel} \]
\[ \text{leg5t} = 5; \quad \text{Time spent on leg 5 of travel} \]
\[ \text{leg6t} = 20; \quad \text{Time spent on leg 6 of travel} \]
\[ \text{leg7t} = 5; \quad \text{Time spent on leg 7 of travel} \]
\[ \text{leg8t} = 25; \quad \text{Time spent on leg 8 of travel} \]
\[ \text{leg9t} = 5; \quad \text{Time spent on leg 9 of travel} \]
\[ \text{leg10t} = 20; \quad \text{Time spent on leg 1 of travel} \]
\[ \text{leg11t} = 5; \quad \text{Time spent on leg 11 of travel} \]

\[ \sum_{i=1}^{11} \text{leg}_i t \]

% Distance of each leg

\[ \text{leg1d} = 3.5; \quad \text{Dist spent on leg 1 of travel} \]
\[ \text{leg2d} = 10.124; \quad \text{Dist spent on leg 2 of travel} \]
\[ \text{leg3d} = 3; \quad \text{Dist spent on leg 3 of travel} \]
\[ \text{leg4d} = 17; \quad \text{Dist spent on leg 4 of travel} \]
\[ \text{leg5d} = 3; \quad \text{Dist spent on leg 5 of travel} \]
\[ \text{leg6d} = 21.401; \quad \text{Dist spent on leg 6 of travel} \]
\[ \text{leg7d} = 3; \quad \text{Dist spent on leg 7 of travel} \]
\[ \text{leg8d} = 17; \quad \text{Dist spent on leg 8 of travel} \]
\[ \text{leg9d} = 3; \quad \text{Dist spent on leg 9 of travel} \]
\[ \text{leg10d} = 20.359; \quad \text{Dist spent on leg 1 of travel} \]
\[ \text{leg11d} = 3.5; \quad \text{Dist spent on leg 11 of travel} \]

% tf = 10; \quad \text{Total time [s] for trajectory execution, change this to whatever you want} \\
% posf = 20; \quad \text{Total dist. [ft] of trajectory, change to whatever, or calculate by supplying two cartesian points and taking the vector length norm between them} \\
% Assumed nominal parameters, would require optimization in future to ensure actuator limits are not breached
PR = (1/5); % ratio of total time spent in the acceleration phase, Phase1

% t1 = PR*tf;     % Time stamp for end of Phase 1 / beginning of Phase 2 [s]
% t2 = tf-t1;     % Time stamp for end of Phase 2 / beginning of Phase 3 [s]

dt = PR/2;       % Time increment [s]

t = [t0:dt:harvesttf]; % Time array

FRAMEPOSE       % Plots static default config to screen

% Animate Harvest Simulation, for full length of time
for k = 1:length(t)

% Leg 1 of travel
    if t(k) < leg1t

% In first loop, define leg-specific motion parameters
    if t(k) == t0
        t1 = PR*leg1t;
        t2 = leg1t-t1;
        posf = leg1d;
        vel1 = posf/(leg1t*(1-PR));
        pos1 = (1/2)*t1*vel1;
        pos2 = posf - pos1;
        phase3counter = 0;
        leg = 1;

        ThisLegt = [t0:dt:leg1t];
        count1 = ((length(ThisLegt)-1)*PR)+1;
        TRAJECTORY_GENERATOR_H
        disp('check1')
    end

% Drive end effector in leg-specific direction, powered by x(k)
    X(k) = 0.5;
    Y(k) = 12.5;
    Z(k) = 0.5 + x(k);

% This option is set in the TRAJECTORYMENU as the view style
    if view3D == 1
        MOTIONPLOT3D
    end

% Create index marker for end of leg 1
    if t(k) == leg1t
        leg1count = k;
    end
end
elseif t(k) > leg1t && t(k) <= (leg1t+leg2t)

%   Leg 2 of travel
%   In first loop of leg 2, define leg-specific motion
parameters
   if k == leg1count+1
      disp('check2')
      t1 = PR*leg2t;
      t2 = leg2t-t1;
      posf = leg2d;
      vel1 = posf/(leg2t*(1-PR));
      pos1 = (1/2)*t1*vel1;
      pos2 = posf - pos1;
      phase3counter = 0;
      leg = 2;

      ThisLegt = [t0:dt:leg2t];
      count1 = ((length(ThisLegt)-1)*PR)+1;
      TRAJECTORY_GENERATOR_H
   end

%   Drive end-effector is leg-specific direction, powered by x(k)
   x(k) = x(k-leg1count)*cos(atan(8.5/5.5)) + 0.5;
   Y(k) = x(k-leg1count)*sin(atan(8.5/5.5)) + 12.5;
   Z(k) = 4;

% View style
   if view3D == 1
      MOTIONPLOT3D
   end

%   Create index count for end of leg 2
   if t(k) == (leg1t+leg2t)
      leg2count = k;
   end

elseif t(k) > (leg1t+leg2t) && t(k) <= (leg1t+leg2t+leg3t)

%   Leg 3 of travel
%   Define leg-specific motion direction, powered by x(k)
if k == leg2count+1
   disp('check3')
   t1 = PR*leg3t;
   t2 = leg3t-t1;
   posf = leg3d;
   vel1 = posf/(leg3t*(1-PR));
   pos1 = (1/2)*t1*vel1;
   pos2 = posf - pos1;
phase3counter = 0;
leg = 3;

ThisLegt = [t0:dt:leg3t];
count1 = ((length(ThisLegt)-1)*PR)+1;
TRAJECTORY_GENERATOR_H

end

%   Leg 3 direction of motion, powered by x(k)
X(k) = 6;
Y(k) = 21;
Z(k) = 4 - x(k-3count);

if view3D == 1
    MOTIONPLOT3D
end

%   Create index marker for end of leg 3
if t(k) == (leg1t+leg2t+leg3t)
    leg3count = k;
end

elseif t(k) > (leg1t+leg2t+leg3t) && t(k) <= (leg1t+leg2t+leg3t+leg4t)

%   Leg 4 of travel
%   Define leg-specific motion direction, powered by x(k)
if k == leg3count+1
disp('check4')
t1 = PR*leg4t;
t2 = leg4t-t1;
posf = leg4d;
vel1 = posf/(leg4t*(1-PR));
pos1 = (1/2)*t1*vel1;
pos2 = posf - pos1;
phase3counter = 0;
leg = 4;

ThisLegt = [t0:dt:leg4t];
count1 = ((length(ThisLegt)-1)*PR)+1;
TRAJECTORY_GENERATOR_H

end

%   Leg 4 direction of motion, powered by x(k)
X(k) = 6;
Y(k) = 21-x(k-leg3count);
Z(k) = 1;

if view3D == 1
    MOTIONPLOT3D
% Create index marker for end of leg 4
if t(k) == (leg1t+leg2t+leg3t+leg4t)
    leg4count = k;
end

elseif t(k) > (leg1t+leg2t+leg3t+leg4t) && t(k) <= (leg1t+leg2t+leg3t+leg4t+leg5t)

% Leg 5 of travel
% Define leg-specific motion direction, powered by x(k)
if k == leg4count+1
    disp('check5')
    t1 = PR*leg5t;
    t2 = leg5t-t1;
    posf = leg5d;
    vel1 = posf/(leg5t*(1-PR));
    pos1 = (1/2)*t1*vel1;
    pos2 = posf - pos1;
    phase3counter = 0;
    leg = 5;
    disp('check5')
    ThisLegt = [t0:dt:leg5t];
    count1 = ((length(ThisLegt)-1)*PR)+1;
    TRAJECTORY_GENERATOR_H
    disp('check5')

    % Leg 3 direction of motion, powered by x(k)
    X(k) = 6;
    Y(k) = 4;
    Z(k) = 1 + x(k-leg4count);

    if view3D == 1
        MOTIONPLOT3D
    end

    % Create index marker for end of leg 3
    if t(k) == (leg1t+leg2t+leg3t+leg4t+leg5t)
        leg5count = k;
    end

    elseif t(k) > (leg1t+leg2t+leg3t+leg4t+leg5t) && t(k) <= (leg1t+leg2t+leg3t+leg4t+leg5t+leg6t)

    % Leg 6 of travel
    % Define leg-specific motion direction, powered by x(k)
    if k == leg5count+1
        disp('check6')
        t1 = PR*leg6t;
        t2 = leg6t-t1;
        posf = leg6d;
        vel1 = posf/(leg6t*(1-PR));
        pos1 = (1/2)*t1*vel1;
pos2 = posf - pos1;
phase3counter = 0;
leg = 6;

ThisLegt = [t0:dt:leg6t];
count1 = ((length(ThisLegt)-1)*PR)+1;
TRAJECTORY_GENERATOR_H
end

% Leg 6 direction of motion, powered by x(k)
X(k) = x(k-leg5count)*cos(atan(17/13)) + 6;
Y(k) = x(k-leg5count)*sin(atan(17/13)) + 4;
Z(k) = 4;

if view3D == 1
    MOTIONPLOT3D
end

% Create index marker for end of leg 6
if t(k) == (leg1t+leg2t+leg3t+leg4t+leg5t+leg6t)
    leg6count = k;
end
elseif t(k) > (leg1t+leg2t+leg3t+leg4t+leg5t+leg6t) && t(k) <=
    (leg1t+leg2t+leg3t+leg4t+leg5t+leg6t+leg7t)

% Leg 7 of travel
% Define leg-specific motion direction, powered by x(k)
if k == leg6count+1
    disp('check7')
t1 = PR*leg7t;
t2 = leg7t-t1;
posf = leg7d;
vel1 = posf/(leg7t*(1-PR));
pos1 = (1/2)*t1*vel1;
pos2 = posf - pos1;
phase3counter = 0;
leg = 7;

ThisLegt = [t0:dt:leg7t];
count1 = ((length(ThisLegt)-1)*PR)+1;
TRAJECTORY_GENERATOR_H
end

% Leg 7 direction of motion, powered by x(k)
X(k) = 19;
Y(k) = 21;
Z(k) = 4 - x(k-leg6count);
if view3D == 1
MOTIONPLOT3D
end

% Create index marker for end of leg 7
if t(k) == (leg1t+leg2t+leg3t+leg4t+leg5t+leg6t+leg7t)
  leg7count = k;
end

elseif t(k) > (leg1t+leg2t+leg3t+leg4t+leg5t+leg6t+leg7t) &&
t(k) <= (leg1t+leg2t+leg3t+leg4t+leg5t+leg6t+leg7t+leg8t)

% Leg 8 of travel
% Define leg-specific motion direction, powered by x(k)
if k == leg7count+1
  disp('check8')
t1 = PR*leg8t;
t2 = leg8t-t1;
posf = leg8d;
vel1 = posf/(leg8t*(1-PR));
pos1 = (1/2)*t1*vel1;
pos2 = posf - pos1;
phase3counter = 0;
leg = 8;

ThisLegt = [t0:dt:leg8t];
count1 = ((length(ThisLegt)-1)*PR)+1;
TRAJECTORY_GENERATOR_H
end

% Leg 8 direction of motion, powered by x(k)
X(k) = 19;
Y(k) = 21 - x(k-leg7count);
Z(k) = 1;

if view3D == 1
  MOTIONPLOT3D
end

% Create index marker for end of leg 8
if t(k) == (leg1t+leg2t+leg3t+leg4t+leg5t+leg6t+leg7t+leg8t)
  leg8count = k;
end

elseif t(k) > (leg1t+leg2t+leg3t+leg4t+leg5t+leg6t+leg7t+leg8t) &&
t(k) <= (leg1t+leg2t+leg3t+leg4t+leg5t+leg6t+leg7t+leg8t+leg9t)

% Leg 9 of travel
% Define leg-specific motion direction, powered by x(k)
if k == leg8count+1
  disp('check9')
t1 = PR*leg9t;
t2 = leg9t-t1;
posf = leg9d;
vel1 = posf/(leg9t*(1-PR));
pos1 = (1/2)*t1*vel1;
pos2 = posf - pos1;
phase3counter = 0;
leg = 9;

ThisLegt = [t0:dt:leg9t];
count1 = ((length(ThisLegt)-1)*PR)+1;
TRAJECTORY_GENERATOR_H

end

% Leg 9 direction of motion, powered by x(k)
X(k) = 19;
Y(k) = 4;
Z(k) = 1 + x(k-leg8count);
if view3D == 1
  MOTIONPLOT3D
end

% Create index marker for end of leg 9
if t(k) ==
  (leg1t+leg2t+leg3t+leg4t+leg5t+leg6t+leg7t+leg8t+leg9t)
  leg9count = k;
end

elseif t(k) >
  (leg1t+leg2t+leg3t+leg4t+leg5t+leg6t+leg7t+leg8t+leg9t) && t(k) <=
  (leg1t+leg2t+leg3t+leg4t+leg5t+leg6t+leg7t+leg8t+leg9t+leg10t)

  % Leg 10 of travel
  % Define leg-specific motion direction, powered by x(k)
  if k == leg9count+1
    disp('check10')
    t1 = PR*leg10t;
    t2 = leg10t-t1;
posf = leg10d;
vel1 = posf/(leg10t*(1-PR));
pos1 = (1/2)*t1*vel1;
pos2 = posf - pos1;
phase3counter = 0;
leg = 10;

ThisLegt = [t0:dt:leg10t];
count1 = ((length(ThisLegt)-1)*PR)+1;
TRAJECTORY_GENERATOR_H
end
% Leg 10 direction of motion, powered by x(k)
X(k) = x(k-leg9count)*cos(pi - atan(8.5/18.5)) + 19;
Y(k) = x(k-leg9count)*sin(pi - atan(8.5/18.5)) + 4;
Z(k) = 4;

if view3D == 1
    MOTIONPLOT3D
end

% Create index marker for end of leg 10
if t(k) ==
(leg1t+leg2t+leg3t+leg4t+leg5t+leg6t+leg7t+leg8t+leg9t+legt10)
    leg10count = k;
end

elseif t(k) >
(leg1t+leg2t+leg3t+leg4t+leg5t+leg6t+leg7t+leg8t+leg9t+legt10) && t(k)
<= harvesttf

% Leg 11 of travel
% Define leg-specific motion direction, powered by x(k)
if k == leg10count+1
    disp('check11')
t1 = PR*leg1lt;
t2 = leg1lt-t1;
posf = leg1ld;
vel1 = posf/(leg1lt*(1-PR));
pos1 = (1/2)*t1*vel1;
pos2 = posf - pos1;
phase3counter = 0;
leg = 11;

ThisLegt = [t0:dt:leg11t];
count1 = ((length(ThisLegt)-1)*PR)+1;
TRAJECTORY_GENERATOR_H

end

% Leg 10 direction of motion, powered by x(k)
X(k) = 0.5;
Y(k) = 12.5;
Z(k) = 4 - x(k-leg10count);

if view3D == 1
    MOTIONPLOT3D
end

% Create index marker for end of leg 10
if t(k) ==
(leg1t+leg2t+leg3t+leg4t+leg5t+leg6t+leg7t+leg8t+leg9t+legt10)
    leg10count = k;
% end
%---------------------------------------------
% MENU0.m
% Main Menu
% Programmer: Collier Fais
% Ohio University, Mechanical Engineering
% All Rights Reserved, March 2016
%---------------------------------------------

% Main Menu, menu number 0
% Creates figure window displaying main-menu button options

close all;

disp('MAIN MENU')

name0 = 'Main Menu';

menu0 = figure('Position',[100 354 350 200], 'name', name0, 'NumberTitle','off');

% Display a .jpeg image on the menu
I = imread('C:\Users\cf277_000\Documents\Cable Robot MATLAB\ISO_SW.jpg');
imshow(I)

% Button 0a is Run Program, return '1'
btn0a = uicontrol('Style','pushbutton','string','Run', 'position',[360 10 70 40], ...
 'fontsize',20,...
 'callback','close all; MENU1;')

% Button 0b is About, provides info on program, return '2'
btn0b = uicontrol('Style','pushbutton','string','About', 'position',[440 10 90 40], ...
 'fontsize',20,...
 'callback','choice0 = 2; ABOUT;')

% Button 0c is EXIT, exits program, return '0'
btn0c = uicontrol('Style','pushbutton','string','Exit', 'position',[540 10 70 40], ...
 'fontsize',20,...
 'callback','EXIT0 = 1; close all');
waitfor(menu0); % Waits to execute the rest of the script until user chooses any option

% MENU0, EXIT
if EXIT0 == 1
disp('PROGRAM EXIT');
EXIT0 = EXIT0 + 1;
end

%----------------------------------------------------------
%   MENU1.m
%   Prompts user to select from 'RUN MENU' options
%   Programmer: Collier Fais
%   Ohio University, Mechanical Engineering
%   All Rights Reserved, February 2016
%----------------------------------------------------------
% Using a uicontrol based menu, the user is prompted to choose from the
% following 4 options: Snapshot mode, Trajectory mode, Manual Mode, and
% Exit

close all;

disp('RUN MENU')

name1 = 'Run Menu';

% Creates figure window displaying main-menu button options
menu1 = figure('Position', [100 354 790 200], 'name', name1, 'NumberTitle', 'off');

% Display a .jpeg image on the menu
I = imread('C:\Users\cf277_000\Documents\Cable Robot MATLAB\ISO_ML.jpg');
imshow(I)

% Button 1a is Snapshot
btn1a = uicontrol('Style', 'pushbutton', 'string', 'Snapshot', ...    'position', [10 10 150 40], ...    'fontsize', 20, ...    'callback', 'close all; SNAP;');

% Button 1b is Trajectory Mode, activated downstream if choice1 = 2
btn1b = uicontrol('Style', 'pushbutton', 'string', 'Trajectory', ...    'position', [170 10 150 40], ...
% Button 1c is Manual Control, activated downstream if choice1 = 3
btn1c = uicontrol('Style','pushbutton','string','Manual Control',...
    'position',[330 10 210 40],...
    'fontsize',20,...
    'callback','choice1 = 3; close all; TRAJECTORYMENU;');

% Button 1d is Main Menu, activated downstream if choice1 = 4
btn1d = uicontrol('Style','pushbutton','string','Main Menu',...
    'position',[550 10 150 40],...
    'fontsize',20,...
    'callback','close all; CSR_SIMULATOR;');

% Button 1e is EXIT, activated downstream if choice1 = 0
btn1e = uicontrol('Style','pushbutton','string','Exit',...
    'position',[710 10 70 40],...
    'fontsize',20,...
    'callback','EXIT1 = 1; close all;');

waitfor(menu1); % Waits to execute the rest of the script until user chooses any option

% MENU1, EXIT
if EXIT1 == 1
    disp('PROGRAM EXIT');
    EXIT1 = EXIT1 + 1;
end

%----------------------------------------------------------------
% MOTIONPLOT3D.m
% Plots the snap-shot
% Programmer: Collier Fais
% Ohio University, Mechanical Engineering
% All Rights Reserved, March 2016
%----------------------------------------------------------------

if k == 1
    nameMOTIONPLOT_PLANAR = '2D View Trajectory';
    FourSnapView = figure('position',[10 10 1520 760], 'name',nameMOTIONPLOT_PLANAR, 'NumberTitle','off');
end

% Figure 1, 3D plot
% Plotting the Booms
plot([B01(1) B11(1)], [B01(2) B11(2)], 'k', 'linewidth', Tlw)
hold on
plot([B02(1) B12(1)], [B02(2) B12(2)], 'k', 'linewidth', Tlw)

% Plotting the Cables
plot ([B11(1) X(k)], [B11(2) Y(k)], 'r', 'linewidth', Clw)
plot ([B12(1) X(k)], [B12(2) Y(k)], 'r', 'linewidth', Clw)

title('3D Plot')
grid on;
grid minor;
pl = gca;
set(gca,'Xcolor',[0 0 0]);  %Sets axis line color, not grid lines
set(gca,'Ycolor',[0 0 0]);  %Sets axis line color, not grid lines
set(gca,'Zcolor',[0 0 0]);  %Sets axis line color, not grid lines
% set(gca,'zlim',[0 10]);
whitebg([1 .94 .81]);       % Changes the color of the background
axis equal;

hold off;
pause(1/64);
if k == 1
  txt = uicontrol('Style','text',...
    'Position',[20 20 400 40],...
    'String','Press ''Enter'' to Begin','fontsize',20);
pause
end
if k == length(t)
  % Button 4a is View Report
  btn4a = uicontrol('Style','pushbutton','string','View Report',...
    'position',[20 20 400 40],...
    'fontsize',20,...
    'callback','TRAJECTORY_PLOTS_PLANAR;');

  btn4b = uicontrol('Style','pushbutton','string','Return',...
    'position',[500 20 150 40],...
    'fontsize',20,...
    'callback','close all; TRAJECTORYMENU;');

  btn4b = uicontrol('Style','pushbutton','string','Exit',...
    'position',[750 20 150 40],...
    'fontsize',20,...
    'callback','close all;');
end

%----------------------------------------------------------------
% MOTIONPLOT3D.m
% Plots the snap-shot
% Programmer: Collier Fais
% Ohio University, Mechanical Engineering
% All Rights Reserved, March 2016
%----------------------------------------------------------------
if k == 1
    nameMOTIONPLOT3D = '3D View Trajectory';
    FourSnapView = figure('position',[10 10 1520 760],'
    name',nameMOTIONPLOT3D,'
    NumberTitle','off');
end

% Figure 1, 3D plot
% Plotting the Booms
plot3([B01(1) B11(1)], [B01(2) B11(2)], [B01(3) B11(3)], 'k',
    'linewidth', Tlw)
hold on
plot3([B02(1) B12(1)], [B02(2) B12(2)], [B02(3) B12(3)], 'k',
    'linewidth', Tlw)
plot3([B03(1) B13(1)], [B03(2) B13(2)], [B03(3) B13(3)], 'k',
    'linewidth', Tlw)
plot3([B04(1) B14(1)], [B04(2) B14(2)], [B04(3) B14(3)], 'k',
    'linewidth', Tlw)

%Edge of plot-space patch
patch([0-Dx 0-Dx W+Dx W+Dx],[0-Dy L+Dy L+Dy 0-Dy],[-0.1 -0.1 -0.1 -
0.1],[0.467 .392 .314])

%Workspace footprint patch
patch([0 0 W W],[0 L L 0],[0.882 1 0.618])

% Plotting the Bottom Beams
plot3([M01(1) M11(1)], [M01(2) M11(2)], [M01(3) M11(3)], 'k',
    'linewidth', Tlw)
plot3([M02(1) M12(1)], [M02(2) M12(2)], [M02(3) M12(3)], 'k',
    'linewidth', Tlw)
plot3([M03(1) M13(1)], [M03(2) M13(2)], [M03(3) M13(3)], 'k',
    'linewidth', Tlw)
plot3([M04(1) M14(1)], [M04(2) M14(2)], [M04(3) M14(3)], 'k',
    'linewidth', Tlw)

% Plotting the Runners
plot3([R01(1) R11(1)], [R01(2) R11(2)], [R01(3) R11(3)], 'k',
    'linewidth', Tlw)
plot3([R02(1) R12(1)], [R02(2) R12(2)], [R02(3) R12(3)], 'k',
    'linewidth', Tlw)
plot3([R03(1) R13(1)], [R03(2) R13(2)], [R03(3) R13(3)], 'k',
    'linewidth', Tlw)
plot3([R04(1) R14(1)], [R04(2) R14(2)], [R04(3) R14(3)], 'k',
    'linewidth', Tlw)

% Plotting the Side Supports [a]
plot3([S01a(1) S11a(1)], [S01a(2) S11a(2)], [S01a(3) S11a(3)], 'k',
    'linewidth', Tlw)
plot3([S02a(1) S12a(1)], [S02a(2) S12a(2)], [S02a(3) S12a(3)], 'k',
    'linewidth', Tlw)
plot3([S03a(1) S13a(1)], [S03a(2) S13a(2)], [S03a(3) S13a(3)], 'k',
    'linewidth', Tlw)
% Plotting the Side Supports [b]
plot3([S01b(1) S11b(1)], [S01b(2) S11b(2)], [S01b(3) S11b(3)], 'k', 'linewidth', Tlw)
plot3([S02b(1) S12b(1)], [S02b(2) S12b(2)], [S02b(3) S12b(3)], 'k', 'linewidth', Tlw)
plot3([S03b(1) S13b(1)], [S03b(2) S13b(2)], [S03b(3) S13b(3)], 'k', 'linewidth', Tlw)
plot3([S04b(1) S14b(1)], [S04b(2) S14b(2)], [S04b(3) S14b(3)], 'k', 'linewidth', Tlw)

% Plotting the Back Support
plot3([A01(1) A11(1)], [A01(2) A11(2)], [A01(3) A11(3)], 'k', 'linewidth', Tlw)
plot3([A02(1) A12(1)], [A02(2) A12(2)], [A02(3) A12(3)], 'k', 'linewidth', Tlw)
plot3([A03(1) A13(1)], [A03(2) A13(2)], [A03(3) A13(3)], 'k', 'linewidth', Tlw)
plot3([A04(1) A14(1)], [A04(2) A14(2)], [A04(3) A14(3)], 'k', 'linewidth', Tlw)

% Plotting the Cables
plot3([B11(1) X(k)], [B11(2) Y(k)], [B11(3) Z(k)], 'r', 'linewidth', Clw)
plot3([B12(1) X(k)], [B12(2) Y(k)], [B12(3) Z(k)], 'r', 'linewidth', Clw)
plot3([B13(1) X(k)], [B13(2) Y(k)], [B13(3) Z(k)], 'r', 'linewidth', Clw)
plot3([B14(1) X(k)], [B14(2) Y(k)], [B14(3) Z(k)], 'r', 'linewidth', Clw)

% Plotting end-effector path
if trajectory == 1
    if k == 1
        % Plotting the Cone end effector, just size here
        cyl1 = [0 .35];
        cyl2 = [.35 0];
    end

    plot3([2.5 X(k)], [W/2 W/2], [1 1], 'c', 'linewidth', Clw) % Plots trajectory path
    [x1,y1,z1] = cylinder(cyl1);
    [x2,y2,z2] = cylinder(cyl2);
    surf(x1+X(k),y1+Y(k),z1+Z(k))
    surf(x2+X(k),y2+Y(k),z2+Z(k)-1)
elseif trajectory == 2
    % Further code for trajectory 2
end
Theta = 2*pi*(x(k)/posf);  % Calculates theta of circular path
ThetaArray = [ThetaArray,Theta]; % Stores theta values, becomes 1 entry larger with each loop
xcir = xyoff(1)+r*cos(ThetaArray);  % Circle plotting engine
ycir = xyoff(2)+r*sin(ThetaArray);
zcir = [zcir,1];
plot3(xcir,ycir,zcir,'c')

if k == 1
  % Plotting the Cone end effector, just size here
  cyl1 = [0 .35];
  cyl2 = [.35 0];
end

[x1,y1,z1] = cylinder(cyl1);
[x2,y2,z2] = cylinder(cyl2);
surf(x1+X(k),y1+Y(k),z1+Z(k))
surf(x2+X(k),y2+Y(k),z2+Z(k)-1)

elseif trajectory == 3
  if leg == 1
    plot3([0.5 X(k)],[12.5 Y(k)],[0.5 Z(k)],'c','linewidth', Plw)

    % End-effector patch work
    patch([X(k)-0.5 X(k)-0.5 X(k)+0.5 X(k)+0.5],[Y(k)-0.5 Y(k)+0.5 Y(k)+0.5 Y(k)-0.5],[Z(k) Z(k) Z(k) Z(k)],[1 1 1])
    patch([X(k)-0.5 X(k)-0.5 X(k)-0.5 X(k)-0.5],[Y(k)+0.3 Y(k)-0.3 Y(k)+0.5 Y(k)-0.5],[Z(k)-0.5 Z(k) Z(k) Z(k)-0.5],[1 1 1])
    patch([X(k)-0.5 X(k)-0.5 X(k)+0.5 X(k)+0.5],[Y(k)+0.3 Y(k)-0.3 Y(k)+0.5 Y(k)-0.5],[Z(k)-0.5 Z(k) Z(k) Z(k)-0.5],[1 1 1])
    patch([X(k)-0.5 X(k)-0.5 X(k)-0.5 X(k)-0.5],[Y(k)-0.5 Y(k)-0.5 Y(k)+0.5 Y(k)+0.5],[Z(k) Z(k) Z(k) Z(k)],[1 1 1])
  elseif leg == 2
    plot3([0.5 0.5],[12.5 12.5],[0.5 4],'k','linewidth', 0.5)
    plot3([0.5 6],[12.5 21],[4 4],'k','linewidth', 0.5)
    patch([X(k)-0.5 X(k)-0.5 X(k)+0.5 X(k)+0.5],[Y(k)-0.5 Y(k)+0.5 Y(k)-0.5 Y(k)+0.5],[Z(k) Z(k) Z(k) Z(k)],[1 1 1])
  elseif leg == 3
    plot3([0.5 6],[12.5 21],[4 4],'k','linewidth', 0.5)
    plot3([6 X(k)],[21 Y(k)],[4 Z(k)],[1 1 1])
    patch([X(k)-0.5 X(k)-0.5 X(k)+0.5 X(k)+0.5],[Y(k)-0.5 Y(k)+0.5 Y(k)-0.5 Y(k)+0.5],[Z(k) Z(k) Z(k) Z(k)],[1 1 1])
patch([X(k)-0.5 X(k)-0.5 X(k)-0.5 X(k)-0.5], [Y(k)+0.3 Y(k)+0.5 Y(k)-0.5 Y(k)-0.5], [Z(k) Z(k) Z(k) Z(k)], [1 1 1])
patch([X(k)+0.5 X(k)+0.5 X(k)+0.5 X(k)+0.5], [Y(k)+0.3 Y(k)+0.5 Y(k)-0.5 Y(k)-0.5], [Z(k) Z(k) Z(k) Z(k)], [1 1 1])
patch([X(k)-0.5 X(k)-0.5 X(k)-0.5 X(k)-0.5], [Y(k)-0.5 Y(k)-0.5 Y(k)-0.5 Y(k)-0.5], [Z(k) Z(k) Z(k) Z(k)], [1 1 1])

elseif leg == 4
plot3([0.5 0.5], [12.5 12.5], [0.5 4], 'k', 'linewidth', 0.5)
plot3([0.5 6], [12.5 21], [4 4], 'k', 'linewidth', 0.5)
plot3([6 6], [21 21], [4 4], 'k', 'linewidth', 0.5)
plot3([6 X(k)], [4 Y(k)], [1 1], 'k', 'linewidth', Plw)

patch([X(k)-0.5 X(k)-0.5 X(k)-0.5 X(k)-0.5], [Y(k)-0.5 Y(k)+0.5 Y(k)+0.5 Y(k)-0.5], [Z(k) Z(k) Z(k) Z(k)], [(1-(0.549*(x(k-leg3count)/posf))) 1 (1-(0.549*(x(k-leg3count)/posf)))])
patch([X(k)-0.5 X(k)-0.5 X(k) X(k)], [Y(k)+0.3 Y(k)+0.5 Y(k)-0.5 Y(k)-0.5], [Z(k) Z(k) Z(k) Z(k)], [(1-(0.549*(x(k-leg3count)/posf))) 1 (1-(0.549*(x(k-leg3count)/posf)))])
patch([X(k)+0.5 X(k)+0.5 X(k)+0.5 X(k)+0.5], [Y(k)+0.3 Y(k)+0.5 Y(k)-0.5 Y(k)-0.5], [Z(k) Z(k) Z(k) Z(k)], [(1-(0.549*(x(k-leg3count)/posf))) 1 (1-(0.549*(x(k-leg3count)/posf)))])
patch([X(k)-0.5 X(k)-0.5 X(k)+0.5 X(k)+0.5], [Y(k)-0.5 Y(k)+0.3 Y(k)+0.3 Y(k)-0.5], [Z(k) Z(k) Z(k) Z(k)], [(1-(0.549*(x(k-leg3count)/posf))) 1 (1-(0.549*(x(k-leg3count)/posf)))])

elseif leg == 5
plot3([0.5 0.5], [12.5 12.5], [0.5 4], 'k', 'linewidth', 0.5)
plot3([0.5 6], [12.5 21], [4 4], 'k', 'linewidth', 0.5)
plot3([6 6], [21 21], [4 4], 'k', 'linewidth', 0.5)
plot3([6 6], [21 4], [1 1], 'k', 'linewidth', Plw)
plot3([6 6], [4 4], [1 4], 'k', 'linewidth', Plw)
plot3([6 X(k)], [4 Y(k)], [4 Z(k)], 'c', 'linewidth', Plw)

patch([X(k)-0.5 X(k)-0.5 X(k)-0.5 X(k)-0.5], [Y(k)-0.5 Y(k)+0.5 Y(k)-0.5 Y(k)-0.5], [Z(k) Z(k) Z(k) Z(k)], [0.451 1 0.859-(0.408*(x(k-leg4count)/posf))])
patch([X(k)-0.5 X(k)-0.5 X(k)-0.5 X(k)-0.5], [Y(k)+0.3 Y(k)+0.5 Y(k)-0.5 Y(k)-0.5], [Z(k) Z(k) Z(k) Z(k)], [0.451 1 0.859-(0.408*(x(k-leg4count)/posf))])
patch([X(k)+0.5 X(k)+0.5 X(k)+0.5 X(k)+0.5], [Y(k)+0.3 Y(k)+0.5 Y(k)-0.5 Y(k)-0.5], [Z(k) Z(k) Z(k) Z(k)], [0.451 1 0.859-(0.408*(x(k-leg4count)/posf))])
patch([X(k)-0.5 X(k)-0.5 X(k)+0.5 X(k)+0.5], [Y(k)-0.5 Y(k)+0.3 Y(k)-0.5 Y(k)-0.5], [Z(k) Z(k) Z(k) Z(k)], [0.451 1 0.859-(0.408*(x(k-leg4count)/posf))])

elseif leg == 6
plot3([0.5 0.5], [12.5 12.5], [0.5 4], 'k', 'linewidth', 0.5)
plot3([0.5 6], [12.5 21], [4 4], 'k', 'linewidth', 0.5)
plot3([6 6], [21 21], [4 4], 'k', 'linewidth', 0.5)
plot3([6 6], [21 4], [1 4], 'k', 'linewidth', 0.5)
plot3([6 6], [4 4], [1 4], 'k', 'linewidth', 0.5)
plot3([6 X(k)], [4 Y(k)], [4 Z(k)], 'c', 'linewidth', Plw)
patch([X(k) - 0.5 X(k) - 0.5 X(k) + 0.5 X(k) + 0.5], [Y(k) - 0.5 Y(k) + 0.5 Y(k) + 0.5 Y(k) - 0.5], [Z(k) Z(k) Z(k) Z(k)], [0.451 1 0.451])
patch([X(k) - 0.5 X(k) - 0.5 X(k) - 0.5 X(k) - 0.5], [Y(k) + 0.3 Y(k) + 0.5 Y(k) - 0.5], [Z(k) - 0.5 Z(k) Z(k) Z(k) - 0.5], [0.451 1 0.451])
patch([X(k) + 0.5 X(k) + 0.5 X(k) + 0.5 X(k) + 0.5], [Y(k) + 0.3 Y(k) + 0.5 Y(k) - 0.5], [Z(k) - 0.5 Z(k) Z(k) Z(k) - 0.5], [0.451 1 0.451])
patch([X(k) - 0.5 X(k) - 0.5 X(k) + 0.5 X(k) + 0.5], [Y(k) - 0.5 Y(k) + 0.3 Y(k) + 0.3 Y(k) - 0.5], [Z(k) - 0.5 Z(k) - 0.5 Z(k) - 0.5 Z(k) - 0.5], [0.451 1 0.451])

elseif leg == 7
plot3([0.5 0.5], [12.5 12.5], [0.5 4], 'k', 'linewidth', 0.5)
plot3([0.5 6], [12.5 21], [4 4], 'k', 'linewidth', 0.5)
plot3([6 6], [21 21], [4 1], 'k', 'linewidth', 0.5)
plot3([6 4], [21 4], [1 1], 'k', 'linewidth', 0.5)
plot3([6 6], [4 4], [1 4], 'k', 'linewidth', 0.5)
plot3([6 19], [4 21], [4 4], 'k', 'linewidth', 0.5)
plot3([19 X(k)], [21 Y(k)], [4 Z(k)], 'c', 'linewidth', Plw)
patch([X(k) - 0.5 X(k) - 0.5 X(k) + 0.5 X(k) + 0.5], [Y(k) - 0.5 Y(k) + 0.5 Y(k) + 0.5 Y(k) - 0.5], [Z(k) Z(k) Z(k) Z(k)], [0.451 1 0.451])
patch([X(k) - 0.5 X(k) - 0.5 X(k) - 0.5 X(k) - 0.5], [Y(k) + 0.3 Y(k) + 0.5 Y(k) - 0.5], [Z(k) - 0.5 Z(k) Z(k) Z(k) - 0.5], [0.451 1 0.451])
patch([X(k) + 0.5 X(k) + 0.5 X(k) + 0.5 X(k) + 0.5], [Y(k) + 0.3 Y(k) + 0.5 Y(k) - 0.5], [Z(k) - 0.5 Z(k) Z(k) Z(k) - 0.5], [0.451 1 0.451])
patch([X(k) - 0.5 X(k) - 0.5 X(k) + 0.5 X(k) + 0.5], [Y(k) - 0.5 Y(k) + 0.3 Y(k) + 0.3 Y(k) - 0.5], [Z(k) - 0.5 Z(k) - 0.5 Z(k) - 0.5 Z(k) - 0.5], [0.451 - (0.451*(x(k-leg7count)/posf)) 1 0.451+(0.137*(x(k-leg7count)/posf))] 

elseif leg == 8
plot3([0.5 0.5], [12.5 12.5], [0.5 4], 'k', 'linewidth', 0.5)
plot3([0.5 6], [12.5 21], [4 4], 'k', 'linewidth', 0.5)
plot3([6 6], [21 21], [4 1], 'k', 'linewidth', 0.5)
plot3([6 4], [21 4], [1 1], 'k', 'linewidth', 0.5)
plot3([6 6], [4 4], [1 4], 'k', 'linewidth', 0.5)
plot3([6 19], [4 21], [4 4], 'k', 'linewidth', 0.5)
plot3([19 19], [21 21], [4 1], 'k', 'linewidth', 0.5)
plot3([19 X(k)], [21 Y(k)], [1 Z(k)], 'c', 'linewidth', Plw)
patch([X(k) - 0.5 X(k) - 0.5 X(k) + 0.5 X(k) + 0.5], [Y(k) - 0.5 Y(k) + 0.5 Y(k) + 0.5 Y(k) - 0.5], [Z(k) Z(k) Z(k) Z(k)], [0.451 - (0.451*(x(k-leg7count)/posf)) 1 0.451+(0.137*(x(k-leg7count)/posf))])
patch([X(k) - 0.5 X(k) - 0.5 X(k) - 0.5 X(k) - 0.5], [Y(k) + 0.3 Y(k) + 0.5 Y(k) - 0.5], [Z(k) - 0.5 Z(k) Z(k) Z(k) - 0.5], [0.451 - (0.451*(x(k-leg7count)/posf)) 1 0.451+(0.137*(x(k-leg7count)/posf))])
patch([X(k) + 0.5 X(k) + 0.5 X(k) + 0.5 X(k) + 0.5], [Y(k) + 0.3 Y(k) + 0.5 Y(k) - 0.5], [Z(k) - 0.5 Z(k) Z(k) Z(k) - 0.5], [0.451 - (0.451*(x(k-leg7count)/posf)) 1 0.451+(0.137*(x(k-leg7count)/posf))])
patch([X(k) - 0.5 X(k) - 0.5 X(k) + 0.5 X(k) + 0.5], [Y(k) - 0.5 Y(k) + 0.3 Y(k) + 0.3 Y(k) - 0.5], [Z(k) - 0.5 Z(k) - 0.5 Z(k) - 0.5 Z(k) - 0.5], [0.451 - (0.451*(x(k-leg7count)/posf)) 1 0.451+(0.137*(x(k-leg7count)/posf))])

elseif leg == 9
plot3([0.5 0.5], [12.5 12.5], [0.5 4], 'k', 'linewidth', 0.5)
plot3([0.5 6], [12.5 21], [4 4], 'k', 'linewidth', 0.5)
plot3([6 6], [21 21], [4 1], 'k', 'linewidth', 0.5)
patch([X(k)+0.5 X(k)+0.5 X(k)+0.5 X(k)+0.5], [Y(k)+0.3 Y(k)+0.5 Y(k)-0.5 Y(k)-0.5], [Z(k)-0.5 Z(k) Z(k) Z(k)-0.5], [0 1 0])
patch([X(k)+0.5 X(k)+0.5 X(k)+0.5 X(k)+0.5], [Y(k)+0.3 Y(k)+0.5 Y(k)-0.5 Y(k)-0.5], [Z(k)-0.5 Z(k) Z(k) Z(k)-0.5], [0 1 0])
patch([X(k)-0.5 X(k)-0.5 X(k)+0.5 X(k)+0.5], [Y(k)-0.3 Y(k)+0.3 Y(k)+0.3 Y(k)-0.5], [Z(k)-0.5 Z(k)-0.5 Z(k) Z(k)-0.5], [0 1 0])

elseif trajectory == 4
    [x_sp,y_sp,z_sp] = sphere(20);
surf(x_sp+X(k),y_sp+Y(k),z_sp+Z(k))
end
end

title('3D Plot')
xlabel('Width [ft]')
ylabel('Length [ft]')
zlabel('Height [ft]')
grid on;
grid minor;
pl = gca;
set(gca,'Xcolor',[0 0 0]);  %Sets axis line color, not grid lines
set(gca,'Ycolor',[0 0 0]);  %Sets axis line color, not grid lines
set(gca,'Zcolor',[0 0 0]);  %Sets axis line color, not grid lines
% set(gca,'zlim',[0 10]);
whitebg([1 .94 .81]);       % Changes the color of the background
ylim([L-Dy L+Dy]);
xlim([W-Dx W+Dx]);
zlim([0 Ht]);
axis equal;
hold off;
pause(1/64);

if k == 1
    txt = uicontrol('Style','text',...
    'Position',[20 20 400 40],...
    'String','Press 'Enter' to Begin','fontsize',20);
pause
end

if k == length(t)
    % Button 4a is View Report
    btn4a = uicontrol('Style','pushbutton','string','View Report',...
    'position',[20 20 400 40],...
    'fontsize',20,...
    'callback','TRAJECTORY_PLOTS;');

    btn4b = uicontrol('Style','pushbutton','string','Return',...
    'position',[500 20 150 40],...
    'fontsize',20,...
    'callback','close all; TRAJECTORYMENU;');
btn4b = uicontrol('Style','pushbutton','string','Exit', ...  
'position',[750 20 150 40], ...  
'fontsize',20,...  
'callback','close all;');

%----------------------------------------------------------------
%   MOTIONPLOT4.m
%   Plots the snap-shot
%   Programmer: Collier Fais
%   Ohio University, Mechanical Engineering
%   All Rights Reserved, March 2016
%----------------------------------------------------------------

nameSNAP_PLOT = 'Font, Top, Right and 3D View Snapshot';
% Figure 1, 3D plot
subplot(222)
% Plotting the Booms
plot3([B01(1) B11(1)], [B01(2) B11(2)], [B01(3) B11(3)], 'k',  
'linewidth', Tlw)
hold on
plot3([B02(1) B12(1)], [B02(2) B12(2)], [B02(3) B12(3)], 'k',  
'linewidth', Tlw)
plot3([B03(1) B13(1)], [B03(2) B13(2)], [B03(3) B13(3)], 'k',  
'linewidth', Tlw)
plot3([B04(1) B14(1)], [B04(2) B14(2)], [B04(3) B14(3)], 'k',  
'linewidth', Tlw)

% Edge of plot-space patch
patch([0-Dx 0-Dx W+Dx W+Dx], [0-Dy L+Dy L+Dy 0-Dy], [-0.1 -0.1 -0.1 -0.1], [0 1 0])

% Workspace footprint patch
patch([0 0 W W], [0 L L 0], [.73 1 .96])

% Plotting the Bottom Beams
plot3([M01(1) M11(1)], [M01(2) M11(2)], [M01(3) M11(3)], 'k',  
'linewidth', Tlw)
plot3([M02(1) M12(1)], [M02(2) M12(2)], [M02(3) M12(3)], 'k',  
'linewidth', Tlw)
plot3([M03(1) M13(1)], [M03(2) M13(2)], [M03(3) M13(3)], 'k',  
'linewidth', Tlw)
plot3([M04(1) M14(1)], [M04(2) M14(2)], [M04(3) M14(3)], 'k',  
'linewidth', Tlw)

% Plotting the Runners
plot3([R01(1) R11(1)], [R01(2) R11(2)], [R01(3) R11(3)], 'k',  
'linewidth', Tlw)
plot3([R02(1) R12(1)], [R02(2) R12(2)], [R02(3) R12(3)], 'k',  
'linewidth', Tlw)
plot3([R03(1) R13(1)], [R03(2) R13(2)], [R03(3) R13(3)], 'k', 'linewidth', Tlw)
plot3([R04(1) R14(1)], [R04(2) R14(2)], [R04(3) R14(3)], 'k', 'linewidth', Tlw)

% Plotting the Side Supports [a]
plot3([S01a(1) S11a(1)], [S01a(2) S11a(2)], [S01a(3) S11a(3)], 'k', 'linewidth', Tlw)
plot3([S02a(1) S12a(1)], [S02a(2) S12a(2)], [S02a(3) S12a(3)], 'k', 'linewidth', Tlw)
plot3([S03a(1) S13a(1)], [S03a(2) S13a(2)], [S03a(3) S13a(3)], 'k', 'linewidth', Tlw)
plot3([S04a(1) S14a(1)], [S04a(2) S14a(2)], [S04a(3) S14a(3)], 'k', 'linewidth', Tlw)

% Plotting the Side Supports [b]
plot3([S01b(1) S11b(1)], [S01b(2) S11b(2)], [S01b(3) S11b(3)], 'k', 'linewidth', Tlw)
plot3([S02b(1) S12b(1)], [S02b(2) S12b(2)], [S02b(3) S12b(3)], 'k', 'linewidth', Tlw)
plot3([S03b(1) S13b(1)], [S03b(2) S13b(2)], [S03b(3) S13b(3)], 'k', 'linewidth', Tlw)
plot3([S04b(1) S14b(1)], [S04b(2) S14b(2)], [S04b(3) S14b(3)], 'k', 'linewidth', Tlw)

% Plotting the Back Support
plot3([A01(1) A11(1)], [A01(2) A11(2)], [A01(3) A11(3)], 'k', 'linewidth', Tlw)
plot3([A02(1) A12(1)], [A02(2) A12(2)], [A02(3) A12(3)], 'k', 'linewidth', Tlw)
plot3([A03(1) A13(1)], [A03(2) A13(2)], [A03(3) A13(3)], 'k', 'linewidth', Tlw)
plot3([A04(1) A14(1)], [A04(2) A14(2)], [A04(3) A14(3)], 'k', 'linewidth', Tlw)

% Plotting the Cables
plot3([B11(1) X(k)], [B11(2) Y(k)], [B11(3) Z(k)], 'r', 'linewidth', Clw)
plot3([B12(1) X(k)], [B12(2) Y(k)], [B12(3) Z(k)], 'r', 'linewidth', Clw)
plot3([B13(1) X(k)], [B13(2) Y(k)], [B13(3) Z(k)], 'r', 'linewidth', Clw)
plot3([B14(1) X(k)], [B14(2) Y(k)], [B14(3) Z(k)], 'r', 'linewidth', Clw)
plot3([2.5 X(k)], [W/2 W/2], [1 1], 'c', 'linewidth', Clw)

% Plotting the Cone end effector

cyl1 = [0 .35];
cyl2 = [.35 0];
[x1,y1,z1] = cylinder(cyl1);
[x2,y2,z2] = cylinder(cyl2);
surf(x1+X(k),y1+Y(k),z1+Z(k))
surf(x2+X(k),y2+Y(k),z2+Z(k)-1)

title('3D Plot')
xlabel('Width [ft]')
ylabel('Length [ft]')
zlabel('Height [ft]')

grid on;
grid minor;
p1 = gca;
set(gca,'Xcolor',[0 0 0]); %Sets axis line color, not grid lines
set(gca,'Ycolor',[0 0 0]); %Sets axis line color, not grid lines
set(gca,'Zcolor',[0 0 0]); %Sets axis line color, not grid lines
% set(gca,'zlim',[0 10]); % Changes the color of the background
axis equal;
hold off;
pause(1/64);

% Figure 2 x-y plane, top view
subplot(221)
% Plotting the Booms
plot([B01(1) B11(1)], [B01(2) B11(2)], 'k', 'linewidth', Tlw)
hold on
plot([B02(1) B12(1)], [B02(2) B12(2)], 'k', 'linewidth', Tlw)
plot([B03(1) B13(1)], [B03(2) B13(2)], 'k', 'linewidth', Tlw)
plot([B04(1) B14(1)], [B04(2) B14(2)], 'k', 'linewidth', Tlw)

%Edge of plot-space patch
patch([0-Dx 0-Dx W+Dx W+Dx], [0-Dy L+Dy L+Dy 0-Dy], [-0.1 -0.1 -0.1 -0.1], [0 1 0 0])

%Workspace footprint patch
patch([0 0 W W], [0 L L 0], [.73 1 .96])

% Plotting the Bottom Beams
plot([M01(1) M11(1)], [M01(2) M11(2)], 'k', 'linewidth', Tlw)
plot([M02(1) M12(1)], [M02(2) M12(2)], 'k', 'linewidth', Tlw)
plot([M03(1) M13(1)], [M03(2) M13(2)], 'k', 'linewidth', Tlw)
plot([M04(1) M14(1)], [M04(2) M14(2)], 'k', 'linewidth', Tlw)

% Plotting the Runners
plot([R01(1) R11(1)], [R01(2) R11(2)], 'k', 'linewidth', Tlw)
plot([R02(1) R12(1)], [R02(2) R12(2)], 'k', 'linewidth', Tlw)
plot([R03(1) R13(1)], [R03(2) R13(2)], 'k', 'linewidth', Tlw)
plot([R04(1) R14(1)], [R04(2) R14(2)], 'k', 'linewidth', Tlw)

% Plotting the Side Supports [a]
plot([S01a(1) S11a(1)], [S01a(2) S11a(2)], 'k', 'linewidth', Tlw)
plot([S02a(1) S12a(1)], [S02a(2) S12a(2)], 'k', 'linewidth', Tlw)
plot([S03a(1) S13a(1)], [S03a(2) S13a(2)], 'k', 'linewidth', Tlw)
plot([S04a(1) S14a(1)], [S04a(2) S14a(2)], 'k', 'linewidth', Tlw)

% Plotting the Side Supports [b]
plot([S01b(1) S11b(1)], [S01b(2) S11b(2)], 'k', 'linewidth', Tlw)
plot([S02b(1) S12b(1)], [S02b(2) S12b(2)], 'k', 'linewidth', Tlw)
plot([S03b(1) S13b(1)], [S03b(2) S13b(2)], 'k', 'linewidth', Tlw)
plot([S04b(1) S14b(1)], [S04b(2) S14b(2)], 'k', 'linewidth', Tlw)

% Plotting the Back Support
plot([A01(1) A11(1)], [A01(2) A11(2)], 'k', 'linewidth', Tlw)
plot([A02(1) A12(1)], [A02(2) A12(2)], 'k', 'linewidth', Tlw)
plot([A03(1) A13(1)], [A03(2) A13(2)], 'k', 'linewidth', Tlw)
plot([A04(1) A14(1)], [A04(2) A14(2)], 'k', 'linewidth', Tlw)

% Plotting the Cables
plot([B11(1) X(k)], [B11(2) Y(k)], 'r', 'linewidth', Clw)
plot([B12(1) X(k)], [B12(2) Y(k)], 'r', 'linewidth', Clw)
plot([B13(1) X(k)], [B13(2) Y(k)], 'r', 'linewidth', Clw)
plot([B14(1) X(k)], [B14(2) Y(k)], 'r', 'linewidth', Clw)

Circle(X(k), Y(k), 0.5);

title('X-Y Plane, Top View')
xlabel('Width [ft]')
ylabel('Length [ft]')

grid on;
grid minor;
pl = gca;
set(gca,'Xcolor',[0 0 0]);  %Sets axis line color, not grid lines
set(gca,'Ycolor',[0 0 0]);  %Sets axis line color, not grid lines
set(gca,'Zcolor',[0 0 0]);  %Sets axis line color, not grid lines
whitebg([1 .94 .81]);       % Changes the color of the background
axis equal;
hold off;
pause(1/64);

% Figure 3, x-z plane, front view
% figure(3);
% subplot(223)
% Plotting the Booms
plot([B01(1) B11(1)], [B01(3) B11(3)], 'k', 'linewidth', Tlw)
hold on
plot([B02(1) B12(1)], [B02(3) B12(3)], 'k', 'linewidth', Tlw)
plot([B03(1) B13(1)], [B03(3) B13(3)], 'k', 'linewidth', Tlw)
plot([B04(1) B14(1)], [B04(3) B14(3)], 'k', 'linewidth', Tlw)

% Plot edge of plot-space with a line instead of a patch
plot([-0.1 -0.1], [-0.1 -0.1], 'g', 'linewidth', 1)

% Plot workspace with a line instead of patch
plot([0 W], [0 0], 'c', 'linewidth', 2)
% Plotting the Bottom Beams
plot([M01(1) M11(1)], [M01(3) M11(3)], 'k', 'linewidth', Tlw)
plot([M02(1) M12(1)], [M02(3) M12(3)], 'k', 'linewidth', Tlw)
plot([M03(1) M13(1)], [M03(3) M13(3)], 'k', 'linewidth', Tlw)
plot([M04(1) M14(1)], [M04(3) M14(3)], 'k', 'linewidth', Tlw)

% Plotting the Runners
plot([R01(1) R11(1)], [R01(3) R11(3)], 'k', 'linewidth', Tlw)
plot([R02(1) R12(1)], [R02(3) R12(3)], 'k', 'linewidth', Tlw)
plot([R03(1) R13(1)], [R03(3) R13(3)], 'k', 'linewidth', Tlw)
plot([R04(1) R14(1)], [R04(3) R14(3)], 'k', 'linewidth', Tlw)

% Plotting the Side Supports [a]
plot([S01a(1) S11a(1)], [S01a(3) S11a(3)], 'k', 'linewidth', Tlw)
plot([S02a(1) S12a(1)], [S02a(3) S12a(3)], 'k', 'linewidth', Tlw)
plot([S03a(1) S13a(1)], [S03a(3) S13a(3)], 'k', 'linewidth', Tlw)
plot([S04a(1) S14a(1)], [S04a(3) S14a(3)], 'k', 'linewidth', Tlw)

% Plotting the Side Supports [b]
plot([S01b(1) S11b(1)], [S01b(3) S11b(3)], 'k', 'linewidth', Tlw)
plot([S02b(1) S12b(1)], [S02b(3) S12b(3)], 'k', 'linewidth', Tlw)
plot([S03b(1) S13b(1)], [S03b(3) S13b(3)], 'k', 'linewidth', Tlw)
plot([S04b(1) S14b(1)], [S04b(3) S14b(3)], 'k', 'linewidth', Tlw)

% Plotting the Back Support
plot([A01(1) A11(1)], [A01(3) A11(3)], 'k', 'linewidth', Tlw)
plot([A02(1) A12(1)], [A02(3) A12(3)], 'k', 'linewidth', Tlw)
plot([A03(1) A13(1)], [A03(3) A13(3)], 'k', 'linewidth', Tlw)
plot([A04(1) A14(1)], [A04(3) A14(3)], 'k', 'linewidth', Tlw)

% Plotting the Cables
plot([B11(1) X(k)], [B11(3) Z(k)], 'r', 'linewidth', Clw)
plot([B12(1) X(k)], [B12(3) Z(k)], 'r', 'linewidth', Clw)
plot([B13(1) X(k)], [B13(3) Z(k)], 'r', 'linewidth', Clw)
plot([B14(1) X(k)], [B14(3) Z(k)], 'r', 'linewidth', Clw)

% Patch triangles that look like the cones for the iso view
patch([X(k) X(k)-cyl1(2) X(k)+cyl1(2)], [Z(k) Z(k)+1 Z(k)+1], [0 0 0], 'g')
patch([X(k) X(k)-cyl1(2) X(k)+cyl1(2)], [Z(k) Z(k)-1 Z(k)-1], [0 0 0], 'g')

title('X-Z Plane, Front View')
xlabel('Width [ft]')
ylabel('Height [ft]')
axis equal;
grid on;
grid minor;
hold off;
pause(1/64);
% Figure 4, y-z plane, Right view
% figure(4);
subplot(224)

% Plotting the Booms
plot([B01(2) B11(2)], [B01(3) B11(3)], 'k', 'linewidth', Tlw)
hold on
plot([B02(2) B12(2)], [B02(3) B12(3)], 'k', 'linewidth', Tlw)
plot([B03(2) B13(2)], [B03(3) B13(3)], 'k', 'linewidth', Tlw)
plot([B04(2) B14(2)], [B04(3) B14(3)], 'k', 'linewidth', Tlw)

% Plot edge of plot-space with a line instead of a patch
plot([0-Dy L+Dy],[-0.1 -0.1], 'g', 'linewidth', 1)
% Plot workspace with a line instead of patch
plot([0 L],[0 0], 'c', 'linewidth', 2)

% Patch triangles that look like the cones for the iso view
patch([Y(k) Y(k)-cyl1(2) Y(k)+cyl1(2)], [Z(k) Z(k)+1 Z(k)+1], [0 0 0], 'g')
patch([Y(k) Y(k)-cyl1(2) Y(k)+cyl1(2)], [Z(k) Z(k)-1 Z(k)-1], [0 0 0], 'g')

% Plotting the Bottom Beams
plot([M01(2) M11(2)], [M01(3) M11(3)], 'k', 'linewidth', Tlw)
plot([M02(2) M12(2)], [M02(3) M12(3)], 'k', 'linewidth', Tlw)
plot([M03(2) M13(2)], [M03(3) M13(3)], 'k', 'linewidth', Tlw)
plot([M04(2) M14(2)], [M04(3) M14(3)], 'k', 'linewidth', Tlw)

% Plotting the Runners
plot([R01(2) R11(2)], [R01(3) R11(3)], 'k', 'linewidth', Tlw)
plot([R02(2) R12(2)], [R02(3) R12(3)], 'k', 'linewidth', Tlw)
plot([R03(2) R13(2)], [R03(3) R13(3)], 'k', 'linewidth', Tlw)
plot([R04(2) R14(2)], [R04(3) R14(3)], 'k', 'linewidth', Tlw)

% Plotting the Side Supports [a]
plot([S01a(2) S11a(2)], [S01a(3) S11a(3)], 'k', 'linewidth', Tlw)
plot([S02a(2) S12a(2)], [S02a(3) S12a(3)], 'k', 'linewidth', Tlw)
plot([S03a(2) S13a(2)], [S03a(3) S13a(3)], 'k', 'linewidth', Tlw)
plot([S04a(2) S14a(2)], [S04a(3) S14a(3)], 'k', 'linewidth', Tlw)

% Plotting the Side Supports [b]
plot([S01b(2) S11b(2)], [S01b(3) S11b(3)], 'k', 'linewidth', Tlw)
plot([S02b(2) S12b(2)], [S02b(3) S12b(3)], 'k', 'linewidth', Tlw)
plot([S03b(2) S13b(2)], [S03b(3) S13b(3)], 'k', 'linewidth', Tlw)
plot([S04b(2) S14b(2)], [S04b(3) S14b(3)], 'k', 'linewidth', Tlw)

% Plotting the Back Support
plot([A01(2) A11(2)], [A01(3) A11(3)], 'k', 'linewidth', Tlw)
plot([A02(2) A12(2)], [A02(3) A12(3)], 'k', 'linewidth', Tlw)
plot([A03(2) A13(2)], [A03(3) A13(3)], 'k', 'linewidth', Tlw)
plot([A04(2) A14(2)], [A04(3) A14(3)], 'k', 'linewidth', Tlw)
% Plotting the Cables
plot ([B11(2) Y(k)], [B11(3) Z(k)], 'r', 'linewidth', Clw)
plot ([B12(2) Y(k)], [B12(3) Z(k)], 'r', 'linewidth', Clw)
plot ([B13(2) Y(k)], [B13(3) Z(k)], 'r', 'linewidth', Clw)
plot ([B14(2) Y(k)], [B14(3) Z(k)], 'r', 'linewidth', Clw)

title('Y-Z Plane, Right View')
xlabel('Length [ft]')
ylabel('Height [ft]')

grid on;
grid minor;
p1 = gca;
set(gca,'Xcolor',[0 0 0]); % Sets axis line color, not grid lines
set(gca,'Ycolor',[0 0 0]); % Sets axis line color, not grid lines
set(gca,'Zcolor',[0 0 0]); % Sets axis line color, not grid lines
whitebg([1 .94 .81]); % Changes the color of the background
axis equal;
hold off;
pause(1/64);

%----------------------------------------------------------------
% OBJECTIVE4_DRIVER.m
% Simulates straight line trajectory of the 4th Objective
% Programmer: Collier Fais
% Ohio University, Mechanical Engineering
% All Rights Reserved, April 2016
%----------------------------------------------------------------

% User defined inputs
t0 = 0; % This shouldn't change for calculation simplicity
tf = 10; % Total time [s] for trajectory execution, change this to what ever you want
posf = (8^2 + 8^2)^(1/2); % Total dist. [ft] of trajectory, unit corner to catty corner unit corner

% ensure actuator limits are not breached

PR = (1/5); % ratio of total time spent in the acceleration phase, Phase 1

% Time stamp for end of Phase 1 / beginning of Phase 2 [s]
t1 = PR*tf;

% Time stamp for end of Phase 2 / beginning of Phase 3 [s]
t2 = tf-t1;

% Time increment [s]
t = [t0:dt:tf]; % Time array

% Solving for cruising velocity, see notes for derivation [DO NOT CHANGE]
vel1 = posf/(tf*(1-PR));

% Distance at which acceleration phase ends, cruising begins [ft]
pos1 = (1/2)*t1*vel1;

% Distance traveled at constant velocity [ft]
CoastD = posf - 2*pos1;

% Distance at which cruising ends, negative acceleration begins [ft]
pos2 = posf - pos1;

% Time interval of phase 2 (constant velocity phase) [seconds]
t_coast = CoastD/vel1;

L = 10;
W = 10;

TRAJECTORY_GENERATOR % Script call
FRAMEPOSE % Script call

for k = 1:length(t)
% lower case 'x' refers to position vector data
% upper case 'X' refers to mode-specific end-effector catesian...
% x-motion
X(k) = x(k)*cos(pi/4) + 1;
Y(k) = x(k)*sin(pi/4) + 1;
Z(k) = 1;

% Calculate active parameters
ACTIVE_PARAM

if view3D == 1
    MOTIONPLOT3D
end

if view4 == 1
    MOTIONPLOT4
end
end

%----------------------------------------------------------
% PLANAR.m
% Root menu for 2-axis configuration
% User defined inputs
\[ t_0 = 0; \quad \text{This shouldn't change for calculation simplicity} \]
\[ t_f = 240; \quad \text{Total time [s] for trajectory execution, change this} \]
\[ \text{to what ever you want} \]
\[ p_{osf} = 10; \quad \text{Total dist. [ft] of trajectory, change to whatever,} \]
\[ \text{or calculate by supplying two cartesian points and taking the vector} \]
\[ \text{length norm between them} \]

% Assumed nominal parameters, would require optimization in future to
% ensure actuator limits are not breached
\[ PR = (1/10); \quad \text{Fraction of total time spent in the acceleration} \]
\[ \text{phase, Phase1} \]
\[ t_1 = PR \times t_f; \quad \text{Time stamp for end of Phase 1 / beginning of Phase 2} \]
\[ \text{[s]} \]
\[ t_2 = t_f - t_1; \quad \text{Time stamp for end of Phase 2 / beginning of Phase} \]
\[ 3 \text{ [s]} \]
\[ dt = .1; \quad \text{Time increment [s]} \]
\[ t = [t_0:dt:t_f]; \quad \text{Time array} \]

% Solving for cruising velocity, see notes for derivation
\[ v_{ell} = p_{osf} / (t_f \times (1-PR)); \]

% Distance at which acceleration phase ends, cruising begins [ft]
\[ p_{osl} = (1/2) \times t_1 \times v_{ell}; \]

% Distance traveled at constant velocity [ft]
\[ C_{oast} = p_{osf} - 2 \times p_{osl}; \]

% Distance at which cruising ends, negative acceleration begins [ft]
\[ p_{os2} = p_{osf} - p_{osl}; \]

% Time interval of phase 2 (constant velocity phase) [seconds]
\[ t_{coast} = C_{oast} / v_{ell}; \]

% Create trajectory
TRAJECTORY_GENERATOR

% Fill workspace with vector poses of the 2-axis configuration
PLANARPOSE

\textbf{for} k = 1:length(t)
\% lower case 'x' refers to position vector data
\% upper case 'X' refers to mode-specific end-effector catesian...
\% x-motion
X(k) = x(k) + 2;
Y(k) = 19/12; % Fixed height of end effector is roughly 19"

% Calculate active parameters
ACTIVE_PARAM_PLANAR
MOTIONPLOT_PLANAR

%------------------------------------------------------------------
% PLANARPOSE.m
% Vector pose of towers and workspace
% Programmer: Collier Fais
% Ohio University, Mechanical Engineering
% All Rights Reserved, November 2016
%------------------------------------------------------------------

Ht = 68/12; % Height of PULLEY, 68 inches from the ground, converted to feet
D  = 14; % Horizontal distance between PULLEYS, [ft]

B01 = [0 0];
B11 = [0 Ht];
B02 = [D 0];
B12 = [D Ht];

GRND1 = [-3 0];
GRND2 = [D+3 0];

%----------------------------------------------------------
% SNAP.m
% Snap-Shot Mode main script
% Programmer: Collier Fais
% Ohio University, Mechanical Engineering
% All Rights Reserved, February 2016
%----------------------------------------------------------

disp('SNAPSHOT MODE')

SnapError = 1;

while SnapError == 1
    % Create array of 'default values'
    % SNAP_DEFAULT = [W L Ht P(1) P(2) P(3) m*g];
    P1 = num2str(W/2);
    P2 = num2str(L/2);
    P3 = num2str(Ht/2);
    W = num2str(W);
    L = num2str(L);
    Ht = num2str(Ht);
    Wt = num2str(Wt);
% Create table of default system parameters, user can change if desired
nameSNAP_PARAM = 'Snapshot Parameters';
valuesSNAP_PARAM = {'Workspace Width [ft]','Workspace Length [ft]','Tower Height [ft]','x-coordinate [ft]','y-coordinate [ft]','z-coordinate [ft]','End-effector Weight [lbf]'};
defaultSNAP_PARAM = {W,L,Ht,P1,P2,P3,Wt};
SNAP_PARAM = inputdlg(valuesSNAP_PARAM,nameSNAP_PARAM,[1 50],defaultSNAP_PARAM);
waitfor(SNAP_PARAM);

% Re-define variables in case they were changed
W = str2num(SNAP_PARAM{1});
L = str2num(SNAP_PARAM{2});
Ht = str2num(SNAP_PARAM{3});
P1 = str2num(SNAP_PARAM{4});
P2 = str2num(SNAP_PARAM{5});
P3 = str2num(SNAP_PARAM{6});
m = str2num(SNAP_PARAM{7})*PtS;
P = [P1 P2 P3];

% Check for errors
if P(1) <= 0
    ERR = errordlg('x-coordinate must satisfy 0 < x < Width');
    waitfor(ERR);
    SnapError = 1; 
    continue;
else
    SnapError = 0;
end
if P(1) >= W
    ERR = errordlg('x-coordinate must satisfy 0 < x < Width');
    waitfor(ERR);
    SnapError = 1;
    continue;
else
    SnapError = 0;
end
if P(2) <= 0
    ERR = errordlg('y-coordinate must satisfy 0 < y < Length');
    waitfor(ERR);
    SnapError = 1;
    continue;
else
    SnapError = 0;
end
if P(2) >= L
    ERR = errordlg('y-coordinate must satisfy 0 < y < Length');
    waitfor(ERR);
    SnapError = 1;
else
    SnapError = 0;
end
if P(3) <= 0
    ERR = errordlg('z-coordinate must satisfy 0 < z < Height');
    waitfor(ERR);
    SnapError = 1;
else
    SnapError = 0;
end
if P(3) >= Ht
    ERR = errordlg('z-coordinate must satisfy 0 < z < Height');
    waitfor(ERR);
    SnapError = 1;
else
    SnapError = 0;
end
continue;
else
  SnapError = 0;
end

if P(3) <= 0
  ERR = errordlg('z-coordinate must satisfy 0 < z < Tower Height');
  waitfor(ERR);
  SnapError = 1;
  continue;
else
  SnapError = 0;
end
if P(3) >= Ht
  ERR = errordlg('z-coordinate must satisfy 0 < z < Tower Height');
  waitfor(ERR);
  SnapError = 1;
  continue;
else
  SnapError = 0;
end

% Populate workspace with Pose variables
SNAPPOSE

% Perform calculation pertaining to snap-mode only
SNAPCALC

% Plot configuration to screen
SNAPPLOT

% % Generate table of corresponding data
% SNAPTABLE1

if EXIT2 == 0
  disp('PROGRAM EXIT');
  close all;
end

%----------------------------------------------------------------
% SNAPCALC.m
% Performs calculations only for Snap-Shot Mode
% Programmer: Collier Fais
% Ohio University, Mechanical Engineering
% All Rights Reserved, March 2016
%----------------------------------------------------------------
% Cable vectors
C1 = B11 - P;
C2 = B12 - P;
C3 = B13 - P;
C4 = B14 - P;

% Magnitude of cable lengths
L1 = norm(C1);
L2 = norm(C2);
L3 = norm(C3);
L4 = norm(C4);

% Matrix of active cable lengths
CL = [L1 L2 L3 L4]';

% Static Jacobian matrix 'A'
% Not to be confused with Back Support Pose info 'Aij'
A = [C1(1)/L1 C2(1)/L2 C3(1)/L3 C4(1)/L4;
    C1(2)/L1 C2(2)/L2 C3(2)/L3 C4(2)/L4;
    C1(3)/L1 C2(3)/L2 C3(3)/L3 C4(3)/L4];

% Transpose of Jacobian
A_trans = A';

% J_inv = inv(J); % Cannot compute b/c matrix must be square

% PseudoInverse of Jacobian
A_pseudoInv = A_trans*(A*A_trans)^-1;

% Matrix of external forces exerted on end-effector
F = [fx fy fz]';

% Cable tension calculation
T = -1*A_pseudoInv*(F+m*g);

% Inverse Velocity Jacobian Matrix
J = -1*A_trans;

% % Singularities
% sing1 = min(abs(A*A_trans));
% if sing1 == 0
%    msg = 'Singularity';
%    error(msg)
% end

det_A = det(A*A_trans);

% Robot Translational Stiffness
E = 12000000;  % Estimated cable modulus of elasticity [lbs per sq inch]
D_cab = 0.16;  % Cable diameter [in]
A_cab = (pi/4)*D_cab^2;  % Cable cross-sectional area [in^2]
ki = (E*A_cab)*[L1^-1 L2^-1 L3^-1 L4^-1]; % Individual-cable spring constant terms
k = diag(ki); % Diagonal matrix of k-values [lb/ft]
K = A*k*A_trans; % 3x3 translational cartesian stiffness matrix
K_eig = eig(K); % Eigen values represent the principal stiffness components
K_tot = norm([K_eig]); % Euclidean norm yields a single stiffness number

% Internal cable stress [ksi]
CS = [T(1)/A_cab T(2)/A_cab T(3)/A_cab T(4)/A_cab]/1000;

% Motor holding torque [lbf/in]
Wr = 2; % Winch radius [in]

%----------------------------------------------------------------
%   SNAPPLOT.m
%   Plots the snap-shot
%   Programmer: Collier Fais
%   Ohio University, Mechanical Engineering
%   All Rights Reserved, March 2016
%----------------------------------------------------------------

nameSNAP_PLOT = 'Font, Top, Right and 3D View Snapshot';

% Figure 1, 3D plot
FourSnapView = figure('position', [10 10 1520 760],'name', nameSNAP_PLOT, 'NumberTitle','off');
subplot(222)

% Plotting the Booms
plot3([B01(1) B11(1)], [B01(2) B11(2)], [B01(3) B11(3)], 'k', 'linewidth', Tlw)
hold on
plot3([B02(1) B12(1)], [B02(2) B12(2)], [B02(3) B12(3)], 'k', 'linewidth', Tlw)
plot3([B03(1) B13(1)], [B03(2) B13(2)], [B03(3) B13(3)], 'k', 'linewidth', Tlw)
plot3([B04(1) B14(1)], [B04(2) B14(2)], [B04(3) B14(3)], 'k', 'linewidth', Tlw)

%Edge of plot-space patch
patch([-Dx 0-Dx W+Dx W+Dx],[-Dy L+Dy L+Dy 0-Dy],[0.467 .392 .314])

%Workspace footprint patch
patch([0 0 W W],[0 L L 0],[.73 1 .96])

% Plotting the Bottom Beams
plot3([M01(1) M11(1)], [M01(2) M11(2)], [M01(3) M11(3)], 'k', 'linewidth', Tlw)
plot3([M02(1) M12(1)], [M02(2) M12(2)], [M02(3) M12(3)], 'k', 'linewidth', Tlw)
plot3([M03(1) M13(1)], [M03(2) M13(2)], [M03(3) M13(3)], 'k', 'linewidth', Tlw)
plot3([M04(1) M14(1)], [M04(2) M14(2)], [M04(3) M14(3)], 'k', 'linewidth', Tlw)

% Plotting the Runners
plot3([R01(1) R11(1)], [R01(2) R11(2)], [R01(3) R11(3)], 'k', 'linewidth', Tlw)
plot3([R02(1) R12(1)], [R02(2) R12(2)], [R02(3) R12(3)], 'k', 'linewidth', Tlw)
plot3([R03(1) R13(1)], [R03(2) R13(2)], [R03(3) R13(3)], 'k', 'linewidth', Tlw)
plot3([R04(1) R14(1)], [R04(2) R14(2)], [R04(3) R14(3)], 'k', 'linewidth', Tlw)

% Plotting the Side Supports [a]
plot3([S01a(1) S11a(1)], [S01a(2) S11a(2)], [S01a(3) S11a(3)], 'k', 'linewidth', Tlw)
plot3([S02a(1) S12a(1)], [S02a(2) S12a(2)], [S02a(3) S12a(3)], 'k', 'linewidth', Tlw)
plot3([S03a(1) S13a(1)], [S03a(2) S13a(2)], [S03a(3) S13a(3)], 'k', 'linewidth', Tlw)
plot3([S04a(1) S14a(1)], [S04a(2) S14a(2)], [S04a(3) S14a(3)], 'k', 'linewidth', Tlw)

% Plotting the Side Supports [b]
plot3([S01b(1) S11b(1)], [S01b(2) S11b(2)], [S01b(3) S11b(3)], 'k', 'linewidth', Tlw)
plot3([S02b(1) S12b(1)], [S02b(2) S12b(2)], [S02b(3) S12b(3)], 'k', 'linewidth', Tlw)
plot3([S03b(1) S13b(1)], [S03b(2) S13b(2)], [S03b(3) S13b(3)], 'k', 'linewidth', Tlw)
plot3([S04b(1) S14b(1)], [S04b(2) S14b(2)], [S04b(3) S14b(3)], 'k', 'linewidth', Tlw)

% Plotting the Back Support
plot3([A01(1) A11(1)], [A01(2) A11(2)], [A01(3) A11(3)], 'k', 'linewidth', Tlw)
plot3([A02(1) A12(1)], [A02(2) A12(2)], [A02(3) A12(3)], 'k', 'linewidth', Tlw)
plot3([A03(1) A13(1)], [A03(2) A13(2)], [A03(3) A13(3)], 'k', 'linewidth', Tlw)
plot3([A04(1) A14(1)], [A04(2) A14(2)], [A04(3) A14(3)], 'k', 'linewidth', Tlw)

% Plotting the Cables
plot3([B11(1) P(1)], [B11(2) P(2)], [B11(3) P(3)], 'r', 'linewidth', Clw)
plot3([B12(1) P(1)], [B12(2) P(2)], [B12(3) P(3)], 'r', 'linewidth', Clw)
plot3 ([B13(1) P(1)], [B13(2) P(2)], [B13(3) P(3)], 'r', 'linewidth', Clw)
plot3 ([B14(1) P(1)], [B14(2) P(2)], [B14(3) P(3)], 'r', 'linewidth', Clw)

% Plotting the Cone end effector

t1 = [0 .35];
t2 = [.35 0];
[x1,y1,z1] = cylinder(t1);
[x2,y2,z2] = cylinder(t2);
surf(x1+P(1),y1+P(2),z1+P(3))
surf(x2+P(1),y2+P(2),z2+P(3)-1)

title('3D Plot')
xlabel('Width [ft]')
ylabel('Length [ft]')
zlabel('Height [ft]')
grid on;
grid minor;
p1 = gca;
set(gca,'Xcolor',[0 0 0]);  %Sets axis line color, not grid lines
set(gca,'Ycolor',[0 0 0]);  %Sets axis line color, not grid lines
set(gca,'Zcolor',[0 0 0]);  %Sets axis line color, not grid lines
% set(gca,'zlim',[0 10]);
whitebg([1 .94 .81]);       % Changes the color of the background
axis equal;

% Figure 2 x-y plane, top view
% figure(2);
% subplot(221)
% Plotting the Booms
plot([B01(1) B11(1)], [B01(2) B11(2)], 'k', 'linewidth', Tlw)
hold on
plot([B02(1) B12(1)], [B02(2) B12(2)], 'k', 'linewidth', Tlw)
plot([B03(1) B13(1)], [B03(2) B13(2)], 'k', 'linewidth', Tlw)
plot([B04(1) B14(1)], [B04(2) B14(2)], 'k', 'linewidth', Tlw)

%Edge of plot-space patch
patch([0-Dx 0-Dx W+Dx W+Dx],[0-Dy L+Dy L+Dy 0-Dy],[0.1 -0.1 -0.1 -0.1], [0.467 .392 .314])

%Workspace footprint patch
patch([0 0 W W],[0 L L 0],[.73 1 .96])

% Plotting the Bottom Beams
plot([M01(1) M11(1)], [M01(2) M11(2)], 'k', 'linewidth', Tlw)
plot([M02(1) M12(1)], [M02(2) M12(2)], 'k', 'linewidth', Tlw)
plot([M03(1) M13(1)], [M03(2) M13(2)], 'k', 'linewidth', Tlw)
plot([M04(1) M14(1)], [M04(2) M14(2)], 'k', 'linewidth', Tlw)

% Plotting the Runners
plot([R01(1) R11(1)], [R01(2) R11(2)], 'k', 'linewidth', Tlw)
plot([R02(1) R12(1)],[R02(2) R12(2)],'k','linewidth',Tlw)
plot([R03(1) R13(1)],[R03(2) R13(2)],'k','linewidth',Tlw)
plot([R04(1) R14(1)],[R04(2) R14(2)],'k','linewidth',Tlw)

% Plotting the Side Supports [a]
plot([S01a(1) S11a(1)],[S01a(2) S11a(2)],'k','linewidth',Tlw)
plot([S02a(1) S12a(1)],[S02a(2) S12a(2)],'k','linewidth',Tlw)
plot([S03a(1) S13a(1)],[S03a(2) S13a(2)],'k','linewidth',Tlw)
plot([S04a(1) S14a(1)],[S04a(2) S14a(2)],'k','linewidth',Tlw)

% Plotting the Side Supports [b]
plot([S01b(1) S11b(1)],[S01b(2) S11b(2)],'k','linewidth',Tlw)
plot([S02b(1) S12b(1)],[S02b(2) S12b(2)],'k','linewidth',Tlw)
plot([S03b(1) S13b(1)],[S03b(2) S13b(2)],'k','linewidth',Tlw)
plot([S04b(1) S14b(1)],[S04b(2) S14b(2)],'k','linewidth',Tlw)

% Plotting the Back Support
plot([A01(1) A11(1)],[A01(2) A11(2)],'k','linewidth',Tlw)
plot([A02(1) A12(1)],[A02(2) A12(2)],'k','linewidth',Tlw)
plot([A03(1) A13(1)],[A03(2) A13(2)],'k','linewidth',Tlw)
plot([A04(1) A14(1)],[A04(2) A14(2)],'k','linewidth',Tlw)

% Plotting the Cables
plot([B11(1) P(1)],[B11(2) P(2)],'r','linewidth',Clw)
plot([B12(1) P(1)],[B12(2) P(2)],'r','linewidth',Clw)
plot([B13(1) P(1)],[B13(2) P(2)],'r','linewidth',Clw)
plot([B14(1) P(1)],[B14(2) P(2)],'r','linewidth',Clw)

Circle(P(1), P(2), 0.5);

title('X-Y Plane, Top View')
xlabel('Width [ft]')
ylabel('Length [ft]')

grid on;
grid minor;
pl = gca;
set(gca,'Xcolor',[0 0 0]);  %Sets axis line color, not grid lines
set(gca,'Ycolor',[0 0 0]);  %Sets axis line color, not grid lines
set(gca,'Zcolor',[0 0 0]);  %Sets axis line color, not grid lines
whitebg([1 .94 .81]);       % Changes the color of the background
axis equal;

% Figure 3, x-z plane, front view
% figure(3);
subplot(223)
% Plotting the Booms
plot([B01(1) B11(1)],[B01(3) B11(3)],'k','linewidth',Tlw)
hold on
plot([B02(1) B12(1)],[B02(3) B12(3)],'k','linewidth',Tlw)
plot([B03(1) B13(1)],[B03(3) B13(3)],'k','linewidth',Tlw)
plot([B04(1) B14(1)],[B04(3) B14(3)],'k','linewidth',Tlw)
% Plot edge of plot-space with a line instead of a patch
plot([0 -Dx W+Dx],[-0.1 -0.1], 'g', 'linewidth', 1)
% Plot workspace with a line instead of patch
plot([0 W], [0 0], 'c', 'linewidth', 2)

% Plotting the Bottom Beams
plot([M01(1) M11(1)], [M01(3) M11(3)], 'k', 'linewidth', Tlw)
plot([M02(1) M12(1)], [M02(3) M12(3)], 'k', 'linewidth', Tlw)
plot([M03(1) M13(1)], [M03(3) M13(3)], 'k', 'linewidth', Tlw)
plot([M04(1) M14(1)], [M04(3) M14(3)], 'k', 'linewidth', Tlw)

% Plotting the Runners
plot([R01(1) R11(1)], [R01(3) R11(3)], 'k', 'linewidth', Tlw)
plot([R02(1) R12(1)], [R02(3) R12(3)], 'k', 'linewidth', Tlw)
plot([R03(1) R13(1)], [R03(3) R13(3)], 'k', 'linewidth', Tlw)
plot([R04(1) R14(1)], [R04(3) R14(3)], 'k', 'linewidth', Tlw)

% Plotting the Side Supports [a]
plot([S01a(1) S11a(1)], [S01a(3) S11a(3)], 'k', 'linewidth', Tlw)
plot([S02a(1) S12a(1)], [S02a(3) S12a(3)], 'k', 'linewidth', Tlw)
plot([S03a(1) S13a(1)], [S03a(3) S13a(3)], 'k', 'linewidth', Tlw)
plot([S04a(1) S14a(1)], [S04a(3) S14a(3)], 'k', 'linewidth', Tlw)

% Plotting the Side Supports [b]
plot([S01b(1) S11b(1)], [S01b(3) S11b(3)], 'k', 'linewidth', Tlw)
plot([S02b(1) S12b(1)], [S02b(3) S12b(3)], 'k', 'linewidth', Tlw)
plot([S03b(1) S13b(1)], [S03b(3) S13b(3)], 'k', 'linewidth', Tlw)
plot([S04b(1) S14b(1)], [S04b(3) S14b(3)], 'k', 'linewidth', Tlw)

% Plotting the Back Support
plot([A01(1) A11(1)], [A01(3) A11(3)], 'k', 'linewidth', Tlw)
plot([A02(1) A12(1)], [A02(3) A12(3)], 'k', 'linewidth', Tlw)
plot([A03(1) A13(1)], [A03(3) A13(3)], 'k', 'linewidth', Tlw)
plot([A04(1) A14(1)], [A04(3) A14(3)], 'k', 'linewidth', Tlw)

% Plotting the Cables
plot ([B11(1) P(1)], [B11(3) P(3)], 'r', 'linewidth', Clw)
plot ([B12(1) P(1)], [B12(3) P(3)], 'r', 'linewidth', Clw)
plot ([B13(1) P(1)], [B13(3) P(3)], 'r', 'linewidth', Clw)
plot ([B14(1) P(1)], [B14(3) P(3)], 'r', 'linewidth', Clw)

% Patch triangles that look like the cones for the iso view
patch([P(1) P(1)-t1(2) P(1)+t1(2)], [P(3) P(3)+1 P(3)+1], [0 0 0], 'g')
patch([P(1) P(1)-t1(2) P(1)+t1(2)], [P(3) P(3)-1 P(3)-1], [0 0 0], 'g')

title('X-Z Plane, Front View')
xlabel('Width [ft]')
ylabel('Height [ft]')
axis equal;
grid on;
grid minor;
% Figure 4, y-z plane, Right view
subplot(224)
% Plotting the Booms
plot([B01(2) B11(2)], [B01(3) B11(3)], 'k', 'linewidth', Tlw)
hold on
plot([B02(2) B12(2)], [B02(3) B12(3)], 'k', 'linewidth', Tlw)
plot([B03(2) B13(2)], [B03(3) B13(3)], 'k', 'linewidth', Tlw)
plot([B04(2) B14(2)], [B04(3) B14(3)], 'k', 'linewidth', Tlw)

% Plot edge of plot-space with a line instead of a patch
plot([0-Dy L+Dy], [-0.1 -0.1], 'g', 'linewidth', 1)
% Plot workspace with a line instead of patch
plot([0 L], [0 0], 'c', 'linewidth', 2)

% Patch triangles that look like the cones for the iso view
patch([P(2) P(2)-t1(2) P(2)+t1(2)], [P(3) P(3)+1 P(3)+1], [0 0 0], 'g')
patch([P(2) P(2)-t1(2) P(2)+t1(2)], [P(3) P(3)-1 P(3)-1], [0 0 0], 'g')

% Plotting the Bottom Beams
plot([M01(2) M11(2)], [M01(3) M11(3)], 'k', 'linewidth', Tlw)
plot([M02(2) M12(2)], [M02(3) M12(3)], 'k', 'linewidth', Tlw)
plot([M03(2) M13(2)], [M03(3) M13(3)], 'k', 'linewidth', Tlw)
plot([M04(2) M14(2)], [M04(3) M14(3)], 'k', 'linewidth', Tlw)

% Plotting the Runners
plot([R01(2) R11(2)], [R01(3) R11(3)], 'k', 'linewidth', Tlw)
plot([R02(2) R12(2)], [R02(3) R12(3)], 'k', 'linewidth', Tlw)
plot([R03(2) R13(2)], [R03(3) R13(3)], 'k', 'linewidth', Tlw)
plot([R04(2) R14(2)], [R04(3) R14(3)], 'k', 'linewidth', Tlw)

% Plotting the Side Supports [a]
plot([S01a(2) S11a(2)], [S01a(3) S11a(3)], 'k', 'linewidth', Tlw)
plot([S02a(2) S12a(2)], [S02a(3) S12a(3)], 'k', 'linewidth', Tlw)
plot([S03a(2) S13a(2)], [S03a(3) S13a(3)], 'k', 'linewidth', Tlw)
plot([S04a(2) S14a(2)], [S04a(3) S14a(3)], 'k', 'linewidth', Tlw)

% Plotting the Side Supports [b]
plot([S01b(2) S11b(2)], [S01b(3) S11b(3)], 'k', 'linewidth', Tlw)
plot([S02b(2) S12b(2)], [S02b(3) S12b(3)], 'k', 'linewidth', Tlw)
plot([S03b(2) S13b(2)], [S03b(3) S13b(3)], 'k', 'linewidth', Tlw)
plot([S04b(2) S14b(2)], [S04b(3) S14b(3)], 'k', 'linewidth', Tlw)

% Plotting the Back Support
plot([A01(2) A11(2)], [A01(3) A11(3)], 'k', 'linewidth', Tlw)
plot([A02(2) A12(2)], [A02(3) A12(3)], 'k', 'linewidth', Tlw)
plot([A03(2) A13(2)], [A03(3) A13(3)], 'k', 'linewidth', Tlw)
plot([A04(2) A14(2)], [A04(3) A14(3)], 'k', 'linewidth', Tlw)

% Plotting the Cables
plot([B11(2) P(2)], [B11(3) P(3)], 'r', 'linewidth', Clw)
plot([B12(2) P(2)], [B12(3) P(3)], 'r', 'linewidth', Clw)
plot ([B13(2) P(2)], [B13(3) P(3)], 'r', 'linewidth', Clw)
plot ([B14(2) P(2)], [B14(3) P(3)], 'r', 'linewidth', Clw)

title('Y-Z Plane, Right View')
xlabel('Length [ft]')
ylabel('Height [ft]')

grid on;
grid minor;
p1 = gca;
set(gca,'Xcolor',[0 0 0]);  %Sets axis line color, not grid lines
set(gca,'Ycolor',[0 0 0]);  %Sets axis line color, not grid lines
set(gca,'Zcolor',[0 0 0]);  %Sets axis line color, not grid lines
whitebg([1 .94 .81]);       % Changes the color of the background
axis equal;

% Add navigation buttons to bottom of plot area
% Button 2a is Return, sends user back to snap-shot mode
btn2a = uicontrol('Style','pushbutton','string','Reset',...
    'position',[10 10 150 40],...
    'fontsize',20,...
    'callback','choice2 = 1; close all; SNAP');

btn2b = uicontrol('Style','pushbutton','string','Show Data Table',...
    'position',[525 10 250 40],...
    'fontsize',20,...
    'callback','choice2 = 2; SNAPTABLE1');

btn2c = uicontrol('Style','pushbutton','string','Main Menu',...
    'position',[800 10 150 40],...
    'fontsize',20,...
    'callback','choice2 = 3; close all; CSR_SIMULATOR');

btn2d = uicontrol('Style','pushbutton','string','Exit',...
    'position',[1400 10 100 40],...
    'fontsize',20,...
    'callback','EXIT2 = 1; close all;')

waitfor(FourSnapView);

if EXIT2 == 1
    disp('PROGRAM EXIT');
    EXIT2 = EXIT2 + 1;
end

%% 3D view only
nameSNAP_PLOT3D = '3D View Snapshot';

View3D = figure('position',[10 10 1520 760],'
    'name',nameSNAP_PLOT3D,'NumberTitle','off');
% Plotting the Booms
% Plotting the Back Support
plot3([A01(1) A11(1)], [A01(2) A11(2)], [A01(3) A11(3)], 'k', 'linewidth', Tlw)
plot3([A02(1) A12(1)], [A02(2) A12(2)], [A02(3) A12(3)], 'k', 'linewidth', Tlw)
plot3([A03(1) A13(1)], [A03(2) A13(2)], [A03(3) A13(3)], 'k', 'linewidth', Tlw)
plot3([A04(1) A14(1)], [A04(2) A14(2)], [A04(3) A14(3)], 'k', 'linewidth', Tlw)

% Plotting the Cables
plot3([B11(1) P(1)], [B11(2) P(2)], [B11(3) P(3)], 'r', 'linewidth', Clw)
plot3([B12(1) P(1)], [B12(2) P(2)], [B12(3) P(3)], 'r', 'linewidth', Clw)
plot3([B13(1) P(1)], [B13(2) P(2)], [B13(3) P(3)], 'r', 'linewidth', Clw)
plot3([B14(1) P(1)], [B14(2) P(2)], [B14(3) P(3)], 'r', 'linewidth', Clw)

% Plotting base reference frame coord system
plot3([0 0], [0 0], [0 3], 'm', 'linewidth', 5)
plot3([0 0], [0 3], [0 0], 'm', 'linewidth', 5)
plot3([0 3], [0 0], [0 0], 'm', 'linewidth', 5)

% Plotting end-effector reference frame coord system
plot3([P1 P1], [P2 P2], [P3 P3+3], 'g', 'linewidth', 5)
plot3([P1 P1], [P2 P2+3], [P3 P3], 'g', 'linewidth', 5)
plot3([P1 P1+3], [P2 P2], [P3 P3], 'g', 'linewidth', 5)

% Plotting downward gravity vector for static FDB
patch([P1-0.5 P1-0.5 P1+0.5 P1+0.5], [P2-0.5 P2+0.5 P2+0.5 P2-0.5], [P3 P3 P3 P3], [1 1 1])
patch([P1+0.5 P1+0.5 P1-0.5 P1-0.5], [P2+0.3 P2-0.5 P2-0.5 P2+0.5], [P3 P3 P3 P3], [1 1 1])
patch([P1-0.5 P1+0.5 P1+0.5 P1+0.5], [P2-0.5 P2+0.5 P2+0.5 P2-0.5], [P3 P3 P3 P3], [1 1 1])
patch([P1-0.5 P1-0.5 P1+0.5 P1+0.5], [P2-0.5 P2+0.3 P2+0.3 P2-0.5], [P3 P3 P3 P3], [1 1 1])

title('3D Plot')
xlabel('Width [ft]')
ylabel('Length [ft]')
zlabel('Height [ft]')
grid on;
grid minor;
pl = gca;
set(gca,'Xcolor',[0 0 0]); %Sets axis line color, not grid lines
set(gca,'Ycolor',[0 0 0]); %Sets axis line color, not grid lines
set(gca,'Zcolor',[0 0 0]); %Sets axis line color, not grid lines
% set(gca,'zlim',[0 10]);
whitebg([1 .94 .81]); % Changes the color of the background
axis equal;

%------------------------------------------------------------------------
% SNAPPOSE.m
% Vector pose of towers and workspace
% Programmer: Collier Fais
% Ohio University, Mechanical Engineering
% All Rights Reserved, March 2016
%------------------------------------------------------------------------

% Static pose of End Effector, first attempt
% This will change when moving...
% P = [7.195 5.9]; % Located at dead center of workspace, default, can be changed

% Pose of Tower Boom
% Syntax ~ Bij where:
% i indicates the X,Y,Z cartesian START and END of the Boom
% and
% j indicates the ID
B01 = [0-dx 0-dy 0]; % Location of base of tower 1
B11 = [0-dx 0-dy Ht]; % Location of top of tower 1
B02 = [0-dx L+dy 0];
B12 = [0-dx L+dy Ht];
B03 = [W+dx L+dy 0];
B13 = [W+dx L+dy Ht];
B04 = [W+dx 0-dy 0];
B14 = [W+dx 0-dy Ht];

% Pose of Bottom Beam, Mij
% i = left, right from a back oriented view, respectively
% j = Tower ID
M01 = [(Lm/2)*cos(135*DR)-dx   (Lm/2)*sin(135*DR)-dy 0];
M11 = [(Lm/2)*cos(315*DR)-dx   (Lm/2)*sin(315*DR)-dy 0];
M02 = [(Lm/2)*cos(45*DR)-dx    ((Lm/2)*sin(45*DR))+L+dy 0];
M12 = [(Lm/2)*cos(225*DR)-dx   ((Lm/2)*sin(225*DR))+L+dy 0];
M03 = [((Lm/2)*cos(315*DR))+W+dx   ((Lm/2)*sin(315*DR))+L+dy 0];
M13 = [((Lm/2)*cos(135*DR))+W+dx   ((Lm/2)*sin(135*DR))+L+dy 0];
M04 = [((Lm/2)*cos(225*DR))+W+dx   (Lm/2)*sin(225*DR)-dy 0];
M14 = [((Lm/2)*cos(45*DR))+W+dx   (Lm/2)*sin(45*DR)-dy 0];

% Pose of Runner, Rij
% i = front, back from a back oriented view, respectively
% j = Tower ID
R01 = [0-dx 0-dy 0];
R11 = [(Lr)*cos(225*DR)-dx (Lr)*sin(225*DR)-dy 0];
R02 = [0-dx L+dy 0];
R12 = [(Lr)*cos(135*DR)-dx ((Lr)*sin(135*DR))+L+dy 0];
R03 = [W+dx L+dy 0];
R13 = [((Lr)*cos(45*DR))+W+dx ((Lr)*sin(45*DR))+L+dy 0];
R04 = [W+dx 0-dy 0];
R14 = [((Lr)*cos(315*DR))+W+dx (Lr)*sin(315*DR)-dy 0];

% Pose of Side Support a (right support, back view) Sija
% i = bottom, top
% j = Tower ID
% a = right Side Support, back view
S01a = [Ls*cos(60*DR)*cos(315*DR)-dx Ls*cos(60*DR)*sin(315*DR)-dy 0];
S11a = [0-dx 0-dy Ls*sin(60*DR)];
S02a = [Ls*cos(60*DR)*cos(225*DR)-dx Ls*cos(60*DR)*sin(225*DR)+L+dy 0];
S12a = [0-dx L+dy Ls*sin(60*DR)];
S03a = [Ls*cos(60*DR)*cos(135*DR)+W+dx Ls*cos(60*DR)*sin(135*DR)+L+dy 0];
S13a = [W+dx L+dy Ls*sin(60*DR)];
S04a = [Ls*cos(60*DR)*cos(45*DR)+W+dx Ls*cos(60*DR)*sin(45*DR)-dy 0];
S14a = [W+dx 0-dy Ls*sin(60*DR)];

% Pose of Side Support b (left support, back view) Sijb
% i = bottom, top
% j = Tower ID
% b = left Side Support, back view
S01b = [Ls*cos(60*DR)*cos(135*DR)-dx Ls*cos(60*DR)*sin(135*DR)-dy 0];
S11b = [0-dx 0-dy Ls*sin(60*DR)];
S02b = [Ls*cos(60*DR)*cos(45*DR)-dx Ls*cos(60*DR)*sin(45*DR)+L+dy 0];
S12b = [0-dx L+dy Ls*sin(60*DR)];
S03b = [Ls*cos(60*DR)*cos(315*DR)+W+dx Ls*cos(60*DR)*sin(315*DR)+L+dy 0];
S13b = [W+dx L+dy Ls*sin(60*DR)];
S04b = [Ls*cos(60*DR)*cos(225*DR)+W+dx Ls*cos(60*DR)*sin(225*DR)-dy 0];
S14b = [W+dx 0-dy Ls*sin(60*DR)];

% Pose of Back Support, Aij
% Not to be confused with Static Jacobian 'A'
% i = bottom, top from a back oriented view, respectively
% j = Tower ID
A01 = [Lb*cos(60*DR)*cos(225*DR)-dx   Lb*cos(60*DR)*sin(225*DR)-dy 0];
A11 = [0-dx                           0-dy
Lb*sin(60*DR)];
A02 = [Lb*cos(60*DR)*cos(135*DR)-dx   Lb*cos(60*DR)*sin(135*DR)+L+dy 0];
A12 = [0-dx                           L+dy
Lb*sin(60*DR)];
A03 = [Lb*cos(60*DR)*cos(45*DR)+W+dx  Lb*cos(60*DR)*sin(45*DR)+L+dy 0];
A13 = [W+dx                           L+dy
Lb*sin(60*DR)];
A04 = [Lb*cos(60*DR)*cos(315*DR)+W+dx Lb*cos(60*DR)*sin(315*DR)-dy 0];
A14 = [W+dx                           0-dy
Lb*sin(60*DR)];

%----------------------------------------------------------
%    SNAPTABLE1.m
%    Creates a figure with 2 different sub-plotted uitabels
%    Programmer: Collier Fais
%    Ohio University, Mechanical Engineering
%    All Rights Reserved, February 2016
%----------------------------------------------------------

% Used only for the snap-shot mode

nameTABLE1 = 'Snap-Shot Data Table';
f = figure('Position', [208 210 1212 209], 'name', nameTABLE1, 'NumberTitle', 'off');
for i = 0:1
    if i == 0
        cnames = {'End-Effector Weight [lbf]', 'Workspace Width [ft]', 'Workspace Length [ft]', 'x-coord. [ft]', 'y-coord. [ft]', 'z-coord. [ft]', 'Determinant of [A*A^T]', 'Cartesian Stiffness [lbf/ft]'};
        rnames = {'>'};
        d = [m/PtS W L P(1) P(2) P(3) det_A K_tot];
        width = 1192;
        height = 49;
    end
    if i == 1
        cnames = {'Cable Tension [lbf]', 'Cable Length [ft]', 'Cable Stress [ksi]'};
        rnames = {'Cable 1', 'Cable 2', 'Cable 3', 'Cable 4'};
        d = [T(1) CL(1) CS(1);T(2) CL(2) CS(2);T(3) CL(3) CS(3);T(4) CL(4) CS(4)];
        width = 514;
        height = 130;
    end
    t = uitable(f, 'Data', d, 'ColumnName', cnames, 'RowName', rnames, 'Position', [(i*194)+10 (i*59)+10 width height], 'ColumnWidth', [145], 'fontsize', 14);
% User defined inputs
t0 = 0; % This shouldn't change for calculation simplicity
tf = 20; % Total time [s] for trajectory execution, change this
to what ever you want
posf = 20; % Total dist. [ft] of trajectory, change to whatever, or
calculate by supplying two cartesian points and taking the vector
length norm between them

% Assumed nominal parameters, would require optimization in future to
% ensure actuator limits are not breached
PR = (1/5); % ratio of total time spent in the acceleration phase, Phasel

t1 = PR*tf; % Time stamp for end of Phase 1 / beginning of Phase 2
[s]
t2 = tf-t1; % Time stamp for end of Phase 2 / beginning of Phase 3 [s]
dt = PR/2; % Time increment [s]

t = [t0:dt:tf]; % Time array

% Solving for cruising velocity, see notes for derivation
vell = posf/(tf*(1-PR));

% Distance at which acceleration phase ends, cruising begins [ft]
pos1 = (1/2)*t1*vell;

% Distance traveled at constant velocity [ft]
CoastD = posf - 2*pos1;

% Distance at which cruising ends, negative acceleration begins [ft]
pos2 = posf - pos1;

% Time interval of phase 2 (constant velocity phase) [seconds]
t_coast = CoastD/vell;
for k = 1:length(t)
    \% lower case 'x' refers to position vector data
    \% upper case 'X' refers to mode-specific end-effector catesian...
    \% x-motion
    X(k) = x(k) + 2.5;
    Y(k) = W/2;
    Z(k) = 1;

    \% Calculate active parameters
    ACTIVE_PARAM

    if view3D == 1
        MOTIONPLOT3D
    end

    if view4 == 1
        MOTIONPLOT4
    end
end

% STRAIGHTLINE.m
% Coordinates a straight line trajectory
% Programmer: Collier Fais
% Ohio University, Mechanical Engineering
% All Rights Reserved, March 2016
%----------------------------------------------------------------

% Acceleration/Deceleration phase time interval [seconds]
t1 = 5;

% Maximum end-effector velocity [ft/s]
vmax = 1;

% Total distance traveled in this specific example [ft]
Xf = 20;

% Distance traveled in Phase 1/2
Phase1Dist = (1/2)*t1*vmax;
Phase2Dist = Phase1Dist;

% Distance traveled at constant velocity [ft]
CoastDist = Xf - 2*Phase1Dist;

% Time interval of phase 2 (constant velocity phase) [seconds]
t_coast = CoastDist/vmax;
% Calculate final time of trajectory based on the above information [seconds]
tf = 2*t1 + t_coast;
t2 = tf - t1;

% Send info to the driver to calculate incremental position data, along
% with it's derivatives
POLY_7_DRIVER

% provide model vectors which represent the tower structures

for k = 1:length(t)
    if k == 1
        nameSNAP_PLOT = 'Font, Top, Right and 3D View Snapshot';
        % close all;

        % Figure 1, 3D plot
        FourSnapView = figure('position',[10 10 1520 760],'
name',nameSNAP_PLOT,'NumberTitle','off');
    End

    X(k) = x(k) + 2.5;
    Y(k) = W/2;
    Z(k) = 1;
    FRAMEPOSE
    FRAMEPLOT
end

% subplot generation
figure; %plots velocity vs t
subplot(511);
plot(t, x, 'b'); grid;
title('Position vs. time');
ylabel('Position');
xlim([0 tf]);

subplot(512);
plot(t, v, 'b'); grid;
title('Velocity vs. time');
ylabel('Velocity');
xlabel('Time');
xlim([0 tf]);

%plots accel vs t
subplot(513);
plot(t, a, 'b'); grid;
title('Acceleration vs. time');
ylabel('Acceleration');
xlim([0 tf]);
% plots jerk vs t
subplot(514);
plot(t, j, 'b'); grid;
title('Jerk vs. time');
ylabel('Jerk');
xlim([0 tf]);

% plots snap v t
subplot(515);
plot(t, s, 'b'); grid;
title('Snap vs. time');
ylabel('Snap');
xlim([0 tf]);

%-----------------------------------
%   TRAJECTORY_DIR.m
%   Directs the trajectory menu input
%   Programmer: Collier Fais
%   Ohio University, Mechanical Engineering
%   All Rights Reserved, April 2016
%-----------------------------------

if trajectory == 1
    STRAIGHT_LINE_DRIVER
end

if trajectory == 2
    CIRCLE_DRIVER
end

if trajectory == 3
    HARVEST_DRIVER
end

if trajectory == 4
    OBJECTIVE4_DRIVER
end

%-----------------------------------
%   Collier Fais
%   7th order polynomial driver
%   Copyright March 2016
%-----------------------------------

% close all;
% clear;
% clc;
% Initial conditions, always true
pos0 = 0;
vel0 = 0;
acc0 = 0;
jerk0 = 0;
snap0 = 0;

% Final conditions, always true
velF = 0;
accF = 0;
jerkF = 0;
snapF = 0;

% Solving Ta = b for 7th order polynomial coef. matrix 'a'
T = [t0^7 t0^6 t0^5 t0^4 t0^3 t0^2 t0 1;
    7*t0^6 6*t0^5 5*t0^4 4*t0^3 3*t0^2 2*t0 1 0;
    42*t0^5 30*t0^4 20*t0^3 12*t0^2 6*t0 2 0 0;
    210*t0^4 120*t0^3 60*t0^2 24*t0 6 0 0 0;
    t1^7 t1^6 t1^5 t1^4 t1^3 t1^2 t1 1;
    7*t1^6 6*t1^5 5*t1^4 4*t1^3 3*t1^2 2*t1 1 0;
    42*t1^5 30*t1^4 20*t1^3 12*t1^2 6*t1 2 0 0;
    210*t1^4 120*t1^3 60*t1^2 24*t1 6 0 0 0];

b = [vel0;acc0;jerk0;snap0;vel1;accF;jerkF;snapF];

% Solve
tcoef = T\b;

% Rename the coef. for ease of use
a0 = coef(8);
a1 = coef(7);
a2 = coef(6);
a3 = coef(5);
a4 = coef(4);
a5 = coef(3);
a6 = coef(2);
a7 = coef(1);

% Initialize counter for Phase 3
phase3counter = 0;

% Calculate i-value corresponding to end of phase 1, used to mirror curve in
% phase 3
count1 = ((length(t)-1)*PR)+1;

% Piecewise loop used to populate the vector kinematics of end-effector
% in specified direction of travel
for i = 1:length(t)
    % Phase 1: acceleration
    if t(i) <= t1
% Phase 2: cruising at constant velocity in vector direction of travel
elseif t(i) > t1 && t(i) < t2
    x(i) = vel1*(t(i)-t1) + pos1;
    v(i) = vel1;
    a(i) = 0;
    j(i) = 0;
    s(i) = 0;

% Phase 3: negative acceleration
else
    phase3counter = 0;
    v(i) = v(1,count1-phase3counter);
    x(i) = -1*x(1,count1-phase3counter) + pos2 + pos1;
    a(i) = -1*a(1,count1-phase3counter);
    j(i) = -1*j(1,count1-phase3counter);
    s(i) = -1*s(1,count1-phase3counter);

    phase3counterNEW = phase3counter + 1;
    phase3counter = phase3counterNEW;

end

end

%------------------------------------------------------------------------
% Collier Fais
% 7th order polynomial driver
% Copyright March 2016
%------------------------------------------------------------------------
% Initial conditions, always true
pos0 = 0;
vel0 = 0;
acc0 = 0;
jer0 = 0;
snap0 = 0;

% Final conditions, always true
velF = 0;
accF = 0;
jerkF = 0;
snapF = 0;

% Solving Ta = b for 7th order polynomial coef. matrix 'a'
T = [t0^7 t0^6 t0^5 t0^4 t0^3 t0^2 t0 1; 
    7*t0^6 6*t0^5 5*t0^4 4*t0^3 3*t0^2 2*t0 1 0; 
    42*t0^5 30*t0^4 20*t0^3 12*t0^2 6*t0 2 0 0; 
    210*t0^4 120*t0^3 60*t0^2 24*t0 6 0 0 0; 
    t1^7 t1^6 t1^5 t1^4 t1^3 t1^2 t1 1; 
    7*t1^6 6*t1^5 5*t1^4 4*t1^3 3*t1^2 2*t1 1 0; 
    42*t1^5 30*t1^4 20*t1^3 12*t1^2 6*t1 2 0 0; 
    210*t1^4 120*t1^3 60*t1^2 24*t1 6 0 0 0];

b = [vel0;acc0;jerk0;snap0;vel1;accF;jerkF;snapF];

% Solve
coef = T\b;

% Rename the coef. for ease of use
a0 = coef(8);
a1 = coef(7);
a2 = coef(6);
a3 = coef(5);
a4 = coef(4);
a5 = coef(3);
a6 = coef(2);
a7 = coef(1);

% Initialize counter for Phase 3
phase3counter = 0;

% Calculate i-value corresponding to end of phase 1, used to mirror curve in
% phase 3
% count1 = ((length(t)-1)*PR)+1;

% Piecewise loop used to populate the vector kinematics of end-effector
% in specified direction of travel
for i = 1:length(ThisLegt)
    % Phase 1: acceleration
    if t(i) <= t1
        x(i) = (1/8)*a7*t(i)^8 + (1/7)*a6*t(i)^7 + (1/6)*a5*t(i)^6 + 
            (1/5)*a4*t(i)^5 + (1/4)*a3*t(i)^4 + (1/3)*a2*t(i)^3 + (1/2)*a1*t(i)^2 + 
            a0*t(i) + pos0;
        v(i) = a7*t(i)^7 + a6*t(i)^6 + a5*t(i)^5 + a4*t(i)^4 + 
            a3*t(i)^3 + a2*t(i)^2 + a1*t(i) + a0;
        a(i) = 7*a7*t(i)^6 + 6*a6*t(i)^5 + 5*a5*t(i)^4 + 4*a4*t(i)^3 + 
            3*a3*t(i)^2 + 2*a2*t(i) + a1;
        j(i) = 42*a7*t(i)^5 + 30*a6*t(i)^4 + 20*a5*t(i)^3 + 
            12*a4*t(i)^2 + 6*a3*t(i) + 2*a2;
\[ s(i) = 210a7t(i)^4 + 120a6t(i)^3 + 60a5t(i)^2 + \\
24a4t(i) + 6a3; \]

% Phase 2: cruising at constant velocity in vector direction of travel
elseif t(i) > t1 && t(i) < t2
    x(i) = vel1*(t(i)-t1) + pos1;
    v(i) = vel1;
    a(i) = 0;
    j(i) = 0;
    s(i) = 0;

% Phase 3: negative acceleration
else

    v(i) = v(1,count1-phase3counter);
    x(i) = -1*x(1,count1-phase3counter) + pos2 + pos1;
    a(i) = -1*a(1,count1-phase3counter);
    j(i) = -1*j(1,count1-phase3counter);
    s(i) = -1*s(1,count1-phase3counter);

    phase3counterNEW = phase3counter + 1;
    phase3counter = phase3counterNEW;

end
end

%----------------------------------------------------------
% TRAJECTORY_PLOTS.m
% Creates report of trajectory by plotting the following:
% End-Effector Velocity, Acceleration, Jerk, Snap vs. Time
% Cable Lengths vs. Time
% Cartesian Trajectory vs. Time
% Cable Tensions vs. Time
% Determinant of \[ A A^T \] vs. Time
% Robot Cartesian Stiffness vs. Time
% Programmer: Collier Fais
% Ohio University, Mechanical Engineering
% All Rights Reserved, March 2016
%----------------------------------------------------------

nameENDEFFECTORKIN = 'End-Effector Kinematics vs. Time';
EndEffectorKin = figure('position',[10 10 1520 760],'
name',nameENDEFFECTORKIN,'NumberTitle','off');
% subplot generation
%plots velocity vs t
subplot(511);
plot(t, x, 'b'); grid;
title('Position vs. time');
ylabel('Position');
xlim([0 tf]);

subplot(512);
plot(t, v, 'b'); grid;
title('Velocity vs. time');
ylabel('Velocity');
% axis equal;
% ylim([-2 2]);
xlim([0 tf]);

% plots accel vs t
subplot(513);
plot(t, a, 'b'); grid;
title('Acceleration vs. time');
ylabel('Acceleration');
xlim([0 tf]);

% plots jerk vs t
subplot(514);
plot(t, j, 'b'); grid;
title('Jerk vs. time');
ylabel('Jerk');
xlim([0 tf]);

% plots snap vs t
subplot(515);
plot(t, s, 'b'); grid;
title('Snap vs. time');
ylabel('Snap');
xlim([0 tf]);
xlabel('Time [sec]');

% Active cable lengths [ft]
nameACTIVECL = 'Active Cable Lengths vs. Time';
ActiveCL = figure('position', [10 10 1520 760], 'name', nameACTIVECL, 'NumberTitle', 'off');
plot(t, L1, 'r', t, L2, 'k', t, L3, 'c', t, L4, 'g'); grid;
title('Active Cable Lengths vs. Time');
legend('Cable 1', 'Cable 2', 'Cable 3', 'Cable 4');
ylabel('Cable Length [ft]');
xlim([0 tf]);
xlabel('Time [sec]');

% Active Cable Tensions [lbf]
nameACTIVECT = 'Active Cable Tensions vs. Time';
ActiveCT = figure('position', [10 10 1520 760], 'name', nameACTIVECT, 'NumberTitle', 'off');
plot(t, T1, 'r', t, T2, 'k', t, T3, 'c', t, T4, 'g'); grid;
title('Active Cable Tensions vs. Time');
legend('Cable 1', 'Cable 2', 'Cable 3', 'Cable 4');
ylabel('Cable Tension [lbf]');
xlim([0 tf]);
xlabel('Time [sec]');
% Active Cable Tensions [lbf]
nameCARTESIAN = 'Cartesian Trajectory vs. Time';
CartT = figure('position',[10 10 1520 760],'
name',nameCARTESIAN,'NumberTitle','off');
plot(t,X,'r',t,Y,'k',t,Z,'c'); grid;
title('Cartesian Trajectory vs. Time');
legend('[X]','[Y]','[Z]')
ylabel('Cartesian Trajectory [ft]');
xlim([0 tf]);
xlabel('Time [sec]');

% Robot Cartesian Stiffness [lbf/ft]
nameSTIFFNESS = 'Robot Cartesian Stiffness vs. Time';
CartT = figure('position',[10 10 1520 760],'
name',nameSTIFFNESS,'NumberTitle','off');
plot(t,Stiff,'r'); grid;
title('Robot Cartesian Stiffness vs. Time');
ylabel('Robot Cartesian Stiffness [lbf/ft]');
xlim([0 tf]);
xlabel('Time [sec]');

% Determinant of [A*A^T]
nameSTIFFNESS = 'Determinant of [A A^T] vs. Time';
CartT = figure('position',[10 10 1520 760],'
name',nameSTIFFNESS,'NumberTitle','off');
plot(t,det_A,'r'); grid;
title('Determinant of [A A^T] vs. Time');
ylabel('Determinant of [A A^T] [1/ft]');
xlim([0 tf]);
xlabel('Time [sec]');

%-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
%- TRAJECTORY_PLOTS.m
%- Creates report of trajectory by plotting the following:
%- End-Effector Velocity, Acceleration, Jerk, Snap vs. Time
%- Cable Lengths vs. Time
%- Cartesian Trajectory vs. TimeS
%- Cable Tensions vs. Time
%- Determinant of [A A^T] vs. Time
%- Robot Cartesian Stiffness vs. Time
%- Programmer: Collier Fais
%- Ohio University, Mechanical Engineering
%- All Rights Reserved, March 2016
%-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-

nameENDEFFECTORKIN = 'End-Effector Kinematics vs. Time';
EndEffectorKin = figure('position',[10 10 1520 760],'
name',nameENDEFFECTORKIN,'NumberTitle','off');
% subplot generation
%plots velocity vs t
subplot(511);
plot(t, x, 'b'); grid;
title('Position vs. time');
ylabel('Position');
xlim([0 tf]);

subplot(512);
plot(t, v, 'b'); grid;
title('Velocity vs. time');
ylabel('Velocity');
% axis equal;
% ylim([-2 2]);
xlim([0 tf]);

% plots accel vs t
subplot(513);
plot(t, a, 'b'); grid;
title('Acceleration vs. time');
ylabel('Acceleration');
xlim([0 tf]);

% plots jerk vs t
subplot(514);
plot(t, j, 'b'); grid;
title('Jerk vs. time');
ylabel('Jerk');
xlim([0 tf]);

% plots snap v t
subplot(515);
plot(t, s, 'b'); grid;
title('Snap vs. time');
ylabel('Snap');
xlim([0 tf]);
xlabel('Time [sec]');

% Active cable lengths [ft]
nameACTIVECL = 'Active Cable Lengths vs. Time';
ActiveCL = figure('position',[10 10 1520 760],'
name',nameACTIVECL,'NumberTitle','off');
plot(t,L1,'r',t,L2,'k'); grid;
title('Active Cable Lengths vs. Time');
legend('Cable 1','Cable 2')
ylabel('Cable Length [ft]');
xlim([0 tf]);
xlabel('Time [sec]');

% Active Cable Tensions [lbf]
nameACTIVECT = 'Active Cable Tensions vs. Time';
ActiveCT = figure('position',[10 10 1520 760],'
name',nameACTIVECT,'NumberTitle','off');
plot(t,T1,'r',t,T2,'k'); grid;
title('Active Cable Tensions vs. Time');
legend('Cable 1','Cable 2')
ylabel('Cable Tension [lbf]');
xlim([0 tf]);
xlabel('Time [sec]');

% Active perpendicular force acting on top of towers [lbf]
nameACTIVECF = 'Active Perpendicular Force on Tower vs. Time';
ActiveCF = figure('position',[10 10 1520 760],'
name',nameACTIVECF,'NumberTitle','off');
plot(t,F1,'r',t,F2,'k'); grid;
title('Active Perpendicular Force on Tower vs. Time');
legend('Tower 1','Tower 2')
ylabel('Perpendicular Force [lbf]');
xlim([0 tf]);
xlabel('Time [sec]');

% Cartesian trajectory vs time [lbf]
nameCARTESIAN = 'Cartesian Trajectory vs. Time';
CartT = figure('position',[10 10 1520 760],'
name',nameCARTESIAN,'NumberTitle','off');
plot(t,X,'r',t,Y,'k'); grid;
title('Cartesian Trajectory vs. Time');
legend('X','Y')
ylabel('Cartesian Trajectory [ft]');
xlim([0 tf]);
xlabel('Time [sec]');

% Active cable rates, velocities [ft/s]
nameACTIVECV = 'Active Cable Velocities [ft/s] vs. Time';
ActiveCV = figure('position',[10 10 1520 760],'
name',nameACTIVECV,'NumberTitle','off');
plot(t,L1_dot,'r',t,L2_dot,'k'); grid;
title('Active Cable Velocities vs. Time');
legend('Cable 1','Cable 2')
ylabel('Cable Velocities [ft/s]');
xlim([0 tf]);
xlabel('Time [sec]');

% Active cable accelerations [ft/s^2]
nameACTIVECA = 'Active Cable Accelerations [ft/s^2] vs. Time';
ActiveCA = figure('position',[10 10 1520 760],'
name',nameACTIVECA,'NumberTitle','off');
plot(t,L1_ddot,'r',t,L2_ddot,'k'); grid;
title('Active Cable Accelerations vs. Time');
legend('Cable 1','Cable 2')
ylabel('Cable Acceleration [ft/s^2]');
xlim([0 tf]);
xlabel('Time [sec]');

% Active reel angular velocities [rad/s]
nameACTIVERV = 'Active Cable Reel Angular Velocities [rad/s] vs. Time';
ActiveRV = figure('position',[10 10 1520 760],'
name',nameACTIVERV,'NumberTitle','off');
plot(t,w_reel1,'r',t,w_reel2,'k'); grid;
title('Active Cable Reel Angular Velocities vs. Time');
ylabel('Cable Reel Velocities [rad/s]');
xlabel('Time [sec]');

legend('Reel 1','Reel 2')

xlim([0 tf]);

nameACTIVERA = 'Active Cable Reel  Angular Accelerations [rad/s^2] vs. Time';
ActiveRA = figure('position',[10 10 1520 760],'

plot(t,a_reel1,'r',t,a_reel2,'k'); grid;
title('Active Cable Reel  Angular Accelerations vs. Time');
legend('Reel 1','Reel 2')
ylabel('Cable Reel Angular Acceleration [rad/s^2]');
xlim([0 tf]);
xlabel('Time [sec]');

nameACTIVEMV = 'Active Motor Axis Angular Velocities [rad/s] vs. Time';
ActiveMV = figure('position',[10 10 1520 760],'

plot(t,w_motor1,'r',t,w_motor2,'k'); grid;
title('Active Motor Axis Angular Velocities vs. Time');
legend('Motor 1','Motor 2')
ylabel('Motor Axis Velocities [rad/s]');
xlim([0 tf]);
xlabel('Time [sec]');

nameACTIVEMA = 'Active Motor Axis  Angular Accelerations [rad/s^2] vs. Time';
ActiveMA = figure('position',[10 10 1520 760],'

plot(t,a_motor1,'r',t,a_motor2,'k'); grid;
title('Active Motor Axis Angular Accelerations vs. Time');
legend('Motor 1','Motor 2')
ylabel('Motor Axis Angular Acceleration [rad/s^2]');
xlim([0 tf]);
xlabel('Time [sec]');

nameACTIVEpulse = 'Active Encoder Pulse Count [rad/s^2] vs. Time';
Activepulse = figure('position',[10 10 1520 760],'

plot(t,pulse1,'r',t,pulse2,'k'); grid;
title('Active Encoder Pulse Count vs. Time');
legend('Motor 1','Motor 2')
ylabel('Encoder Pulse count');
xlim([0 tf]);
xlabel('Time [sec]');
disp('TRAJECTORY MENU')

nameTRAJECTORYMENU = 'Trajectory Menu';

% Creates figure window displaying Trajectory Menu options
TMENU = figure('Position',[100 300 610 260],'name',nameTRAJECTORYMENU,'NumberTitle','off');

% Display a .jpeg image on the menu
I = imread('C:\Users\cf277_000\Documents\Cable Robot MATLAB\ISO_ML.jpg');
imshow(I)

txt1 = uicontrol('Style','text','Position',[10 220 180 30],'String','Select a Trajectory','fontsize',14);
drop1 = uicontrol('style','popup','string',{'Straight Line','Circle','Harvest Example','Objective 4'},'position',[200 220 150 30]);

txt2 = uicontrol('Style','text','Position',[10 180 180 30],'String','View Style','fontsize',14);
r1 = uicontrol('style','radiobutton','string','3D View Only (Recommended)','Position',[200 180 180 30],'HandleVisibility','off');
r2 = uicontrol('style','radiobutton','string','3D View and Orthographic Views','Position',[200 150 200 30],'HandleVisibility','off');

% Button 3a is START
btn3a = uicontrol('Style','pushbutton','string','Start','position',[10 10 70 40],'fontsize',20,'callback','trajectory = get(drop1,''value''); view3D = get(r1,''value''); view4 = get(r2,''value''); close all;TRAJECTORY_DIR;')

% Button 1e is RETURN
btn3b = uicontrol('Style','pushbutton','string','Return','position',[110 10 110 40],...
'fontsize', 20, ...
'callback', 'MENU1');

% Button 1e is EXIT, activated downstream if choice1 = 0
btn3c = uicontrol('Style', 'pushbutton', 'string', 'Exit', ...
'position', [250 10 70 40], ...
'fontsize', 20, ...
'callback', 'clear; close all;');

waitfor(TMEN);  % Waits to execute the rest of the parent script
until user chooses any option
// Forward planar straight line trajectory
// Program file

vel1 := xf/(tf*(1-PR));
//t1 := tf*PR;
pos1 := (0.5)*t1*vel1;
pos2 := xf - pos1;

a4 := (35*vel1)/EXPT(t1,4);
a5 := (-84*vel1)/EXPT(t1,5);
a6 := (70*vel1)/EXPT(t1,6);
a7 := (-20*vel1)/EXPT(t1,7);

// Fill time array [0:0.1:120] (seconds)

FOR k:= k TO 2400 BY 1 DO
  t[k]:= count1;
  count1:= count1 + 0.1;

  g_PTArray1[k].delta_time:= TIME#13MS; //Time increment is constant at 0.1 seconds or 100 ms
  g_PTArray2[k].delta_time:= TIME#13MS; //Time increment is constant at 0.1 seconds or 100 ms
END_FOR

FOR i:= i TO 2400 DO
IF i <= 240 THEN
    pos[i] := (1.0/8.0)*a7*EXPT(t[i],8) +
    (1.0/7.0)*a6*EXPT(t[i],7) +
    (1.0/6.0)*a5*EXPT(t[i],6) +
    (1.0/5.0)*a4*EXPT(t[i],5);
END_IF

IF i > 240 AND i < 2160 THEN
    pos[i] := vel1*(t[i-240]) + pos1;
END_IF

IF i>= 2160 THEN
    pos[i] := -1*pos[240-count2] + pos1 + pos2;
    count2:=count2+1;
END_IF

END_FOR

FOR j:= j TO 2400 BY 1 DO
    L1[j] := SQRT((EXPT((pos[j]+2),2))+(EXPT((47.0/12.0),2)));
    IF j = 0 THEN // Initial cable lengths
        L1o:= L1[j];
    END_IF
    L1d[j] := L1[j] - L1o;
    M1POS[j] := L1d[j]*229.5*1.0;
    g_PTArray1[j].position:= M1POS[j];
END_FOR

FOR n:= n TO 2400 DO
    IF n = 0 THEN
        L2o:= L1[2400];
    END_IF
END_FOR
END_IF

L2[n]:= L1[2400-n];
L2d[n]:= L2[n]-L2o;

M2POS[n]:= L2d[n]*244.5*1.0;
g_PTArray2[n].position:= M2POS[n];

END_FOR

//**********STEPS TO RUN***********************
// 1. ENABLE DRIVE USING i_ON FLAG
// 2. HOME AXIS USING i_HOME FLAG
// 3. PROGRAM WILL THEN BEGIN EXECUTION AUTOMATICALLY
//@**********END STEPS TO RUN***************************/

v_TIMEPOSITION_1.IsAbsolute   :=TRUE;
v_TIMEPOSITION_1.Number_of_pairs :=100;
v_TIMEPOSITION_1.MC_TP_Array  :=g_MOVEPROFILE1;
fb_POSITIONPROFILE_1.ArraySize  :=100;
fb_DRIVE_ON_1.bRegulatorOn   :=i_ON;
fb_DRIVE_ON_1.bDriveStart   :=i_ON;
oPOS_1 :=fb_STATUS_1.Position;
oDRIVE_ON :=fb_DRIVE_ON_1.bRegulatorRealState;
oDRIVE_ON :=fb_DRIVE_ON_1.bDriveStartRealState;

v_TIMEPOSITION_2.IsAbsolute   :=TRUE;
v_TIMEPOSITION_2.Number_of_pairs :=100;
v_TIMEPOSITION_2.MC_TP_Array  :=g_MOVEPROFILE2;
fb_POSIPOSITIONPROFILE_2.ArraySize  :=100;
fb_DRIVE_ON_2.bRegulatorOn   :=i_ON;
fb_DRIVE_ON_2.bDriveStart   :=i_ON;
oPOS_2 := fb_STATUS_2.Position;

oDRIVE_ON := fb_DRIVE_ON_2.bRegulatorRealState;

oDRIVE_ON := fb_DRIVE_ON_2.bDriveStartRealState;

//*********************************************************************
********************
IF oDRIVE_ON = FALSE OR iSTOP = TRUE THEN
    STATE := 0;
END_IF

IF fb_HOME_1.Done = TRUE THEN
    i_HOME := FALSE;
    oHOMED := TRUE;
END_IF

//*********************************************************************
********************
CASE STATE OF

0://CHECK IF DRIVE IS ENABLED BEFORE MOVING TO NEXT STATE; OTHERWISE STAY IN STATE 0
    IF iSTOP = FALSE AND oDRIVE_ON = TRUE AND oHOMED = TRUE AND oDONE_MOVE = FALSE THEN
        oFILL_AGAIN := TRUE;
        STATE := STATE+1;
    ELSE
        STATE:=0;
    END_IF

1://TRANSFER BUFFER ARRAY INTO MOVE PROFILE ARRAY
    IF oMOV_READY = TRUE AND NOT(INDEX = vN_INDEX) THEN
2://EXECUTE MOVE PROFILE
   IF fb_POSITIONPROFILE_1.Busy = TRUE THEN
      fb_POSITIONPROFILE_1.Execute := FALSE;
      fb_POSITIONPROFILE_2.Execute := FALSE;
      oMOV_READY                     := FALSE;
      oHOMED                           := FALSE;
      COUNTER2            := 0;
      VAR3                  := 1;
      INDEX             := INDEX + 1;
      oREADY_AGAIN     := FALSE;
      STATE       := STATE + 1;
   END_IF
3://WAIT FOR COMPLETION
   IF fb_POSITIONPROFILE_1.Done = TRUE THEN
      oFILL_AGAIN     := TRUE;
      STATE      := 1;
   END_IF

END_CASE
IF oFILL_AGAIN = TRUE THEN

    WHILE COUNTER2<100 DO
        g_MOVEPROFILE1[VAR3].delta_time :=
        g_PTArray1[VAR2].delta_time;
        g_MOVEPROFILE1[VAR3].position := g_PTArray1[VAR2].position;
        
        g_MOVEPROFILE2[VAR3].delta_time :=
        g_PTArray2[VAR2].delta_time;
        g_MOVEPROFILE2[VAR3].position := g_PTArray2[VAR2].position;
        
        VAR3:= VAR3+1;
        VAR2:= VAR2+1;
        COUNTER2:=COUNTER2+1;
    END_WHILE
END_IF

IF COUNTER2 = 100 THEN
    oMOV_READY := TRUE;
END_IF

//INITIALIZE FUNCTION BLOCKS
fb_POSITIONPROFILE_1 (AXIS:=X1, TIMEPOSITION:=v_TIMEPOSITION_1);
fb_DRIVE_ON_1 (AXIS:=X1, ENABLE:=TRUE);
fb_RESET_1 (AXIS:=X1, EXECUTE:=i_RESET);
fb_HOME_1 (AXIS:=X1, EXECUTE:=i_HOME);
fb_ZERO_1 (AXIS:=X1, EXECUTE:=i_ZERO, POSITION:=0);
fb_STOP_1 (AXIS:=X1, EXECUTE:=iSTOP, DECELERATION:=1000);
fback_STATUS_1 (AXIS:=X1, ENABLE:=TRUE);

fb_POSITIONPROFILE_2 (AXIS:=X2, TIMEPOSITION:=v_TIMEPOSITION_2);
fback_DRIVE_ON_2 (AXIS:=X2, ENABLE:=TRUE);
fback_RESET_2 (AXIS:=X2, EXECUTE:=i_RESET);
fback_HOME_2 (AXIS:=X2, EXECUTE:=i_HOME);
fback_ZERO_2 (AXIS:=X2, EXECUTE:=i_ZERO, POSITION:=0);
fback_STOP_2 (AXIS:=X2, EXECUTE:=iSTOP, DECELERATION:=1000);
fback_STATUS_2 (AXIS:=X2, ENABLE:=TRUE);

VAR_GLOBAL

{ 
t: ARRAY[1..2400] OF LREAL; // Time array
L1: ARRAY[1..2400] OF LREAL; //Active length of cable 1
L2: ARRAY[1..2400] OF LREAL; //Active length of cable 2
L1_o: LREAL; // Initial cable lengths
L2_o: LREAL;
L1_d: ARRAY[1..2400] OF LREAL; // Difference in cable length with respect to the initial length
L2_d: ARRAY[1..2400] OF LREAL;
M1POS: ARRAY[1..2400] OF LREAL; // Array will contain position profile of motor in revolutions
M2POS: ARRAY[1..2400] OF LREAL;
L2R: LREAL:= 218.570; // Motor revolutions per unit length of cable, CHANGE THIS
L2R_1: LREAL:= 229.5;
L2R_2: LREAL:= 244.8;

L2R_3: LREAL:= 250.0;

// gRATIO: LREAL:= 153.0; // Winch gear ratio
// Phase correction factors, used only for minor adjustments.
Slow down or speed up a certain phase

c11: LREAL:= 1.0;
c12: LREAL:= 1.0;
c13: LREAL:= 1.0;
c14: LREAL:= 1.0;
c15: LREAL:= 1.0;
c16: LREAL:= 1.0;
c17: LREAL:= 1.0;
c18: LREAL:= 1.0;
c19: LREAL:= 1.0;
c110: LREAL:= 1.0;

c21: LREAL:= 1.0;
c22: LREAL:= 1.0;
c23: LREAL:= 1.0;
c24: LREAL:= 1.0;
c25: LREAL:= 1.0;
c26: LREAL:= 1.0;
c27: LREAL:= 1.0;
c28: LREAL:= 1.0;
c29: LREAL:= 1.0;
c210: LREAL:= 1.0;

\[ \text{g\_MOVEPROFILE1: } \text{ARRAY}[1..100] \text{ OF SMC\_TP}; \text{ //Aptly} \]
\[ \text{named to comply with positionProfile demo code} \]
\[ \text{g\_PTArray1: } \text{ARRAY}[1..2400] \text{ OF SMC\_TP;} \]

\[ \text{g\_MOVEPROFILE2: } \text{ARRAY}[1..100] \text{ OF SMC\_TP}; \text{ //Aptly} \]
\[ \text{named to comply with positionProfile demo code} \]
\[ \text{g\_PTArray2: } \text{ARRAY}[1..2400] \text{ OF SMC\_TP;} \]

count1: LREAL:= 0; //This gets incremented by 0.1 and fills the time increment array
count2: UINT:= 0;

xf: LREAL:= 10.0; //Net distance traveled of end-effector
tf:  LREAL:= 240; //Final time in seconds, 240 sec = 4 min
PR:   REAL:= 0.1;
vel1: LREAL;//:= xf/(tf*(1-PR));
t1:   LREAL:= 24; //This is defined as one tenth o the final time
pos1: LREAL;//:= (1/2)*t1*vel1;
pos2: LREAL;//:= xf - pos1;
D:    INT:= 14;
a0: INT:= 0;
a1: INT:= 0;
a2: INT:= 0;
a3: INT:= 0;
a4: LREAL;//:= (35*vel1)/EXPT(t1,4);
a5: LREAL;//:= (-84*vel1)/EXPT(t1,5);
a6: LREAL;//:= (70*vel1)/EXPT(t1,6);
a7: LREAL;//:= (-20*vel1)/EXPT(t1,7);
pos: ARRAY[0..2400] OF LREAL;
i: INT;//:=0;
j: INT;
k: INT;//:=0;
n: INT;

//FUNCTION BLOCKS
fb_POSITIONPROFILE_1:  MC_POSITIONPROFILE;
v_TIMEPOSITION_1:  MC_TP_REF;
fb_DRIVE_ON_1:       MC_POWER;
fb_RESET_1:          MC_RESET;
fb_HOME_1:           MC_HOME;
fb_ZERO_1: MC_SETPOSITION;
fb_STOP_1: MC_STOP;
fb_STATUS_1: MC_ReadActualPosition;

fb_POSITIONPROFILE_2: MC_POSITIONPROFILE;
v_TIMEPOSITION_2: MC_TP_REF;
fb_DRIVE_ON_2: MC_POWER;
fb_RESET_2: MC_RESET;
fb_HOME_2: MC_HOME;
fb_ZERO_2: MC_SETPOSITION;
fb_STOP_2: MC_STOP;
fb_STATUS_2: MC_ReadActualPosition;

//INPUTS
i_ON: BOOL;
i_RESET: BOOL;
i_HOME: BOOL;
i_ZERO: BOOL;
iFILL_BUFFER: BOOL;
iREFILL_BUFFER: BOOL;
iok: BOOL;
io_AGAIN: BOOL;
io_STOP: BOOL;

//OUTPUTS
oDONE_MOVE: BOOL;
oMOV_READY: BOOL;
oPOS_1: LREAL;
oPOS_2: LREAL;
Driven: BOOL;
oHOMED: BOOL := FALSE;
oREADY_NEXT_MOV: BOOL := FALSE;
oPTArray_READY: BOOL;
ofILL_AGAIN: BOOL;
ofREADY_AGAIN: BOOL;

//VARIABLES
COUNTER1: UINT := 0;
COUNTER2: UINT := 0;
STATE: INT := 0;
VAR1: INT := 1;
VAR2: INT := 1;
VAR3: INT := 1;
P1: LREAL := 0;
//T1: TIME := TIME\#10MS;
INDEX: UINT := 1;
vN_INDEX: UINT := 24;

oARRAY_READY: BOOL;

END_VAR
int pinA = 3; // Connected to CLK on KY-040
int pinB = 4; // Connected to DT on KY-040
int encoderPosCount = 0;
int pinALast;
int aVal;
boolean bCW;
unsigned long t;

void setup() {
  pinMode (pinA, INPUT);
  pinMode (pinB, INPUT);
pinALast = digitalRead(pinA);
  Serial.begin (9600);
}
void loop() {
aVal = digitalRead(pinA);
  if (aVal != pinALast){ // Means the knob is rotating
    if (digitalRead(pinB) != aVal) { // Means pin A Changed first - We're Rotating Clockwise
      encoderPosCount ++;
bCW = true;
    } else { // Otherwise B changed first and we're moving CCW
      bCW = false;
      encoderPosCount--;
    }
  t = millis();
  Serial.println(t);
}
pinALast = aVal;