Major Qualifying Projects (All Years)   Major Qualifying Projects

March 2013

Design and Construction of a Tree-Climbing Robot

Eric Cobane
Worcester Polytechnic Institute

Ian A. Campbell
Worcester Polytechnic Institute

Ryan Giovacchini
Worcester Polytechnic Institute

Thomas Grant Murray
Worcester Polytechnic Institute

Follow this and additional works at: https://digitalcommons.wpi.edu/mqp-all

Design and Construction of a Tree-Climbing Robot

A Major Qualifying Project Report

Submitted to the faculty of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of **Bachelor of Science**

Advisors:

Michael A. Gennert

Zhikun Hou

Submitted By:

Ian Campbell

Eric Cobane

Ryan Giovacchini

Thomas Murray

Date: Tuesday March 12th 2013

Abstract

This report describes the research, mechanical analysis, design methodology, and testing procedures that were used to design and build a tree-climbing robot. The goal of this project was to build a tree-climbing robot to satisfy the requirements established by the USDA and aid in the detection of Asian Longhorn Beetles. The following report details the threat that invasive beetle species pose to the United States, how tree climbing robots may help eliminate invasive species, a review of robots that have successfully climbed trees, and how effective they may be at locating beetles, our considerations when developing a tree climbing robot design, the preliminary robot design, the final robot design, mechanical analysis, programming structure, and the results that were achieved by the robot.

Executive Summary

The Asian Longhorn Beetle (ALB) is an extremely destructive invasive species native to China, Japan, and Korea, which was brought to the United States and now threatens to destroy many hardwood trees. The current methods of beetle detection involve workers climbing trees to find evidence of beetles and are hazardous, expensive, time consuming, and ineffective. The USDA endorsed the potential solution of using robots equipped with cameras to detect beetles instead of using humans.

Different research projects have resulted in the creation of robots such as RISE, Treebot, and many others, which have successfully climbed trees. However, each of these designs has limitations in functionality, which would prevent it from being used for beetle detection. This project is a continuation of last year's tree-climbing robot MQP, both of which sought to design and construct a tree-climbing robot that meets the requirements specified by the USDA.

After comparing various tree climbing strategies for their pros and cons, a robot platform with a gait similar to that of an inchworm was chosen. Subsequently, a prototype was designed and a concept gripper was produced. The robot and gripper were analyzed mechanically, programmed, and tested. After testing the prototype gripper, a new robot chassis was redesigned, analyzed, built, programmed, and tested. The end result of this project was proof of concept, useful analysis that can contribute to next year's MQP, and a plethora of future recommendations.

Table of Contents

Table of Figures

Table of Tables

Authorship

Primary Editor: Ian Campbell

1.   Introduction

The Asian Longhorn Beetle (ALB) is an extremely destructive invasive species native to China, Japan, and Korea, which was brought to the United States on shipping containers in the late 1980's. These beetles have killed off more than 50 million hardwood trees over a three-year period in the province of Ningxia, China. Their more recent presence in Worcester, Massachusetts resulted in the removal of 18,000 trees from 2008 to 2011 (Nisley, 2012). These pests were also found in New York in 1996, Illinois in 1998, New Jersey in 2004, and Ohio in 2011. If the infestation continues to spread throughout the United States, scientists believe the beetles could kill a third of the country's trees (Reardon, 2012; Daniel 2011). If this were to happen, the national parks and hardwood forests of America could be destroyed, which would affect the production of furniture, maple syrup, as well as other goods made from trees and wood. (Reardon, 2012; Kenny, 2011).

The ALB is a larger beetle with a shiny black body that is covered in irregular white spots. Adults can range in length from 3/4 to 1-1/4 inches and have two long white and black antennae (Drew, 2008). In its native habitat, the beetle usually takes up residence in poplar and willow trees, however the beetles taste expanded when it was introduced to North America. In the United States the beetle typically inhabits maple, box elder, buckeye, willow, elm, birch, and sycamore trees (Sawyer, 2010).

Thus far it has been difficult to contain the ALB infestation because traditional pest control methods, such as the application of pesticides, are not very effective in reducing the population of the beetles. This is because the beetle spends most of its life cycle inside the tree, eating it from the inside out. The proven method of successfully eliminating an ALB population requires locating the infested trees, cutting down the infested trees and some surrounding

potential host trees, chipping the trees up into fine particles, incinerating the remains, and applying pesticides to the surrounding trees (Reardon, 2012). Currently the only way to locate an infested tree is to hire individuals to climb trees and look for the telltale signs of infestation, small 3/8-1/2" entrance holes in the tree, and/or locate frass, sawdust like shavings which can accumulate near the hole openings (Nisley, 2012). Tree inspections can be hazardous to workers, expensive for small towns, and are labor intensive.

While the current "chip and burn" methods of ALB detection and elimination could potentially slow the progression of an infestation, these methods are only effective if the infected trees are properly identified. Since there is a high probability of a worker missing a small hole in a large tree, a more reliable detection method is needed to improve chances of eliminating the infestation. A robot could be used as an alternative to humans for tree inspections, this would improve chances of locating an infestation, lower the cost of tree inspections, and make it more safe for workers to inspect trees.

Different research projects have resulted in the creation of robots such as RISE, Treebot, and many others, which have successfully climbed trees. However, each of these designs has limitations in functionality, which would prevent it from being used for beetle detection. Other robots are either too big, too complex, cannot climb around branches, or do not have room to mount a camera. This project, which is a continuation of last year's MQP, seeks to design and construct a tree climbing robot that meets the requirements specified by the USDA.

2.   Background

2.1   Researching the Problem

After reviewing the USDA's requirements, extensive research was conducted to locate as much information as possible on robot designs that could climb trees and similar surfaces. The following section contains a review of these robot designs. After compiling a list of tree climbing robots, they were individually analyzed for their advantages, disadvantages, and ability to meet the USDA's requirements. Each design tree climbing robot design was evaluated based on how well it could climb, the surfaces it could climb, and whether or not it could maneuver around branches.

2.1.1   RiSE

The RiSE project was funded by the Defense Advanced Research Projects Agency (DARPA). DARPA's biodynamic robotics program consists entirely of robots that are biologically inspired and are designed to function and maneuver in a variety of conditions. One of the goals is the development of a robot that can climb vertically. The possible applications for such robots include surveillance, retrieval, and inspection. Boston Dynamics Inc., in collaboration with several Universities, has at this point created three versions of the RiSE robot which can climb straight up trees and wooden poles (University of Pennsylvania, 2012).

RiSE V1 was first announced in 2005 (University of Pennsylvania, 2012). Each of its six legs is actuated by two electric motors, giving them each two degrees of freedom. The robot was tested mainly on carpeted walls to analyze and enhance its climbing ability. This robot maintains stability while climbing by using a tripod gait, meaning that at least three legs are in contact with the climbing surface at any given time. The robot maintains that grip by using a tail, which is

attached to the rear of the chassis and has the ability to push the robot. Figure 1 below shows how the tail works by pushing towards the climbing surface, which allows the front of the robot to remain within reach of the tree.

Figure 1: RiSE V1

RiSE V2 was the successor to RiSE V1, and was very similar in structure to the original version, and it uses the same six legged configuration, each leg is powered by two actuators each. It reuses the tripod gait for climbing, in which three legs maintain contact with the surface at all times. This robot also has several end effector modules, which allow it to climb a variety of surfaces including outdoor walls and trees, as shown in Figure 2 below.

Figure 2: RiSE V2

The gripping method for this robot includes spines made from modified medical needles installed at the end of each leg. These micro-spine covered feet allow penetration to be made into the climbing surface with minimal damage. With two degrees of actuated freedom on each leg, the designers are able to determine and utilize the best direction in which to apply force through the spiny feet for maximum gripping.

RiSE V3 brought about some major changes from the previous versions. This robot employs a Quadrupedal configuration, which means it only has four legs instead of the original six. Different brushless DC motors are used in this version to increase power. Coupled with a dramatically different leg mechanism and unique gaited behavior, this robot exhibits rapid climbing (upward of 22 cm/s) up a vertical surface such as a telephone pole. (University of Pennsylvania, 2012). This chassis offers another degree of freedom over the old design. A

pivoting joint in the backbone of the robot allows it to adjust its upper body toward or away from the climbing surface. This gives it even more ability to adjust to the optimal gripping position during climbing. This can be seen in Figure 3 below.

Figure 3: RiSE V3

2.1.2 WOODY

The WOODY project began in 2004 in the Sugano Lab at Waseda University in Japan, and since then there have been three generations of prototypes (Waseda University 2003). Unlike the RiSE

project, the only desired application for WOODY is in forest preservation. Trees need to be periodically pruned to reduce the number of branches on them. Too many branches can have negative effects on the forest by blocking sunlight, as well as accumulating precipitation which in turn can cause the trees to fall.

WOODY adheres to tree trunks by wrapping its two arms all the way around them. Because of this, the robot is limited to a certain range of tree diameters it can climb. The robot climbs by alternating grip on the upper and lower arms, and by using a worm gear to generate vertical motion, similar to an inchworm. The tree side of the arm has wheels mounted to it which allow for rotational motion. Due to its configuration, the robot can only climb up straight trees, and cannot avoid branches. Thus, WOODY is equipped with a saw mounted at the highest point of the robot to allow it to remove obstructing branches and proceed upward, as depicted in Figure 4 below.

Figure 4: WOODY: "Robot Assisting Forestry Work"

2.1.3 Uncle Sam

Carnegie Mellon University developed a modular hyper-redundant robot named Uncle Sam that mimics the motions of a snake. Using universal joints with three degrees of freedom, the robot is

able to move in many different ways including rolling, wiggling, and side winding, depending on the terrain being encountered (Carnegie Mellon University, 2008). The way this robot climbs trees is one method that is not actually borrowed from the snake. Instead, it wraps around the tree trunk and applies inward pressure while rolling its body to generate vertical motion up the tree, as seen in Figure 5 below. This climbing method is effective in certain situations but also has some inherent limitations. First, the body of the robot must be long enough to wrap all the way around the tree trunk, and second, it is not able to overcome branches. However, researchers think that if they increase the length of the robot, they may enhance the ability of the robot to maneuver onto and off of branches (Carnegie Mellon University, 2008).

Figure 5: Uncle Sam, courtesy of Carnegie Mellon University

2.1.4 Kawasaki's Pruning Robot

A tree-pruning robot was developed at the Kawasaki & Mouri Lab of Gifu University in Japan. This robot climbs up cylindrical objects, whether trees or metal posts, yet it does not possess the ability to transition onto branches that may obstruct the robot's climbing progress. It is designed

so that its center of gravity resides within the tree when it is mounted around the tree. It has four wheels in contact with the tree that it is climbing, two of which are in contact with the tree below the robot's center of gravity and adjacent to each other and two of which are in contact with the tree above the robot's center of gravity. This design negates the need for a pushing force on the tree to be exerted by on-board actuators. The static state of the robot naturally creates the pushing forces needed to maintain the four wheels' traction on the tree and prevent the robot from falling off of the tree (Kawasaki & Mouri Lab). The orientation of these four contact points can be seen in Figure 6, below (Kawasaki & Mouri Lab).

Figure 6: Kawasaki's Tree Pruning Robot at Gifu University, Japan

The robot uses a worm gear drive of each wheel independently in order to prevent them from being back-driven, which would otherwise allow the robot to roll back down the tree when the motors were not active. The first prototype of the robot was developed in 2008. This first iteration exhibited fixed wheels that were aligned vertically, thereby allowing exclusively vertical travel up and down the tree through one degree of freedom. The second prototype incorporated wheels that were capable of actively steering, thereby adding a second degree of

freedom. This allowed the robot to switch between varying degrees of vertical versus spiral

climbing patterns to potentially improve climbing efficiency (Kawasaki & Mouri Lab).

2.1.5   Seirei Industry's Automatic Pruning Machine

Seirei Industry Co.'s AB232R Automatic Pruning Machine is a commercialized tree-climbing

robot. Its wheels are mounted at fixed angles that, when driven, move the robot up a tree in a

fixed spiral pattern. Since the wheel orientation cannot be changed, neither can the spiral pattern

with which the robot climbs the tree. This skewed orientation of the wheels can be seen in Figure

7, below (Seirei Industry Co.).

Figure 7: Seirei Industries' Automatic Pruning Machine (AB232R)

This robot generates its gripping force on the tree from pre-loaded springs. This approach

to providing the necessary gripping force restricts the domain of tree diameters that the robot can

climb. This particular model is designed to climb trees with diameters between 70mm and

230mm. This robot is not able to climb onto any branches other than the primary tree trunk and will instead indiscriminately cut off any branches that it runs into with its cutting tool.

2.1.6 TREPA

In 2006, a robot called TREPA was developed at Miguel Hernandez University. This robot uses a version of the six-degree-of-freedom Gough-Stewart platform. This platform uses six linear actuators connected to a platform at each of their ends via universal joints (Aracil, 2006). In this configuration, each of the "platforms" is actually hollow. The robot is mounted around the tree with and the tree trunk occupies the cylindrical void in the middle of the robot. The rings, hollow "platforms" on the top and bottom of the robot, grip the tree with actuated grippers that fold in to apply pressure on the tree from multiple sides of the tree. The robot's structure can be seen in Figure 8, below (Aracil, 2006).

Figure 8: TREPA Robot Climbing a Tree Trunk

TREPA uses a general repeated four-step process when climbing up a tree, as illustrated in Figure 9, below (Aracil, 2006). First, with the bottom ring gripping the tree and the actuators contracted, the top ring releases its grip. The linear actuators then extend to move the top ring into its new position. Next, the top ring engages its grippers to grab the tree. Once this has

14

gripped, the bottom ring releases its grip on the tree. The linear actuators then contract to raise

the lower ring to a higher position where it engages its grippers to grab the tree. This process is

repeated continuously.

Figure 9: The Four-Cycle Climbing Steps of the TREPA Parallel Climbing Robot

2.1.7 DIGbot

A climbing robot was developed by Eric David Diller at Case Western University with the

intended functionality of being able to climb various surfaces and transition between orthogonal

surfaces. The robot's gripping system used a principle called Distributed Inward Gripping (DIG)

where the rot's hexapod legs grip the climbing surface by exerting pulling forces from opposite

legs in opposite directions to enhance grip and stability. A functional version of the robot can be

seen in Figure 10, below (Diller, 2010)
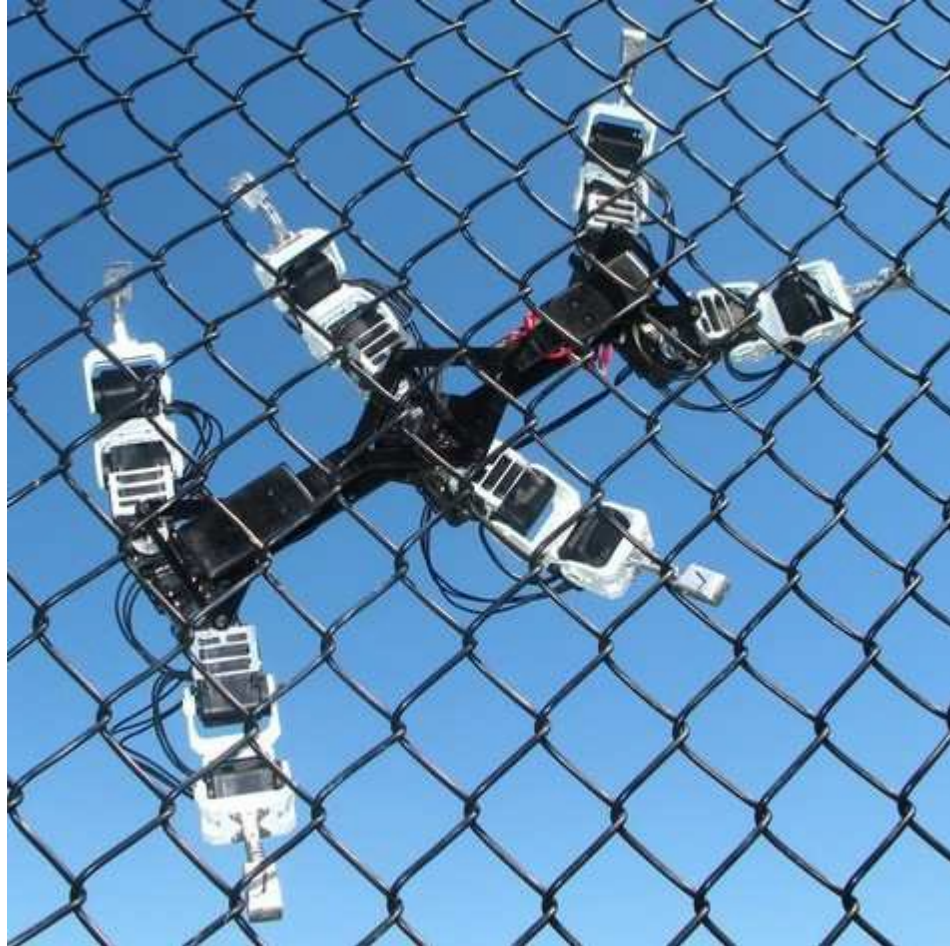
The final design of DIGbot's legs incorporated physical compliance into the design. An exploded view of this leg design can be seen in Figure 11, below. Two different sets of legs were constructed for DIGbot, one with compliant materials and one with rigid materials. Depending on the surface DIGbot was climbing, the legs could be switched out to enable for better adhesion. DIGbot was tested as tree-climber, researchers concluded that, "when climbing on tree bark, a stiff spine is required to allow the spine to penetrate the bark." However, based on the testing of stiff legs, the stiff spine "has never resulted in robust climbing." (Diller, 2010)
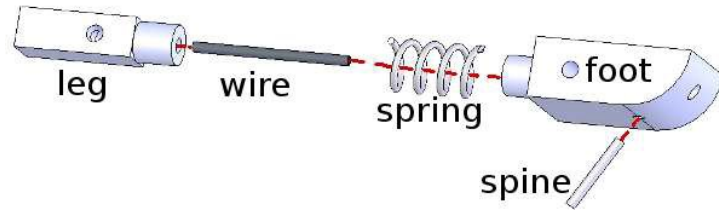
Figure 11: Exploded foot design
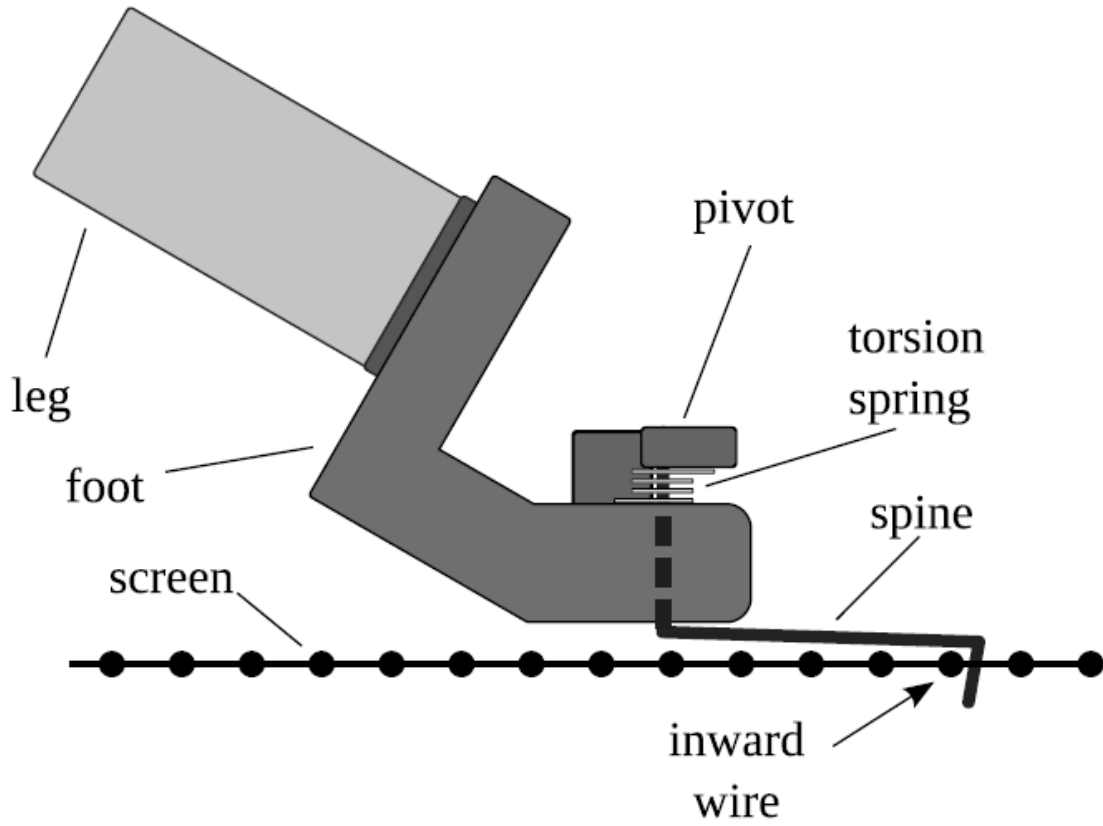
Figure 12: One foot design: single-spine for climbing a screen
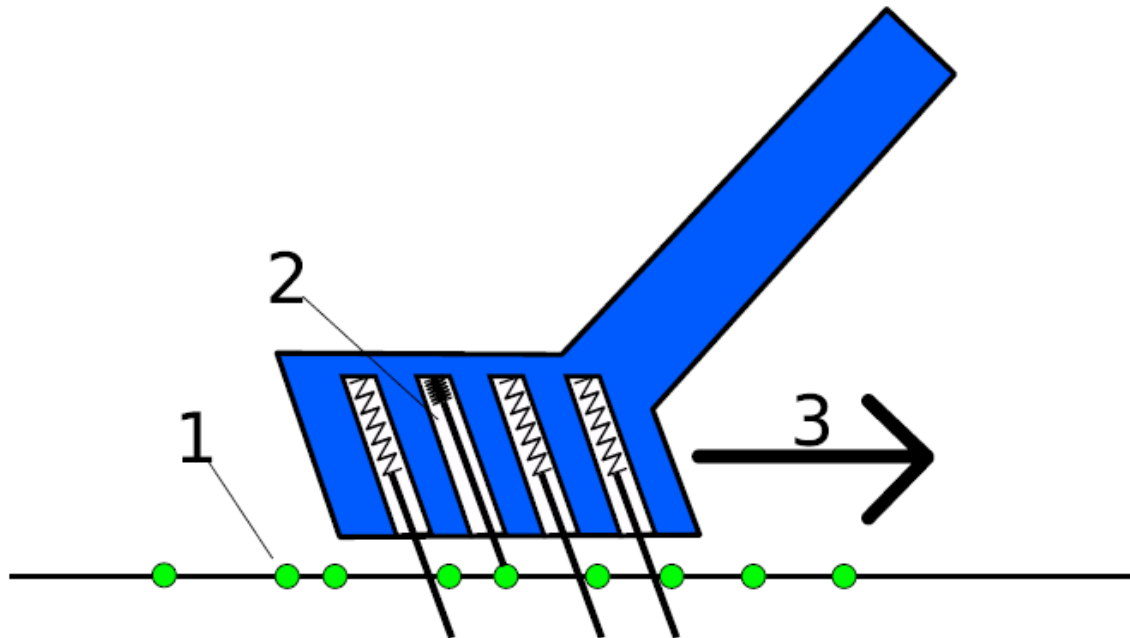
Figure 13: Another foot design: cross sectional view of a foot with multiple spring-loaded, retractable spines that passively adjust to the surface that the foot is in contact with

Figure 14: DIGbot climbing up a tree (left) and a telephone pole (right)

2.1.8 TreeBot

Treebot was developed by Tin Lun Lam and Yangsheng Xu from the University of Hong Kong to assist and/or replace humans in tree related tasks (Lam, 2012). Treebot was designed to be a highly skilled climber capable of traversing the bark and branches of a many tree species. Treebot's design is broken down into three main assemblies, the two tree grippers, the continuum body, and the semi passive joint. Treebot has three active, and two passive, degrees of freedom utilizing a total of five actuators. Two of these actuators are located in the grippers, and one resides in the continuum body. Maneuverability was a top design priority in order for Treebot to be able to traverse irregularly shaped trees, to enable turning, and allow for transitions to different climbing postures. The only way for Treebot to maintain stability was to remain strongly adhered to a tree. Without solid adhesion, Treebot would fall.
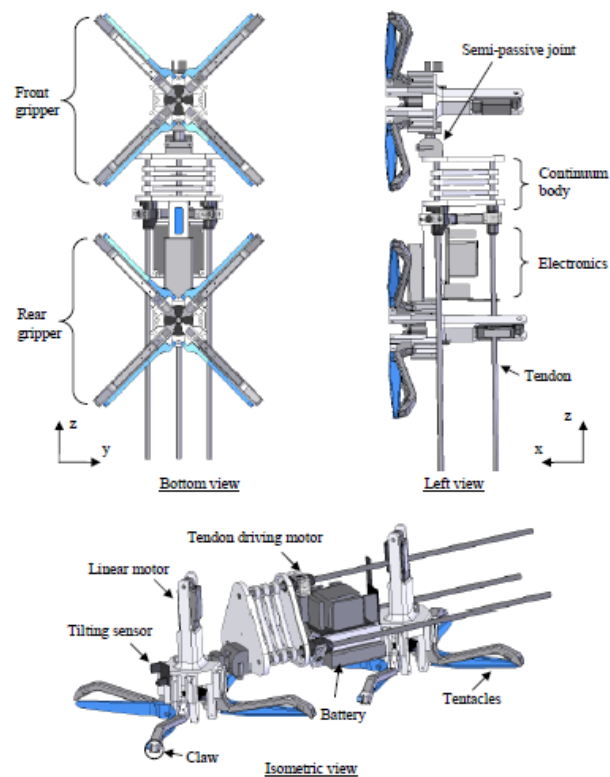
Figure 15: Treebot Overview

The makers of Treebot wanted to facilitate easy transportation by focusing on creating a lightweight and compact design. The designers of Treebot ended up manufacturing a 6.5 gram robot with a payload capacity. This high power to weight ratio allowed Treebot to carry a significant amount of additional equipment to perform a variety of tasks in trees.

The tree grippers are made up of four claws separated by 90 degrees. This allows for Omni-directional gripping, or a relatively similar amount of grip that is not heavily impacted by the orientation of the claw relative to the tree. Each claw is made up of two parts, named phalanx 1 and 2, and is arranged in a two bar linkage configuration. At the tips of each claw are surgical needles that are used for tree surface penetration, and generate surface adhesion when the gripper closes.

As seen in Figure 16, in order for the claw to open, the linear motor presses down on a plate that in turn pushes all four phalanx ones. As a result of phalanx 1 being pressed, phalanx 2 moves up and compresses a spring at joint (A) on phalanx 1. When the gripper is closed, the linear motor releases the plate and the spring force applied to joint A presses phalanx 2 into the tree. By using the spring at joint A to close the gripper, Treebot can maintain adhesion to the tree surface with zero energy expenditures.

The continuum body of Treebot can extend up to ten times its contracted length, and has three degrees of freedom. The continuum body moves similar to an inchworm, but rather than bend its body, Treebot contracts and extends. The continuum body uses three mechanical springs connected in parallel, separated by 120 degrees as a rack, and combines a pinion gear attached to a DC motor to provide bendable movement. Treebot is equipped with a variety of sensors that monitor the position and condition of the robot. Treebot has encoders mounted on each tendon

motor in the continuum body, which are used to measure its extension length. On the claws of each gripper are tactile sensors used to map the tree as Treebot climbs. Mounted on the forward gripper is a triple axis tilt sensor used to measure the direction of gravity relative to Treebot.

Treebot moves up the tree in an inchworm style motion. First, Treebot anchors its rear gripper to the tree and extends its front gripper up the tree. Then the front gripper is engaged and the rear released. The continuum body contracts and raises the rear gripper up before it is reengaged with the tree. Once this process is complete it begins again and continues to move up the tree.
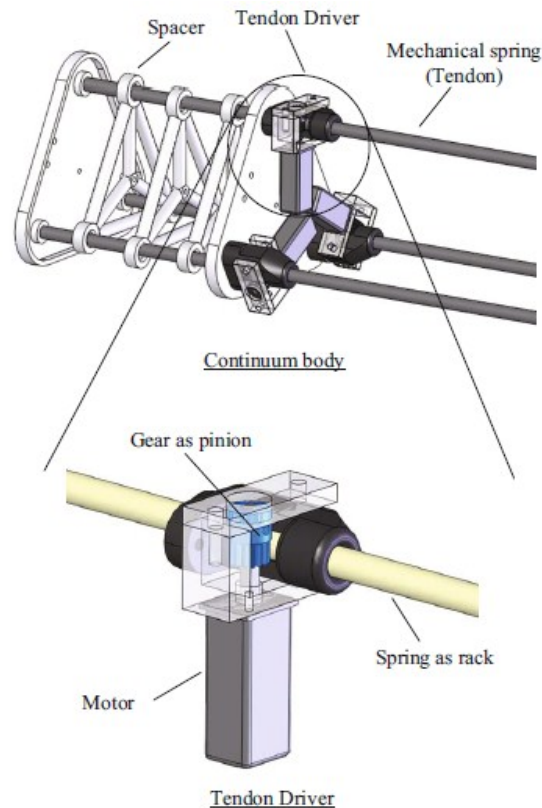
Figure 16: Treebot Continuum Body

3.   Project Strategy

A list of robot chassis ideas which took inspiration from the previously review robot designs was developed. Concurrently, a list of specifications was created from the requirements set forth by the USDA. The following list of specifications provided a way to evaluate the proposed designs and eliminate the least feasible designs.

- Be small and lightweight as to facilitate transportation.

- Be able to transport a camera to the canopy of a tree and back.

- Not damage the tree it is surveying.

- Be able to navigate around branches and other limbs.

- Have a control interface that is intuitive and easy to use.

- Have built in safety features to protect its operators.

Using the list of design specifications, the team created and compared a list of proposed ideas. If a particular design element did failed to meet any specifications it was removed from consideration. The remaining design ideas were examined for their advantages and disadvantages, and were subsequently incorporated into a single preliminary design.

3.1 Potential Robot Designs

3.1.1 Wheeled Design

One of the design possibilities the team considered was to use wheels. Looking back at the background research section, several tree-pruning robots can be seen that utilize wheels. However, these robots are all designed to trim trees and are only capable of climbing straight up the trunks of trees. Due to these undesirable design features, these ideas would not be suitable for the application at hand.

Another potential wheeled robot design could utilize a wheel design similar to the one in Figure 17 below. This wheel designed by the Jet Propulsion Laboratory at the California Institute of Technology, has four micro-spines which are fixed to the wheel by highly elastic red rubbery material. When the wheel rotates during climbing, the red material stretches and allows for the weight of the robot to hang from the spike and pull it into the surface. This design proved quite effective for the Durable Reconnaissance and Observation Platform (DROP) robot.

ROTARY MICROSPINE MECHANISM

Figure 17: Wheel that can climb up walls, courtesy of NASA/JPL-Caltech.

There are several advantages to a wheeled design. Eliminating the legs makes the design much simpler. With a legged robot comes the problem of how to build the legs, which need many degrees of freedom to allow for proper maneuverability on varying tree diameters. Also,

every joint needs to be actuated and a proper gait must also be devised, thereby complicating programming. With a wheeled design, there is no gait. Instead, the driving motors directly rotate the wheels.

Despite the simplicity a wheeled design offers, the team decided against it for a few reasons. First, the design and construction of the wheels could be a project in itself. Not only would the correct materials have to be studied and engineered, but a way to attach the plastic and rubber materials would have to be devised. Also, the micro-spine wheels may need to be delicate and could possibly need to be replaced often. Lastly, a wheeled design would limit the maneuverability of the robot. These wheels are designed to climb straight up, but the USDA specified a robot with the ability to turn and navigate all around the tree. The team considered a four-wheeled mobile robot. In order for a robot of this configuration to turn, the wheels either need the ability to steer or the ability to slip. Since steering the wheels would add more actuation and thus more weight and complexity of programming, it would defeat the purpose of using wheels in the first place. To use four fixed wheels would require the wheels to slip orthogonally to the direction of rotation per the equations derived in the figure below. Since the purpose of these wheels is to dig in and grip sturdily, the idea of allowing for some slippage did not seem fitting.
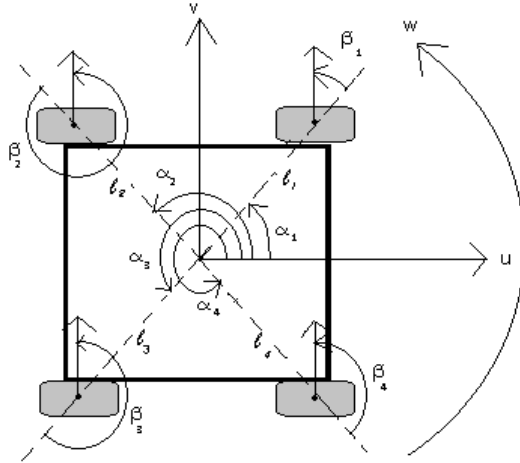
$$\alpha_1 := 45 \text{deg} \qquad \beta_1 := 45 \text{deg} \qquad l_1 := 1$$

$$\alpha_2 := 135 \text{deg} \qquad \beta_2 := 315 \text{deg} \qquad l_2 := l_1$$

$$\alpha_3 := 225 \text{deg} \qquad \beta_3 := 225 \text{deg} \qquad l_3 := l_2$$

$$\alpha_4 := 315 \text{deg} \qquad \beta_4 := 135 \text{deg} \qquad l_4 := l_3$$

Sliding Constraint: $\left(\cos(\alpha + \beta) \quad \sin(\alpha + \beta) \quad l \cdot \sin(\beta) \right) \cdot \begin{pmatrix} u \\ v \\ w \end{pmatrix} = 0$

Wheel 1: $SC_1 := \left(\cos(\alpha_1 + \beta_1) \quad \sin(\alpha_1 + \beta_1) \quad l_1 \cdot \sin(\beta_1) \right) \cdot \begin{pmatrix} u \\ v \\ w \end{pmatrix} = 0$

Wheel 2: $SC_2 := \left(\cos(\alpha_2 + \beta_2) \quad \sin(\alpha_2 + \beta_2) \quad l_2 \cdot \sin(\beta_2) \right) \cdot \begin{pmatrix} u \\ v \\ w \end{pmatrix} = 0$

Wheel 3: $SC_3 := \left(\cos(\alpha_3 + \beta_3) \quad \sin(\alpha_3 + \beta_3) \quad l_3 \cdot \sin(\beta_3) \right) \cdot \begin{pmatrix} u \\ v \\ w \end{pmatrix} = 0$

Wheel 4: $SC_4 := \left(\cos(\alpha_4 + \beta_4) \quad \sin(\alpha_4 + \beta_4) \quad l_4 \cdot \sin(\beta_4) \right) \cdot \begin{pmatrix} u \\ v \\ w \end{pmatrix} = 0$

Combined: $SC_{robot} := \begin{pmatrix} 0 & 1 & \dfrac{\sqrt{2} \cdot 1}{2} \\ 0 & 1 & \dfrac{-\sqrt{2} \cdot 1}{2} \\ 0 & 1 & \dfrac{-\sqrt{2} \cdot 1}{2} \\ 0 & 1 & \dfrac{\sqrt{2} \cdot 1}{2} \end{pmatrix} \begin{pmatrix} u \\ v \\ w \end{pmatrix} = 0$

$$v + \frac{\sqrt{2} \cdot 1 \cdot w}{2} = 0 \qquad v + \frac{-\sqrt{2} \cdot 1 \cdot w}{2} = 0$$

Figure 18: Sliding constraint for a four fixed wheeled robot

3.1.2 Legged Design

The legged design is essentially defined by a robot with a stationary central body and legs on the sides of this central body. Each leg is individually able to hook onto a tree that is being climbed. The robot would climb using a repetitive gait process in which one or more legs is anchored onto the climbing surface while one or more of the other legs are released from the climbing surface to move to a new, higher position, and anchor into the surface. This process can repeat a number of times for each leg's position change depending on how many legs are used on the robot.

The primary advantage of the legged design is its proof of concept, in both natural and man-made instances. Almost all insects and animals that are able to climb utilize this principle (the notable exception being the inchworm, discussed in the following section). Their bodies or torsos ascend at a relatively constant rate while their legs on each side of their bodies execute a gait that involves legs alternatingly gripping, releasing, moving, and re-gripping. This concept has also proven successful in the instance of Boston Dynamics' RISE robots. The proven re-creation of these climbing motions could have been applied to the design of this project's robot with the advantage of being able to directly relate insect and animal climbing patterns to the climbing dynamics of the robot.

Although the legged design has been proven in concept, it is largely unfeasible for the scope of this project for some notable shortcomings. First and foremost, the motions that have to be executed by the legs are very complicated compared to other designs. This directly translates to more complex and precise actuator design, more expensive actuators and mechanisms, and much more complex programming and control design for the gait. The robot is also not as weight-efficient as possible because the heavy central body does not serve a purpose in the gait.

3.1.3 Inchworm Design

The inchworm design is similar to the legged design. However, it differs in the sense that the inchworm robot has no stationary central body. It is essentially one system of legs that all move relative to each other in a gait. The simplest form of this includes two gripping points at each end of the robot along the climbing axis with an actuated, bendable set of connections between the grip points that allow the grippers to move relative to each other. With this design's gait, one gripper releases to move to a new position and re-grip while the other gripper remains attached to the climbing surface to support the robot.

The main advantage of the inchworm design is that it is a simpler mechanism than the legged style described in the previous section. It has many of the benefits of this legged design, while also being a mechanically simpler design. This greatly simplifies the mechanical dynamics and their design along with the corresponding overall cost, actuator programming, and weight. It has also been proven in concept in the case of Treebot. The inchworm design also offers optimal functionality to weight factor because the mechanical system is comprised exclusively of components that contribute to the climbing actions because the entire robot is essentially made out of actuated legs, a notable shortcoming of the central-bodied legged design.

Although the inchworm design's main advantage may be its simpler design, this property can also serve as a disadvantage because of the reduced number of grippers. Such a property reduces the complexity of many aspects of the robot, but also reduces the ability of the robot to maintain its grip in the case of slippage relative to, or detachment from, the climbing surface. If one of the two grippers loses its grip from the tree while the other is detached and in the gait phase of moving to a new gripping position, the robot will have no connection with the tree and will therefore fall off of the tree. Another advantage that also serves as a potential disadvantage

is the overall lightweight construction of the robot. The lighter, less dense design infers less overall toughness and could make the robot more fragile and susceptible to damage should it lose grip from the tree. Although these properties of the inchworm design possess both advantages and disadvantages, the team deduced that the benefits of this design outweigh its detriments and selected this design as the final design foundation for the tree-climbing robot.

3.2   Preliminary Robot Body Design

One of the main design constraints for the proposed robot design was flexibility and maneuverability. In order for the robot to be able to successfully navigate a tree, the chassis of the robot would need to be able to maneuver the gripping mechanisms to reach many local locations in order to maintain adhesion to the tree surface. To achieve this, the robot grabbing mechanisms were designed around a flexible "spine", which would allow the robot to place its grippers in numerous locations, while providing a place to mount the necessary electronics.
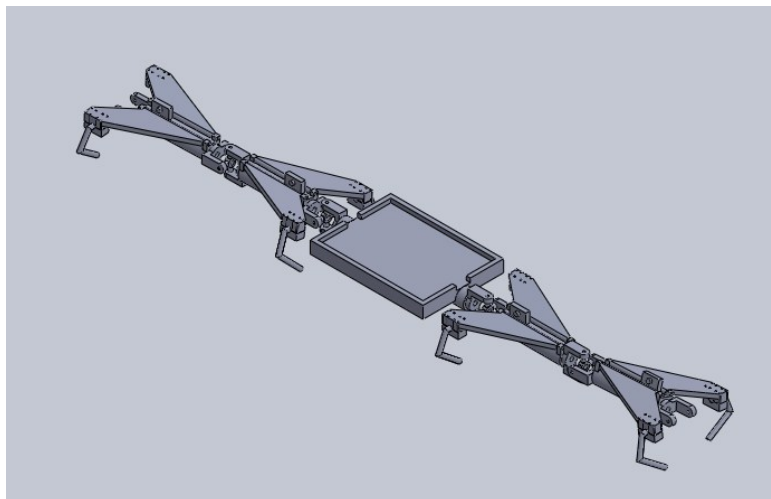
Figure 19: Robot Overview (No cables shown)

The robot is made up of two separate sections: the vertebrae and the central body. In the configuration shown in Figure 19, there are four identical vertebrae. Each vertebra is connected to another and to the central body. This is accomplished via a universal joint that can also rotate inside the vertebra. Between each vertebra is a series of cables arranged in pairs, such as those shown in Figure 20, which would both be connected to the same actuator. As the actuator rotates one way, it would wind up one cable while simultaneously unwinding the other. This allows the vertebra to lift up or go down (away from or towards the tree) based upon the direction of rotation. As seen in Figure 21, this pair of cables would both be both connected to the same actuator.
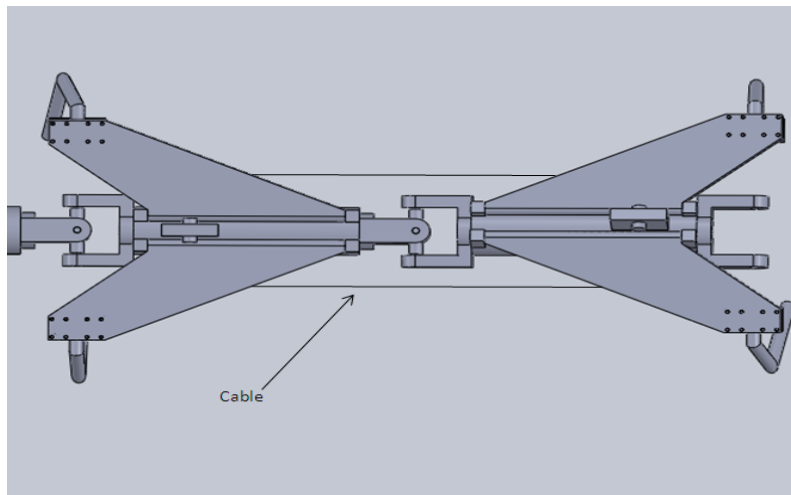
Figure 20: Robot Vertebra-Vertebra Connection (Top View)
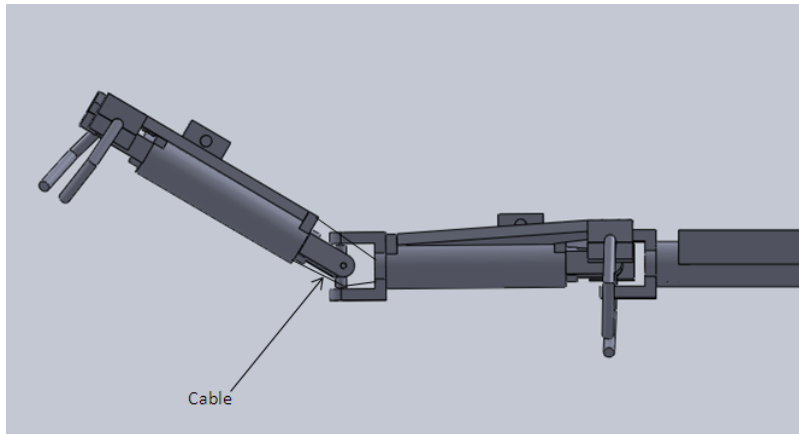
Figure 21: Robot Vertebra-Vertebra Connection (Side View)
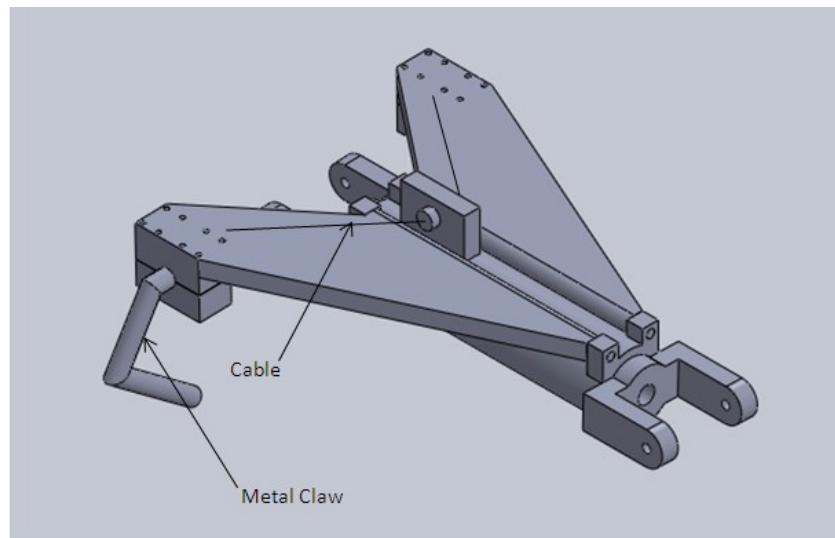
Figure 22: Robot Vertebra-Vertebra Connection (Side View)

The claw is connected to the vertebra via a removable bottom plate that snaps into place. On either side of the claw block is a spring, which can be seen in Figure 23. The purpose of the two springs is to realign the claw when the vertebra is not in contact with the tree. The claw itself would be made of a high strength material with sharpened ends for the purposes of penetrating into the tree and generating an adhesive force.
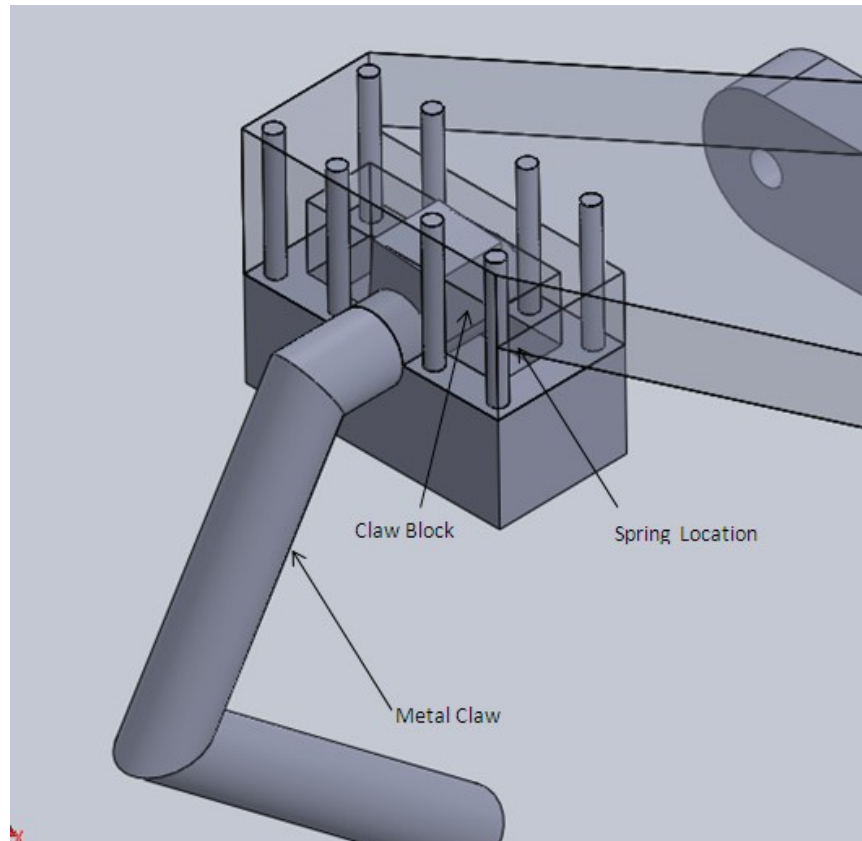
Figure 23: Vertebrae Claw Connection

3D printing could be used to create the complex universal axle connections that are located between the vertebrae and central body, as shown in Figure 24, because this is the only way to create this part and make it one solid piece. The universal axle would need to rotate within this coupling to allow for an additional degree of freedom allowing the vertebrae to better align with the surface of the tree. This motion is not powered but would be limited in order to keep the vertebrae claws in a position that can reach the tree and grab hold. The gripping action, which occurs when the actuator winds and constricts the cables, is shown in Figure 25.
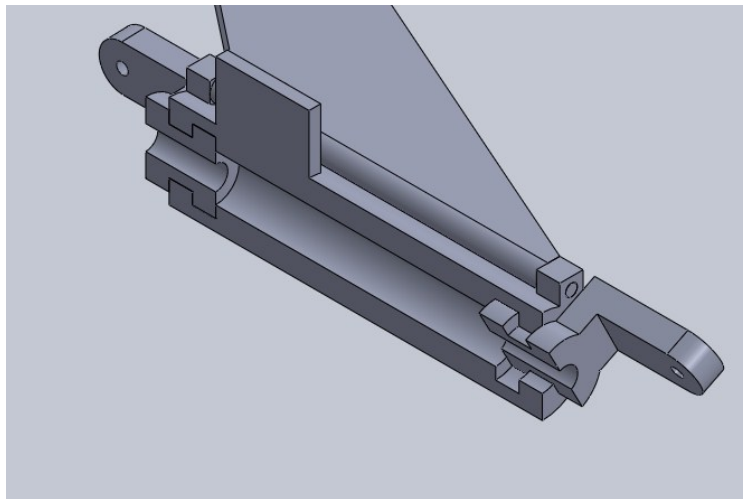
Figure 24: Section View of Vertebrae Universal Joint Connection
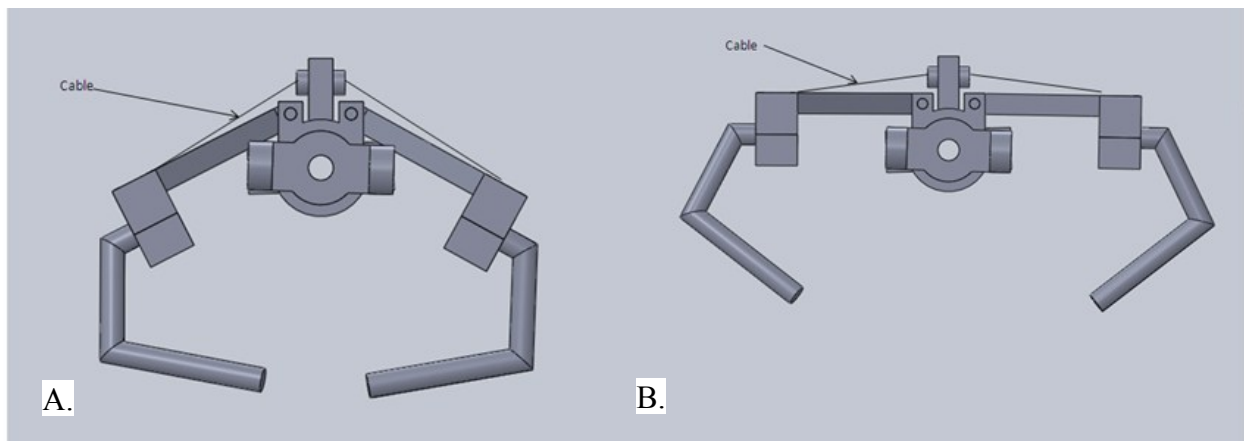
Figure 25: Robot Vertebra (Front view)

3.2.1 Sensors

In order to achieve a closed loop control system, there needs to be sensor feedback from the robot. The first issue to address is whether or not a gripper is correctly placed against the tree before allowing its claws to close and clamp onto the bark. To know when this occurs, there is a push-button installed in the 'palm' area of each gripper. Being simple binary devices consisting of only an on or off state, the buttons are the most efficient method for sensing contact.   When the push-button is depressed, the system knows that the gripper is in place and that it is ok to allow the gripper to close on the tree.

The second set of data desired is the robot's orientation at any given time. To accomplish this, the team thought of attaching accelerometers to each section of the robot. The accelerometers allow constant feedback of what the angle of each section of the robot is, relative about the x, y and z axes, and otherwise known as its pitch, roll and yaw. The accelerometers also provide a secondary desired function by being incorporated into the fail safe mode and triggering a response. If all the values change rapidly at once, the robot has lost grip and is free falling. Figure 26 illustrates potential reference coordinate frames where accelerometers could be placed.
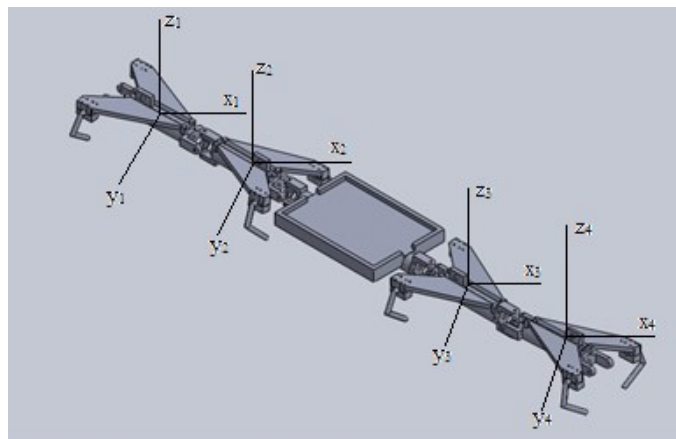
3.2.2 Robot Movement Steps

The robot moves up the tree using a series of steps. To complete one gait there are four separate steps.

Robot Movement Step 1

The first position that the robot would be in would be a straight configuration with all four sets of claws, in their rest position, attached to the tree (Figure 27A)

Robot Movement Step 2

To get to the second position, first the last vertebra (V4) would release the tree. Then, the actuator inside the central body corresponding to the cables connected to the third and fourth universal axles (U3, U4) would rotate, winding and unwinding the corresponding cables to achieve the desired position. Lastly, to get into the second position, the last vertebra would then reengage the tree (Figure 27B).

Robot Movement Step 3

To get to the third position, the second and third vertebra (V2, V3) would release the tree. Then, the actuator inside the central body corresponding to the cables connected to all four of the universal axles would rotate, winding and unwinding the corresponding cables to achieve the desired position. Lastly, to get into position, the second and third vertebra (V2, V3) would then reengage the tree (Figure 27C).

Robot Movement Step 4

To get back to the first configuration, the first vertebra (V1) would release the tree. Then, the actuator inside the central body corresponding to the cables connected to the first and second

universal axles would rotate, winding and unwinding the corresponding cables to achieve the desired position. Lastly, to get into position, the first vertebra (V1) would then reengage that tree (Figure 27D).
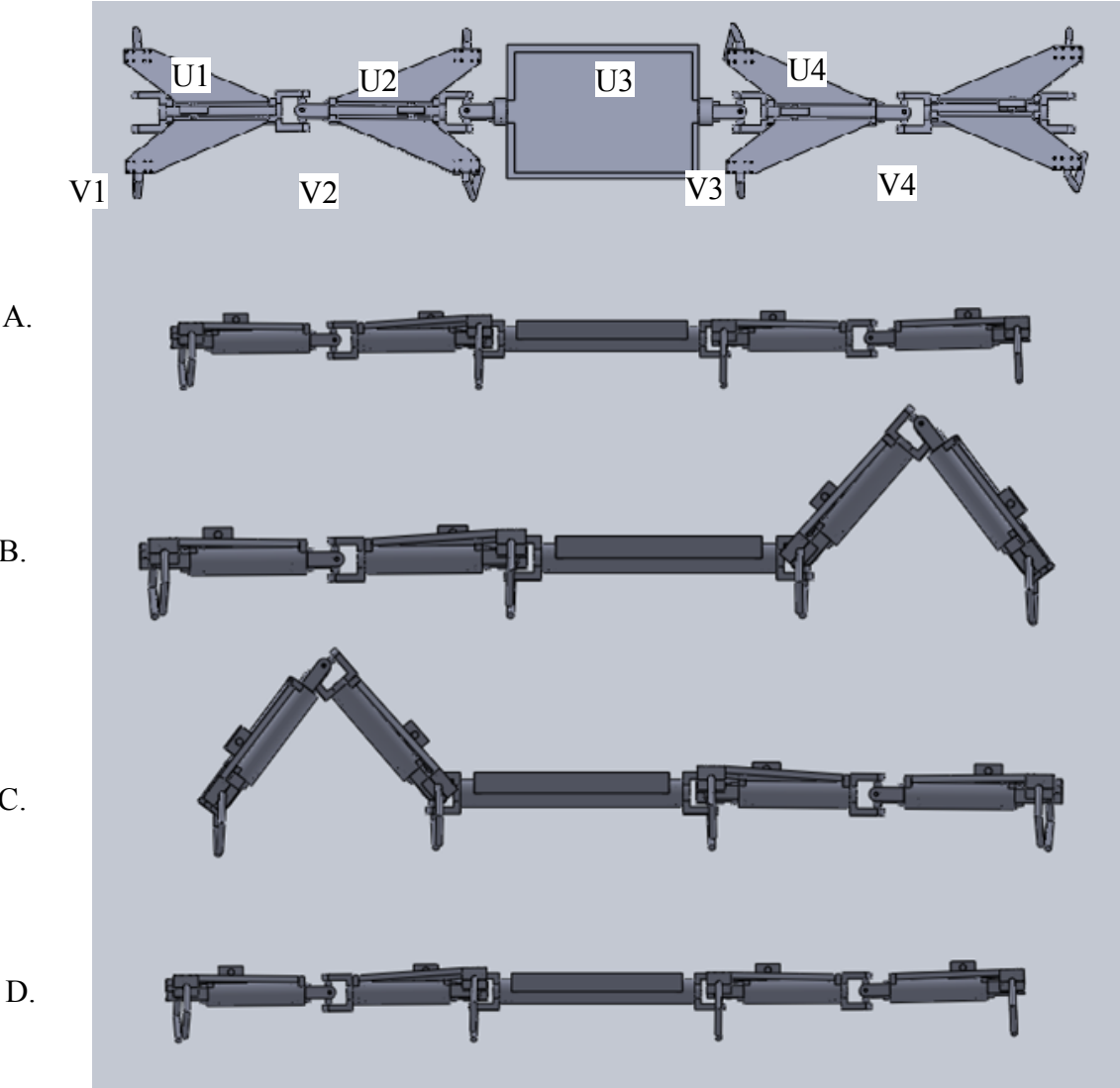
Figure 27: Robot Gait

3.2.3   Design Concerns

After taking a deeper look into the preliminary design, many concerns about the proposed design began to surface. One potential problem was the placement of the cables; the cables could cause the body of the robot to pull in on itself, causing possible jamming issues. One way to overcome this issue would be to reroute the cables to the exterior of the robot chassis. However, repositioning the cables did not eliminate the possibility of having the cables get tangles in branches or other objects during normal operation.

Another concern with the prototype design was the ability of the vertebrae to maneuver the spikes to various locations. The vertebrae design would only allow the grippers to be positioned in a limited range of orientations in order to achieve any gripping force. This design would impair the robots ability to climb irregular shaped trees, and was not satisfactory for our purposes.

After re-evaluating the gait proposed robot gait, the team decided that same gait could be accomplished with only two vertebrae and a central body to house the electronics. If the robot were to be redesigned around this principle, it would be possible to significantly lower the complexity of the robot while reducing the weight. From these decisions the team decided to focus solely on the way in which the robot would grab onto the tree and build the chassis around the gripping mechanism.

3.3   Preliminary Gripper Design

In order to be able to firmly attach to a wide variety of trees, the robot would need a gripper designed to firmly attach to a tree in any orientation. To achieve this, the gripper design was made to be radially symmetrical. Taking design principles from the DIG robot, the proposed

gripper design would utilize opposite inward pulling forces to generate a holding force against the climbing surface. Also, like Treebot, the concept gripper was designed to be able to remain attached to the climbing surface with no external power to prevent the robot from falling if it ran out of power. The gripper was also designed to be small and light to allow for maximum maneuverability. As seen in Figure 28, the housing is roughly 1.25 cubic inches, and the gripper arm span is about 4.5 inches.
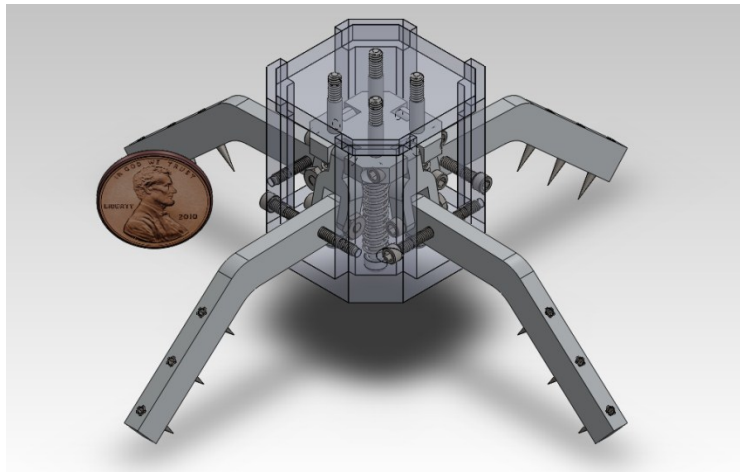
Figure 28: Gripper Design

The gripper is made up of 4 four bar linkages all sharing the same top link (see Figure 29A). With the springs generating a force upward on Link A, the gripper links C are forced into the climbing surface. There are three spikes located on each gripper arm (see Figure 29C), and each spike is threaded, allowing for easy adjustment of the spike length.
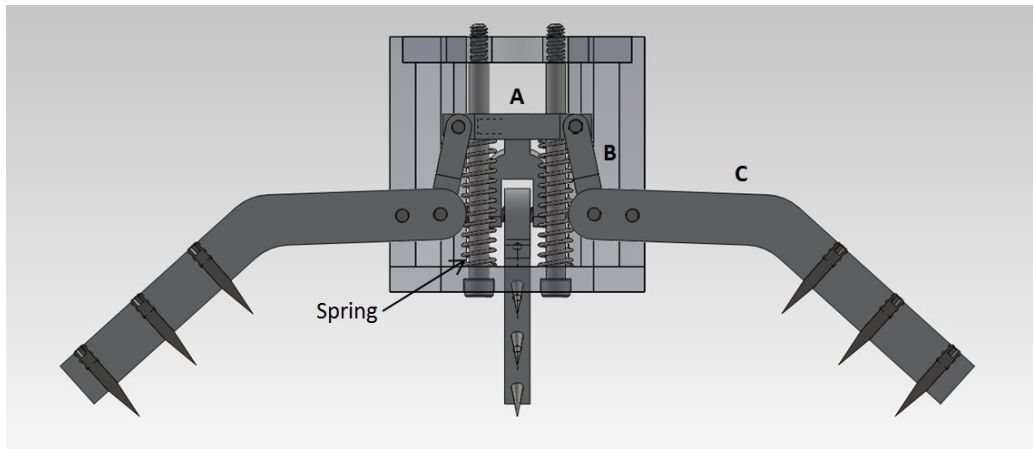
3.3.1 Gripper Construction and Testing

In order to keep on track with the project timeline, the machining of the parts necessary for the gripper and the body was done off site by Robert Symcak and Edward Healey. Together, they have over 40 years of machining experience. In collaboration with Mr. Healey, the Solidworks drawings of each part were verified for machinability, as well as drawing completeness. Also, during the machining process, a number of phone consultations were conducted to make minor adjustments to the design and to verify these changes would not interfere with the function of the gripper when assembled. After receiving the machined parts and hardware for the concept gripper and completing assembly, a variety of tests were designed and subsequently implemented to test the effectiveness of the gripper design.

First Iteration

The initial tests consisted of placing the gripper on the surface of a log and manually driving the gripper's spikes into the bark. The results from this test showed it would be possible for the gripper's spikes to penetrate the surface of the tree and generate a holding force if a great enough force was generated. This test revealed that the amount of force required to generate a significant

38

holding force was approximately 5 to 10lbs. After this test the team found and purchased a pair of linear actuators that could generate 45N or approximately 10.12lbs. Unfortunately, because of the how the testing took place, inaccurate conclusions about the amount of force required for the spike to penetrate the tree were concluded.

Second Iteration

After the linear actuators arrived, a mounting bracket was constructed in order to attach the linear actuator to the gripper. A sample program was written to the Arduino to enable the actuator to open and close, while a bench top power supply supplied power to the actuator. Then, the gripper was held against the tree and the program triggered the linear actuator to close, creating the gripping force needed to press the spikes against the tree. The results of this test showed that the purchased actuators were not strong enough to force the spikes into the tree surface.

Third Iteration

Following the results from the previous iteration of testing, the next set of tests was designed to figure out how much force would actually be required to drive the spikes into the tree. To determine this force, the gripper was modified to accommodate a mounting point to which a spring scale or hanging weights could be attached. A piece of scrap 2x3 pine was clamped to the bench top, and the gripper was clamped to the scrap wood. Then, a spring scale was attached to the mounting point and pulled upwards to simulate the linear actuator force. The spring scale maximum force of 25lb was applied, and was found to be inadequate to penetrate the wood and provide any significant holding force. Then, the gripper was clamped with the gripper upside down to the wood and weights were hung from the constructed mounting point. After various weights of ranging from 30 to 75lbs were hung from the constructed gripper mounting point, the gripper spikes were manually checked for their removal resistance, by feeling the force required

to pull the spikes from the wood. It was found that the weights were hard to stabilize and did not appear to give consistent results; therefore modifications to this testing procedure to improve the accuracy of the test were implemented for the next iteration of testing.

Fourth Iteration

The previous test showed that a spring scale gave more accurate and consistent results then did hanging weights from the gripper. In light of this finding, a digital spring scale with a maximum of 110lbs was purchased for use in this test. Also new for this iteration of testing, a different testing apparatus was constructed which had a higher degree of controllability than the previous design. The new testing apparatus utilized several quick-clamps to hold the gripper to a piece of 1x6 maple, the hardest wood favored by the Asian Long-horned Beetle. The maple was clamped to the bench top and a scrap piece of wood was attached to the shelf hanging over the bench. One side of a quick-clamp was connected to the overhanging piece of scrap wood, while the other side was attached to the spring scale and to the mounting point within the gripper. This way, the quick-clamp handle could be compressed, and the resulting force being applied to the gripper could be read off of the digital scale. This setup can be seen in Figure 30.

Once the setup was constructed, a force of 52.2lbs was applied to the gripper, and the spikes began to slightly penetrate the wood. Next, the location of the gripper was moved to a new location on the wood and a force of 62.5lbs was applied. With this force the spikes began to penetrate but still did not show any measurable resistance to removal.

Finally, a force of 71.98lbs was applied. This force was enough to cause the spikes to penetrate the wood with sufficient depth to warrant a holding force test. Without disturbing the spikes, the clamps were carefully removed from the gripper and a spring scale was attached to the body of the robot. The spring scale was subsequently pulled in a direction parallel to the maple board, in order to simulate the force of gravity when the gripper was on a tree as seen in

Figure 31. From this test the spring scale was able to apply between 5 and 6lbs before the spikes were released their hold from the wood.
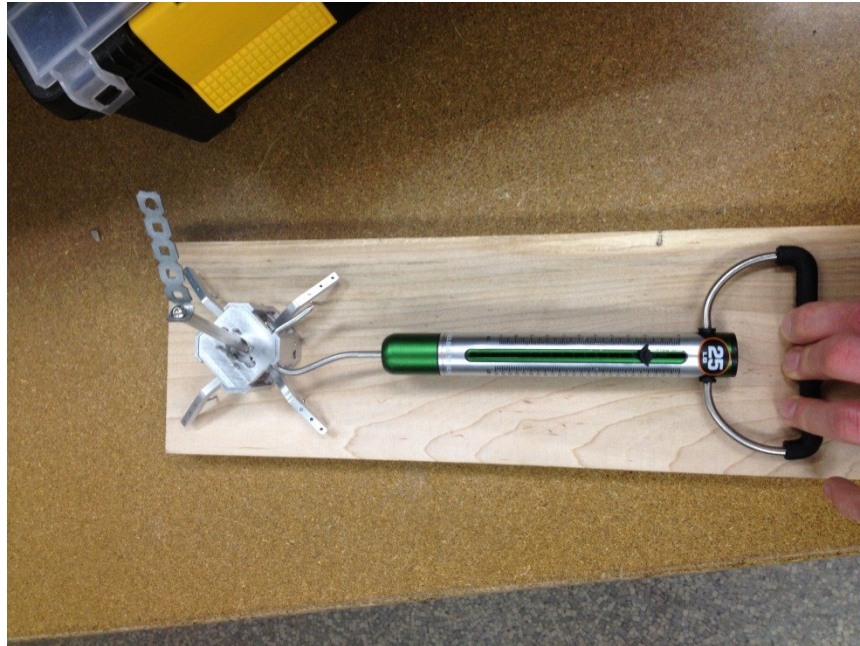
Fifth Iteration

Since the 10lb linear actuators were not capable of generating 70lbs of force with our current configuration, the effects of different spike insertion angles were empirically tested. Three different sets of gripper arms were machined and subsequently tested to see how the varying insertion angle would affect the amount of force required to penetrate the wood. The three different arm sets were shorter than the original arms to generate additional penetrating force at the end of the spike, and had bend angles of plus and minus 15 degrees from the original 140-degree set. The part drawings for the spike angles can be seen in Figure 32.
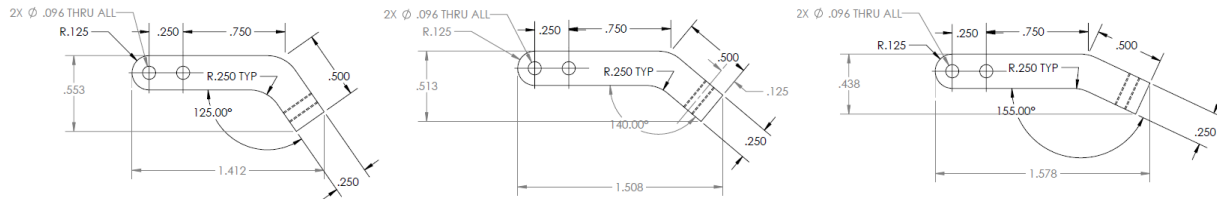
Figure 32: Gripper Arm Angles

The fifth iteration of testing utilized the same set up as the previous iteration of testing, and was carried out to test the new sets of gripper arms. Unfortunately, no conclusive results were drawn on the effectiveness of varying spike insertion angles, as all three angles appeared to drive the spikes into the wood roughly the same distance. Since the spike angles did not have a serious impact on the force required to penetrate the wood, a new linear actuator capable of generating a much greater force was purchased.

After the second set of linear actuators was received, additional testing with the gripper and linear actuator was performed. The gripper was placed on the tree log, held in place, and the actuator was signaled to close. Fortunately, this test showed that the actuator by itself could generate enough force to grip onto the log and hold roughly one pound.

3.3.2 Gripper Spring Force Requirements

The gripper is designed with compatibility for four springs within the mechanism that assist with closing the gripper. The springs lie between the base of the frame of the gripper and the connecting block (the same part that the linear actuator's shaft is connected to). As the linear actuator opens the gripper's claws, the connecting block compresses the springs, thereby linearly increasing the amount of force that each of the four springs applies onto the connecting block. If selected properly, these springs could assist the gripper with digging the spikes into a tree.

The design choice for these four parallel springs is based on the two parameters of the spring constant and the natural length of the spring. The two free variables specified in this situation are the spring constant (k) and natural length (L_o). These two values may be selected based on the desired resultant force for the four identical springs. This analysis yields a value for both the spring constant and the natural length based on a desired resultant spring force. However, both the spring constant and natural length may be independently selected. In order to prevent physical interference within the gripper, these springs would need to have radii between 2.5mm and 4.5mm to allow them to fit simultaneously around the bolts and inside the pre-cut holes.

Assuming that the linear actuator on the gripper is able to exert 40lbs of force, the springs will need to exert a reaction force that is a reasonable amount less than 40lbs to allow the linear actuator to compress the connecting block all the way down to move the gripper into the fully open position. To incorporate a safety factor, the maximum exertion force of the springs was chosen to be 30lbs. This initial condition can then be used to solve for the desired spring constants and unloaded spring lengths for a given F_c. F_c (or F_{closed}) is defined as the net force that all four of the springs collectively exert onto the connecting block when the gripper is in the fully closed configuration. Ideally, there would be some force exerted by the springs in this situation to keep the claw as still as possible in its resting position. Therefore, F_c should be chosen to be at least 5 lbs. or greater. For the analysis of these design parameters, some variables must be defined.

L: the general length between the base of the frame of the gripper and the connecting block. This represents the length of the spring in any given situation because that is the exact length of the space that the spring has to occupy.

L_{min} = 0.567 inches: minimum length of the spring that is seen in any situation. This occurs when the gripper is fully open.

L_{max} = 0.803 inches: maximum length of the spring that is seen in any situation. This occurs when the gripper is fully closed (measured on gripper)

L_o: represents the natural length of the spring (design choice)

x_{open}: the displacement of the spring relative to L_o when the gripper is in the open position

x_{closed}: the displacement of the spring relative to L_o when the gripper is in the closed position

k: the spring constant of one of the four springs within the gripper (design choice)

     First, the overall spring constant of this system needs to be related to the spring constant of one of the four springs comprising the system. Following the rules of springs acting in parallel, the overall spring constant of this system would be all of the spring constants added together with the value of "4k" used to describe the overall spring constant. This value of "4k" can then be used to describe the overall force of this spring system where x is the displacement of the springs from their would-be unloaded length.

$$F_{springs} = (4k)x$$

The displacements x_{open} and x_{closed} need to be calculated to properly relate to the spring force equation. These calculations are as follows:

$$L_o = L_{max} + x_{closed} \qquad L_o = L_{min} + x_{open}$$

$$x_{closed} = L_o - L_{max} \qquad x_{open} = L_o - L_{min}$$

The spring force equation can now be solved for each of the two states of the gripper when it is either fully closed or completely open. When the gripper is closed, the combined force of the springs is equal to the free-choice variable F_c, which is the independent variable that L_o and k are plotted against for this analysis. The maximum force that the springs can output under any circumstance is 30lbs. This combined spring force occurs when the gripper is completely open.

$$F_c = (4k)(x_{closed}) \qquad 30\ lbs = (4k)(x_{open})$$

$$F_c = (4k)(L_o - L_{max}) \qquad 30\ lbs = (4k)(L_o - L_{min})$$

Solving each of these equations for "k", setting these resultant equations equal to each other, and solving for L_o yields:

$$L_o = \frac{F_c * L_{min} - 30 * L_{max}}{F_c - 60}$$

Similarly, solving each of these equations for "L_o", setting the resultant equations equal to each other, and solving for "k" yields:

$$k = \frac{30 - F_c}{4(L_{max} - L_{min})}$$

Each of these equations was enumerated in Mathcad and the resulting equations were plotted against F_c. The results of this analysis are shown below in Figure 33. The final plot of these relationships is shown in Figure 34.

$F_c := 0lb, 1lb .. 30lb$

$L_{min} := 0.567in$

$L_{max} := 0.803in$

$$L_o(F_c) := \frac{\left(F_c \cdot L_{min} - 30lb \cdot L_{max}\right)}{\left(F_c - 60lb\right)}$$

$$k(F_c) := \frac{\left(\left(30lb - F_c\right)\right)}{4 \cdot \left(L_{max} - L_{min}\right)}$$

$L_o(F_c) =$ $L_o(10lb) = 0.368\, in$ $k(F_c) =$ $k(10lb) = 21.186\,\dfrac{lb}{in}$

$L_o(F_c)$		$k(F_c)$	
0.402	in	31.78	lb/in
0.399		30.72	
0.396		29.661	
0.393		28.602	
0.39		27.542	
0.386	5	26.483	5
0.383		25.424	
0.38		24.364	
0.376		23.305	
0.372		22.246	
0.368	10	21.186	10
0.364		20.127	
0.36		19.068	
0.356		18.008	
0.351		16.949	
0.346	15	15.89	15
0.341		14.831	
0.336		13.771	
0.331		12.712	
0.325		11.653	
0.319		10.593	
0.312		9.534	
0.306		8.475	
0.299		7.415	
0.291		6.356	
0.283		5.297	
0.275		4.237	
0.266		3.178	
0.257		2.119	
0.247		1.059	
0.236		0	

Figure 33: Mathcad calculations of natural spring length (Lo) and spring constant (k)
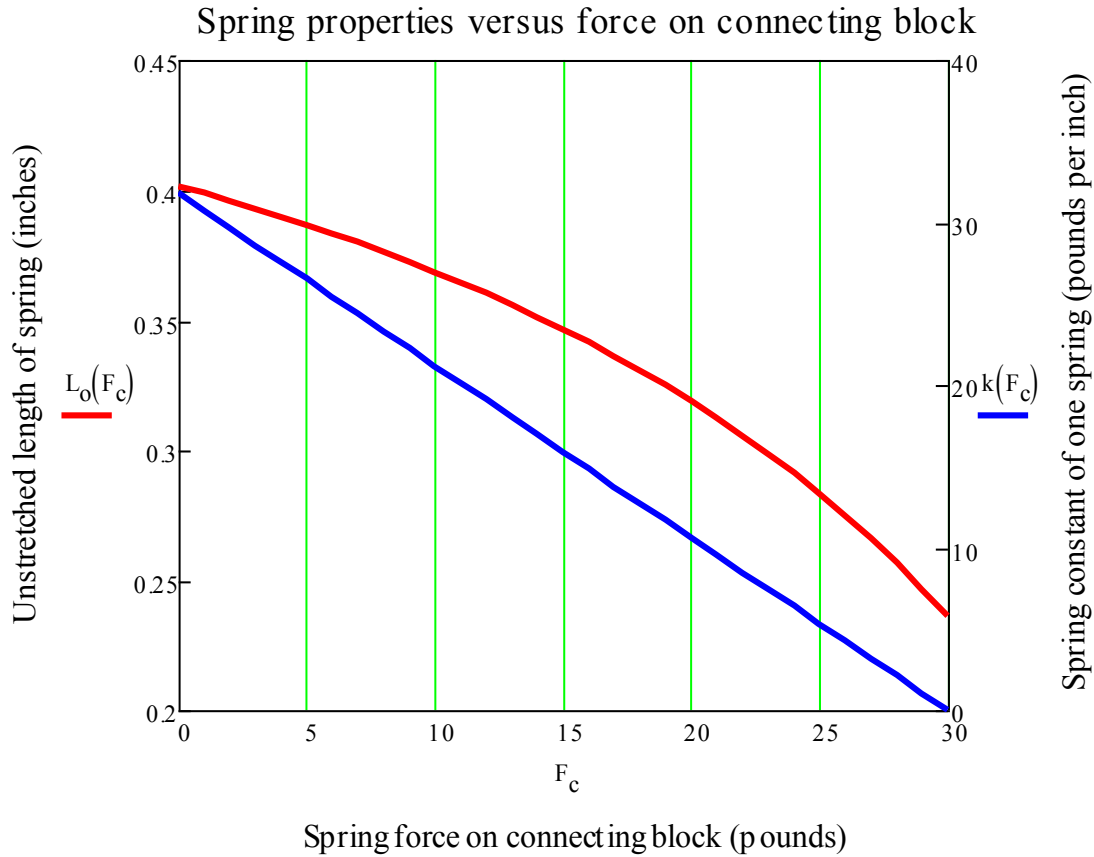
Figure 34: Plots of two spring parameters versus the desired spring force in the gripper's closed position

As a result of this spring analysis, the possibility of being able to develop more than 70lbs of force at the connecting block was realized. The 70lb force required to penetrate a flat maple board is significantly greater than the force it would take to penetrate the bark of a tree. Using the linear actuator that was sourced which was capable of developing 40lbs, it may not be possible to penetrate the tree bark. However, the actuator could be supplemented by springs to generate in excess of 60lbs if necessary. Our calculations and testing proved that the concept gripper was effective in its ability to adhere to a tree. Following the success of the gripper design, the process of designing a body that was capable of utilizing the gripper was started.

3.4   TCR12 Body Design

Taking from the preliminary design and the gripper design, a new design was constructed that would provide the flexibility and degrees of freedom that are needed to scale the uneven surface of a tree. This new design is shown in Figure 35. The overall length of the TCR12 is roughly 14 inches, with an approximate width of 5 inches and a height of 6 inches. The overall weight of the robot has been calculated to be just over 2 pounds. With its compact dimensions and light weight the TCR12 easily fulfills the requirements of being light and portable. The robot consists of two grippers connected via a ladder frame, which is connected to a center spherical wrist; in total the entire robot has 5 degrees of freedom with respect of one gripper relative to the other.
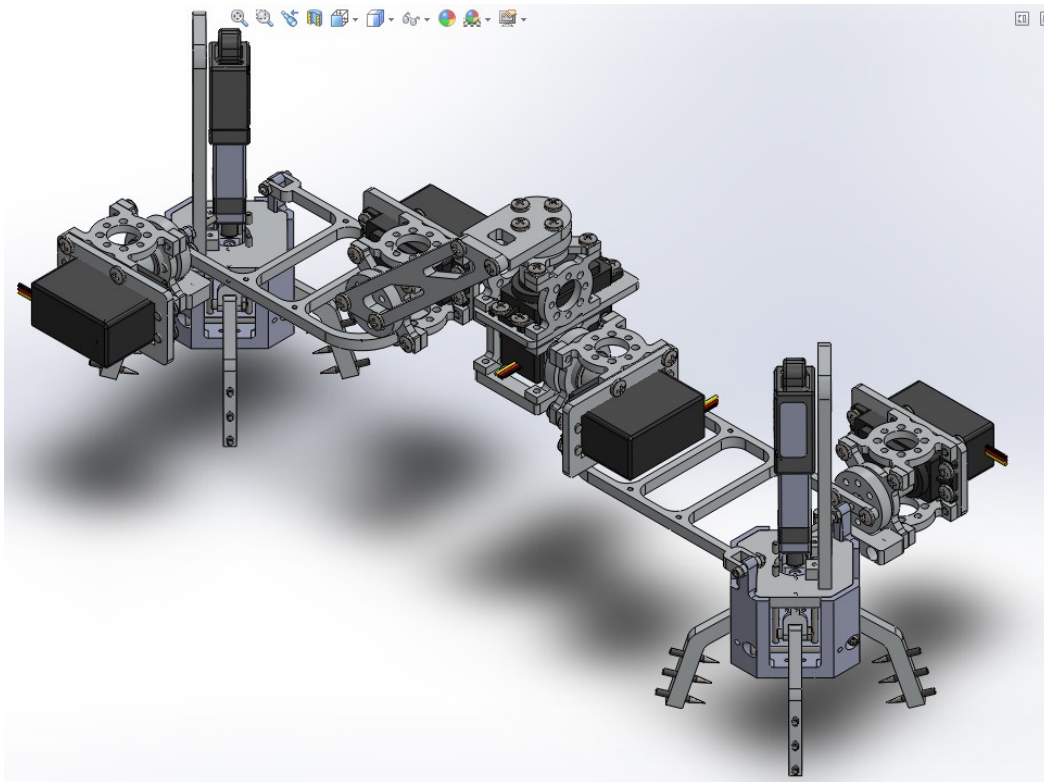
Figure 35: Tree-Climbing-Robot-2012 SolidWorks Rendering

The robot is a modular design consisting of two different subassemblies, the gripper frame assembly, and the spherical wrist assembly, as shown in Figure 36. The robot is made up

of two copies of the gripper frame assembly and one spherical wrist assembly. By creating a

design that utilizes repeating subassemblies it was possible to minimize the number of unique

parts, and subsequently ease the manufacturing effort. With this design, only 13 parts needed to

be machined, allowing for a low production line cost.
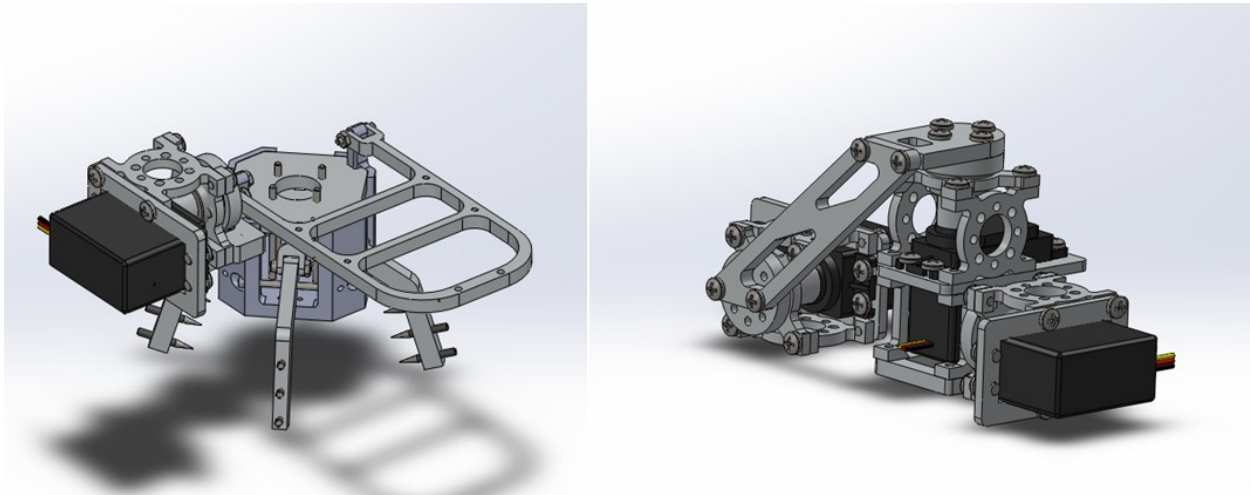
Figure 36: Gripper Frame Assembly Solidworks Rendering Left/ Spherical Wrist Assembly Solidworks Rendering Right

TCR12 is equipped with multiple sensors that allow it to observe its surroundings,

monitor its own health, and warn people in the event of a gripping failure. On the palm of each

gripper is a series of 5 push button sensors as seen in Figure 37.
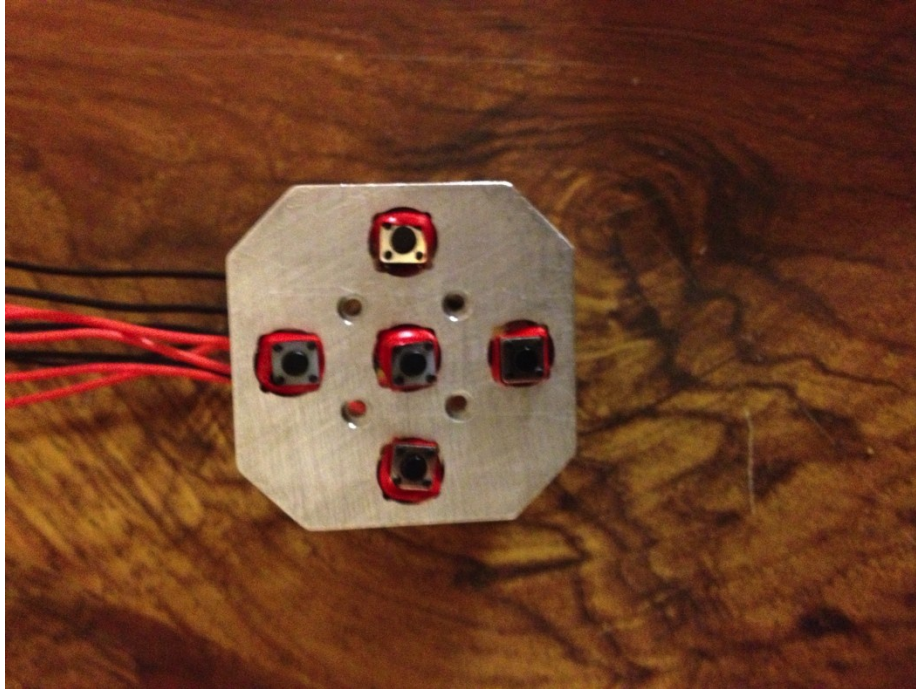
Figure 37: Gripper Palm Push-buttons

The push buttons allow the robot to know when it has made full contact with the climbing surface. By having multiple pushbuttons on the palm, the robot not only knows when it has come into contact with the climbing surface, but also the orientation relative to the climbing surface. If the linear actuator begins to draw too much current it will trip the current sensor, the robot can take preventative actions to avoid damaging the linear actuator by opening its gripper, repositioning it, and trying again. Also, a safety system was designed to warn people in the event of a grip failure. Accelerometers could be added to allow the robot to know when it is falling, close the grippers to prevent it from digging into a person if it fell on them, and activate a warning sound to notify people that they need to clear the area.

3.5   TCR12 Gait

TCR12's climbing motion is a series of steps that create the robot's gait. First, TCR12 is placed on the tree's surface, and after the pushbuttons on the gripper's palms sense that they are in

contact with the tree, both grippers close (Figure 38.1). The next step is for the bottom gripper to release from the tree (Figure 38.2). Next, the servos on each gripper and the spherical wrist move to pre-determined set points, creating a V-shape with the open side against the tree surface (Figure 38.3). Then, the bottom gripper clamps down onto the tree surface (Figure 38.4), followed by the upper gripper releasing from the tree (Figure 38.5). After that, the two servos and spherical wrist move back to the straight out configuration (Figure 38.6). The last step of each gait is to reengage the upper gripper with the tree returning to the first step (Figure 38.1). This process is repeated as the robot moves up the tree, and the only variation comes when the robot needs to turn left or right, or if the upper gripper needs to move in or out compared to the lower gripper. By giving the upper gripper 5 degrees of freedom with respect to the lower gripper, the upper gripper can be placed in a wide variety of places to traverse even the most complicated tree surface.
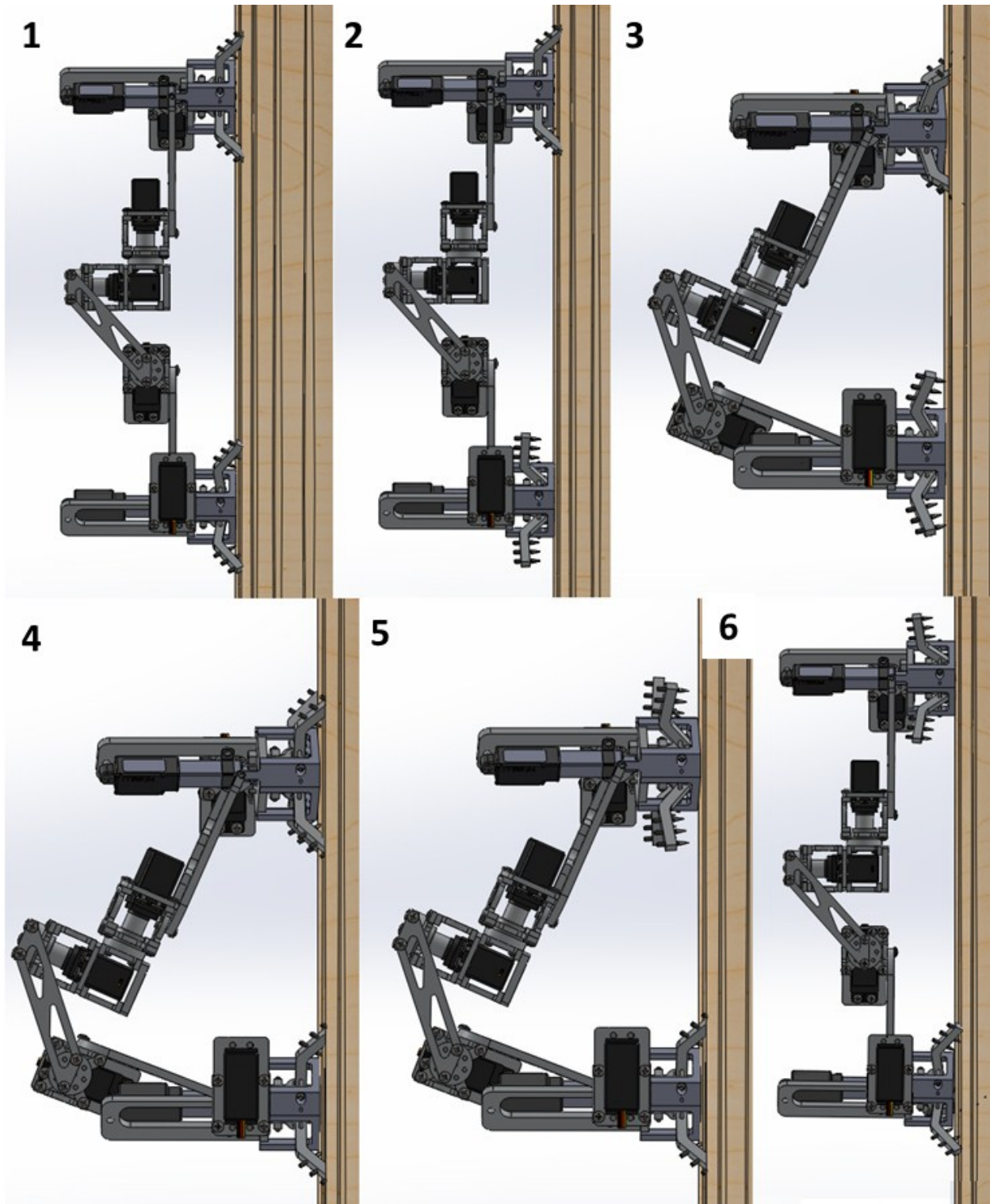
Figure 38: TCR12 State Diagram

3.6   Robot Design Analysis

After the preliminary design was finalized, the next effort was to examine and evaluate the robot's ability to meet our established goals. Various analytical methods were used to determine the required specifications for servomotors, find the amount of force developed at the gripper spikes in various situations, and solve for the factor of safety when utilizing specific components. These analytical methodologies and their respective results are described in the following sections.

3.6.1   Maximum Torque Analysis

To allow the robot to operate properly it was necessary to determine the greatest torque that any servo would ever have to exert to enable the robot to move in a situation. To solve for the worst-case scenario when the robot was lifting itself in a vertical direction, multiple free body diagrams were created and analyses were performed.

In the horizontal worst-case scenario, only one of the robot's grippers would be adhered to the tree while the body would be fully outstretched. The free body diagram in Figure 39 illustrates this worst-case scenario. In this diagram, W_c and W_b represent simplified centers of mass while L_c and L_b represent simplified lengths of robot components. Equations of equilibrium were then established and solved to determine the angle α at which the greatest possible moment occurs about point Z.
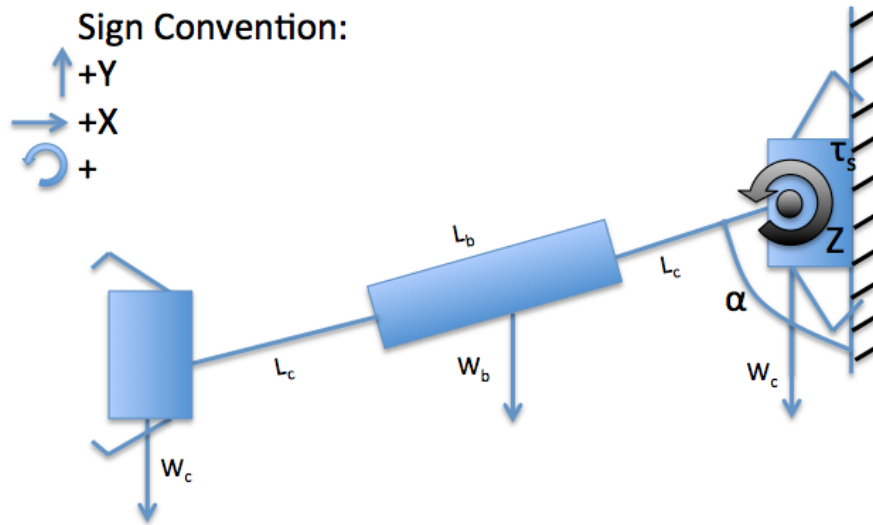
Figure 39: Using Generalized Free Body Diagram to Determine The Max Torque Situation

Solving for required τ_s to maintain equilibrium.

$\Sigma MZ = \tau_s - W_b(L_c + .5L_b)\sin(\alpha) - W_c(2L_c + L_b)\sin(\alpha)$

$\tau_s = -W_b(L_c + .5L_b)\sin(\alpha) - W_c(2L_c + L_b)\sin(\alpha)$

In order to find the maximum torque, τ_s, required to keep this system at equilibrium, it was necessary to select an angle α that will maximize the value of the sine function and create the greatest moment about point Z. When $\alpha=90°$, the sine function outputs it's highest possible value, 1, and τ_s has to balance out the greatest possible moment exerted by the weight of the body. To solve for τ_s, a weight distribution diagram, Figure 40, was constructed to determine how much torque a servo located at point Z would have to generate in order to maintain equilibrium.

55

3.6.2 TCR Weight Distribution

The precise locations of the centers of mass of each modeled robot component each of their respective weights were obtained through SolidWorks. The remaining component weights were obtained from their respective manufactures and their centers of mass were estimated. The diagram in Figure 40 was constructed to provide an estimate of the torque that would have to be generated at the outermost servo in order to lift the body of the robot straight out if it were only gripping from one end.
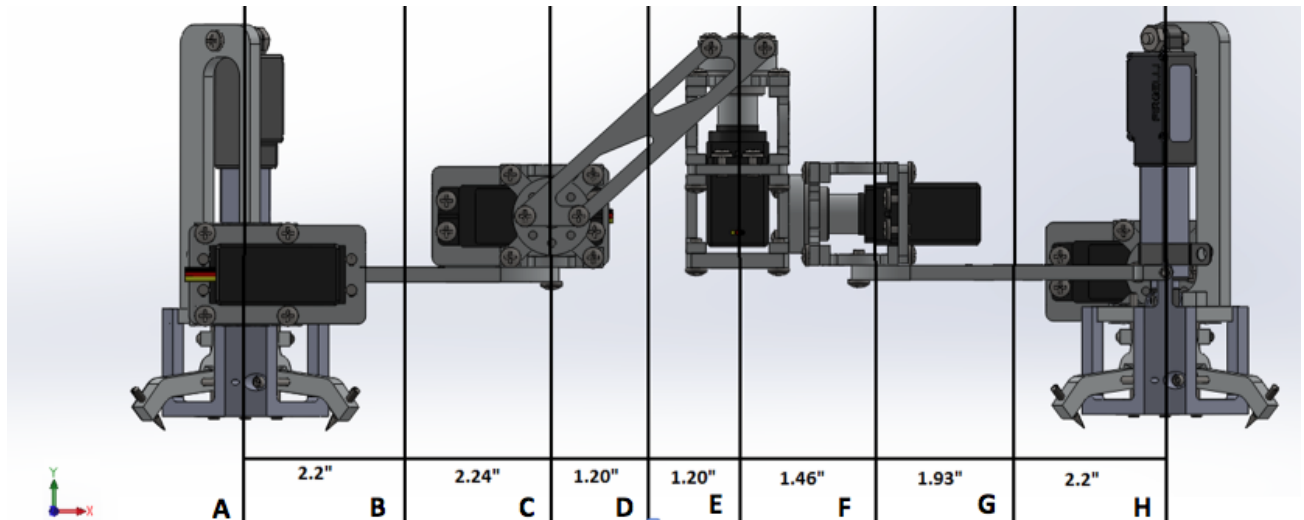
Figure 40: Determining the Weight Distribution Diagram

A = gripper assembly (236g) + linear actuator circuit (20g) + servo with servo block (102g) = 358g

B = ladder guide (18g)

C = servo with servo block (102g)

D = servo block guide (16g) + bread board (35g) = 51g

E = central spherical wrist servo assembly (122g)

F = servo with servo block (102g)

G = ladder guide (18g) + Arduino (143g) = 161g

H = gripper assembly (236g) + linear actuator circuit (20g) + servo with servo block (102g) = 358g

Total weight = 1272 grams=44.87 ounces=2.80 pounds

Required torque = 0 *A +2.2" *B + 4.44"*C + 5.64"*D + 6.84"*E + 8.3"*F + 10.23"*G + 12.43"*H

Required torque = 0 + 1.39 + 15.98 + 10.15 + 29.41 + 29.88 +58.11 + 156.99 = 301.91 oz-in (1.57 ft-lb)

As shown in the calculations above, it was estimated that the outermost servos would need to exert a force of 1.57 ft-lb to lift the weight of the body. After computing and rechecking this calculation, servos capable of producing 2.68 ft-lb of torque were found online and were considered for use in the body of the robot. If a servo capable of producing 2.68ft-lb were used in the claw, it would be using 59% of its maximum torque output to hold the body horizontal.

After the greatest possible moment required for movement in the vertical direction was calculated, it was necessary to check and solve for the maximum moment necessary to allow the robot to pivot left and right. In Figure 41 below, servo B rotates while the servo at A holds the

robot body horizontal. In Figure 42, the centers of mass of the respective robot components located at C, B, E, and F change locations are shown. After the rotation these masses move to their new locations at C', B', E', F' after the rotation at B.
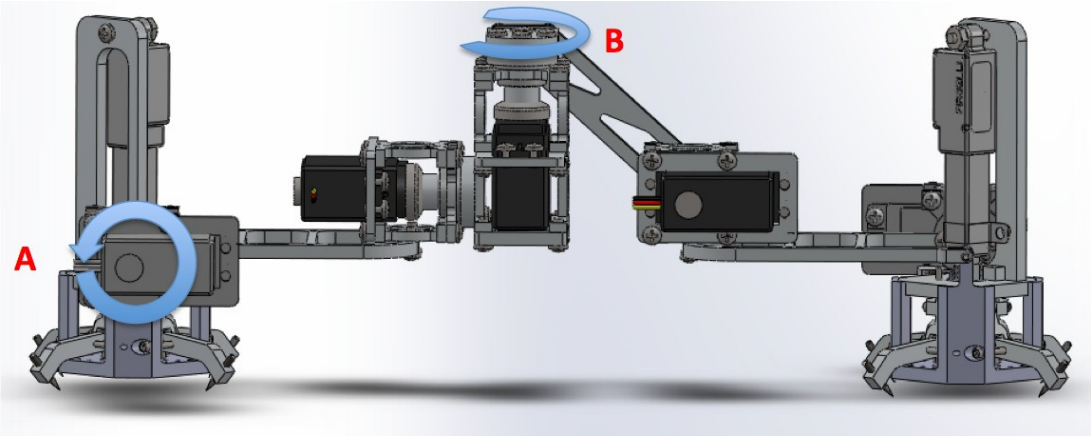
Figure 41: Determining Max Torque (Side View)
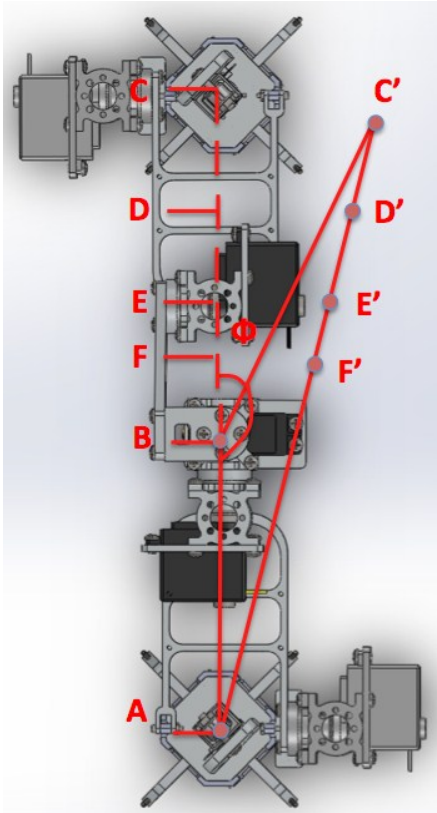
Figure 42: Determining Moment Arm Length AC' After a Rotation of Φ° (Top View)

Point	Weight (lb)	Distance from Point B to Center of Mass (inches) when Φ=0°
C	0.789	6.840
D	0.040	4.640
E	0.225	2.400
F	0.112	1.200

Table 1: Weights and Distances

As the servo at point B rotates clockwise, the resultant moment required to hold up end C, which is developed at servo A, changes in accordance with angle Φ. While servo B rotates, the mass at point C moves to point C', as do the other respective centers of mass shown in Table 1. The lengths BC and AC do not change during the rotation about B. However, the moment arm from point A to point C' changes, indicating a change in the amount of force required to keep the body at vertical equilibrium during the rotation. These assumptions were made when solving for moment arm AC:

- Lengths AB and BC are known
- Angle ABC' is 180°-Φ°

To solve for length AC' the following equation is used:

$$AC' = \sqrt{AC^2 + BC^2 - 2(AC)(BC)\cos(180 - \phi)}$$

$$AC' = \sqrt{5.59^2 + 6.84^2 - 2(5.59)(6.84)\cos(180 - 60)}$$

$$AC' = 10.8"$$

The moment arms for all of the centers of mass of each of the robot's components were calculated using the formula above and substituting in the appropriate value of BC for each component, which is the distance from each respective center of mass to the rotation point B. After all calculated moment arms were solved for, they were multiplied by their respective

weight to determine the required moment at servo A to maintain equilibrium the result of these calculations is in Table 2. Subsequently, the moment generated by the rest of the robot was added to the resultant force, everything was converted to foot-pounds, and the resulting graph is shown in Figure 43.

Rotation Φ° at Point B	0	10	20	30	40
Distance BC (inches)	12.430	12.382	12.242	12.009	11.686
Distance BD (inches	10.230	10.191	10.075	9.883	9.616
Distance BE (inches)	7.990	7.964	7.887	7.761	7.586
Distance BF (inches)	6.790	6.775	6.730	6.656	6.554
Resultant force (in-lb)	12.777	12.730	12.592	12.364	12.047

Table 2: Resultant Forces after Rotations
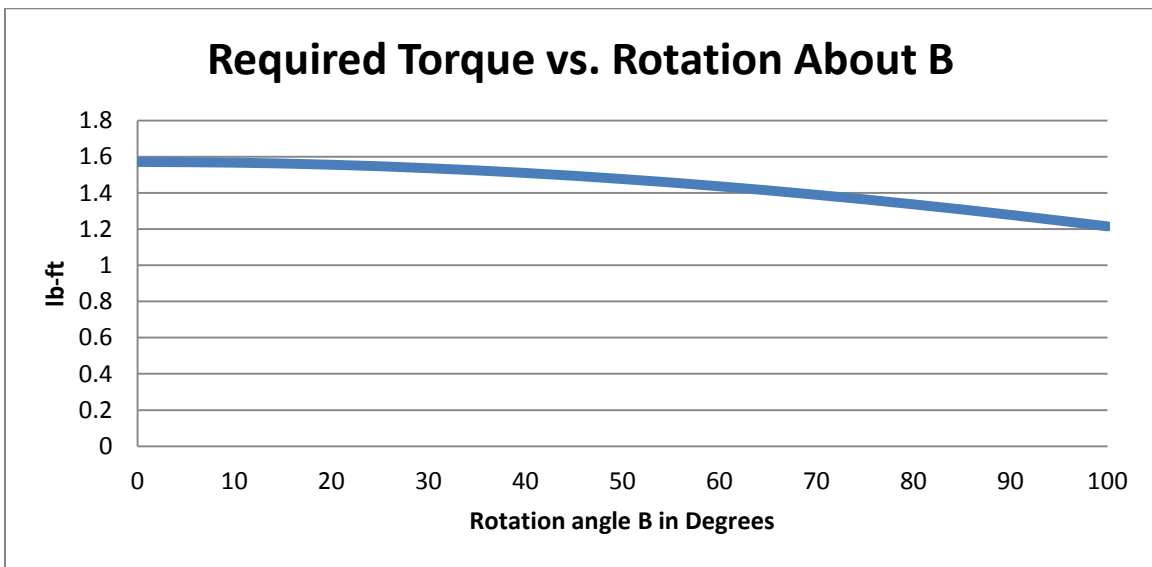
Figure 43: Required Torque vs. Rotation Graph

By the two using the two analytical methods detailed above, it was determined that the maximum required torque output of any servo during any body positioning movement is 1.57 ft-lb. This situation is encountered when a servo at one end of the robot is holding the opposing end of the body out parallel with the horizontal plane. Next, it was necessary to determine what forces the gripper could exert on the body of the TCR while in operation.

3.6.3 Gripper Force Analysis

In order to determine the force that would be generated at the spike for any given spike insertion angle, θ, force equations were derived using geometric relations measured in SolidWorks. The following images, taken from the SolidWorks CAD models that the team created, show the geometric relations between the forces developed at various points and the effects of various component geometries of the robot claw components.
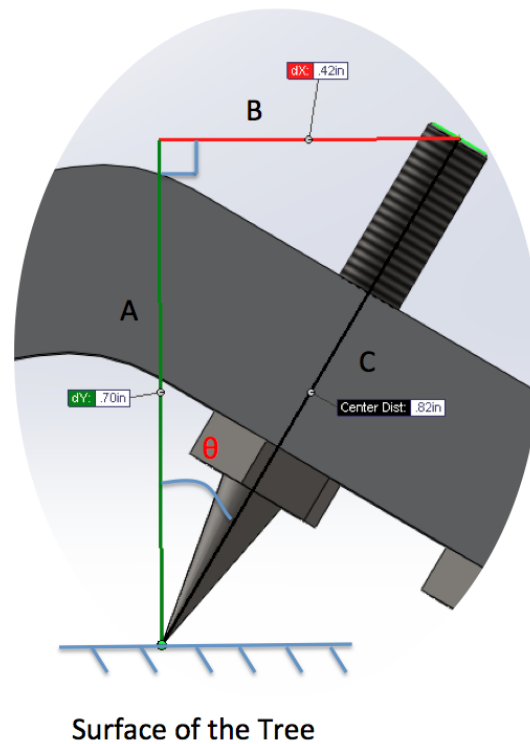
Figure 44: Determining Spike Angle

Solving for spike insertion angle θ:

$$\frac{B}{\sin(\theta)} = \frac{C}{\sin(90)}$$

$$\theta = sin^{-1}(\frac{B}{C}\sin(90))$$

$$\theta = 30°$$
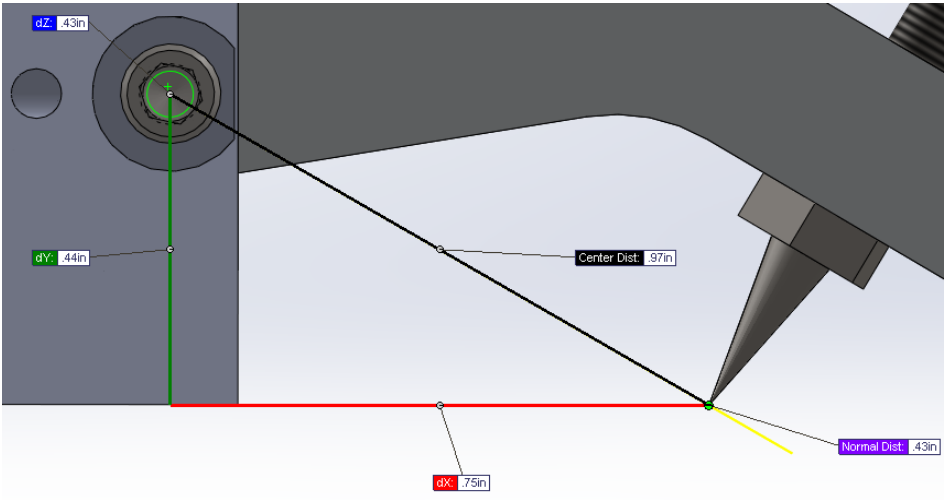
Figure 45: Determining Spike Moment Arm 1
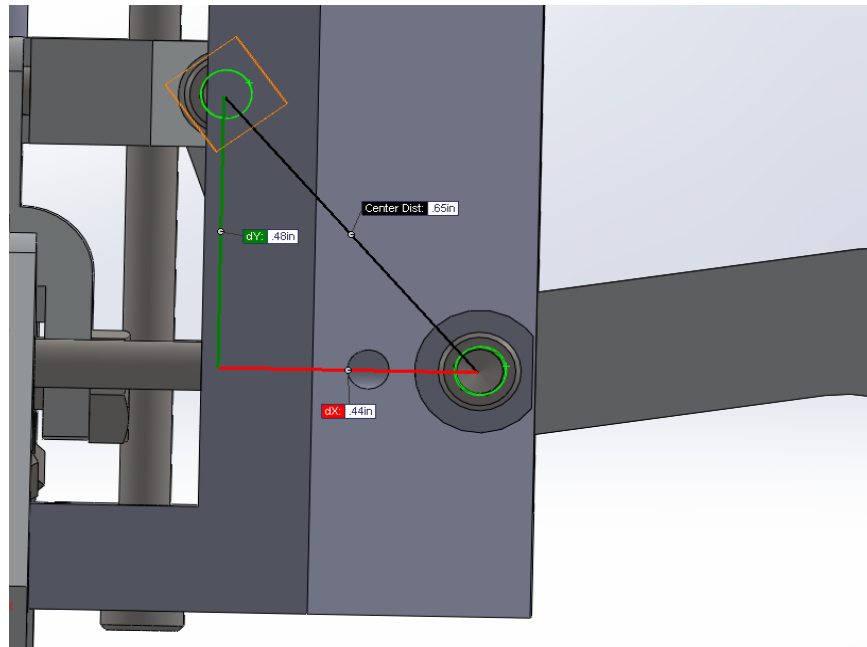
Figure 46: Determining Spike Moment Arm 2

The distance from the center connecting block to the tip of the spike is: .57in+ .75in = 1.32in. A minimum of 40lbs of force is developed at the connecting block. In this situation, 10lbs of force are developed at each of the connecting linkages.

Solving for the resulting X and Y component forces developed the end of each spike when 40lbs of force is applied to the center connecting block:

X-force=10(1.32)cos(30)=11.1inlb

Y-force=10(1.32)sin(30)=6.6inlb

Since these forces are developed at one end of the robot while the claw is closing, the servomotor at the opposing end of the robot needs to maintain equilibrium. The two farthest spikes are 13.24in from the axis of rotation of the opposing servo, while the two closer spikes are 11.06 inches away from the axis of rotation the opposing servo. The torque that needs to be developed at the opposing servo and maintain equilibrium while the other claw closes is:

$$\frac{6.6(2(13.24))}{12} = 14.564\text{ftlb}$$

$$\frac{6.6(2(11.06))}{12} = 12.66\text{ftlb}$$

The following graph shows the relationship of the angle of insertion to the amount of force, developed in the X and Y directions at each spike when the claw closes. The forces developed in the X and Y directions are equal when the insertion angle is 80°, and the Y force goes to zero when the spike is completely horizontal. The production claw was designed with a 30° insertion angle to reduce force in the Y direction, while providing a strong force in the X direction. This was done to help prevent the robot from pushing itself off of the tree when the claw closes.
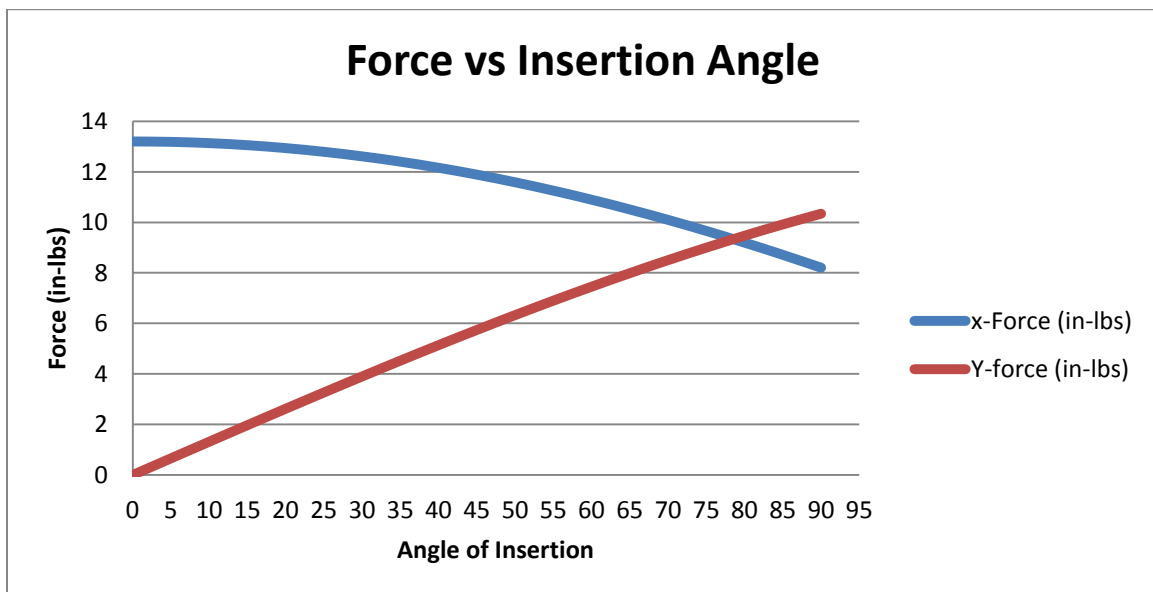
Figure 47: Force vs. Insertion Angle Graph

3.6.4 Determining Spike Reaction Forces

In order for the robot to be able to actually stay adhered to a tree that it is climbing, the spikes gripping the tree must experience reaction forces from the tree that are able to cancel out the forces applied to these spikes by the robot's weight. The realistic worst case that the robot can experience when the forces on the spikes are the greatest is when the robot's bottom claw is anchored into the tree while the other claw is not attached to any surface and the body extends away from the vertical tree at a 45° angle. This configuration is illustrated in Figure 49. This is the realistic worst case because the robot is physically unable to drop the body lower than this 45° position due to various components of the chassis colliding with each other and limiting the range of motion. This model simplifies the robot to two dimensions and simplifies the weight distribution of the robot into three discrete elements. These three elements are the overall central body and each of the two grippers. The model also assumes that the only two points of contact between the robot and the tree are points A and B. Since there are four gripping spikes on the claw, each of those two points represents the effective contact point of two spikes. The goal of this analysis was to determine the spike reaction forces at points A and B. Since there are four reaction forces to solve for ($R_{A,x}$, $R_{A,y}$, $R_{B,x}$ and $R_{B,y}$) and this model is only two-dimensional, this problem is statically indeterminate.

$$W_c = 358 \text{ g} = 12.63 \text{ oz}$$
$$W_b = 556 \text{ g} = 19.61 \text{ oz}$$
$$L_c + \tfrac{1}{2}L_b = 6.215 \text{ in}$$
$$L_z = 0.68 \text{ in}$$
$$L_s = 2 \text{ in}$$
$$\alpha = 135°$$

Figure 48: Dimensions for the 2-D model
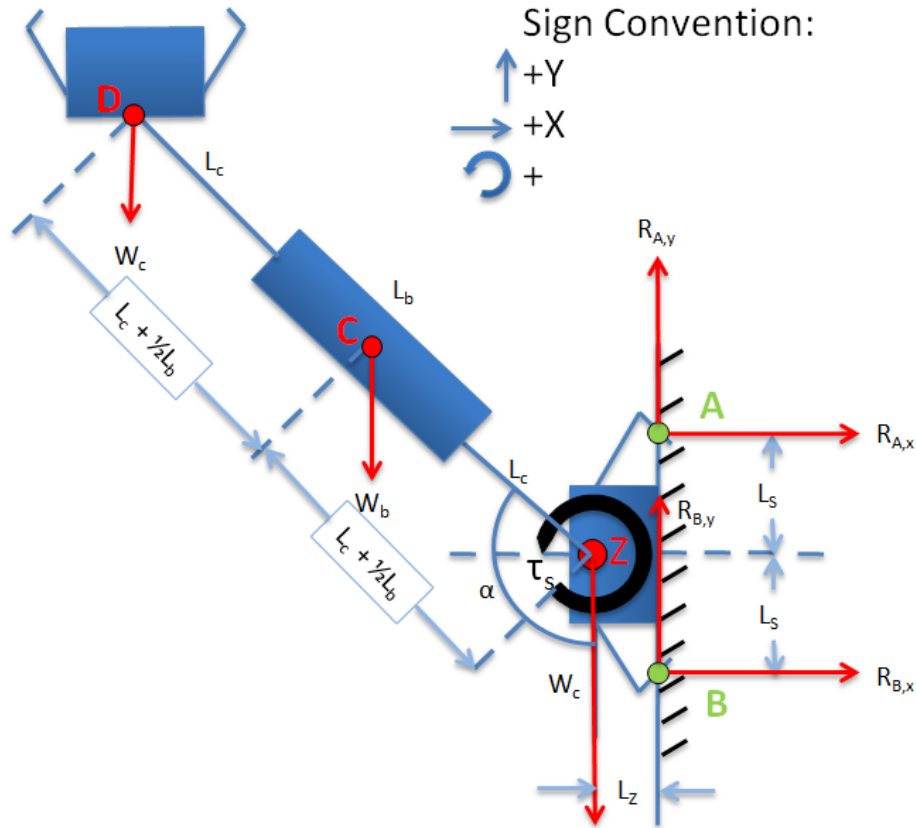
Figure 49: Worst case free-body diagram for spike holding/reaction forces - statically indeterminate

In order to solve for the spike reaction forces, the number of unknowns must be reduced from four to three. This was achieved by assuming that there is no y-component of the reaction force at point B ($R_{B,y} = 0$). Since the bottom spike would actually bear some of the force in the y-direction with the real robot, this assumption will cause $R_{A,y}$ to appear greater in magnitude than in the real-world situation. This overshoot maintains this analysis as a worst-case scenario because the top spike at point A would be more likely to lose its grip on the tree and dislocate, causing the robot to fall. This is because the robot's weight trying to rotate the gripper counter-clockwise, thereby pulling the top spike at point A *out* of the tree while pushing the bottom spike

at point B *into* the tree. The new statically determinate free-body diagram of this model is illustrated in Figure 50. All evident forces are represented as red arrows.
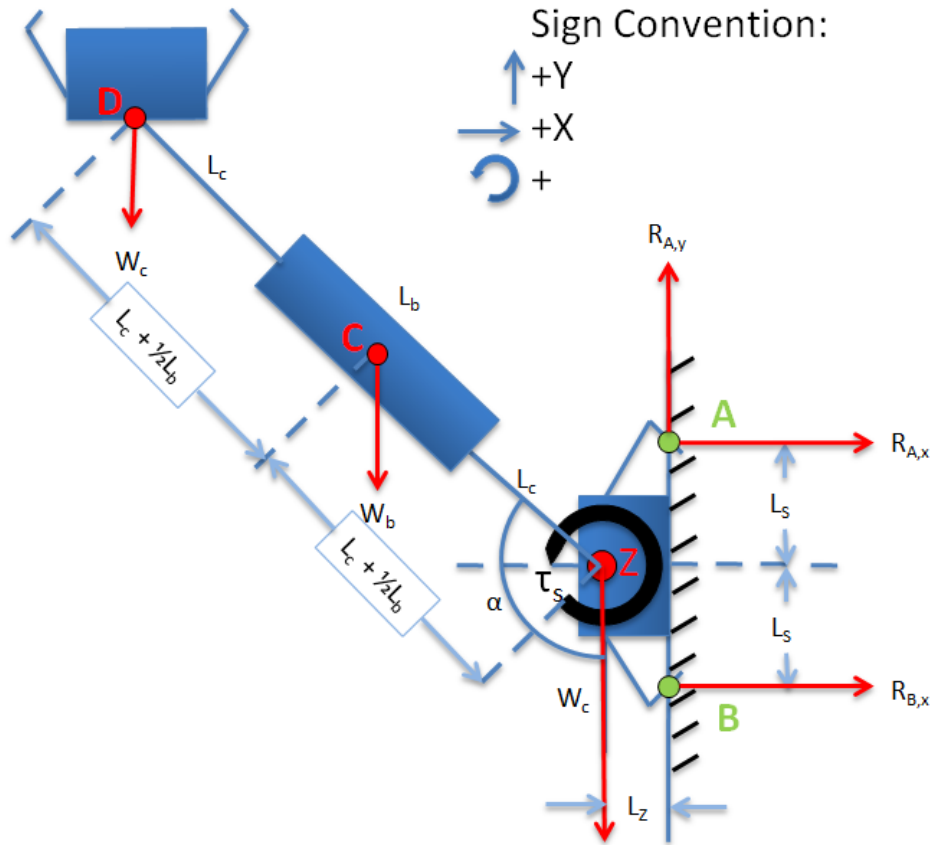
Figure 50: Worst case free-body diagram for spike holding/reaction forces - statically determinate

Using this free-body diagram, the forces in the y-direction can be related to solve for $R_{A,y}$.

$$\sum F_y = 0 = R_{A,y} - W_b - 2 * W_c$$

$$R_{A,y} = W_b + 2 * W_c = 19.61\ oz + 2 * 12.63\ oz$$

$$\boldsymbol{R_{A,y} = 44.87\ oz = 2.80\ lb}$$

Similarly, the forces in the x-direction can be summed to determine the relationship between $R_{A,x}$ and $R_{A,y}$.

$$\sum F_x = 0 = R_{A,x} + R_{B,x}$$

$$R_{A,x} = -R_{B,x}$$

The final equation of static equilibrium that can be applied is the sum of moments. Since the moments were calculated about point A, the relationship between A and all of the other points needed to be determined. These measurements are illustrated in Figure 51, Figure 52, and Figure 53.
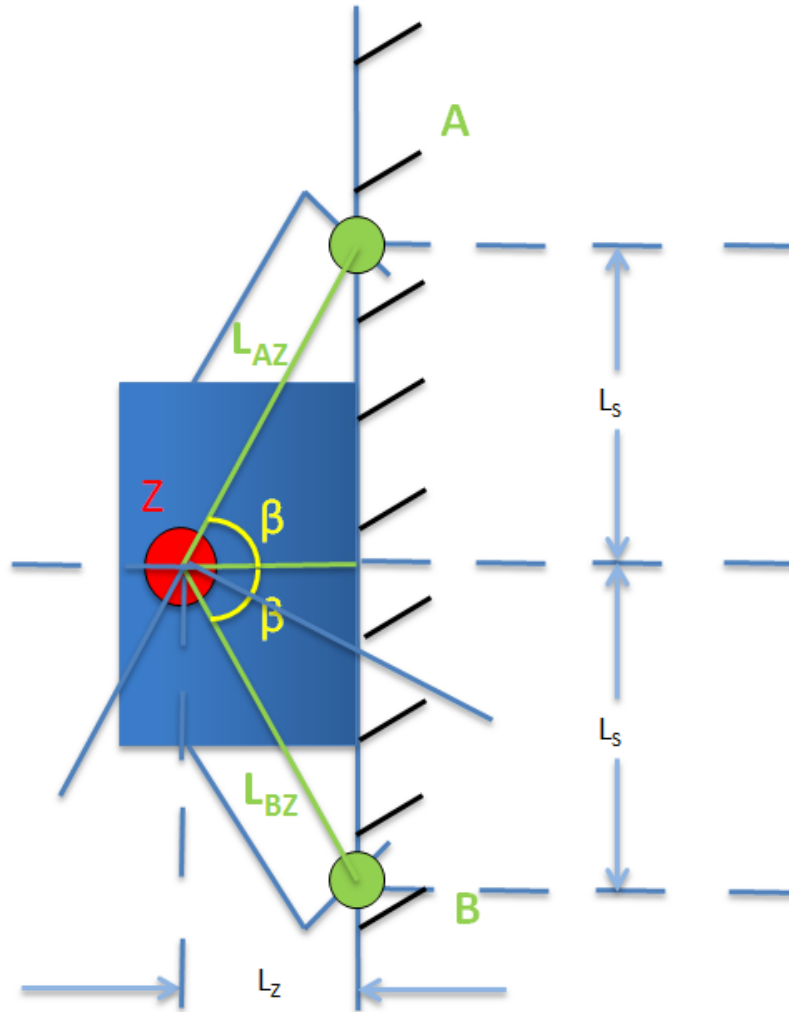
Figure 51: Measurements of point Z relative to point A

$$\beta = tan^{-1}\frac{L_S}{L_Z} = tan^{-1}\frac{2\ in}{0.68\ in}$$

$$\boldsymbol{\beta = 71.22°}$$

$$L_{AZ} = L_{BZ} = \sqrt{L_S{}^2 + L_Z{}^2}$$

$$L_{AZ} = \sqrt{(2\ in)^2 + (0.68\ in)^2}$$

$$\boldsymbol{L_{AZ} = L_{BZ} = 2.11\ in}$$
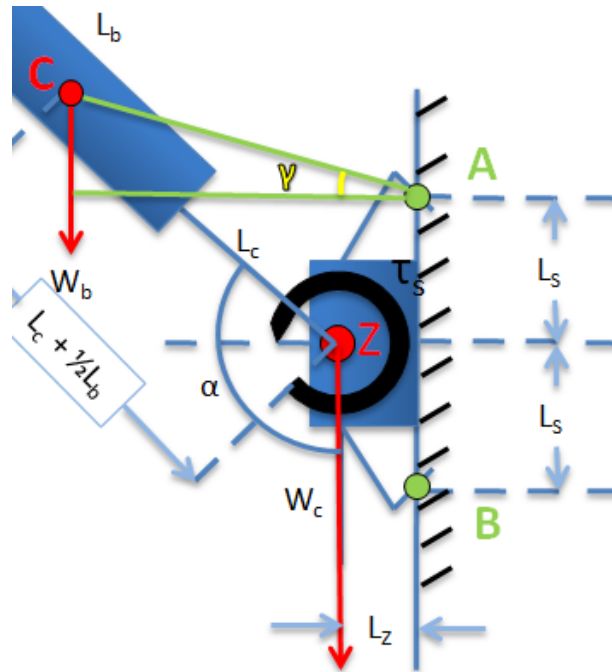
Figure 52: Measurements of point C relative to point A

$$L_{AC,Y} = \left(L_c + \frac{1}{2} L_b\right) * \sin(45°) - L_s$$

$$L_{AC,Y} = (6.215\ in) * \sin(45°) - 2\ in$$

$$\boldsymbol{L_{AC,Y} = 2.39\ in}$$

$$L_{AC,X} = \left(L_c + \frac{1}{2} L_b\right) * \cos(45°) + L_Z$$

$$L_{AC,X} = (6.215\ in) * \cos(45°) + 0.68\ in$$

$$\boldsymbol{L_{AC,X} = 5.07\ in}$$

$$L_{AC} = \sqrt{L_{AC,X}^2 + L_{AC,Y}^2}$$

$$L_{AC} = \sqrt{(5.07\ in)^2 + (2.39\ in)^2}$$

$$\boldsymbol{L_{AC} = 5.61\ in}$$

$$\gamma = \tan^{-1}\left(\frac{L_{AC,Y}}{L_{AC,X}}\right) = tan^{-1}\left(\frac{2.39\ in}{5.07\ in}\right)$$

$$\boldsymbol{\gamma = 25.24°}$$

70
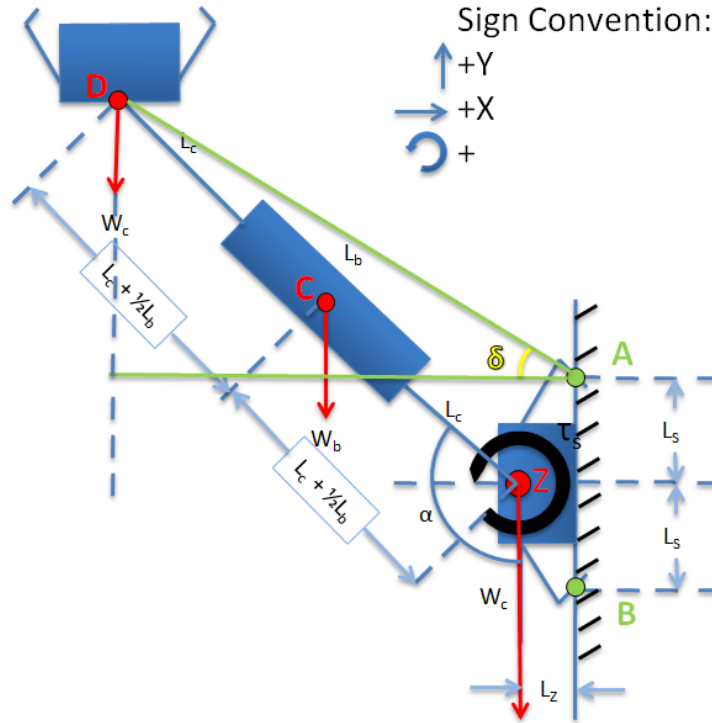
Figure 53: Measurements of point D relative to point A

$$L_{AD,Y} = 2 * \left(L_c + \frac{1}{2}L_b\right) * \sin(45°) - L_s$$

$$L_{AD,Y} = 2 * (6.215\ in) * \sin(45°) - 2\ in$$

$$\boldsymbol{L_{AD,Y} = 6.79\ in}$$

$$L_{AD,X} = 2 * \left(L_c + \frac{1}{2}L_b\right) * \cos(45°) + L_Z$$

$$L_{AD,X} = 2 * (6.215\ in) * \cos(45°) + 0.68\ in$$

$$\boldsymbol{L_{AD,X} = 9.47\ in}$$

$$L_{AD} = \sqrt{L_{AD,X}{}^2 + L_{AD,Y}{}^2}$$

$$L_{AD} = \sqrt{(9.47\ in)^2 + (6.79\ in)^2}$$

$$\boldsymbol{L_{AD} = 11.65\ in}$$

$$\delta = \tan^{-1}\left(\frac{L_{AD,Y}}{L_{AD,X}}\right) = \tan^{-1}\left(\frac{6.79\ in}{9.47\ in}\right)$$

$$\boldsymbol{\delta = 35.64°}$$

With all of these values describing the location of all points relative to point A, the sum of

moments about point A could then be calculated.

$$\sum M_A = 0 = R_{B,x} * (2 * L_S) + W_c * \cos(\beta) * (L_{AZ}) + W_b * \cos(\gamma) * (L_{AC}) + W_c * \cos(\delta) * (L_{AD})$$

$$0 = R_{B,x} * (2 * 2\ in) + 12.63\ oz * \cos(71.22°) * (2.11\ in) + 19.61\ oz * \cos(25.24°) * (5.61\ in) + 12.63\ oz$$
$$* \cos(35.64°) * (11.65\ in)$$

$$R_{B,x} * (2 * 2\ in) = -[12.63\ oz * \cos(71.22°) * (2.11\ in) + 19.61\ oz * \cos(25.24°) * (5.61\ in) + 12.63\ oz$$
$$* \cos(35.64°) * (11.65\ in)]$$

$$R_{B,x} = \frac{-[12.63\ oz * \cos(71.22°) * (2.11\ in) + 19.61\ oz * \cos(25.24°) * (5.61\ in) + 12.63\ oz * \cos(35.64°) * (11.65\ in)]}{(2 * 2\ in)}$$

$$\boldsymbol{R_{B,x} = -56.92\ oz = -3.56\ lb}$$

This value for $R_{B,x}$ can then be plugged into the resultant equation from the sum of the forces in

the y-direction to solve for $R_{A,x}$.

$$R_{A,x} = -R_{B,x}$$

$$\boldsymbol{R_{A,x} = 56.92\ oz = 3.56\ lb}$$

In summary:

$$R_{A,x} = 3.56\ lb,\ R_{A,y} = 2.80\ lb,\ \text{and}\ R_{B,x} = -3.56\ lb$$

The magnitude of the overall reaction force exerted by the tree onto the spike at point A is then:

$$|R_A| = \sqrt{(R_{A,x})^2 + (R_{A,y})^2} = \sqrt{(3.56\ lb)^2 + (2.80\ lb)^2}$$

$$\boldsymbol{|R_A| = 4.53\ lb}$$

$|R_A|$ represents the maximum magnitude of the reaction force exerted on the spike at point A. The direction in which this force acts can be calculated as follows with θ representing the angle between the positive x-axis and the direction of the maximum force (measured in the clockwise direction).

$$\theta = \tan^{-1}\left(\frac{R_{A,y}}{R_{A,x}}\right) = \tan^{-1}\left(\frac{2.80\ lb}{3.56\ lb}\right)$$

$$\boldsymbol{\theta = 38.19°}$$

The torque required by the servo at point Z to hold the robot in this 45° position was calculated based on the dimensions previously specified in Figure 40.

A = gripper assembly (236g) + linear actuator circuit (20g) + servo with servo block (102g) = 358g

B = ladder guide (18g)

C = servo with servo block (102g)

D = servo block guide (16g) + bread board (35g) = 51g

E = central spherical wrist servo assembly (122g)

F = servo with servo block (102g)

G = ladder guide (18g) + Arduino (143g) = 161g

H = gripper assembly (236g) + linear actuator circuit (20g) + servo with servo block (102g) = 358g

Total weight = 1272 grams=44.87 ounces=2.80 pounds

Required torque = 0 *A + sin(45°)*2.2" *B + sin(45°)*4.44"*C + sin(45°)* 5.64"*D + sin(45°)* 6.84"*E + sin(45°)* 8.3"*F + sin(45°)* 10.23"*G + sin(45°)* 12.43"*H

Required torque = 0 + 0.98 + 11.30 + 7.18 + 20.80 + 21.13 + 41.09 + 111.01 = 213.49 oz-in (1.11 ft-lbs)

Therefore, the minimum torque that the servo at point Z must exert to hold the robot statically stable on the tree in the 45° offset position is **1.11 ft-lbs**.

3.6.5 TCR Component Stress Simulations

After the design of the gripper was finalized and prototype testing had been completed, it was determined that the design could feasibly adhere to a tree with minimal design changes. In order to allow the gripper to penetrate wood it would be necessary to generate a force of approximately 70lbs at the connecting block. Since this load was significantly higher than expected when the gripper was designed, all parts that would be placed under an increased load were analyzed to ensure that they could bear the required loads without deformation. SolidWorks Simulationxpress was used to carry out these static load simulations. The assumptions made during these tests, the simulation settings, results, and conclusions are presented below.

Assumptions:

- All loads are static
- Applied loads are estimated and exaggerated to simulate an extreme circumstance
- All loads are applied as close to their real location as possible

- All parts are modeled out of 6061 aluminum alloy with a yield strength of 1150000 lb/ft^2 and a tensile strength of 2515000 lb/ft^2

3.6.6 Stress Simulation Results

All but one of the parts that underwent the SolidWorks SimulationXpress were readily able to hold their exaggerated loads, had an acceptable margin of safety, and exhibited relatively minimal displacement. Screenshots of these tests are shown in Appendix C. The lowest factor of safety experienced by any part during the simulations was the y-block, which had a minimum factor of safety of .84. However, the y-block test load of 17.5 lb was significantly higher than it would experience in actuality and it was also applied to the face at end of the y-block. In reality the load would be transmitted to the part via two bolts, capped with nuts, which would pass through the holes at each end of the y-block. The additional support provided by aforementioned nuts and bolts will prevent the applied load from spreading the fork end of the y-block, which is the area exhibiting the highest deformation and the lowest factor of safety in the simulation. With this in mind, it is reasonable to assume that the y-block will not deform under load as shown in the simulation results.

3.7 Software and Controller Design

TCR12 is a fairly simple state machine with four sensors and seven actuators. Two types of sensing occur on the robot. Whether or not either gripper is in contact with the climbing surface is sensed with pushbuttons on both grippers and amount of force being exerted by the linear actuators is measured by current-sensing circuits wired in series with them. The force sensing is mainly used to monitor the actuators' performance and determine how well the robot is gripping various surfaces. The pushbuttons, however, play an important role in state-to-state transitions

when climbing. They are what determine whether or not the robot is ready to make the next motion in the gait.

The seven actuators are broken down to 5 rotational servos and 2 linear actuators, each of which requires independent controllability. Depending on the current mode of the robot, whether climbing or turning, one or several of these actuators may need to respond to the gait sequence or direct input from the user. A flowchart overview of the control program for TCR12 can be found in Appendix D.

All components were taken into consideration when evaluating potential system controllers. The controller would need to allow for analog and digital input, PWM outputs, and live communication to a user interface device. For this an Arduino Mega microcontroller was selected. The Arduino has more than adequate Input/Output capabilities as well as built in ADC to convert the analog signals from the current-sensing circuits. It also has several options for servo control. It is an inexpensive option with more than enough ability for this application.

3.8   Programming

The control program development began with defining a coordinate system, and naming I/O components accordingly for clarity. Figure 54 depicts the naming convention used. The three servos at the center of the robot, which control chassis movement, are named based on which axis about which they generate rotation.
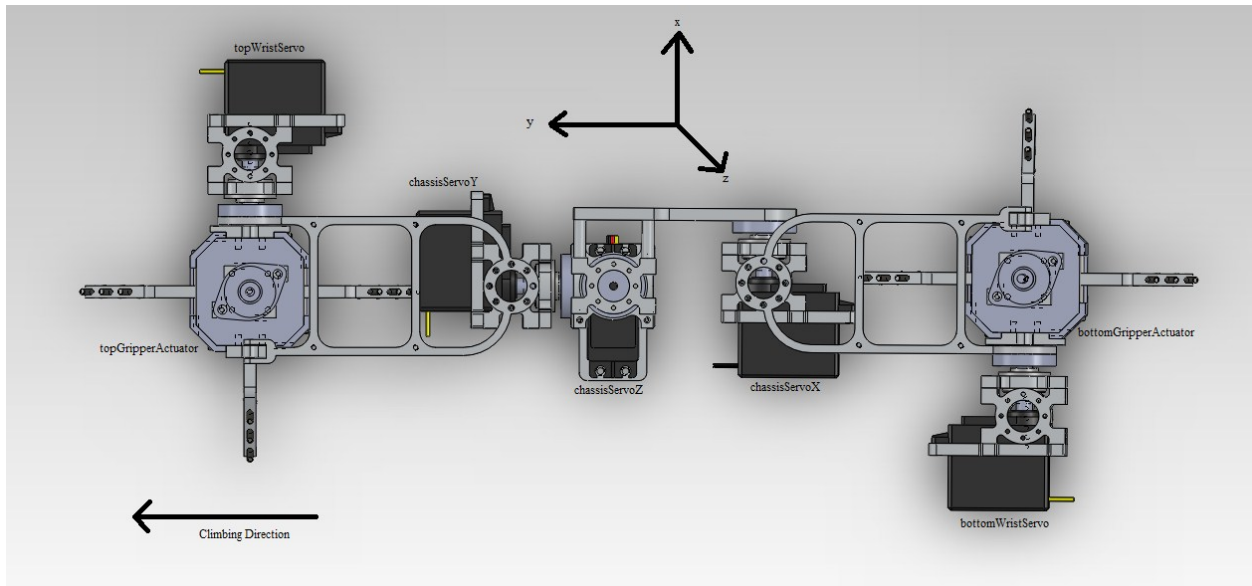
Figure 54: Orientation and component naming convention

Once the components were defined and initial device setup coded, serial communication was established to enable manual control via PC. Using a PuTTY command window, the user can supply inputs for several motions. An example of the command window in action is shown below in Figure 55, along with a sample of the code, depicting one of the motion functions.
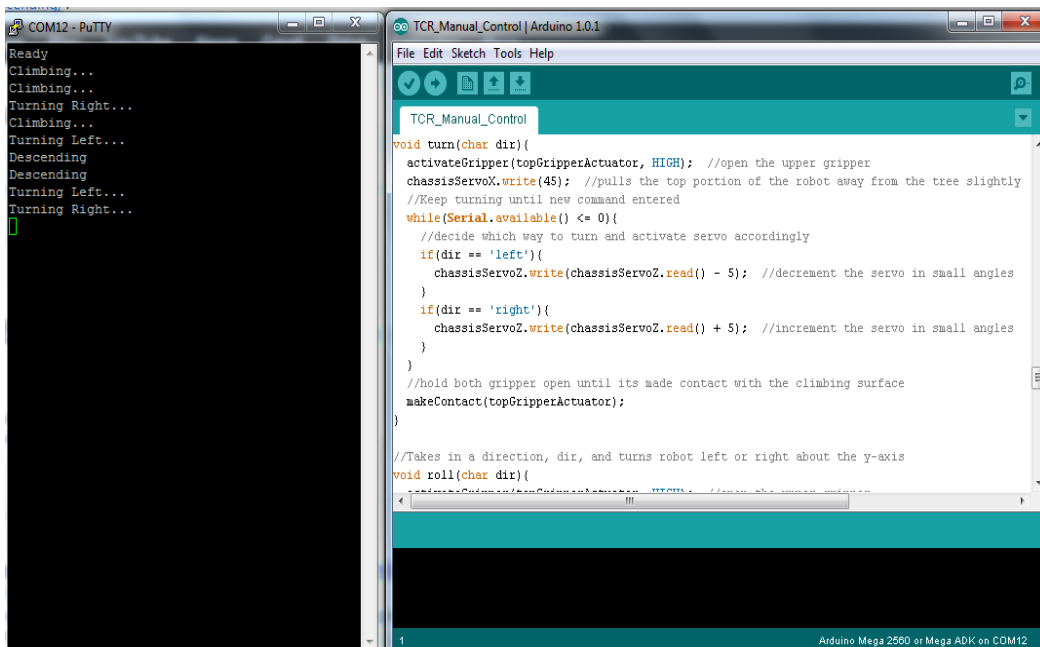
Figure 55: Left: PuTTY Command window in action. Right: Snippet of code showing turning function.

As mentioned previously, there are six different states for the climbing (and descending) gait. Transitions from one state to another are triggered by input from the pushbuttons on the bottom of either gripper. To perform the climbing motion there are two functions, upperStep() and lowerStep(). Starting in state one, the lowerStep() function makes all necessary calls to open the bottom gripper, raise the lower half of the chassis, and close the gripper on the tree again ending in state four. The upperStep() function then moves all servos and actuators required to raise the top half of the chassis and return the robot back to the first state. For transitions between these steps, helper-functions named makeContact() and checkContact() run to slowly move the desired portion of the chassis towards the tree until the pushbuttons are depressed, then close the gripper. All of this loops continuously when the user has commanded the robot to be in 'vertical mode' until a new command is entered. To descend, 'vertical mode' simply takes in a variable based on user input that reverses the order of the sequence described above.

The other two motion modes are used for turning purposes. When either of these modes is initiated in climbing mode, the top gripper opens and the user then has direct control of the upper half of the chassis. Conversely, the bottom gripper and lower half of the chassis is controlled when initiated in descending mode. By using the commands for 'turning mode', chassisServoZ is activated to turn the robot around the z-axis, to change the robots direction in the same plain as it was already in. Using different input commands, the user can initiate 'rolling mode' which rolls the robot around the y-axis and the circumference of the tree, using chassisServoY. Using this command will re-orient the robot correctly so that the gripper is facing the tree and can make contact after a turn is performed. Figure 56 depicts the keyboard input for each command.

Figure 56: User controls

Finally, a program was incorporated to enable users to monitor forces applied to either gripper's center block. To do so, current drawn by the gripper actuators is measured through circuitry and converted to force through code. This data is then displayed on the console for the user to monitor while operating the robot.

Originally, an autonomous program was started, but never finished since the robot will be tethered. It may be further developed for future versions of the robot, if more sensing and obstacle avoidance is added.

3.9 Electronics

3.9.1 Current Sensing:

As mentioned above, the system allows for the user to monitor how much force is being applied by the gripper. This is used to determine optimal force required for gripping different surface properties, or different kinds of trees. Displaying this data live to the user enables them to determine if the gripper has applied enough force to hold on at any given time during operation.

In order to sense the force on a gripper, the current drawn by the linear actuator which drives the gripper needs be measured. To do so the circuit below was designed. It measures the current going through the actuator directly by adding a shunt resistor in series with the actuator's ground lead. The voltage drop is then measured across this resistor, and is converted to current using Ohm's Law, V=iR. In order to get an accurate measurement that would not waste too much power in the process, a high precision, extremely low resistance (5 milliohms) shunt resistor was selected. This resistor is labled R_{sense} in Figure 57 below.
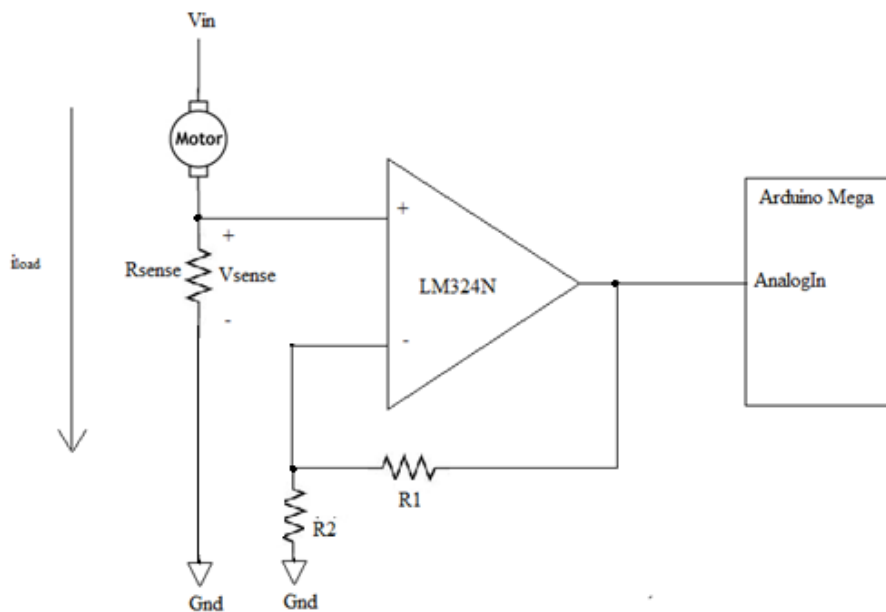
Figure 57: Current Sense Circuit Diagram

Since the voltage drop, V_{sense}, across such a small resistor is very small, an operational amplifier is needed to boost the voltage into a range that the Arduino microcontroller can make use of. Using the resistor value of R_{sense}, and the 650mA stall current of the actuators, the maximum voltage drop across the shunt resistor was determined. Comparing that voltage to the maximum voltage of 5V that the Arduino's ADC uses, the required gain from the op amp was determined to be 1538. To get this gain, a voltage divider needed to be set up between the op amp's output, inverting input, and ground. Using available resistors, a very close gain of approximately 1500 was achieved. These calculations are depicted in Figure 58 below.

Op Amp Gain Calculations:

$$R_{sense} = 5m\Omega \qquad i_{maxLoad} = 650mA$$

$$V_{maxSense} = i_{maxLoad} * R_{sense}$$

$$V_{maxSense} = 650mA * 0.005\Omega$$

$$V_{maxSense} = 0.00325V$$

$$Gain_{required} = \frac{V_{maxArduinoReading}}{V_{maxSense}}$$

$$Gain_{required} = \frac{5}{0.00325}$$

$$Gain_{required} = 1538$$

$$Gain = 1 + \frac{R_1}{R_2}$$

$$R_1 = 1M\Omega$$

$$R_2 = 680\Omega$$

$$Gain_{actual} = 1472$$

Figure 58: Op-Amp Gain Calculations

With this gain, the voltage range that is input to the Arduino's ADC is 0-4.78V. Comparing this range to the 0-1023 range of ADC values allowed for calculations to be made which would map any value read from the ADC to the actual voltage drop across the shunt resistor. From there, simple algebra was used to determine the actual current running through the shunt resistor, and thus the actuator as seen below.

Mapping ADC Readings to current:

AnalogIn Range: 0-1023 units   Voltage Range: 0-4.78V

$$R = 0.005\ \Omega \qquad Gain = 1472$$

$$V_{sense} = \frac{(AnalogIn * 0.0049)}{Gain}$$

$$i_{load} = \frac{Vsense}{R}$$

$$i_{load} = \frac{\frac{AnalogIn * 0.0047}{1472}}{0.005}$$
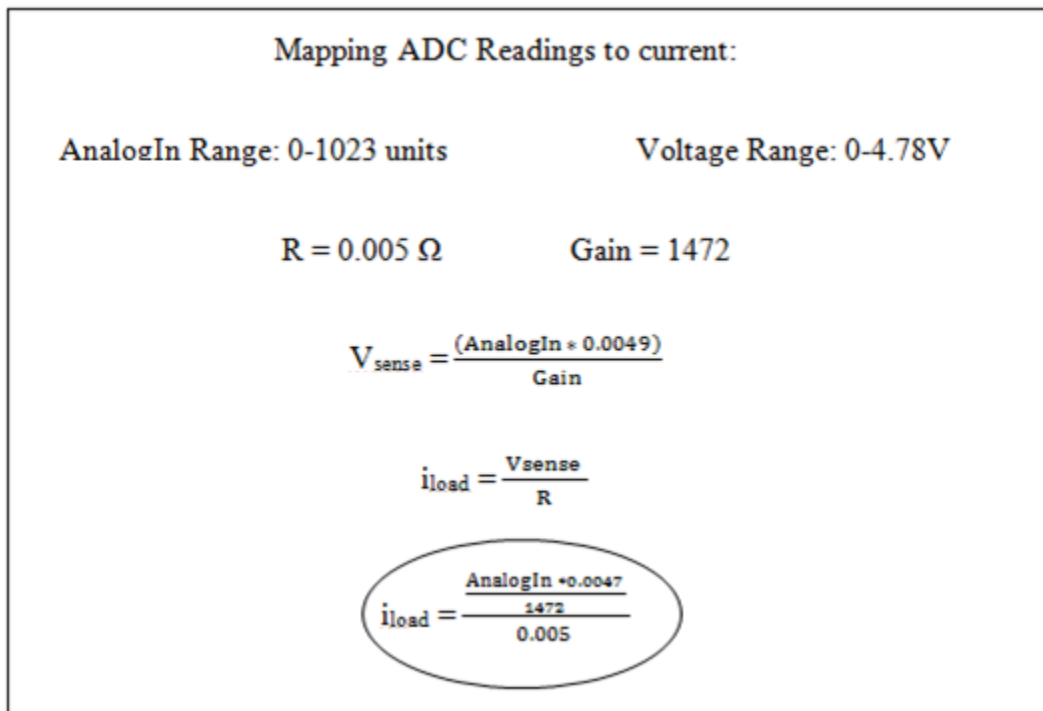
Figure 59: Mapping ADC values to Curent

Using the Current vs. Force graph obtained from the actuator's manufacturer, as shown below in Figure 60, the current being sensed was then converted to an actual force being exerted on the gripper's center block. A force of 0N is exerted by the actuator until 50mA is achieved, then it linearly climbs to 200N, or 44.96 lbs., at 400mA. This range was used to calculate the force in pounds per amp.
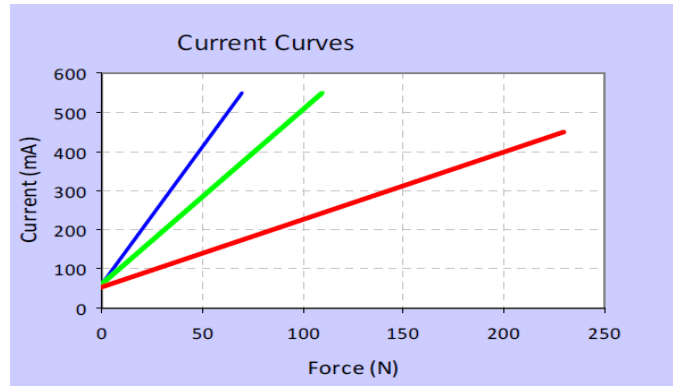
Figure 60: Current vs Force graph for Firgelli L16 actuators, courtesy of Firgelli Technologies Inc.

Current Range: 50mA − 400mA   Force Range: 0lbs − 44.96lbs

$$\text{Resolution} = 44.96\text{lbs} / 0.350\text{A} = 128.5 \text{ lbs./A}$$

$$\text{Force} = \left(\left(\frac{\frac{\text{AnalogIn} * 0.0047}{1472}}{0.005} \right) - .05 \right) * 128.5$$

Quantity	Parts Required
2	0.005Ω Shunt Resistors
2	680Ω Resistors
2	1MΩ Resistors
1	LM324N Op Amps

Table 3: Current-sensing circuit parts list

83

Finally, the parts were ordered according to the list in Table 3 and a test circuit was constructed. A test program was written to manually run one actuator and print the current and force being sensed to the user console. A multimeter was wired in series with the test circuit in order to compare values for proof of concept. The tests yielded positive results, and thus a final set of circuits was constructed. All of the calculations were then coded into the main program in order to display real time force readings to the user. For TCR12, the circuits will remain on a prototyping breadboard to allow for flexibility incase additional circuitry is needed. Future teams may consider switching to a printed circuit board for added robustness and weight reduction. The prototype test circuit, the test setup, and the final circuits are shown in Figure 61, Figure 62, and Figure 63, respectively
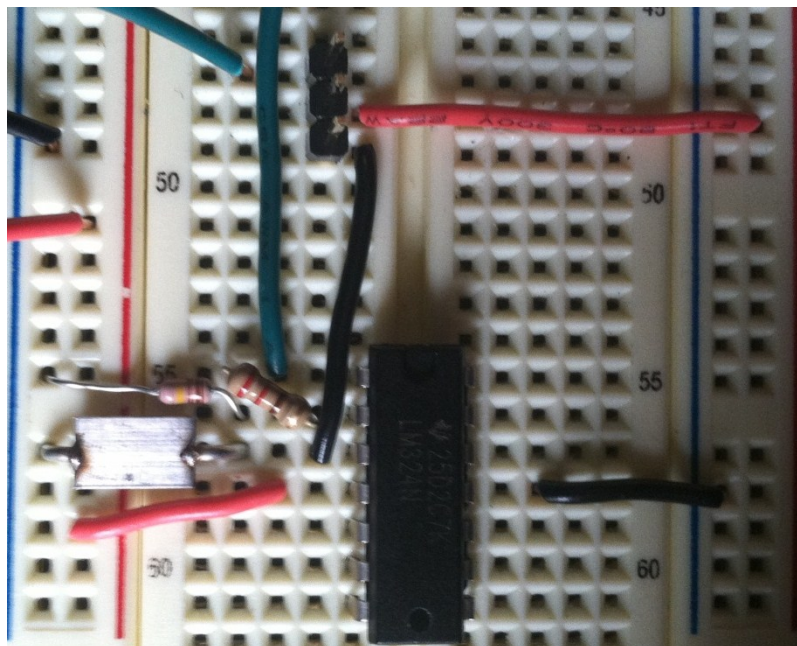
Figure 61: Current Sense Circuit Prototype
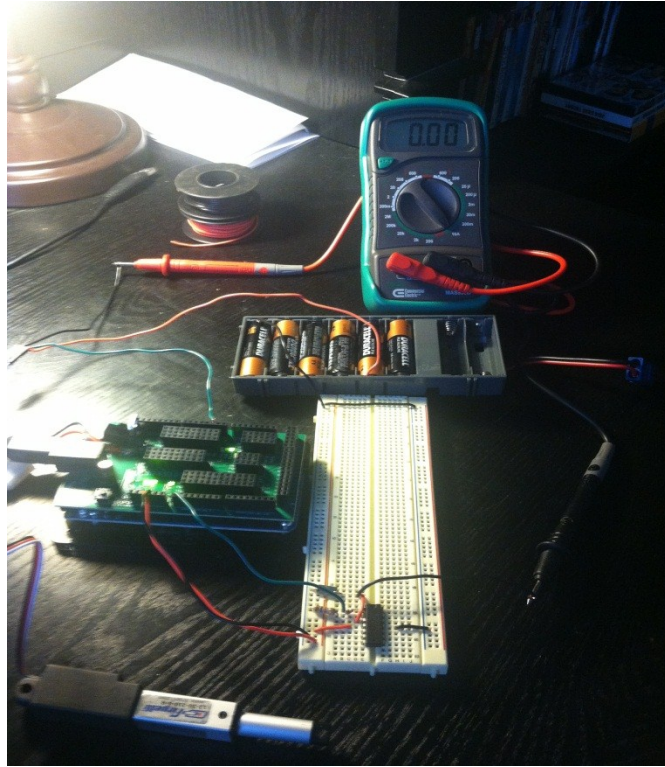
Figure 62: Current Sense test setup
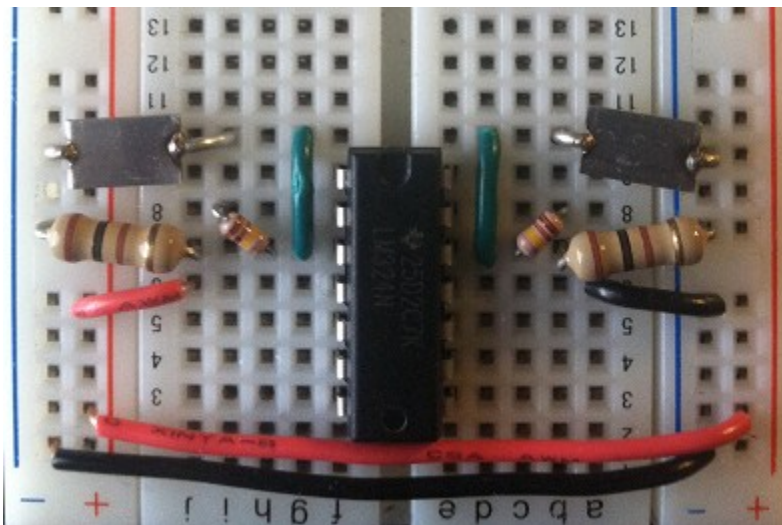
Figure 63: Finalized current sense circuits

4.   Final Testing

Once final assembly of the prototype was complete, testing began. First it was necessary to manually jog each joint servo to find the set points for centered and extreme positions. These values were subsequently entered into the robot control program. After all correct values were in place, the program was ready for testing. Initially, all individual functions of the program, such as the climbing command, were tested separately to ensure proper operation. This stage of testing was conducted on a horizontal surface to ensure that the robot had the proper gait before attempting a vertical climb. Once functionality was established, the manual control program was tested in whole, with user input as described in the Programming section. Figure 64 depicts the robot in its initial state on the left, and then in the middle of its stride on the right.
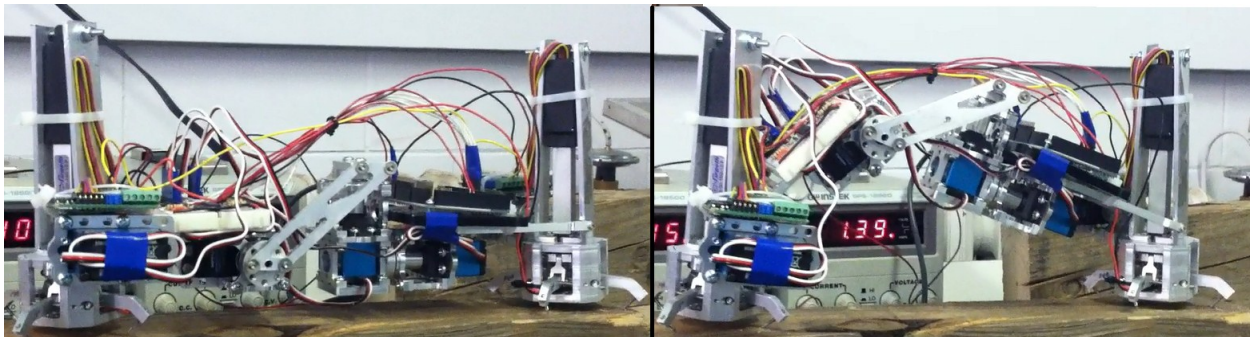
Figure 64: TCR12 half-stride

After fine tuning the program to get a smooth and consistent gait, vertical testing began. A new testing apparatus was constructed, as shown in Figure 65, to allow for safe vertical testing.
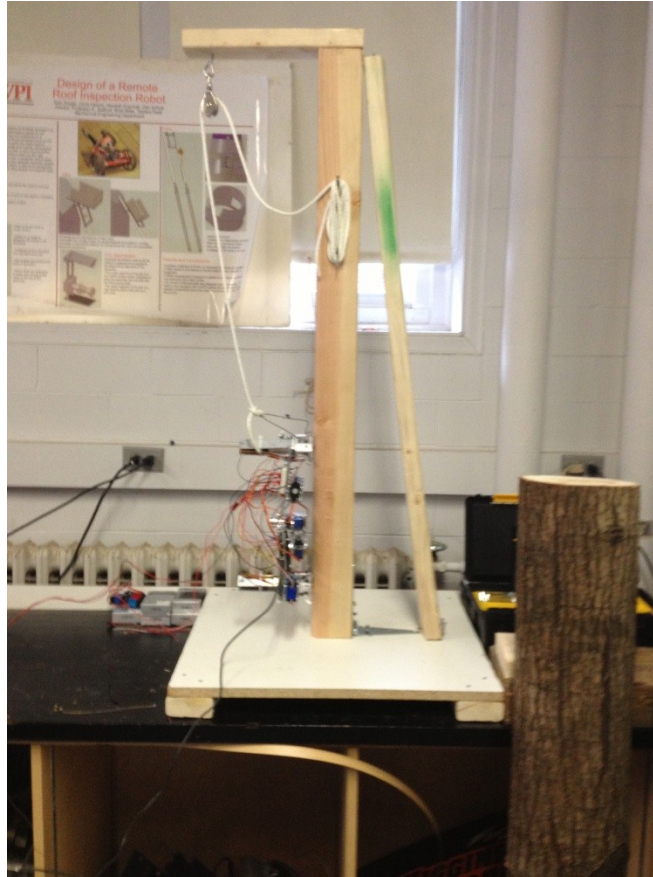
Figure 65: Testing apparatus

The testing apparatus was capable of holding the upright post at any angle ranging from a vertical position to a completely horizontal position. To tether the robot to the post, a rope was looped over a pulley at the top of the post, and attached to the robot at the other end. This was a precautionary measure taken to prevent the robot from crashing to the ground if something were to go awry.

To test the vertical climbing ability of the robot, it was held against the tower and the start position command was given to close both of the grippers. With both grippers closed, they were able to hold the robot to the tower and subsequently the climbing function was called. When one gripper opened and the robot attempted to climb, a slight jolt occurred. This sudden application of force was enough to dislodge the spikes of the other gripper, causing the robot to

fall. The team realized that although the grippers had sufficient force to drive the spikes in enough to hold the robot in place, they were not driven in deep enough to withstand the motion of the next gait step with only one gripper holding it. The team considered installing springs in the grippers to increase the force developed at the spike points. However, the calculations in the Gripper Force Analysis section proved that the servos would not have enough torque to withstand the push-off force the tower would exert back on the gripper with increased closing force.

At this point the team decided that vertical climbing would not be attainable for this iteration of the robot. The team then moved to establish proof of concept for individual functions of the robot. In the earlier stages of testing, shown in the Gripper Assembly and Testing section, the gripper mechanism had proven it was capable of holding substantial weight when enough force was generated to drive its spikes in. Also, the chassis design and control program demonstrated the ability to perform a smooth and consistent gait.

5. Conclusions

5.1 Project Review

Few robots in existence can climb vertically, and none of them could satisfy our established list of requirements. Having abandoned altogether the initial prototype from the previous project team, this team started from scratch. The team had to consider both proven and unproven methods of climbing to determine the best design features. Then the team needed to decide how to execute the chosen approach and come up with a design. The design underwent analysis in many areas before being machined and assembled for testing. During testing, unexpected issues arose, requiring much trouble shooting and a minor redesign. Finally a finished prototype was assembled, tested, and analyzed in order to establish a list of things that worked well and things that future project teams should take into consideration.

5.2 Results

Throughout the project, the team was able to demonstrate their expertise in designing and analyzing mechanical, electrical, and computer-controlled systems. That being said, the robot was unable to climb a vertical surface due to one factor, the force required to drive the gripper spikes into the wood. This is something that the team grappled with for the duration of the project. No resources could be found that provided an acceptable model of the mechanical properties of tree bark, requiring that the team estimate the force instead. The team did not correctly estimate the amount of force required to drive in the spikes, and this reduced the effectiveness of the design. To compensate for this, the spikes were sharpened to make them more effective at penetrating the wood, and several sets of legs were machined with different spike insertion angles with the hope that they might also reduce the force required. However,

these modifications did not have enough of an impact to allow the gripping function to operate as intended. It was determined that stronger actuation must be added to the grippers, which in turn would require replacing the chassis servos to compensate for the additional weight of said actuators and the additional push back force that accompanies a stronger closing force. Another option would be to use micro-spine style spikes, as seen on some of the robots in the Research section of this paper. Future teams might consider investigating the possibility of combining both of these ideas.

Although vertical climbing was not achieved, the team still made solid progress in several aspects. Through research and analysis, an exceptional gripper was designed, built, and proven successful. A chassis of high mobility was designed and constructed in a way that kept it extremely light weight, which was decidedly an important feature. The chassis effectively demonstrated its ability to perform a smooth gait, as well as turn and twist to navigate around the circumference of a convex surface. Along with progress made mechanically, there were steps taken to enhance the control system as well. A program was written that established full motion control of the robot via user input through a laptop. This program was tested and tuned to provide a consistent gait. A feedback system was incorporated into this iteration of the robot, with sensors built into the grippers to enable further control of actuation for optimum grip. Also, the team designed and constructed circuitry that allowed the user to monitor the force being applied by the grippers, which future iterations of this project can build and improve on.

6.   Recommendations and Future Work

6.1   Gripping

As mentioned numerous times in this report, the gripping needs to be addressed. The mechanism itself works great, but needs to be able to drive the spikes in further. Two suggestions to accomplish this are to look at the actuation and the spikes. If the robot can be modified to handle adding more force at the gripper actuators, the spikes should be able to sink in enough to hold while climbing. When the team manually exerted 70lb of force to the center block where the actuation occurs, the gripper was then able to hold a substantial force before being pulled out. Also, teams might consider looking into micro-spines. Something like a medical syringe or the end of a fishing hook may be easier to drive into the wood due to its smaller diameter and sharpness.

6.2   3D Printing

Future iterations of this project should seek to utilize lighter materials such as 3D printing resin save weight and lower the amount of force required to move the robot in any given situation. In order for 3D printed parts to be a viable alternative to machined aluminum parts, several parts, such as the Y blocks, would need to be redesigned. When the 3D printed robot was created multiple Y blocks were damaged during assembly and needed to be re-printed. However, if the current design was modified properly and aluminum parts were used in high stress areas, 3D printing could be a feasible way to lower the weight of the robot and increase it's effectiveness.

6.3   Safety Features

Since this was not a production design, several safety features were omitted that could easily be included in next year's design. The next tree-climbing robot could be made to be a highly visible

color, such as construction orange, or vibrant green, to make it easy to see. Additionally, a slip

notification system, such as an accelerometer wired to set off a warning siren of a fall is detected

should be employed in future designs. This is another effective way to warn people that there is a

falling robot overhead and they should relocate themselves immediately.

6.4   Power Distribution

There are several necessary considerations when deciding on a power supply. The suggested

supply for the Arduino is anywhere between 7-12V. An on-board regulator will reduce that to 5V

which it operates on, and also provides extra pins to pull that 5V to other components that need

it, provided they do not draw more than 40mA of current. This can be used to power the op amps

necessary for the current sensing circuits. There are 5 servos that require 8.4V and draw roughly

1.5-2A of current when stalled. This totals 10A in a 'worst-case' scenario of all 5 being stalled at

once. Finally, the linear actuators run on 12V, and will draw up to 650mA when stalled. There

are two of these, so combined they could draw up to 1.3A.

Fortunately, since this version of the robot is tethered, battery weight was not a concern.

The supply can sit on the ground with the user, with just the wires running up the tether to the

robot. Due to this fact, this iteration of the robot utilizes a bench-top power supply. Ideally,

however, there would be an on-board power supply for the robot. To accomplish this, future

teams could consider using a 12V battery with a maximum discharge rate of at least 12A.

Although it will probably never be in a situation that requires it to power everything at the same

time, it is advised to allow for the possibility to do so. In order for a single 12V supply to power

all components, step-down voltage regulators could be purchased to bring the voltage down to

the 7.4V that the servos need. It will be important to consider the output current ratings when

selecting regulators as to not restrict flow to the servos.

Future teams should also consider light weight battery options if they intend to go tether-less. They will need to keep in mind discharge rates and capacity in order to have longer lasting run times that still allow for full power for all of the robot's components.

7.   Appendix A: Rapid Prototyping

In an attempt to reduce the weight, cost, and time of construction a 3D printed version of the robot chassis was created. This 3D printed version was created using the MakerBot Replicator, on loan from Scott Innocenzi at Nottingham High School - North. By converting the SolidWorks part files into .Stl files, these parts were printed and assembled using model glue. The finished model can be seen below in Figure 66.
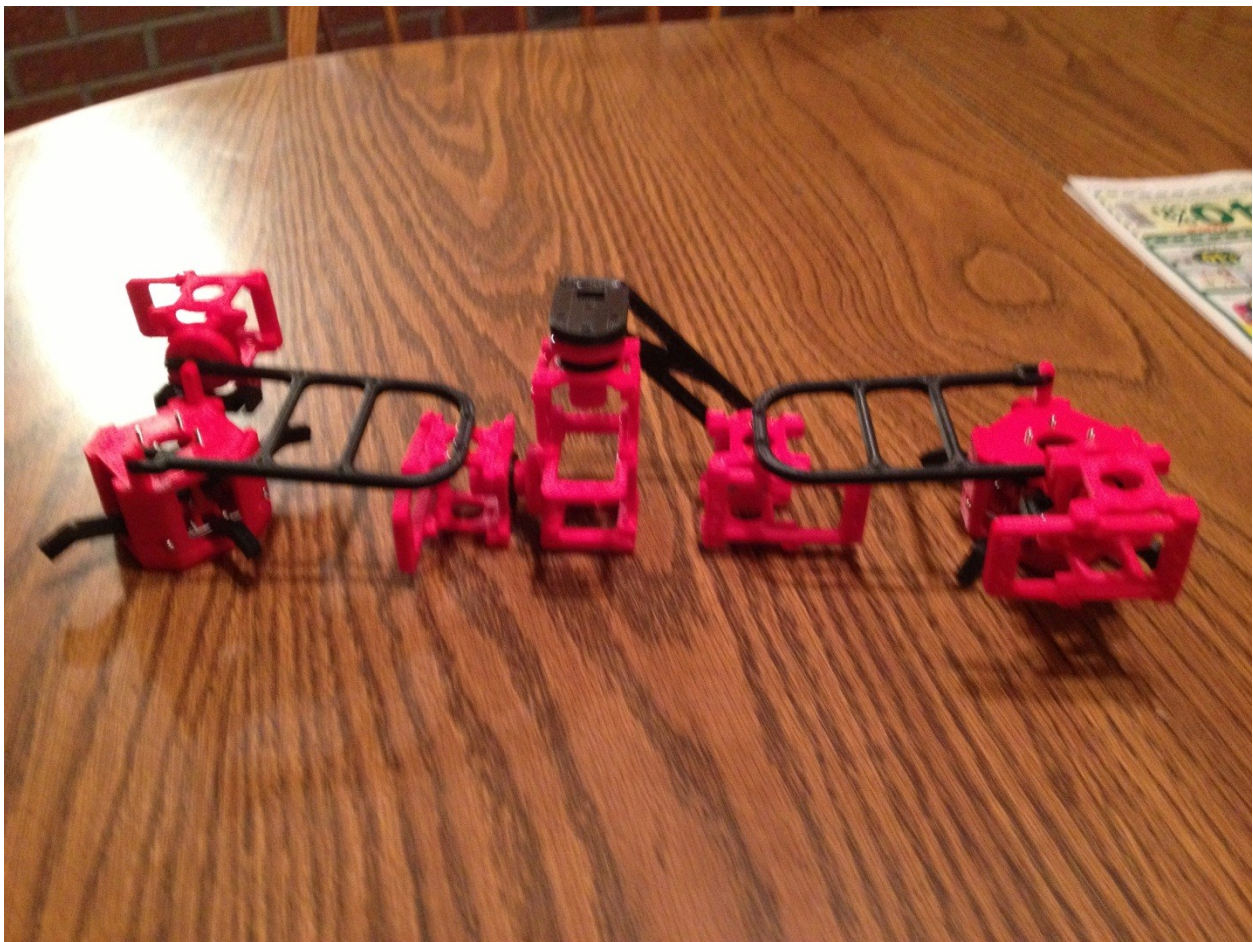
Figure 66: 3D printed TCR12 Chassis

From this 3D printed chassis the team learned that that time to construct each part was drastically reduced when compared with the time required to machine the same part out of aluminum. Also the 3D printed part was roughly $1/3^{rd}$ the weight of its aluminum counterpart. The problem with

3D printed parts is that they could not withstand enough force to be a viable alternative to aluminum parts for our design.

8.   Appendix B: Budget for Initial Design

Price per item	Qty	Total Price	Part	Note
$43.80	5	$219.00	Servos	*Small size*
$28.40	5	$142.00	Servo Blocks	*load-bearing servo case*
$117.00	2	$234.00	Linear actuators	*rated at 3 lbs of force*
$2.35	5	$11.75	H-bridge chip	*2 Motors per controller*
$200.00	1	$200.00	Chassis	*machining*
		$0.00	Hardware	*nuts, bolts, etc.*
$50.00	1	$50.00	Cables/Wires	
$40.00	8	$320.00	Claws	*High strength metal*
$50.00	1	$50.00	Arduino Mega	*(if necessary)*
$50.00	2	$100.00	Cameras	*Laptop webcam*
		$1,327	**Total Requested**	

Table 4: Estimated Budget for Initial Prototype Design
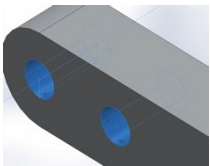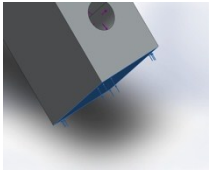
9. Appendix C: Initial Proposed Project Timeline

#	Task Name	Duration	Start	Finish
1	Proposal Writing	17 days	Thu 8/23/12	Fri 9/14/12
2	Design Initial Concept	19 days	Mon 9/17/12	Thu 10/11/12
3	A-term Ends	1 day	Thu 10/11/12	Thu 10/11/12
4				
5	B-term Starts	1 day	Tue 10/23/12	Tue 10/23/12
6	Re-evaluate Initial Concept	3 days	Tue 10/23/12	Thu 10/25/12
7	Prototype Construction	35 days	Fri 10/26/12	Thu 12/13/12
8	Construct end effectors	11 days	Fri 10/26/12	Fri 11/9/12
9	Construct legs	10 days	Mon 11/12/12	Fri 11/23/12
10	Construct chassis	10 days	Mon 11/26/12	Fri 12/7/12
11	Final assembly of prototype	4 days	Mon 12/10/12	Thu 12/13/12
12	B-term Ends	1 day	Thu 12/13/12	Thu 12/13/12
13				
14	C-term Starts	1 day	Thu 1/10/13	Thu 1/10/13
15	Preliminary Testing and Evaluation of Prototype	22 days	Thu 1/10/13	Fri 2/8/13
16	Modifications and updates to prototype	10 days	Mon 1/28/13	Fri 2/8/13
17	Final Testing of Robot	10 days	Mon 2/4/13	Fri 2/15/13
18	Write Final Report	15 days	Mon 2/11/13	Fri 3/1/13
19	C-term Ends	1 day	Fri 3/1/13	Fri 3/1/13
20				
21	D-term Starts	1 day	Mon 3/11/13	Mon 3/11/13
22	Prepare final presentation	14 days	Mon 4/1/13	Thu 4/18/13
23	PROJECT PRESENTATION DAY	1 day	Thu 4/18/13	Thu 4/18/13
24	D-term Ends	1 day	Tue 4/30/13	Tue 4/30/13

10. Appendix D: Component Stress Simulations

10.1 Claw Arm Simulation

Fixtures and Loads
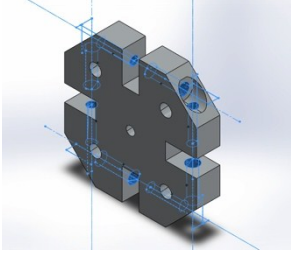
Fixture Image	Fixture Details	
	Entities:	**2 face(s)**
	Type:	**Fixed Geometry**
Load Image	Load Details	
	Entities:	**1 face(s)**
	Type:	**Apply normal force**
	Value:	**17.5 lbf**

Name	Type	Min	Max
Stress	VON: von Mises Stress	319.22 lb/ft^2	719641lb/ft^2
Displacement	URES: Resultant Displacement	0 in	0.00222 in
Factor of Safety	Max von Mises Stress	1.60051	3608.2

Model name: Claw_arm
Study name: SimulationXpress Study
Plot type: Static nodal stress Stress
Deformation scale: 101.428

von Mises (N/m^2)

34,456,732.0
31,686,612.0
28,716,492.0
25,846,370.0
22,976,250.0
20,106,130.0
17,236,008.0
14,365,888.0
11,495,767.0
8,625,646.0
5,755,525.5
2,885,405.0
15,284.2

→ Yield strength: 55,148,500.0
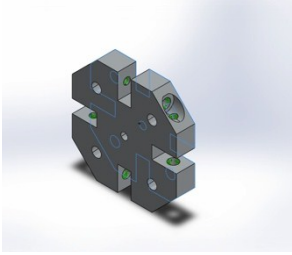
10.2 Center Block Simulation

Fixtures and Loads

Fixture Image	Fixture Details	
	Entities:	**8 face(s)**
	Type:	**Fixed**
		Geometry
Load Image	Load Details	
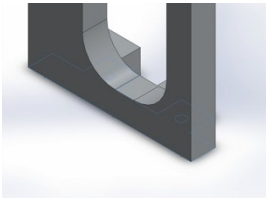
	Entities:	1 face(s)
	Type:	Apply normal force
	Value:	70 lbf (Load applied to front face of part)

Name	Type	Min	Max
Stress	VON: von Mises Stress	560.3 lb/ft^2	115504.4 lb/ft^2
Displacement	URES: Resultant Displacement	0 in	.000002 in
Factor of Safety	Max von Mises Stress	9.9719	2056.99

Model name: Connecting_block
Study name: SimulationXpress Study
Plot type: Static nodal stress Stress
Deformation scale: 5750.65

Max: 5,530,389.5
Min: 26,810.3

von Mises (N/m^2)

5,530,389.5
5,071,758.0
4,613,126.5
4,154,494.5
3,695,863.0
3,237,231.5
2,778,599.8
2,319,968.3
1,861,336.6
1,402,705.0
944,073.5
485,441.9
26,810.3

→ Yield strength: 55,148,500.0
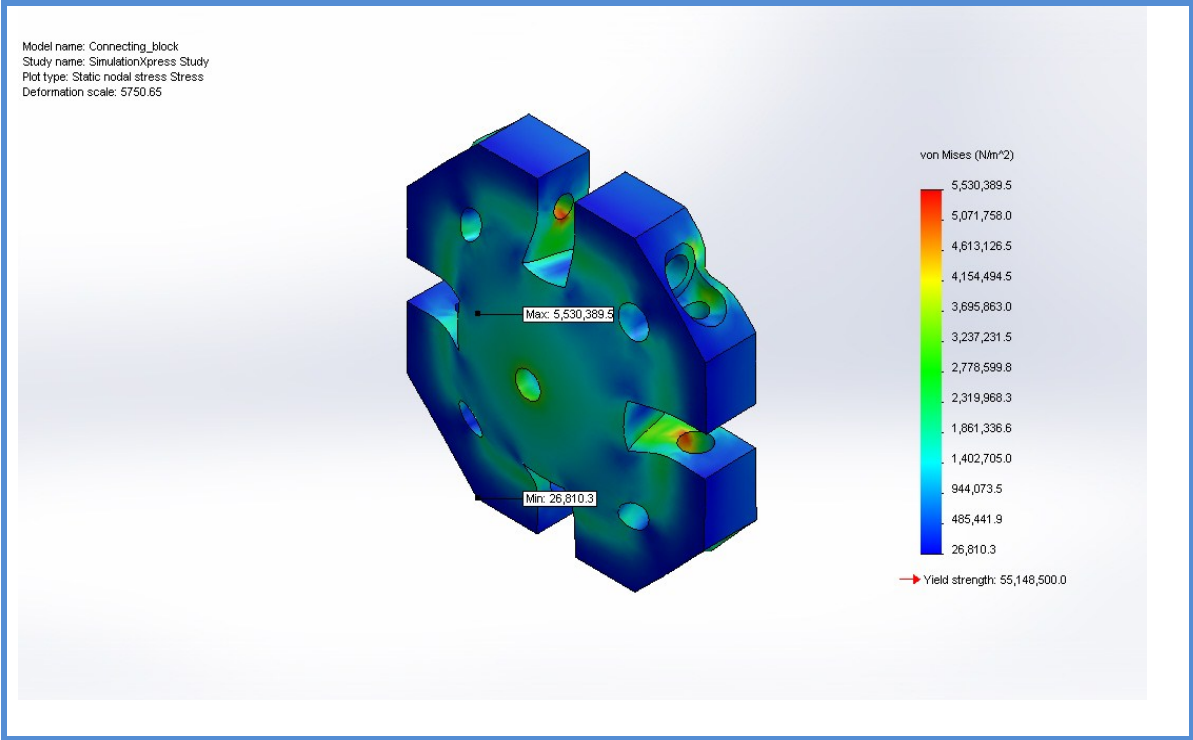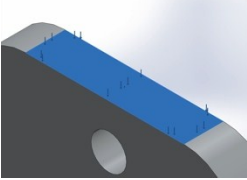
10.3   Linear Actuator Bracket Simulation

Fixtures and Loads

Fixture Image	Fixture Details	
	Entities:	**1 face(s)**
	Type:	**Fixed Geometry** (Fixture located on the underside of this part)

Load Image	Load Details	
	Entities:	**1 face(s)**
	Type:	**Apply**
		normal force
	Value:	**40 lbf**
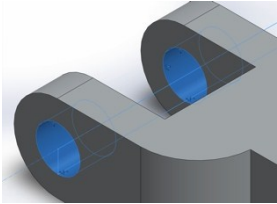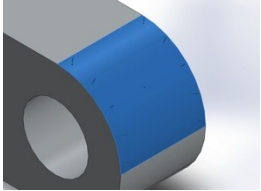
Name	Type	Min	Max
Stress	VON: von Mises Stress	44.528 lb/ft^2	162737.2 lb/ft^2
Displacement	URES: Resultant Displacement	0 in	.0003343 in
Factor of Safety	Max von Mises Stress	7.078	25866.8

10.4 Y-Block Simulation

Fixtures and Loads

Fixture Image	Fixture Details	
	Entities:	**2 face(s)**
	Type:	**Fixed**
		Geometry
Load Image	Load Details	
	Entities:	**1 face(s)**
	Type:	**Apply normal**
		force
	Value:	**17.5 lbf**

Name	Type	Min	Max
Stress	VON: von Mises Stress	4929.33 lb/ft^2	1377805.7 lb/ft^2
Displacement	URES: Resultant Displacement	0 in	.00126 in
Factor of Safety	Max von Mises Stress	.8356	233.662
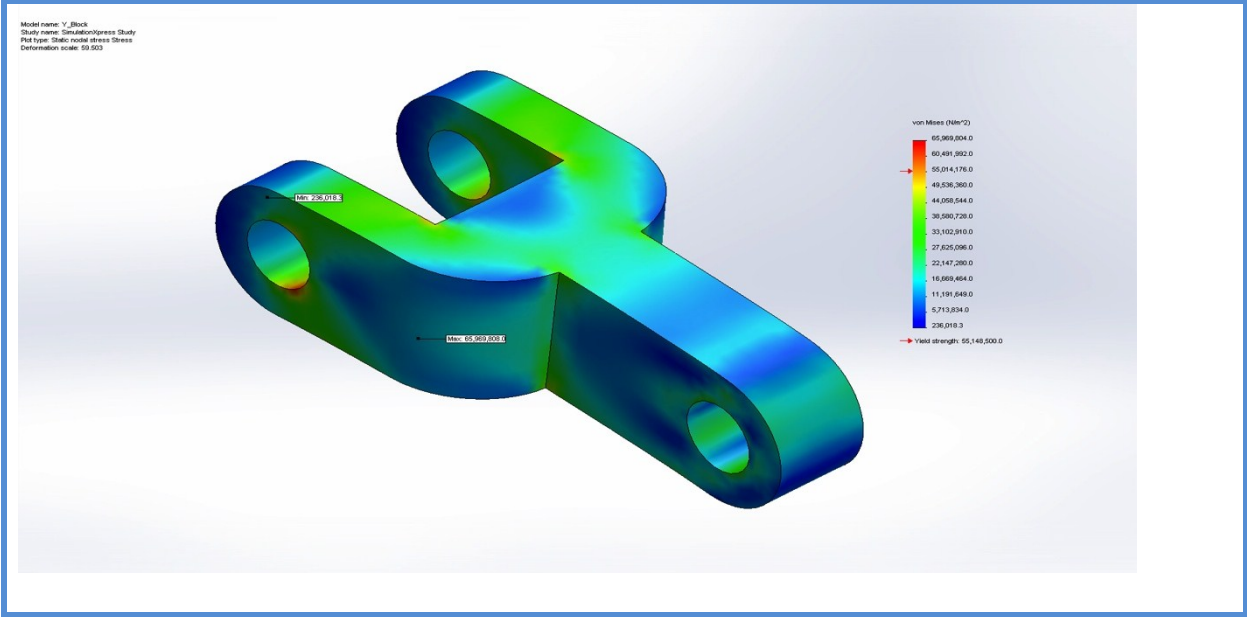
11. Appendix E: TCR12 Program Flowcharts

11.1 Climbing routine

```
Start
  │
  ▼
Both grippers in contact? ──No──┐
  │                             │
  │ Yes                         │
  ▼                             │
Wait for user inptut            │
  │                             │
  ▼                             │
State 1                         │
  │                             │
  ▼                             │
Open bottom gripper             │
  │                             │
  ▼                             │
Bottom gripper in contact? ──No──► State 2
  │                                   │
  │ Yes                               ▼
  ▼                             Move servos:
Move servo                       -top wrist servo
Chassis servo-X                  -bottom wrist servo
away from tree                   -chassis servo-X
                                 into curled position
                                   │
                                   ▼
                                 State 3
                                   │
                                   ▼
                                 Bottom gripper in contact? ──Yes──┐
                                   │                                │
                                   │ No                             │
                                   ▼                                │
                                 Move servo                         │
                                 Chassis servo-X                    │
                                 toward tree                        │
                                   │                                │
                                   ▼                                │
                                 Close bottom gripper ◄─────────────┘
                                   │
                                   ▼
                                 State 4
                                   │
                                   ▼
                                 Open top gripper
                                   │
                                   ▼
                                 Top gripper in contact? ──No──┐
                                   │                            │
                                   │ Yes                        │
                                   ▼                            │
                                 Move servo:                    │
                                 -chassis servo-X ◄─────────────┘
                                   │
                                   ▼
                                 State 5
                                   │
                                   ▼
                                 Move servos:
                                 -top wrist servo
                                 -bottom wrist servo
                                 -chassis servo-X
                                 into straight position
                                   │
                                   ▼
                                 State 6
                                   │
                                   ▼
                                 Top gripper in contact? ──Yes──┐
                                   │                             │
                                   │ No                          │
                                   ▼                             │
                                 Move servo:                     │
                                 -chassis servo-X                │
                                   │                             │
                                   ▼                             │
                                 Close top gripper ◄─────────────┘
```

11.2 ISR1: Turning

```
                          ┌──────────────────┐
                          │   ISR1: Turning  │
                          └──────────────────┘
                                    │
                                    ▼
        Up              ◇ Direction        Down       ◇ Bottom gripper    Yes
◇ Top gripper ◇ ◄─────── ◇  heading?  ◇ ──────────► ◇  in contact?  ◇ ─────►
   in contact?                                                        
        │ No                                              │ No
        ▼                                                 ▼
┌─────────────────┐      ┌──────────────────┐    ┌─────────────────┐
│ Open top gripper,│ ──► │  Enable direct   │◄── │ Open gripper,   │
│ rotate chassis   │     │  manual control  │    │ rotate chassis  │
│ servo-X away     │     │  of chassis      │    │ servo-X slightly│
│ from tree        │     │  servo-Z         │    │ away from tree  │
└─────────────────┘      └──────────────────┘    └─────────────────┘
                                    │
                                    ▼
                          ◇ New command? ◇ ── No ──►
                                    │ Yes
                                    ▼
                          ◇ Entered ISR with  ── No ──►
                          ◇ gripper in
                          ◇ contact?
                                    │ Yes
                                    ▼
                          ┌──────────────────┐
                          │ Rotate chassis   │
                          │ servo-X toward   │
                          │ tree and close   │
                          │ gripper          │
                          └──────────────────┘
                                    │
                                    ▼
                          ┌──────────────────┐
                          │     Exit ISR     │
                          └──────────────────┘
```

11.3 ISR2: Rolling

```
                          ┌─────────────────┐
                          │  ISR2: Rolling  │
                          └─────────────────┘
                                   │
                                   ▼
            Yes      ◆ Top gripper ◆   Up    ◆ Direction ◆   Down   ◆ Bottom gripper ◆    Yes
         ┌─────────  ◆ in contact? ◆ ◄─────  ◆  heading? ◆ ─────►   ◆  in contact?   ◆  ─────┐
         │                  │                      │                        │
         │                 No                     No                       No
         │                  │                      │                        │
         ▼                  ▼                      ▼                        ▼
┌──────────────────┐  ┌──────────────────┐  ┌──────────────────┐
│ Open top gripper,│  │  Enable direct   │  │ Open gripper,    │
│ rotate chassis   │  │  manual control  │  │ rotate chassis   │
│ servo-X away     │─►│  of chassis      │◄─│ servo-X slightly │
│ from tree        │  │  servo-Y         │  │ away from tree   │
└──────────────────┘  └──────────────────┘  └──────────────────┘
                               │
                               ▼
                       ◆ New command? ◆   No
                       ◆              ◆ ──────┐
                               │
                              Yes
                               │
                               ▼
                    ◆ Entered ISR with ◆   No
                    ◆   gripper in     ◆ ──────┐
                    ◆    contact?      ◆
                               │
                              Yes
                               │
                               ▼
                    ┌──────────────────┐
                    │ Rotate chassis   │
                    │ servo-X toward   │
                    │ tree and close   │
                    │ gripper          │
                    └──────────────────┘
                               │
                               ▼
                        ┌─────────────┐
                        │  Exit ISR   │
                        └─────────────┘
```

11.4   Tree-Climbing Robot 2012-2013 Manual Control Program

```
1.  /* FILE:           TCRManualControl.pde
2.  // TITLE:          Tree-Climbing Robot 2012-2013 Manual Control
3.  // WRITTEN BY:     Eric Cobane
4.  // COMMENTED BY:   Ryan Giovacchini
5.  // DATE COMPLETED: February 28, 2013
6.  //
7.  // PURPOSE:
8.  // This program allows the user to control the TCR12's motions using a
9.  // computer keyboard.
10. //
11. // FUNCTIONS:
12. //
13. // initialize
14. //      will initialize the system, setting up the correct input and output
15. //      ports on the Arduino
16. //
17. // main
18. //      will continously loop waiting for user input from the computer
19. //      keyboard
20. //
21. // startPosition
22. //      will allow the user to get robot attached to the climbing surface
23. //      in its starting position
24. //
25. // lowerStep
26. //      will open the lower gripper, then will move lower chassis segment
27. //      according to direction, then will close the lower gripper in it's
28. //      new position
29. //
30. // upperStep
31. //      will open the upper gripper, then will move upper chassis segment
32. //      according to direction, then will close the upper gripper in it's
33. //      new position
34. //
35. // turn
36. //      will turn the robot left or right about the z-axis
37. //
38. // roll
39. //      will turn the robot left or right about the y-axis
40. //
41. // activateGripper
42. //      will open or close the gripper specified
43. //
44. // makeContact
45. //      will rotate the chassis around the x-axis until the specified
46. //      gripper is in contact with the tree's surface
47. //
48. // slowServo
49. //      will control the speed of a specified servo from its current
50. //      position to a desired position
51. //
52. // INCLUDED FILES:
53. //   Servo.h
54. */
55.
56. #include <Servo.h>
57.
```

```
58.  // Set up each of the five servos and two linear actuators:
59.  Servo upperGripperActuator;      // The linear actuator for upper gripper
60.  Servo lowerGripperActuator;      // The linear actuator for lower gripper
61.  Servo upperWristServo;           // The servo for rotating the upper gripper platform
62.  Servo lowerWristServo;           // The servo for rotating the lower gripper platform
63.  Servo chassisServoX;             // The chassis servo for rotation about the x-axis
64.  Servo chassisServoY;             // The chassis servo for rotation about the y-axis
65.  Servo chassisServoZ;             // The chassis servo for rotation about the z-axis
66.
67.  // Define constants for upper and lower linear actuators boundries
68.  const int upperGripperOpen = 1300;      // Value to open upper gripper in Âµs
69.                                          //(microseconds)
70.  const int upperGripperClosed = 1125;    // Value to close upper gripper in Âµs
71.                                          //(microseconds)
72.  const int lowerGripperOpen = 1275;      // value to open lower gripper in Âµs
73.                                          //(microseconds)
74.  const int lowerGripperClosed = 1100;    // value to close lower gripper in Âµs
75.                                          //(microseconds)
76.
77.  // Define constants for servo centers and boundaries
78.  const int xServoCenter = 1450;       // Value to center chassisServoX in Âµs
79.                                          //(microseconds)
80.  const int yServoCenter = 1750;       // Value to center chassisServoY in Âµs
81.                                          //(microseconds)
82.  const int zServoCenter = 1300;       // Value to center chassisServoX in Âµs
83.                                          //(microseconds)
84.  const int upperServoCenter = 1500;   // Value to center upperWristServo in Âµs
85.                                          //(microseconds)
86.  const int lowerServoCenter = 1650;   // Value to center lowerWristServo in Âµs
87.                                          //(microseconds)
88.
89.  // Define constants for servo positions for lower step
90.  const int xServoLS = 800;        // Value for chassisServoX to make Lower Step in Âµs
91.                                          //(microseconds)
92.  const int lowerServoLS = 1450;   // Value to for bottomWristServo to make Lower Step in
93.                                          // Âµs(microseconds)
94.  const int upperServoLS = 1100;   // Value to for topWristServo to make Lower Step in
95.                                          // Âµs(microseconds)
96.
97.  // Global variables to keep track of servos' current position
98.  int upperServoPos;  // Variable to hold position for upperWristServo in Âµs
99.                      //(microseconds)
100.     int lowerServoPos;  // Variable to hold position for lowerWristServo in Âµs
101.                         //(microseconds)
102.     int xServoPos;      // Variable to hold position for chassisServoX in Âµs
103.                         //(microseconds)
104.     int yServoPos;      // Variable to hold position for chassisServoY in Âµs
105.                         //(microseconds)
106.     int zServoPos;      // Variable to hold position for chassisServoZ in
107.                         // Âµs(microseconds)
108.
109.     // Variable used for speed control
110.     int lowerIncrement = 10;                      // Amount to increment
111.                                                   // lowerWristServo
112.                                                   // during slowServo()
113.     double upperIncrement = lowerIncrement * 2.2; // Amount to increment
114.                                                   // upperWristServo
115.                                                   // during slowServo()
116.     double xIncrement = lowerIncrement * 3.5;     // Amount to increment
117.                                                   // xWristServo during
118.                                                   // slowServo()
```

```
119.
120.        // Constant values for program
121.        const int up = 1;       // Used to signify robot is climbing
122.        const int down = 0;     // Used to signify robot is descending
123.        const int upper = 1;    // Used as parameter to choose upper gripper to operate
124.                                // on, in activateGripper() and makeContact()
125.        const int lower = 0;    // Used as parameter to choose lower gripper to operate
126.                                // on, in activateGripper() and makeContact()
127.
128.        // Variables used for program
129.        char userInput;         // Variable for reading serial data from user input
130.        int vertDirection = 1;  // Variable to store which direction the robot is going
131.                                // vertically (used for turning and rolling functions)
132.
133.        // Function prototypes
134.        void initialize();
135.        void startPosition();
136.        void lowerStep(int direction);
137.        void upperStep(int direction);
138.        void turn(char direction);
139.        void roll(char direction);
140.        void activateGripper(int gripper, int pos );
141.        void makeContact(int gripper);
142.        void slowServo(int servoSelect, int startPoint, int setPoint, int increment);
143.
144.        void main() {
145.            initialize();
146.         // Wait for serial input
147.         if (Serial.available() > 0) {
148.            // Read the incoming byte:
149.            userInput = Serial.read();
150.         }
151.         /*Serial input description:
152.          *Key - Function
153.          *'b' - enable user to attach robot to tree
154.          *'w' - enable climbing gait
155.          *'s' - enable descending gait
156.          *'a' - turn chassis left about the z-axis
157.          *'d' - turn chassis right about the z-axis
158.          *'q' - roll chassis left about the y-axis
159.          *'e' - roll chassis right about the y-axis
160.          *ENTERING ANY OTHER KEY WILL STOP ALL MOTION
161.          */
162.         switch(userInput){
163.         case 'b':
164.           Serial.println("Attach Robot to Tree");
165.           startPosition();    // Allows the user to attach robot to the tree in start
166.                               // position
167.           userInput = Serial.read();
168.           break;
169.         case 'w':
170.           Serial.println("Climbing...");
171.           // Climb upward until the user inputs a new command
172.           while(userInput == 'w'){
173.             lowerStep(up);  // Lifts the lower chassis segment to perform first half
174.                             // of climbing sequence
175.             upperStep(up);  // Lifts the upper chassis segment to perform second half
176.                             // of climbing sequence
177.             userInput = Serial.read();
178.           }
179.           break;
```

```
180.          case 's':
181.             Serial.println("Descending...");
182.             // Descends until the user inputs a new command
183.             while(userInput == 's'){
184.               upperStep(down);  // Lowers the upper chassis segment to perform first
185.                                 // half of descending sequence
186.               lowerStep(down);  // Lowers the lower chassis segment to peroform second
187.                                 // half of descending sequence
188.               userInput = Serial.read();
189.             }
190.             break;
191.          case 'a':
192.             Serial.println("Turning Left...");
193.             // Turns the robot to the left until the user inputs a new command
194.             turn('left');  // Turns the chassis to the left about z-axis
195.             userInput = Serial.read();
196.             break;
197.          case 'd':
198.             Serial.println("Turning Right...");
199.             // Turns the robot to the right until the user inputs a new command
200.             turn('right');  // Turns the chassis to the right about z-axis
201.             userInput = Serial.read();
202.             break;
203.          case 'q':
204.             Serial.println("Rolling Left...");
205.             // Rolls the robot to the left until the user inputs a new command
206.             roll('left');  // Turns the chassis to the left about y-axis
207.             userInput = Serial.read();
208.             break;
209.          case 'e':
210.             Serial.println("Rolling Right...");
211.             // Rolls the robot to the right until the user inputs a new command
212.             roll('right');  // Turns the chassis to the right about y-axis
213.             userInput = Serial.read();
214.             break;
215.          default:
216.             // If no button is pressed then the robot stops
217.             Serial.println("Stopped");
218.             delay(500);
219.          }
220.       }
221.       /* name OF FUNCTION: initialize
222.       // CREDIT:
223.       // PURPOSE:
224.       //      Set desired pins on the Ardino to inputs and outputs, and start
225.       //      serial communication with the computer
226.       // PARAMETERS: none
227.       // RETURN VALUE: none
228.       // CALLS TO:
229.       //      attach()
230.       //      writeMicroseconds()
231.       //      Serial.begin()
232.       //      Serial.println()
233.       // CALLED FROM: main
234.       */
235.       void initialize() {
236.
237.         // Set pins as input
238.         pinMode(36, INPUT);  // The pushbuttons on upper gripper
239.         pinMode(38, INPUT);  // The pushbuttons on lower gripper
240.
```

```arduino
241.        // Set pins as outputs
242.        upperGripperActuator.attach(7);
243.        lowerGripperActuator.attach(8);
244.        upperWristServo.attach(3);
245.        lowerWristServo.attach(6);
246.        chassisServoX.attach(9);
247.        chassisServoY.attach(5);
248.        chassisServoZ.attach(4);
249.
250.        // Actuate robot into it's starting configuration
251.        upperWristServo.writeMicroseconds(upperServoCenter);
252.        lowerWristServo.writeMicroseconds(lowerServoCenter);
253.        chassisServoX.writeMicroseconds(xServoCenter);
254.        chassisServoY.writeMicroseconds(yServoCenter);
255.        chassisServoZ.writeMicroseconds(zServoCenter);
256.        upperGripperActuator.writeMicroseconds(upperGripperOpen);
257.        lowerGripperActuator.writeMicroseconds(lowerGripperOpen);
258.
259.        // Initiate serial communication
260.        Serial.begin(9600);
261.        Serial.println("Ready...");
262.        Serial.println();
263.      }
264.      /* name OF FUNCTION: startPosition
265.      // CREDIT:
266.      // PURPOSE:
267.      //      To allow the user to get robot attached to the climbing surface
268.      //      in its starting position
269.      //
270.      // PARAMETERS: none
271.      // RETURN VALUE: none
272.      // CALLS TO:
273.      //      writeMicroseconds
274.      //      activateGripper
275.      // CALLED FROM: main
276.      */
277.      void startPosition(){
278.
279.        // Configure robot in it's starting position
280.        upperWristServo.writeMicroseconds(upperServoCenter);
281.        lowerWristServo.writeMicroseconds(lowerServoCenter);
282.        chassisServoX.writeMicroseconds(xServoCenter);
283.        chassisServoY.writeMicroseconds(yServoCenter);
284.        chassisServoZ.writeMicroseconds(zServoCenter);
285.        upperGripperActuator.writeMicroseconds(upperGripperOpen);
286.        lowerGripperActuator.writeMicroseconds(lowerGripperOpen);
287.
288.        while(digitalRead(upperGripperButton) && digitalRead(lowerGripperButton)){
289.          // Wait for both gripper to be in contact with tree's surface
290.        }
291.        // Once both grippers are in contact, close them
292.        activateGripper(upper, upperGripperClosed);
293.        activateGripper(lower, lowerGripperClosed);
294.      }
295.       /* name OF FUNCTION: lowerStep
296.      // CREDIT:
297.      // PURPOSE:
298.      //      Will open the lower gripper, then will move lower chassis segment
299.      //      according to direction, then will close the lower gripper in it's
300.      //      new position
301.      //
```

```
302.        // PARAMETERS:
303.        // name                  type      description
304.        // ----------------------------------------------------------------
305.        // direction             int       Which direction the robot is going vertically
306.        //
307.        // RETURN VALUE: none
308.        // CALLS TO:
309.        //      writeMicroseconds
310.        //      activateGripper
311.        //      slowServo
312.        //      makeContact
313.        // CALLED FROM: main
314.        */
315.        void lowerStep(int direction){
316.          activateGripper(lower, lowerGripperOpen);  // Open the lower gripper
317.
318.          // Determine if robot is climbing
319.          if(direction == up){
320.            // Rotate  the nessacary servos to raise bottom chassis segment and set
321.            // position holder variables
322.
323.            // This while loop is used in conjunction with the slowServo function to
324.            // allow for the speed of the servos to be controlled
325.            while(upperServoPos != upperServoLS  || xServoPos != xServoLS || lowerServoP
    os != lowerServoLS){
326.              slowServo(1, upperServoPos, upperServoLS, upperIncrement);
327.              upperWristServo.writeMicroseconds(upperServoPos);
328.              slowServo(2, xServoPos, xServoLS, xIncrement);
329.              chassisServoX.writeMicroseconds(xServoPos);
330.              slowServo(3, lowerServoPos, lowerServoLS, lowerIncrement);
331.              lowerWristServo.writeMicroseconds(lowerServoPos);
332.              delay(50);
333.            }
334.            vertDirection = up;  //Set variable to remember robot is climbing
335.            // (used for turning functions)
336.          }
337.          // Determine if robot is descending
338.          if(direction == down){
339.            // Rotate the nessacary servos to lower bottom chassis segment and set
340.            // position holder variables
341.
342.            // This while loop is used in conjunction with the slowServo function to
343.            // allow for the speed of the servos to be controlled
344.            while(upperServoPos != upperServoCenter || xServoPos != xServoCenter || lowe
    rServoPos != lowerServoCenter){
345.              slowServo(1, upperServoPos, upperServoCenter, upperIncrement);
346.              upperWristServo.writeMicroseconds(upperServoPos);
347.              slowServo(2, xServoPos, xServoCenter, xIncrement);
348.              chassisServoX.writeMicroseconds(xServoPos);
349.              slowServo(3, lowerServoPos, lowerServoCenter, lowerIncrement);
350.              bottomWristServo.writeMicroseconds(lowerServoPos);
351.              delay(50);
352.            }
353.            vertDirection = down;   //Set variable to remember robot is descending
354.                                    // (used for turning functions)
355.          }
356.          delay(500);
357.          // Hold the lower gripper open until it has made contact with the climbing
358.          // surface
359.          makeContact(lower);
360.        }
```

```
361.        /* name OF FUNCTION: upperStep
362.        // CREDIT:
363.        // PURPOSE:
364.        //      Will open the upper gripper, then will move upper chassis segment
365.        //      according to direction, then will close the upper gripper in it's
366.        //      new position
367.        //
368.        // PARAMETERS:
369.        // name                  type     description
370.        // -----------------------------------------------------------------
371.        // direction             int      Which direction the robot is going vertically
372.        //
373.        // RETURN VALUE: none
374.        // CALLS TO:
375.        //      writeMicroseconds
376.        //      activateGripper
377.        //      slowServo
378.        //      makeContact
379.        // CALLED FROM: main
380.        */
381.        void upperStep(int direction){
382.          activateGripper(upper, upperGripperOpen);  // Open the upper gripper
383.
384.          // Determine if robot is climbing
385.          if(direction == up){
386.            // Rotate  the nessacary servos to raise upper chassis segment and set
387.            // position holder variables
388.
389.            // This while loop is used in conjunction with the slowServo function to
390.            // allow for the speed of the servos to be controlled
391.            while(upperServoPos != upperServoCenter || xServoPos != xServoCenter || lowe
    rServoPos != lowerServoCenter){
392.              slowServo(1, upperServoPos, upperServoCenter, upperIncrement);
393.              upperWristServo.writeMicroseconds(upperServoPos);
394.              slowServo(2, xServoPos, xServoCenter, xIncrement);
395.              chassisServoX.writeMicroseconds(xServoPos);
396.              slowServo(3, lowerServoPos, lowerServoCenter, lowerIncrement);
397.              lowerWristServo.writeMicroseconds(lowerServoPos);
398.              delay(50);
399.            }
400.            vertDirection = up;  //Set variable to remember robot is climbing
401.                                 //(used for turning functions)
402.          }
403.          // Determine if robot is climbing
404.          if(direction == down){
405.            // Rotate the nessacary servos to lower upper chassis segment and set
406.            // position holder variables
407.
408.            // This while loop is used in conjunction with the slowServo function to
409.            // allow for the speed of the servos to be controlled
410.            while(upperServoPos != upperServoLS || xServoPos != xServoLS || upperServoPo
    s != upperServoLS){
411.              slowServo(1, upperServoPos, upperServoLS, upperIncrement);
412.              upperWristServo.writeMicroseconds(upperServoPos);
413.              slowServo(2, xServoPos, xServoLS, xIncrement);
414.              chassisServoX.writeMicroseconds(xServoPos);
415.              slowServo(3, lowerServoPos, lowerServoLS, lowerIncrement);
416.              lowerWristServo.writeMicroseconds(lowerServoPos);
417.              delay(50);
418.            }
419.            vertDirection = down;  //Set variable to remember robot is descending (used
```

```
420.                                    //for turning functions)
421.          }
422.          delay(500);
423.          // Hold the upper gripper open until it has made contact with the climbing
424.          // surface
425.          makeContact(top);
426.        }
427.        /* name OF FUNCTION: turn
428.        // CREDIT:
429.        // PURPOSE:
430.        //      Will turn the robot left or right about the z-axis
431.        //
432.        // PARAMETERS:
433.        // name                 type     description
434.        // ----------------------------------------------------------------
435.        // direction            int      Which direction the robot is going vertically
436.        //
437.        // RETURN VALUE: none
438.        // CALLS TO:
439.        //      writeMicroseconds
440.        //      activateGripper
441.        //      makeContact
442.        // CALLED FROM: main
443.        */
444.        void turn(char direction){
445.          // Determine if the robot was previously climbing
446.          if(vertDirection == up){
447.            activateGripper(upper, upperGripperOpen);   // Open the upper gripper
448.            delay(200);
449.            xServoPos = xServoCenter + 100;             // Set position holder to value
450.                                                       // that holds gripper slightly
451.                                                       // off tree
452.            chassisServoX.writeMicroseconds(xServoPos); // Set position
453.          }
454.          // Determine if the robot was previously descending
455.          if(vertDirection == down){
456.            activateGripper(lower, lowerGripperOpen);   // Open the lower gripper
457.            delay(200);
458.            xServoPos = xServoCenter -
      100;              // Set position holder to value
459.                                                       // that holds gripper slightly
460.                                                       // off tree
461.            chassisServoX.writeMicroseconds(xServoPos); // Set position
462.
463.          }
464.          // Determine if the robot needs to turn left and activate servo to turn left
465.          if(direction == 'left'){
466.            zServoPos = zServoPos -
      10;                // Increment position holder by
467.                                                       // small angle
468.            chassisServoZ.writeMicroseconds(zServoPos); // Set position
469.          }
470.          // Determine if the robot needs to turn right and activate servo to turn right
471.          if(direction == 'right'){
472.            zServoPos = zServoPos + 10;                 // Increment position holder by
473.                                                       // small angle
474.            chassisServoZ.writeMicroseconds(zServoPos); // Set position
475.          }
476.          // Determine if a new turning commands been entered, if not make contact with
477.          // the gripper
478.          if(Serial.available() > 0 && Serial.read() != 'a' && Serial.read() != 'd'){
```

```
479.          if(vertDirection == up){
480.             makeContact(upper);
481.          }
482.          else makeContact(lower);
483.       }
484.    }
485.    /* name OF FUNCTION: roll
486.    // CREDIT:
487.    // PURPOSE:
488.    //      Will turn the robot left or right about the y-axis
489.    //
490.    // PARAMETERS:
491.    // name                 type      description
492.    // -----------------------------------------------------------------
493.    // direction            int       Which direction the robot is going vertically
494.    //
495.    // RETURN VALUE: none
496.    // CALLS TO:
497.    //      writeMicroseconds
498.    //      activateGripper
499.    //      makeContact
500.    // CALLED FROM: main
501.    */
502.    void roll(char direction){
503.       // Determine if the robot was previously climbing
504.       if(vertDirection == up){
505.          activateGripper(upper, upperGripperOpen);   // Open the upper gripper
506.          delay(200);
507.          xServoPos = xServoCenter + 100;             // Set position holder to value
508.                                                      // that holds gripper slightly
509.                                                      // off the tree
510.          chassisServoX.writeMicroseconds(xServoPos); // Set position
511.       }
512.       // Determine if the robot was previously descending
513.       if(vertDirection == down){
514.          activateGripper(lower, lowerGripperOpen);   // Open the lower gripper
515.          delay(200);
516.          xServoPos = xServoCenter -
    100;             // Set position holder to value
517.                                                      // that holds gripper slightly
518.                                                      // off the tree
519.          chassisServoX.writeMicroseconds(xServoPos); // Set position
520.       }
521.       // Determine if the robot needs to turn left and activate servo to turn left
522.       if(direction == 'left'){
523.          yServoPos = yServoPos + 10;                 // Increment position holder by
524.                                                      // small angle
525.          chassisServoY.writeMicroseconds(yServoPos);   // Set position
526.       }
527.       // Determine if the robot needs to turn right and activate servo to turn right
528.       if(direction == 'right'){
529.          yServoPos = yServoPos - 10;   //decrement position holder by small angle
530.          chassisServoY.writeMicroseconds(yServoPos);   //set position
531.       }
532.       // Determine if a new turning commands been entered, if not make contact with
533.       // the gripper
534.       if(Serial.available() > 0 && Serial.read() != 'a' && Serial.read() != 'd'){
535.          if(vertDirection == up){
536.             makeContact(upper);
537.          }
538.          else makeContact(lower);
```

```
539.          }
540.        }
541.        /* name OF FUNCTION: activateGripper
542.        // CREDIT:
543.        // PURPOSE:
544.        //      Will open or close the selected gripper
545.        //
546.        // PARAMETERS:
547.        // name                  type        description
548.        // -------------------------------------------------------------------
549.        // gripper            int          Which linear actuator is being selected
550.        // pos                int          The value the linear actuator is to be set to
551.        //
552.        // RETURN VALUE: none
553.        // CALLS TO:
554.        //      writeMicroseconds
555.        // CALLED FROM: makeContact
556.        */
557.        void activateGripper(int gripper, int pos ){
558.
559.          // Determine which linear actuator is being selected
560.          if(gripper == upper){
561.            upperGripperActuator.writeMicroseconds(pos);
562.            delay(1500);   // Delay to ensure the gripper has time to open/close
563.          }
564.          // Determine which linear actuator is being selected
565.          if(gripper == lower){
566.            lowerGripperActuator.writeMicroseconds(pos);
567.            delay(1500);   // Delay to ensure the gripper has time to open/close
568.          }
569.        }
570.        /* name OF FUNCTION: makeContact
571.        // CREDIT:
572.        // PURPOSE:
573.        //      Will open the upper gripper, then will move upper chassis segment
574.        //      according to direction, then will close the upper gripper in it's
575.        //      new position
576.        //
577.        // PARAMETERS:
578.        // name                  type        description
579.        // -------------------------------------------------------------------
580.        // gripper            int              Which gripper pushbutton is being monitored
581.        //
582.        // RETURN VALUE: none
583.        // CALLS TO:
584.        //      writeMicroseconds
585.        // CALLED FROM:
586.        //      lowerStep
587.        //      upperStep
588.        //      turn
589.        //      roll
590.        */
591.        void makeContact(int gripper){
592.          // Determine the pushbutton is being monitored
593.          if(gripper == upper){
594.            //Check that the corresponding gripper is in contact before re-closing
595.            // the gripper,
596.            // while gripper is not in contact, slowly move gripper closer to the tree
597.            while(digitalRead(upperGripperButton)){
598.              upperServoPos = upperServoPos -
     5;              // Decrement position holder
```

117

```
599.                                                        // by small angle
600.            xServoPos = xServoPos -
     5;                        // Decrement position holder
601.                                                        // by small angle
602.            lowerServoPos = lowerServoPos -
     5;                    // Decrement position holder
603.                                                        // by small angle
604.            upperWristServo.writeMicroseconds(upperServoPos); // Set position
605.            chassisServoX.writeMicroseconds(xServoPos);       // Set position
606.            lowerWristServo.writeMicroseconds(lowerServoPos); // Set position
607.            delay(500);
608.          }
609.          // Increment the servos an additonal time to increase torque while closing
610.          // the gripper
611.          xServoPos = xServoPos -
     20;                       // Increment position holder
612.                                                        // by small angle
613.          lowerServoPos = lowerServoPos -
     10;                  // Increment position holder
614.                                                        // by small angle
615.          chassisServoX.writeMicroseconds(xServoPos);       // Set position
616.          lowerWristServo.writeMicroseconds(lowerServoPos);  // Set position
617.          activateGripper(upper, upperGripperClosed);        // Close upper gripper
618.        }
619.        if(gripper == lower){
620.          //Check that the corresponding gripper is in contact before
621.          // re-closing the gripper,
622.          // while gripper is not in contact, slowly move gripper closer to the tree
623.          while(digitalRead(lowerGripperButton)){
624.            upperServoPos = upperServoPos + 5;                    // Increment position
625.                                                                 // holder by
626.                                                                 // small angle
627.            xServoPos = xServoPos + 5;                           // Increment position
628.                                                                 // holder by
629.                                                                 // small angle
630.            lowerServoPos = lowerServoPos + 5;                   // Increment position
631.                                                                 // holder by
632.                                                                 // small angle
633.            upperWristServo.writeMicroseconds(upperServoPos); // Set position
634.            chassisServoX.writeMicroseconds(xServoPos);       // Set position
635.            lowerWristServo.writeMicroseconds(lowerServoPos); // Set position
636.            delay(500);
637.          }
638.          // Increment the servos an additonal time to increase torque while closing
639.          // gripper
640.          xServoPos = xServoPos + 20;                          // Increment position
641.                                                              // holder by
642.                                                              // small angle
643.          lowerServoPos = lowerServoPos + 10;                 // Increment position
644.                                                              // holder by
645.                                                              // small angle
646.          chassisServoX.writeMicroseconds(xServoPos);       // Set position
647.          lowerWristServo.writeMicroseconds(lowerServoPos);  // Set position
648.          activateGripper(lower, lowerGripperClosed);        // Close lower gripper
649.        }
650.      }
651.      /* name OF FUNCTION: slowServo
652.      // CREDIT:
653.      // PURPOSE:
654.      //      Will control the speed of a specified servo from its current
655.      //      position to a desired position by changing the value of global
```
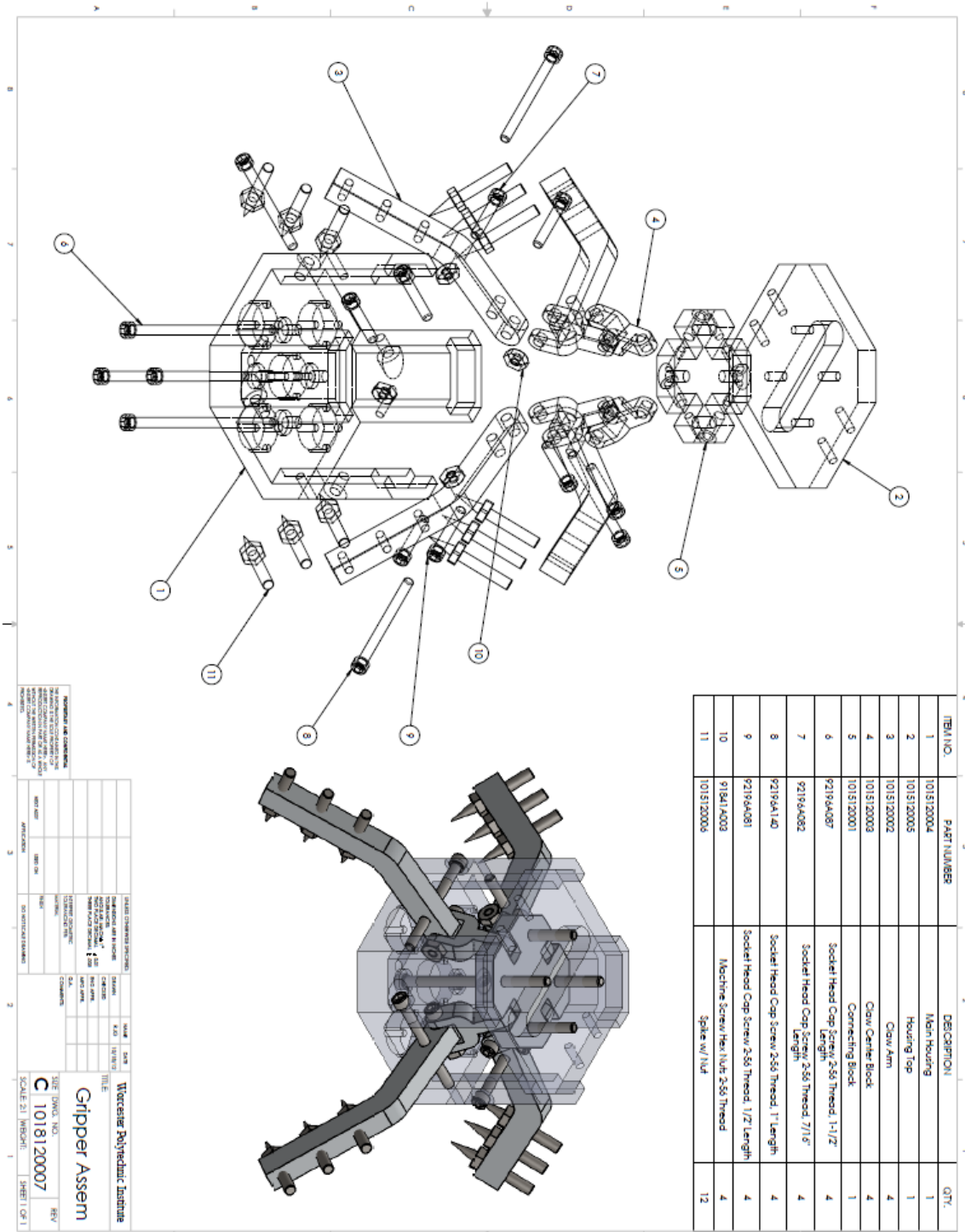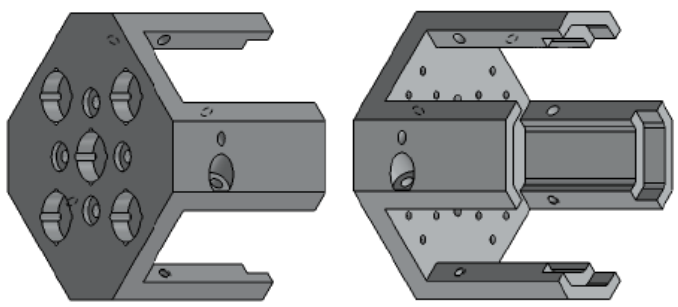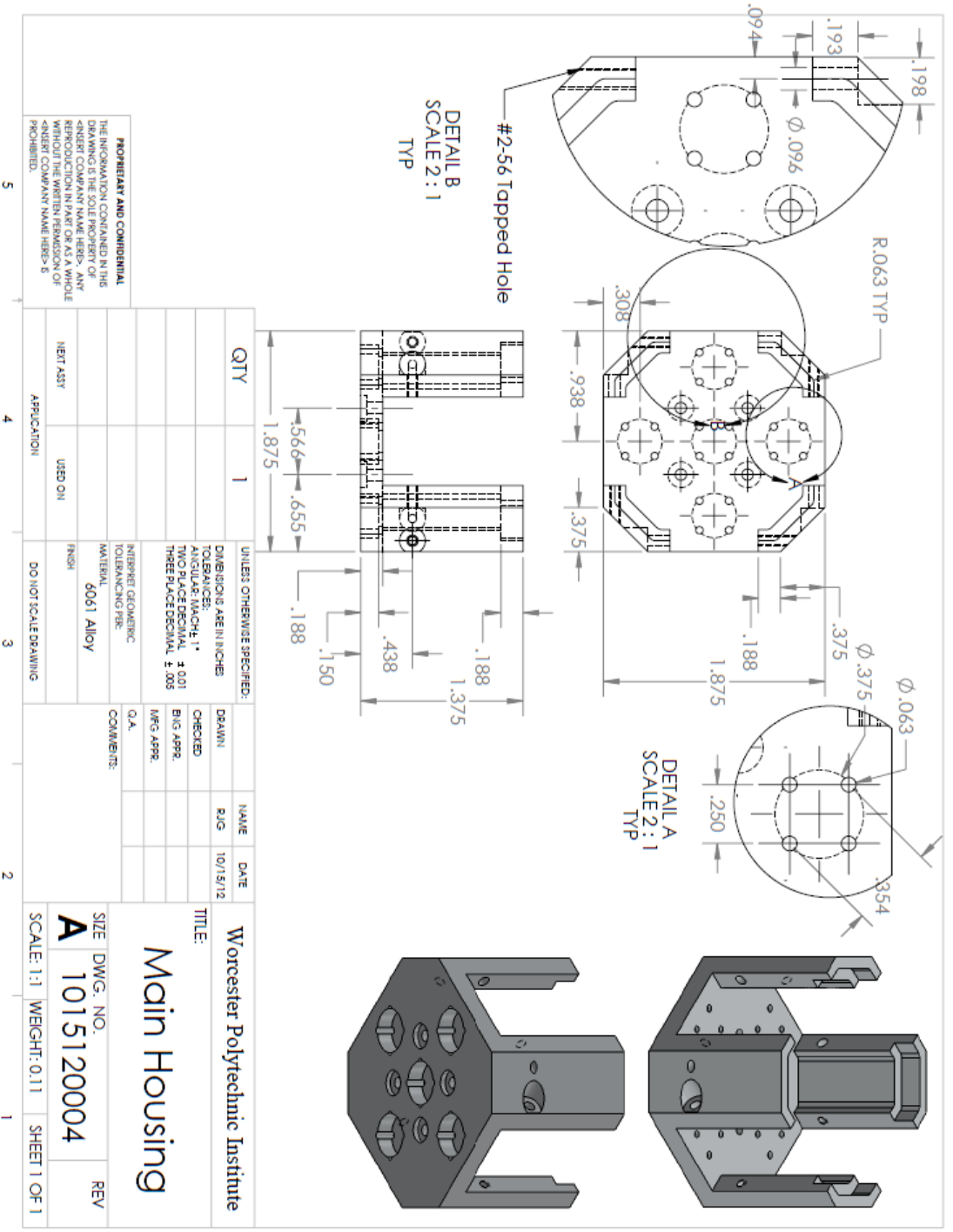
```
656.        //      variables
657.        //
658.        // PARAMETERS:
659.        // name                type      description
660.        // ----------------------------------------------------------------
661.        // servoSelect         int       Which servo is being selected
662.        // startPoint          int       The starting point for servo
663.        // setPoint            int       The new desired servo position
664.        // increment           int       How large of a step the servo positon changes
665.        //
666.        // RETURN VALUE: none
667.        // CALLS TO: none
668.        // CALLED FROM:
669.        //      lowerStep
670.        //      upperStep
671.        //      turn
672.        //      roll
673.        */
674.        void slowServo(int servoSelect, int startPoint, int setPoint, int increment){
675.          // Determines if the startPoint is less then the setPoint and that the servo
676.          //  won't overshoot the setPoint
677.          if (startPoint < setPoint  && startPoint > (setPoint - increment)){
678.            int difference = setPoint - startPoint;
679.            startPoint += difference;
680.          }
681.          else{
682.            startPoint +=increment;
683.          }
684.          // Determines if the startPoint is greater then the setPoint and that the
685.          // servo won't overshoot the setPoint
686.          if (startPoint > setPoint && startPoint < (setPoint + increment)){
687.            int difference = startPoint - setPoint;
688.            startPoint -=difference;
689.          }
690.          else{
691.            startPoint -=increment;
692.          }
693.          // Selects the desired servo and sets the global variable for that servo's
694.          // position
695.          switch(servoSelect){
696.            case 1:
697.            upperServoPos = startPoint;
698.            break;
699.            case 2:
700.            xServoPos = startPoint;
701.            break;
702.            case 3:
703.            lowerServoPos = startPoint;
704.            break;
705.          }
706.        }
```

12. Appendix F: Solidworks Drawings

12.1 Prototype Gripper

ITEM NO.	PART NUMBER	DESCRIPTION	QTY.
1	1015120004	Main Housing	1
2	1015120005	Housing Top	1
3	1015120002	Claw Arm	4
4	1015120003	Claw Center Block	4
5	1015120001	Connecting Block	1
6	92196A087	Socket Head Cap Screw 2-56 Thread, 1-1/2" Length	4
7	92196A082	Socket Head Cap Screw 2-56 Thread, 7/16" Length	4
8	92196A140	Socket Head Cap Screw 2-56 Thread, 1" Length	4
9	92196A081	Socket Head Cap Screw 2-56 Thread, 1/2" Length	4
10	91841A003	Machine Screw Hex Nuts 2-56 Thread	4
11	1015120006	Spike w/ Nut	12

Worcester Polytechnic Institute

TITLE: Gripper Assem

SIZE C   DWG. NO. 101812007   REV

SCALE: 2:1   WEIGHT: SHEET 1 OF 1

DETAIL B
SCALE 2 : 1
TYP

#2-56 Tapped Hole

.094

.193

.198

Ø.096

R.063 TYP

.308

.938

.375

.566
.655

1.875

.188

.150

.438

.188

1.375

1.875

.188

.375

Ø.375

Ø.063

DETAIL A
SCALE 2 : 1
TYP

.250

.354

	QTY
	1

NEXT ASSY | USED ON

APPLICATION

UNLESS OTHERWISE SPECIFIED:

DIMENSIONS ARE IN INCHES
TOLERANCES:
ANGULAR: MACH ± 1°
TWO PLACE DECIMAL   ± .01
THREE PLACE DECIMAL ± .005

INTERPRET GEOMETRIC
TOLERANCING PER:

MATERIAL
6061 Alloy

FINISH

DO NOT SCALE DRAWING

	NAME	DATE
DRAWN	RJG	10/15/12
CHECKED		
ENG APPR.		
MFG APPR.		
Q.A.		
COMMENTS:		

Worcester Polytechnic Institute

TITLE:

Main Housing

SIZE | DWG. NO. | REV
A | 1015120004 |

SCALE: 1:1 | WEIGHT: 0.11 | SHEET 1 OF 1

Material Thickness 3/16

#2-56 Tapped Through

R.063 TYP

1.000

.566

.561

.561

45.00°

R.125

.320

1.688

UNLESS OTHERWISE SPECIFIED:

DIMENSIONS ARE IN INCHES
TOLERANCES:
ANGULAR: MACH ± 1°
TWO PLACE DECIMAL   ± .001
THREE PLACE DECIMAL ± .005

INTERPRET GEOMETRIC
TOLERANCING PER:

MATERIAL
6061 Alloy

FINISH

DO NOT SCALE DRAWING

	NAME	DATE
DRAWN	RJG	10/15/12
CHECKED		
ENG. APPR.		
MFG. APPR.		
Q.A.		
COMMENTS:		

QTY	1

NEXT ASSY   USED ON

APPLICATION

Worcester Polytechnic Institute

TITLE:

Housing Top

SIZE DWG. NO.   REV
A   1015120005

SCALE: 2:1   WEIGHT: 0.04   SHEET 1 OF 1

5   4   3   2   1

DETAIL A TYP
SCALE 4 : 1

⌀ .096 ▽ .455
⌀ .160 ▽ .200
2-56 UNC THRU ALL
⌀ .070 THRU ALL

#2-56 Tapped Hole

.151
.250
.125

1.061
1.061
.250
45.00°

4X ⌀ .096 THRU ALL

.188

| QTY | 1 |

NEXT ASSY	USED ON
APPLICATION	DO NOT SCALE DRAWING

UNLESS OTHERWISE SPECIFIED:

DIMENSIONS ARE IN INCHES
TOLERANCES:
ANGULAR: MACH ± 1°
TWO PLACE DECIMAL   ± .01
THREE PLACE DECIMAL   ± .005

INTERPRET GEOMETRIC
TOLERANCING PER:

MATERIAL
6061 Alloy

FINISH

	NAME	DATE
DRAWN	RJG	10/15/12
CHECKED		
ENG APPR.		
MFG APPR.		
Q.A.		
COMMENTS:		

Worcester Polytechnic Institute

TITLE:

Connecting Block

SIZE	DWG. NO.	REV
A	1015120001	

SCALE: 2:1 | WEIGHT: 0.01 | SHEET 1 OF 1

5 4 3 2 1

.375

.078TYP

.219 .250

R.094

.400

.750

3X ∅ .096

.558

R.125

.125

.188

QTY		4				UNLESS OTHERWISE SPECIFIED:				NAME	DATE		Worcester Polytechnic Institute
						DIMENSIONS ARE IN INCHES		DRAWN		RJG	10/15/12		
						TOLERANCES:		CHECKED					TITLE:
NEXT ASSY		USED ON				ANGULAR: MACH± 1°		ENG APPR.					
						TWO PLACE DECIMAL ± .01		MFG APPR.					Y Block
APPLICATION						THREE PLACE DECIMAL ± .005							
						INTERPRET GEOMETRIC		Q.A.					
						TOLERANCING PER:		COMMENTS:					SIZE DWG. NO. REV
						MATERIAL							A 10151 20003
						6061 Alloy							
						FINISH							SCALE: 4:1 WEIGHT: 0.00 SHEET 1 OF 1
						DO NOT SCALE DRAWING							

5

4

3

2

1

124

R.125

.995

2X Ø .096

R.250 TYP

.250

.750

2.061

140.00°

2.083

1.250

.250

.188

#2-56 Tapped Hole 3x

QTY				UNLESS OTHERWISE SPECIFIED:		NAME	DATE
4				DIMENSIONS ARE IN INCHES:	DRAWN	RJG	10/15/12
				TOLERANCES:	CHECKED		
				ANGULAR: MACH± 1°	ENG APPR.		
				TWO PLACE DECIMAL ± .001	MFG APPR.		
				THREE PLACE DECIMAL ± .005			
NEXT ASSY		USED ON		INTERPRET GEOMETRIC	Q.A.		
APPLICATION		DO NOT SCALE DRAWING		TOLERANCING PER:	COMMENTS:		
				MATERIAL:			
				6061 Alloy			
				FINISH			

Worcester Polytechnic Institute

TITLE:

Claw Arm

SIZE	DWG. NO.		REV
A	101512002		
SCALE: 2:1	WEIGHT: 0.01	SHEET 1 OF 1	

5   4   3   2   1

#2-56 Machine Threads

.217
.109
.188
Ø.096

.250
.070
.500
45.00°
.005

QTY	
12	

NEXT ASSY | APPLICATION | USED ON

UNLESS OTHERWISE SPECIFIED:

DIMENSIONS ARE IN INCHES
TOLERANCES:
ANGULAR: MACH± 1°
TWO PLACE DECIMAL   ± .01
THREE PLACE DECIMAL ± .005

INTERPRET GEOMETRIC
TOLERANCING PER:

MATERIAL
AISI 4130 Steel, normalized at 870C

FINISH

DO NOT SCALE DRAWING

	NAME	DATE
DRAWN	RJG	10/15/12
CHECKED		
ENG APPR.		
MFG APPR.		
Q.A.		
COMMENTS:		

Worcester Polytechnic Institute

TITLE:

Spike w/ Nut

SIZE: A | DWG. NO. 101512006 | REV

SCALE: 4:1 | WEIGHT: 0.00 | SHEET 1 OF 1

5 | 4 | 3 | 2 | 1

12.2 Prototype Robot Chassis

DETAIL B
SCALE 2 : 1

SECTION C-C

DETAIL A
SCALE 2 : 1

#2-56 Tapped Hole

#2-56 Tapped Hole

Ø.063 TYP
Ø.375
Ø.375

1.937
R.094
Ø.096
.188
.350
1.284

1.875
.566
.655
.150
.188
.438
.188
1.375

.094
Ø.096
.193
.198
.375
.188
.250
.354

Worcester Polytechnic Institute

TITLE:

Main Housing

| SIZE | DWG. NO. | REV |
| A | 12031 2011 | 2 |

SCALE: 1:1 | WEIGHT: 51.704 | SHEET 1 OF 1

	NAME	DATE
DRAWN	RJG	12/03/12
CHECKED		
ENG APPR.		
MFG APPR.		
Q.A.		
COMMENTS:		

UNLESS OTHERWISE SPECIFIED:

DIMENSIONS ARE IN INCHES
TOLERANCES:
ANGULAR: MACH± 1°
TWO PLACE DECIMAL ±0.01
THREE PLACE DECIMAL ±.005

INTERPRET GEOMETRIC
TOLERANCING PER:

MATERIAL
6061 Alloy

FINISH

DO NOT SCALE DRAWING

| QTY | | |
| 2 | | |

NEXT ASSY | USED ON

APPLICATION

5 4 3 2 1

Material Thickness 3/16

4X #2-56 Tapped Through

Ø.600

.566

.844

.561

45.00°

R.063 TYP

.561

.844

1.127

1.688

DETAIL A TYP
SCALE 4 : 1

.300

.018

2X R.031

2X R.063

.125

.105

2X #2-56 Tapped

		QTY	2

NEXT ASSY

USED ON

APPLICATION

DO NOT SCALE DRAWING

UNLESS OTHERWISE SPECIFIED:

DIMENSIONS ARE IN INCHES:
TOLERANCES:
ANGULAR: MACH± 1°
TWO PLACE DECIMAL   ±0.01
THREE PLACE DECIMAL ±.005

INTERPRET GEOMETRIC
TOLERANCING PER:

MATERIAL
6061 Alloy

FINISH

	NAME	DATE
DRAWN	RJG	12/03/12
CHECKED		
ENG APPR.		
MFG APPR.		
Q.A.		
COMMENTS:		

Worcester Polytechnic Institute

TITLE:

Main Housing Top

SIZE	DWG. NO.	REV
A	1203120 10	2

SCALE: 2:1   WEIGHT: 18.992   SHEET 1 OF 1

Ø.800

.135 .135 .250

.125

Ø.063

#2-56 Tapped Hole

DETAIL A TYP
SCALE 4 : 1

1.061

.250

45.00°

Ø.188

1.061

4X Ø.096 THRU ALL

.247

.247

.188

Ø .070 THRU ALL
2-56 UNC THRU ALL

QTY			
2			

NEXT ASSY	USED ON

APPLICATION

DO NOT SCALE DRAWING

UNLESS OTHERWISE SPECIFIED:

DIMENSIONS ARE IN INCHES
TOLERANCES:
ANGULAR: MACH± 1°
TWO PLACE DECIMAL   ±0.01
THREE PLACE DECIMAL ±.005

INTERPRET GEOMETRIC
TOLERANCING PER:

MATERIAL
6061 Alloy

FINISH

	NAME	DATE
DRAWN	RJG	12/03/12
CHECKED		
ENG APPR.		
MFG APPR.		
Q.A.		
COMMENTS:		

Worcester Polytechnic Institute

TITLE:

Connecting Block

SIZE	DWG. NO.	REV
A	12031 2002	2

SCALE: 2:1   WEIGHT: 5.926   SHEET 1 OF 1

5   4   3   2   1

R.094

.078 TYP

.354

.198

.250

.400

.750

R.125

.125

3X ⌀.096

.558

.188

QTY	8

NEXT ASSY	USED ON
APPLICATION	

UNLESS OTHERWISE SPECIFIED:

DIMENSIONS ARE IN INCHES
TOLERANCES:
ANGULAR: MACH± 1°
TWO PLACE DECIMAL ±0.01
THREE PLACE DECIMAL ±.005

INTERPRET GEOMETRIC
TOLERANCING PER:

MATERIAL
6061 Alloy

FINISH

DO NOT SCALE DRAWING

	NAME	DATE
DRAWN	RJG	12/03/12
CHECKED		
ENG APPR.		
MFG APPR.		
Q.A.		
COMMENTS:		

Worcester Polytechnic Institute

TITLE:

Y Block

SIZE A	DWG. NO. 120312001	REV 2
SCALE: 4:1	WEIGHT: 0.935	SHEET 1 OF 1

5 4 3 2 1

2X ∅ .096 THRU ALL

R.125

#2-56 Tapped Hole

.553

.250

.750

1.412

125.00°

R.250 TYP

.500

.250

.188

QTY		8
NEXT ASSY	USED ON	
APPLICATION		DO NOT SCALE DRAWING

UNLESS OTHERWISE SPECIFIED:

DIMENSIONS ARE IN INCHES
TOLERANCES:
ANGULAR: MACH± 1°
TWO PLACE DECIMAL   ±0.01
THREE PLACE DECIMAL ±.005

INTERPRET GEOMETRIC
TOLERANCING PER:

MATERIAL:
6061 Alloy

FINISH

	NAME	DATE
DRAWN	RJG	12/03/12
CHECKED		
ENG APPR.		
MFG APPR.		
Q.A.		
COMMENTS:		

Worcester Polytechnic Institute

TITLE:

Gripper Arm 125

SIZE	DWG. NO.	REV
A	120812002	2

SCALE: 2:1 WEIGHT: 2.882 SHEET 1 OF 1

5   4   3   2   1

131

2X Ø .096 THRU ALL

R.125

.513

.250

.750

1.508

140.100°

R.250 TYP

.500

.250

.125

#2-56 Tapped Hole

.188

QTY	8

UNLESS OTHERWISE SPECIFIED:			NAME	DATE
DIMENSIONS ARE IN INCHES	DRAWN	RJG	12/08/12
TOLERANCES:	CHECKED		
ANGULAR: MACH± 1°	ENG APPR.		
TWO PLACE DECIMAL ±0.01	MFG APPR.		
THREE PLACE DECIMAL ±.005	Q.A.		
	COMMENTS:		
INTERPRET GEOMETRIC TOLERANCING PER:			
MATERIAL			
6061 Alloy			
FINISH			

NEXT ASSY | USED ON

APPLICATION

DO NOT SCALE DRAWING

Worcester Polytechnic Institute

TITLE:

Gripper Arm 140

SIZE	DWG. NO.	REV
A	12081 2001	2

SCALE: 2:1 | WEIGHT: 2.963 | SHEET 1 OF 1

132

2X ∅ .096 THRU ALL

R.125

.250

.750

R.250 TYP

.500

155.00°

1.578

.438

.250

#2-56 Tapped Hole

.188

	NAME	DATE
DRAWN	RJG	12/03/12
CHECKED		
ENG APPR.		
MFG APPR.		
Q.A.		
COMMENTS:		

UNLESS OTHERWISE SPECIFIED:

DIMENSIONS ARE IN INCHES
TOLERANCES:
ANGULAR: MACH± 1°
TWO PLACE DECIMAL   ±0.01
THREE PLACE DECIMAL ±.005

INTERPRET GEOMETRIC
TOLERANCING PER:

MATERIAL
6061 Alloy

FINISH

DO NOT SCALE DRAWING

QTY	8

NEXT ASSY	USED ON

APPLICATION

Worcester Polytechnic Institute

TITLE:

Gripper Arm 155

SIZE **A** | DWG. NO. 120812003 | REV 2

SCALE: 2:1 | WEIGHT: 3.037 | SHEET 1 OF 1

5

4

3

2

1

133

.25

.50

ϕ .09

#2-56 Machine Threads

QTY		24

NEXT ASSY

USED ON

APPLICATION

DO NOT SCALE DRAWING

UNLESS OTHERWISE SPECIFIED:

DIMENSIONS ARE IN INCHES
TOLERANCES:
ANGULAR: MACH ± 1°
TWO PLACE DECIMAL   ±0.01
THREE PLACE DECIMAL ±.005

INTERPRET GEOMETRIC
TOLERANCING PER:

MATERIAL:
AISI 4130 Steel, normalized at 870C

FINISH

	NAME	DATE
DRAWN	RJG	12/03/12
CHECKED		
ENG APPR.		
MFG APPR.		
Q.A.		
COMMENTS:		

Worcester Polytechnic Institute

TITLE:

Spike

SIZE | DWG. NO. | REV
A | 12031 2007 | 2

SCALE: 4:1 | WEIGHT: 0.00 | SHEET 1 OF 1

134

.50
.19
.19
.38
.54
.72
1.43
.25
.10
R.06 TYP

.25

2X ∅ .10 THRU ALL
⌴ ∅ .16 ▽ .10

2X ∅ .15 THRU ALL
⌴ ∅ .23 ▽ .25

QTY			
2			

NEXT ASSY | USED ON
APPLICATION | DO NOT SCALE DRAWING
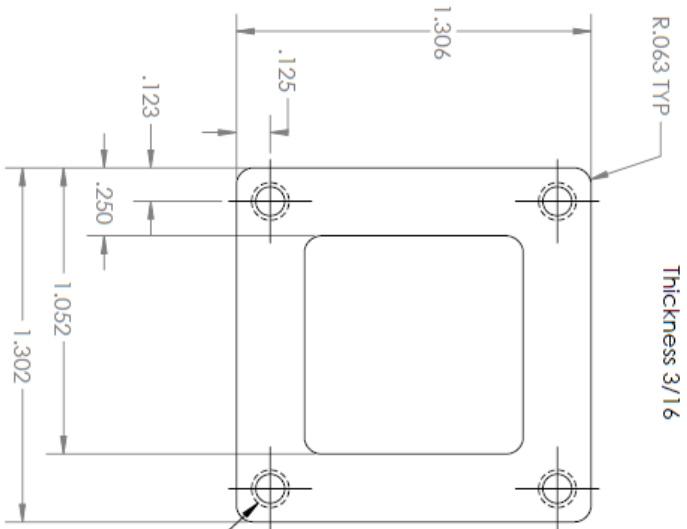
UNLESS OTHERWISE SPECIFIED:

DIMENSIONS ARE IN INCHES
TOLERANCES:
ANGULAR: MACH ± 1°
TWO PLACE DECIMAL   ± 0.01
THREE PLACE DECIMAL ± .005

INTERPRET GEOMETRIC
TOLERANCING PER:

MATERIAL
6061 Alloy

FINISH

	NAME	DATE
DRAWN	RJG	12/03/12
CHECKED		
ENG APPR.		
MFG APPR.		
Q.A.		
COMMENTS:		

Worcester Polytechnic Institute

TITLE:

Gripper Body Spacer

SIZE | DWG. NO. | REV
A | 12031 2003 | 1

SCALE: 2:1 | WEIGHT: 3.982 | SHEET 1 OF 1

Thickness 3/16

R.063 TYP

1.306

.125

.123

.250

1.052

1.302

4X #6-32 Tapped Hole

		QTY	1

NEXT ASSY	USED ON
APPLICATION	

DO NOT SCALE DRAWING

UNLESS OTHERWISE SPECIFIED:

DIMENSIONS ARE IN INCHES
TOLERANCES:
ANGULAR: MACH± 1°
TWO PLACE DECIMAL   ±0.01
THREE PLACE DECIMAL ±.005

INTERPRET GEOMETRIC
TOLERANCING PER:

MATERIAL
6061 Alloy

FINISH

	NAME	DATE
DRAWN	RJG	12/03/12
CHECKED		
ENG APPR.		
MFG APPR.		
Q.A.		
COMMENTS:		

Worcester Polytechnic Institute

TITLE:

Servo Block Bottom Custom

SIZE DWG. NO.   REV
A   120312005   1

SCALE: 2:1 WEIGHT: 0.02   SHEET 1 OF 1

137

UNLESS OTHERWISE SPECIFIED:

DIMENSIONS ARE IN INCHES
TOLERANCES:
ANGULAR: MACH± 1°
TWO PLACE DECIMAL   ±0.01
THREE PLACE DECIMAL ±.005

INTERPRET GEOMETRIC
TOLERANCING PER:

MATERIAL
6061 Alloy

FINISH

DO NOT SCALE DRAWING

	NAME	DATE
DRAWN	RJG	12/03/12
CHECKED		
ENG APPR.		
MFG APPR.		
Q.A.		
COMMENTS:		

Worcester Polytechnic Institute

TITLE:

Servo Block Guide

SIZE	DWG. NO.	REV
A	12031 2006	1

SCALE: 1:1   WEIGHT: 0.03   SHEET 1 OF 1

2.40

2.15

4X R.13

6X R.10

4X Ø .11 THRU ALL

.19 TYP

.49
.69
1.01
1.57

.25
.35
.82
1.09

2X R.06

.09

.71

QTY
1

NEXT ASSY   USED ON
APPLICATION

138

1.56
1.41
1.06
.72
.47
.19

R.06 TYP

.19

1.00
.84
.69
.50
.31
.16

2X Ø.07 ▼.50
2-56 UNC ▼.17

4X Ø.11 THRU ALL

QTY		
	1	

NEXT ASSY | USED ON
APPLICATION

DO NOT SCALE DRAWING
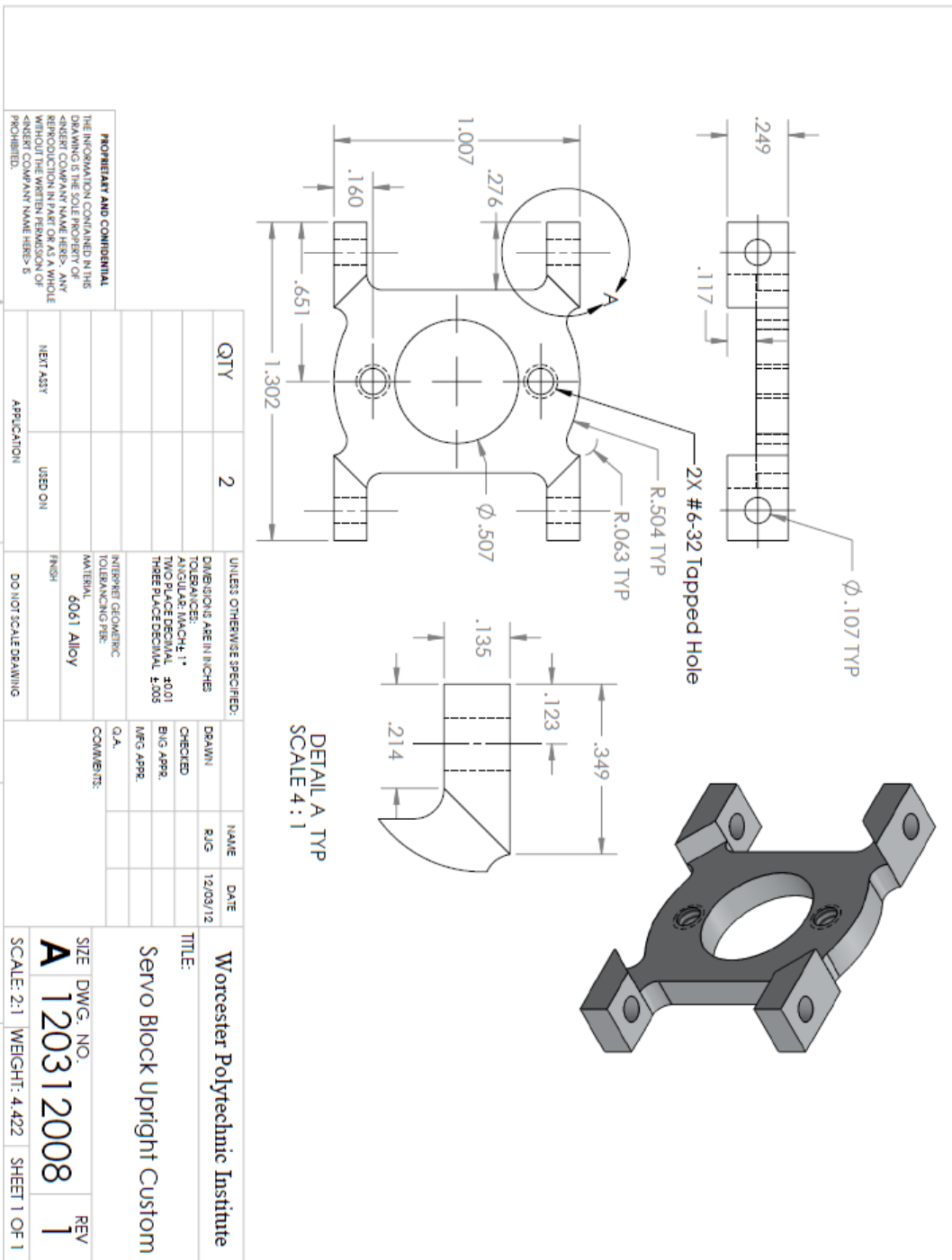
UNLESS OTHERWISE SPECIFIED:

DIMENSIONS ARE IN INCHES
TOLERANCES:
ANGULAR: MACH± 1°
TWO PLACE DECIMAL   ±0.01
THREE PLACE DECIMAL ±.005

INTERPRET GEOMETRIC
TOLERANCING PER:

MATERIAL:
6061 Alloy

FINISH:

	NAME	DATE
DRAWN	RJG	12/03/12
CHECKED		
ENG APPR.		
MFG APPR.		
Q.A.		
COMMENTS:		

Worcester Polytechnic Institute

TITLE:

Servo Block Top Guide

SIZE **A** | DWG. NO. 120312009 | REV 1

SCALE: 2:1 | WEIGHT: 10.757 | SHEET 1 OF 1

5   4   3   2   1

1.007

.160

.276

.651

1.302

2X #6-32 Tapped Hole

R.504 TYP

R.063 TYP

Ø.507

Ø.107 TYP

.249

.117

DETAIL A TYP
SCALE 4 : 1

.135

.214

.123

.349

QTY		
		2

NEXT ASSY | APPLICATION | USED ON

DO NOT SCALE DRAWING

UNLESS OTHERWISE SPECIFIED:		NAME	DATE			
DIMENSIONS ARE IN INCHES	DRAWN	RJG	12/03/12			
TOLERANCES:	CHECKED					
ANGULAR: MACH ± 1°	ENG APPR.					
TWO PLACE DECIMAL ± 0.01	MFG APPR.					
THREE PLACE DECIMAL ± .005	Q.A.					
	COMMENTS:					

INTERPRET GEOMETRIC
TOLERANCING PER:

MATERIAL
6061 Alloy

FINISH

Worcester Polytechnic Institute

TITLE:

Servo Block Upright Custom

SIZE A   DWG. NO. 12031 2008   REV 1

SCALE: 2:1   WEIGHT: 4.422   SHEET 1 OF 1

5   4   3   2   1

140

Bibliography

Aracil, R., R. J. Saltaren, and O. Reinoso. "A Climbing Parallel Robot: A Robot to Climb Along Tubular and Metallic Structures." *Robotics & Automation Magazine, IEEE* 13.1 (2006): 16-22.

Automation, Robotics, and Computer Vision Laboratory, Miguel Hernández University. "TREPA: Palmtree Climbing Teleoperated Robot." 2006. <http://arvc.umh.es/proyectos/trepa/index.php?type=proy&dest=inicio&lang=en&idp=trepa&ficha=on>.

Biorobotics Laboratory, Carnegie Mellon University. "Snake Robots." 2008. <http://www.cs.cmu.edu/~biorobotics//>.>.

Kawasaki & Mouri Lab, Gifu University. "Research on developing the pruning robot." 2011. <http://robo.mech.gifu-u.ac.jp/content/research/research_pruning_e.html>.

Kenny, Daniel. "Asian Longhorned Beetle." 2011. <http://www.agri.ohio.gov/topnews/asianbeetle/>.

Kod*lab, University of Pennsylvania. "RiSE Robotic Hexapod Version 2.0." 2012. <http://kodlab.seas.upenn.edu/RiSE/RiSEV2>.

---. "RiSE V1 Climbing Robot." 2012. <http://kodlab.seas.upenn.edu/RiSE/RiSEV1>.

---. "RiSE Version 3 Prototype." 2012. <http://kodlab.seas.upenn.edu/RiSE/RiSEV3>.

Lam, Tin, and Yangsheng Xu. *Tree Climbing Robot: Design, Kinematics and Motion Planning*. Eds. Bruno Siciliano and Oussama Khatib. 1st ed. New York: Springer-Verlag Berlin Heidelberg 2012, 2012.

Nisley, Rebecca G. "New Pheromone Traps Lure Asian Longhorned Beetles Out of Hiding." *US Forest Service Northern Research Station*.15 (2012).

Reardon, Brendon. "Plant Health: Asian Longhorned Beetle." *USDA Animal and Plant Health Inspection Service* (2012)

Sawyer, Alan. "Annotated Categorization of ALB Host Trees." 2010. <http://www.uvm.edu/albeetle/hosts.htm>.

Seirei Industry Co. "Seirei Industry [Forestry Machinery]." <http://www.seirei.com/en/products/frst.html>.

Sugano Laboratory, Waseda University, Japan. "Robot Assisting Forestry Work." 2003. <http://www.sugano.mech.waseda.ac.jp/project/forest/index.html>.

Todd, Drew. "Asian Longhorned Beetle." 2008.
<http://ohiodnr.com/DNN/health/asianlonghorned/tabid/5197/Default.aspx>.